

A Brain-Friendly Guide

Head First PHP & MySQL



Discover the secrets
behind dynamic,
database-driven sites

Avoid
embarrassing
mishaps with
web forms



Load all the key
syntax directly
into your brain



Hook up your
PHP and
MySQL code



Flex your scripting
knowledge with dozens
of exercises

O'REILLY®

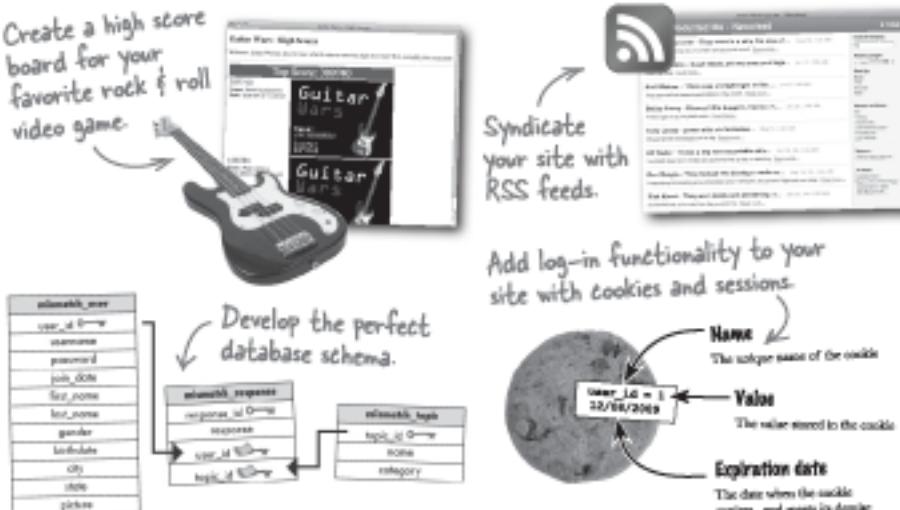
Lynn Beighley & Michael Morrison

Head First PHP & MySQL

Web Programming/PHP

What will you learn from this book?

Ready to take your static HTML web pages to the next level, and build database-driven sites using PHP and MySQL? Then *Head First PHP & MySQL* is your hands-on guide to getting dynamic sites running, fast. Get your hands dirty building real applications, ranging from a video game high-score message board to an online dating site. By the time you're through, you'll be validating forms, working with session IDs and cookies, performing database queries and joins, handling file I/O operations, and more.



Why does this book look so different?

We think your time is too valuable to spend struggling with new concepts. Using the latest research in cognitive science and learning theory to craft a multi-sensory learning experience, *Head First PHP & MySQL* uses a visually rich format designed for the way your brain works, not a text-heavy approach that puts you to sleep.

US \$44.99

CAN \$44.99

ISBN: 978-0-596-00630-3



5 4 4 9 9

Safari
Books Online

Free online edition
for 45 days with
purchase of this book.
Details on last page.

O'REILLY®
www.oreilly.com
www.headfirstlabs.com

"PHP and MySQL are two of today's most popular web development technologies, and this book shows readers why. Building a site without them is now as unthinkable as doing web design without CSS. This book is a great introduction and is laugh-out-loud funny. It's the book I wish I had learned from."

—Harvey Quamen,
Associate Professor
of English and
Humanities Computing,
University of Alberta

"Reading Head First PHP & MySQL is like taking a class from the 'cool' teacher. It makes you look forward to learning."

—Stephanie Liese,
Web Developer

Head First PHP & MySQL

by Lynn Beighley and Michael Morrison

Copyright © 2009 O'Reilly Media, Inc. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (safari.oreilly.com). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.**Series Creators:** Kathy Sierra, Bert Bates**Series Editor:** Brett D. McLaughlin**Editor:** Sanders Kleinfeld**Design Editor:** Louise Barr**Cover Designers:** Louise Barr, Steve Fehler**Production Editor:** Brittany Smith**Proofreader:** Colleen Gorman**Indexer:** Julie Hawks**Page Viewers:** Julien and Drew**Printing History:**

December 2008: First Edition.



Michael's nephew Julien
generously lent his
Superman powers to help
get this book finished.



Drew is, at this very
moment, installing
a new kitchen in
Lynn's new old house.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. The *Head First* series designations, *Head First PHP & MySQL*, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and the authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

No hardwood floors, UFOs, Elvis look-alikes, or virtual guitars were harmed in the making of this book. But a few broken hearts were mended thanks to some careful mismatching!

ISBN: 978-0-596-00630-3

[M]

For my parents, who frequently use web applications and are always there for me.

- Lynn Beighley

To Rasmus Lerdorf, who single-handedly sparked the language that would eventually become PHP as we know it now. Enduring proof that it really only takes one person to lead us all down a new, more enlightened path.

- Michael Morrison

the author(s)

Author(s) of Head First PHP & MySQL



Lynn Beighley is a fiction writer stuck in a technical book writer's body. Upon discovering that technical book writing actually paid real money, she learned to accept and enjoy it. After going back to school to get a Masters in Computer Science, she worked for the acronyms NRL and LANL. Then she discovered Flash, and wrote her first bestseller. A victim of bad timing, she moved to Silicon Valley just before the great crash. She spent several years working for Yahoo! and writing other books and training courses. Finally giving in to her creative writing bent, she moved to the New York area to get an MFA in Creative Writing. Her Head First-style thesis was delivered to a packed room of professors and fellow students. It was extremely well received, and she finished her degree, finished *Head First SQL*, and just finished *Head First PHP & MySQL*. Whew!

Lynn loves traveling, writing, and making up elaborate background stories about complete strangers. She's a little scared of UFOs.

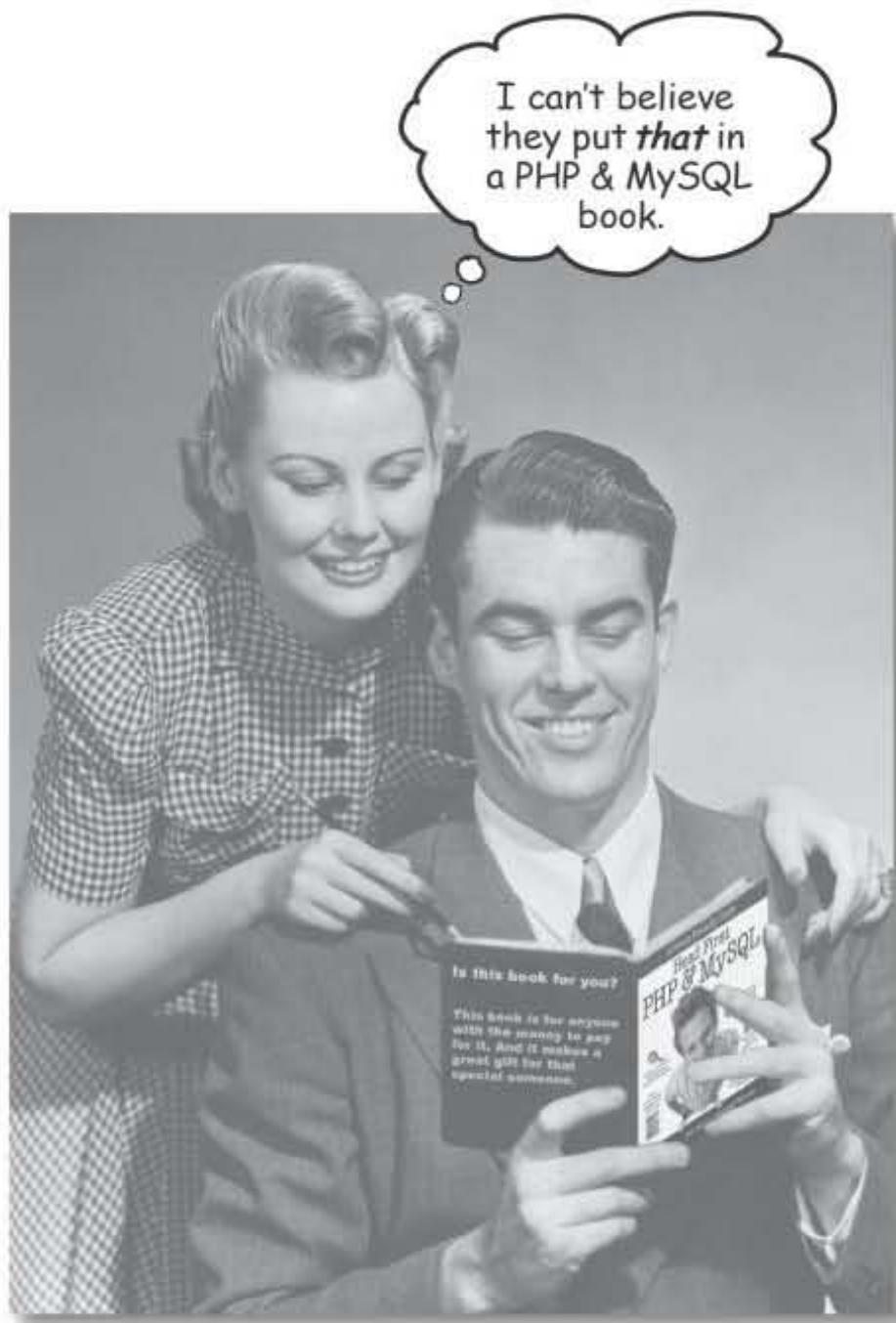


Michael Morrison has been a contributor to the online world even on his Commodore 64 way back when. While it was far less cool than it is these days, he baud later, he still marvels at how fast things have changed. Michael doesn't run a blog, but he is still very much involved in the many communities the tools we use to build them. He also finds time "official" time writing about web-related topics, having authored or co-authored over 20 books from mobile game programming to the Head First foray with *Head First Java*. When he looks back.

Michael is also the founder of Stalefish Labs (stalefishlabs.com), an entertainment company specializing in games, toys, and interactive media. He's been known to actually spend time skateboarding, playing ice hockey, and tending to his koi pond with his wife, Marlene, every once in a while.

how to use this book

Intro



In this section we answer the burning question: "So why DID they put that in a PHP & MySQL book?"

how to use this book

Who is this book for?

If you can answer “yes” to all of these:

- ① Are you a web designer with HTML or XHTML experience and a desire to take your web pages to the next level?
- ② Do you want to go beyond simple HTML pages to **learn, understand, and remember** how to **use PHP and MySQL to build web applications?**
- ③ Do you prefer **stimulating dinner party conversation** to **dry, dull, academic lectures?**

this book is for you.

Who should probably back away from this book?

If you can answer “yes” to any of these:

- ① **Are you completely unfamiliar with basic programming concepts like variables and loops?**
(But even if you've never programmed before, you'll probably be able to get the key concepts you need from this book.)
- ② Are you a kick-butt PHP web developer looking for a **reference book?**
- ③ Are you **afraid to try something different?** Would you rather have a root canal than mix stripes with plaid? Do you believe that a technical book can't be serious if it creates an alien abduction database?

this book is not for you.



[Note from marketing: this book is for anyone with a credit card.]

We know what you're thinking

“How can *this* be a serious PHP and MySQL book?”

“What’s with all the graphics?”

“Can I actually *learn* it this way?”

We know what your *brain* is thinking

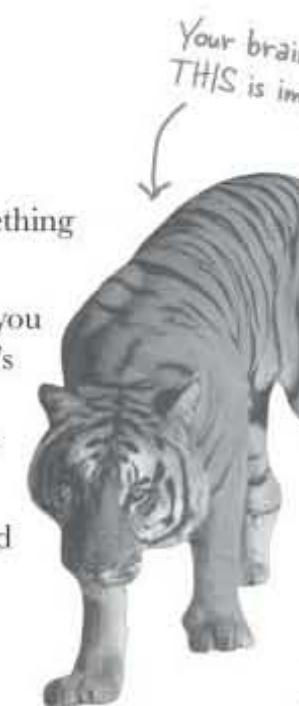
Your brain craves novelty. It’s always searching, scanning, *waiting* for something unusual. It was built that way, and it helps you stay alive.

So what does your brain do with all the routine, ordinary, normal things you encounter? Everything it *can* to stop them from interfering with the brain’s *real* job—recording things that *matter*. It doesn’t bother saving the boring things; they never make it past the “this is obviously not important” filter.

How does your brain *know* what’s important? Suppose you’re out for a day hike and a tiger jumps in front of you, what happens inside your head and body?

Neurons fire. Emotions crank up. *Chemicals surge*.

And that’s how your brain knows...



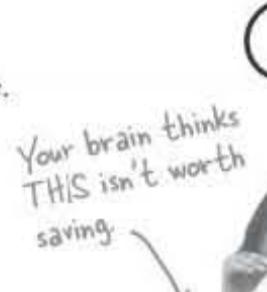
This must be important! Don't forget it!

But imagine you’re at home, or in a library. It’s a safe, warm, tiger-free zone. You’re studying. Getting ready for an exam. Or trying to learn some tough technical topic your boss thinks will take a week, ten days at the most.

Just one problem. Your brain’s trying to do you a big favor. It’s trying to make sure that this *obviously* non-important content doesn’t clutter up scarce resources. Resources that are better spent storing the really *big* things. Like tigers. Like the danger of fire. Like how to quickly hide the browser window with the YouTube video of space alien footage when your boss shows up.

And there’s no simple way to tell your brain, “Hey brain, thank you very much, but no matter how dull this book is, and how little I’m registering on the emotional Richter scale right now, I really *do* want you to keep this stuff around.”

UFO footage on YouTube is obviously more interesting to your brain than some computer book.



how to use this book

We think of a “Head First” reader as a learner.

So what does it take to *learn* something? First, you have to *get it*, then make sure you don’t *forget it*. It’s not about pushing facts into your head. Based on the latest research in cognitive science, neurobiology, and educational psychology, learning takes a lot more than text on a page. We know what turns your brain on.

Some of the Head First learning principles:



user_id = 1

Make it visual. Images are far more memorable than words alone, and make learning more effective (up to 89% improvement in recall and transfer studies). It also makes things more understandable. **Put the words within or near the graphics** rather than on the bottom or on another page, and learners will be up to twice as likely to solve problems related to the content.

Use a conversational and personalized style. In recent studies, students performed up to 40% better on post-learning tests if the content spoke directly to the reader, using a first-person, conversational style rather than taking a formal tone. Tell stories instead of lecturing. Use casual language. Don’t take yourself too seriously. Which would you pay more attention to: a stimulating dinner party companion, or a lecture?

Get the learner to think more deeply. In other words, unless you actively flex your neurons, nothing much happens in your head. A reader has to be motivated, engaged, and inspired to solve problems, draw conclusions, and generate new knowledge. And for that, you need challenges, exercises, and thought-provoking questions, and activities that involve both sides of the brain and multiple senses.

Get—and keep—the reader’s attention. We’ve all had the “I really want to learn this but I can’t stay awake past page one” experience. Your brain pays attention to things that are out of the ordinary, interesting, strange, eye-catching, unexpected. Learning a new technical topic doesn’t have to be boring. Your brain will learn much more quickly if it’s involved.

Touch their emotions. We now know that your ability to remember something is dependent on its emotional content. You remember what you care about. You remember what you feel. No, we’re not talking heart-wrenching stories about a boy and his dog. We’re talking emotions like surprise, curiosity, fun, “what the...?”, and the feeling of “I Rule!” that comes from solving a puzzle, learning something everybody else thinks is hard, or realizing you know something that “I’m more technical than thou” Bob from engineering doesn’t.



Small correction. We actually do have a heart-wrenching story about a boy and his dog – the dog was abducted by aliens, and you’ll be helping the boy find him!

Metacognition: thinking about thinking

If you really want to learn, and you want to learn more quickly and more deeply, pay attention to how you pay attention. Think about how you think. Learn how you learn.

Most of us did not take courses on metacognition or learning theory when we were growing up. We were *expected* to learn, but rarely *taught* to learn.

But we assume that if you're holding this book, you really want to learn how to build database-driven web sites with PHP and MySQL. And you probably don't want to spend a lot of time. If you want to use what you read in this book, you need to *remember* what you read. And for that, you've got to *understand* it. To get the most from this book, or *any* book or learning experience, take responsibility for your brain. Your brain on *this* content.

The trick is to get your brain to see the new material you're learning as Really Important. Crucial to your well-being. As important as a tiger. Otherwise, you're in for a constant battle, with your brain doing its best to keep the new content from sticking.

So just how **DO** you get your brain to treat PHP & MySQL like it was a hungry tiger?

There's the slow, tedious way, or the faster, more effective way. The slow way is about sheer repetition. You obviously know that you *are* able to learn and remember even the dullest of topics if you keep pounding the same thing into your brain. With enough repetition, your brain says, "This doesn't *feel* important to him, but he keeps looking at the same thing *over* and *over* and *over*, so I suppose it must be."

The faster way is to do **anything that increases brain activity**, especially different *types* of brain activity. The things on the previous page are a big part of the solution, and they're all things that have been proven to help your brain work in your favor. For example, studies show that putting words *within* the pictures they describe (as opposed to somewhere else in the page, like a caption or in the body text) causes your brain to try to make sense of how the words and picture relate, and this causes more neurons to fire. More neurons firing = more chances for your brain to *get* that this is something worth paying attention to, and possibly recording.

A conversational style helps because people tend to pay more attention when they perceive that they're in a conversation, since they're expected to follow along and hold up their end. The amazing thing is, your brain doesn't necessarily *care* that the "conversation" is between you and a book! On the other hand, if the writing style is formal and dry, your brain perceives it the same way you experience being lectured to while sitting in a roomful of passive attendees. No need to stay awake.

But pictures and conversational style are just the beginning...



how to use this book

Here's what WE did:

We used **pictures**, because your brain is tuned for visuals, not text. As far as your brain's concerned, a picture really *is* worth a thousand words. And when text and pictures work together, we embedded the text *in* the pictures because your brain works more effectively when the text is *within* the thing the text refers to, as opposed to in a caption or buried in the text somewhere.

We used **redundancy**, saying the same thing in *different* ways and with different media types, and *multiple senses*, to increase the chance that the content gets coded into more than one area of your brain.

We used concepts and pictures in **unexpected** ways because your brain is tuned for novelty, and we used pictures and ideas with at least *some emotional content*, because your brain is tuned to pay attention to the biochemistry of emotions. That which causes you to *feel* something is more likely to be remembered, even if that feeling is nothing more than a little **humor, surprise, or interest**.

We used a personalized, **conversational style**, because your brain is tuned to pay more attention when it believes you're in a conversation than if it thinks you're passively listening to a presentation. Your brain does this even when you're *reading*.

We included more than 80 **activities**, because your brain is tuned to learn and remember more when you **do** things than when you *read* about things. And we made the exercises challenging-yet-do-able, because that's what most people prefer.

We used **multiple learning styles**, because *you* might prefer step-by-step procedures, while someone else wants to understand the big picture first, and someone else just wants to see an example. But regardless of your own learning preference, *everyone* benefits from seeing the same content represented in multiple ways.

We include content for **both sides of your brain**, because the more of your brain you engage, the more likely you are to learn and remember, and the longer you can stay focused. Since working one side of the brain often means giving the other side a chance to rest, you can be more productive at learning for a longer period of time.

And we included **stories** and exercises that present **more than one point of view**, because your brain is tuned to learn more deeply when it's forced to make evaluations and judgments.

We included **challenges**, with exercises, and by asking **questions** that don't always have a straight answer, because your brain is tuned to learn and remember when it has to *work* at something. Think about it—you can't get your *body* in shape just by *watching* people at the gym. But we did our best to make sure that when you're working hard, it's on the *right* things. That **you're not spending one extra dendrite** processing a hard-to-understand example, or parsing difficult, jargon-laden, or overly terse text.

We used **people**. In stories, examples, pictures, etc., because, well, because *you're* a person. And your brain pays more attention to *people* than it does to *things*.



Cut this out and stick it
on your refrigerator.

Here's what YOU can do to bend your brain into submission

So, we did our part. The rest is up to you. These tips are a starting point; listen to your brain and figure out what works for you and what doesn't. Try new things.

① Slow down. The more you understand, the less you have to memorize.

Don't just *read*. Stop and think. When the book asks you a question, don't just skip to the answer. Imagine that someone really *is* asking the question. The more deeply you force your brain to think, the better chance you have of learning and remembering.

② Do the exercises. Write your own notes.

We put them in, but if we did them for you, that would be like having someone else do your workouts for you. And don't just *look* at the exercises. **Use a pencil.** There's plenty of evidence that physical activity *while* learning can increase the learning.

③ Read the “There are No Dumb Questions”

That means all of them. They're not optional sidebars—***they're part of the core content!*** Don't skip them.

④ Make this the last thing you read before bed. Or at least the last challenging thing.

Part of the learning (especially the transfer to long-term memory) happens *after* you put the book down. Your brain needs time on its own, to do more processing. If you put in something new during that processing time, some of what you just learned will be lost.

⑤ Drink water. Lots of it.

Your brain works best in a nice bath of fluid. Dehydration (which can happen before you ever feel thirsty) decreases cognitive function.

⑥ Talk about it. Out loud.

Speaking activates a different part of your brain. If you're trying to understand something, increase your chance of remembering it by talking it out loud. Better still, try to talk about it with someone else. You'll learn a lot, and you might uncover ideas you didn't even know were there when you were reading.

⑦ Listen to your brain.

Pay attention to whether your brain is overloaded. If you find yourself getting tired or losing focus, take a break. The surface of your brain can get tired or forget what you were learning. For a break. Once you go past the point of exhaustion, you won't learn faster by trying to push through it. In fact, you might even hurt the process.

⑧ Feel something.

Your brain needs to know that you're involved with the stories. Make up captions for the photos. Groan when you see something you don't like. Smiling is *still* better than feeling nothing.

⑨ Write a lot of code!

There's only one way to learn how to code: **a lot of code.** And that's what you'll do throughout this book. Coding is the only way to get good at it. It's also the only way to give you a lot of practice: there are lots of exercises that pose problems for you to solve. Just skip over them—a lot of people do—but when you solve the exercises, you'll learn a lot. And to each exercise—don't be afraid to ask for help or look at the **solution** if you get stuck! (It's okay to look at something small.) But try to look at the solution after you've solved the problem. And before you move on to the next exercise.

how to use this book

Read Me

This is a learning experience, not a reference book. We deliberately stripped out everything that might get in the way of learning whatever it is we're working on at that point in the book. And the first time through, you need to begin at the beginning, because the book makes assumptions about what you've already seen and learned.

We begin by teaching simple programming concepts and database connection basics, then more complicated PHP functions and MySQL statements, and finally more complex application concepts.

While it's important to create applications that allow users to add data to and retrieve data from your web application, before you can do that you need to understand the syntax of both PHP and MySQL. So we begin by giving you PHP and MySQL statements that you can actually try yourself. That way you can immediately do something with PHP and MySQL, and you will begin to get excited about them. Then, a bit later in the book, we show you good application and database design practices. By then you'll have a solid grasp of the syntax you need, and can focus on *learning the concepts*.

We don't cover every PHP and MySQL statement, function, or keyword.

While we could have put every single PHP and MySQL statement, function, and keyword in this book, we thought you'd prefer to have a reasonably liftable book that would teach you the most important statements, functions, and keywords. We give you the ones you need to know, the ones you'll use 95 percent of the time. And when you're done with this book, you'll have the confidence to go look up that function you need to finish off that kick-ass application you just wrote.

We support PHP 5 and MySQL 5.0.

Because so many people still use PHP 4 or 5, we avoid any PHP 4, 5, or 6 specific code wherever possible. We suggest you use PHP 5 or 6 and MySQL 5 or 6 while learning the concepts in this book. In developing this book, we focused on PHP 5 and MySQL 5, while making sure our code was compatible with later versions.

You need a web server that supports PHP.

PHP has to be run through a web server to work correctly. You need Apache or some other web server installed on your local machine or a machine to which you have some access so that you can run MySQL commands on the data. Check out Appendixes ii and iii for instructions on how to install and extend PHP and MySQL.

You can
with th
few mo
code. C
of APP

We use MySQL.

While there's Standard SQL language, in this book we focus on the particular syntax of MySQL. With only a few syntax changes, the code in this book should work with Oracle, MS SQL Server, PostgreSQL, DB2, and quite a few more Relational Database Management Systems (RDBMSs) out there. You'll need to look up the particular PHP functions and syntax if you want to connect to these other RDBMSs. If we covered every variation in syntax for every command in the book, this book would have many more pages. We like trees, so we're focusing on MySQL.

The activities are NOT optional.

The exercises and activities are not add-ons; they're part of the core content of the book. Some of them are to help with memory, some are for understanding, and some will help you apply what you've learned. ***Don't skip the exercises.*** The crossword puzzles are the only thing you don't *have* to do, but they're good for giving your brain a chance to think about the words and terms you've been learning in a different context.

The redundancy is intentional and important.

One distinct difference in a Head First book is that we want you to *really* get it. And we want you to finish the book remembering what you've learned. Most reference books don't have retention and recall as a goal, but this book is about *learning*, so you'll see some of the same concepts come up more than once.

The examples are as lean as possible.

Our readers tell us that it's frustrating to wade through 200 lines of an example looking for the two lines they need to understand. Most examples in this book are shown within the smallest possible context, so that the part you're trying to learn is clear and simple. Don't expect all of the examples to be ultra robust, or always complete—they are written specifically for learning, and aren't necessarily fully-functional.

We've placed all of the example code and applications on the Web so you can copy and paste parts of them into your text editor or MySQL Terminal, or upload them as-is to your own web server for testing. You'll find it all at
<http://www.headfirstlabs.com/books/hfphp/>

The Brain Power exercises don't have answers.

For some of them, there is no right answer, and for others, part of the learning experience of the Brain Power activities is for you to decide if and when your answers are right. In some of the Brain Power exercises, you will find hints to point you in the right direction.

the review team

The technical review team

Jereme Allen



David Briggs



Will Harris



Stephanie Liese



Technical Reviewers:

Jereme Allen is a senior level web developer with experience utilizing state of the art technologies to create web applications. He has nine plus years of experience utilizing PHP, MySQL, as well as various other frameworks, operating systems, programming languages and development software.

David Briggs is a technical author and software localization engineer living in Birmingham, England. When he's not being finicky about how to guide users through a particularly tricky piece of software, he likes nothing better than to get out in the local park with his wife, Paulette, and Cleo, the family dog.

Will Harris spends his days running an IT department that provides services to 11 companies on 4 continents, and he is the Vice President of the Las Vegas PASS (Professional Association for SQL Server) chapter. At night, he hops into a phone booth and puts on his web 2.0 suit, helping the designers and developers at Powered By Geek ensure that their data platforms are flexible, portable, maintainable, and FAST, using MySQL and Rails. He also enjoys spending time with his wife, Heather, his beautiful children, Mara and Ellie, and his dog, Swiper.

Stephanie Liese is a technical trainer and web developer in Sacramento, California. When she isn't extolling the virtues of standards compliant code or debugging a CSS layout, you will find her sweating it out in a hot yoga class.

Harvey Quamen



If **Steve Milano** isn't slinging code or playing punk rock with his band, The Rings, in some unventilated basement at home with his laptop, neglecting his dog, Ralph, and human companion, Linda,

Harvey Quamen gave up a career in academia to join the jet-setting, paparazzi-profile world of academia. He's currently a Professor of English and Human Studies at the University of Alberta, where he writes on cyberculture, 20th-century literature, and development—including PHP and MySQL.

Chris Shiflett is the Chief Technical Officer at OmniTI, where he leads the web development practice and guides web development best practices. Chris is a thought leader in the PHP and security communities—a widely-known speaker at conferences worldwide, and the founder of the PHP Security Consortium. His books include *PHP Security Best Practices* (O'Reilly) and *HTTP Developer's Guide*.

Acknowledgments

Our editors:

Many thanks go to **Brett McLaughlin** for the awesome storyboarding session that got us on the right track, and his ruthless commitment to cognitive learning.

The book would not exist if not for the heroic effort, patience, and persistence of **Sanders Kleinfeld**. He always managed to catch the balls, or was it cats, we were juggling when we inevitably dropped one (or three!), and we appreciate it. We hope he gets a chance to put his feet up for a couple of days before taking on another project as difficult as this one.

The O'Reilly team:



Thanks to **Lou Barr** for her phenomenal design skill, making this book such a visual treat.

Thanks also to **Brittany Smith** for all her hard work at the last minute, and to **Caitrin McCullough** for getting the example web sites up and running. And to **Laurie Petrycki** for having faith that we could write another great Head First book.



Lou Barr

And more:



Finally, a big thanks goes out to **Elvis Wilson** for putting together the alien YouTube videos for Chapter 12. Excellent job! Especially seeing as how he's merely a simple caveman art director.

safari books online

Safari® Books Online



When you see a Safari® icon on the cover of your favorite technology book that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at <http://safari.oreilly.com>.

1 add life to your static pages

It's Alive



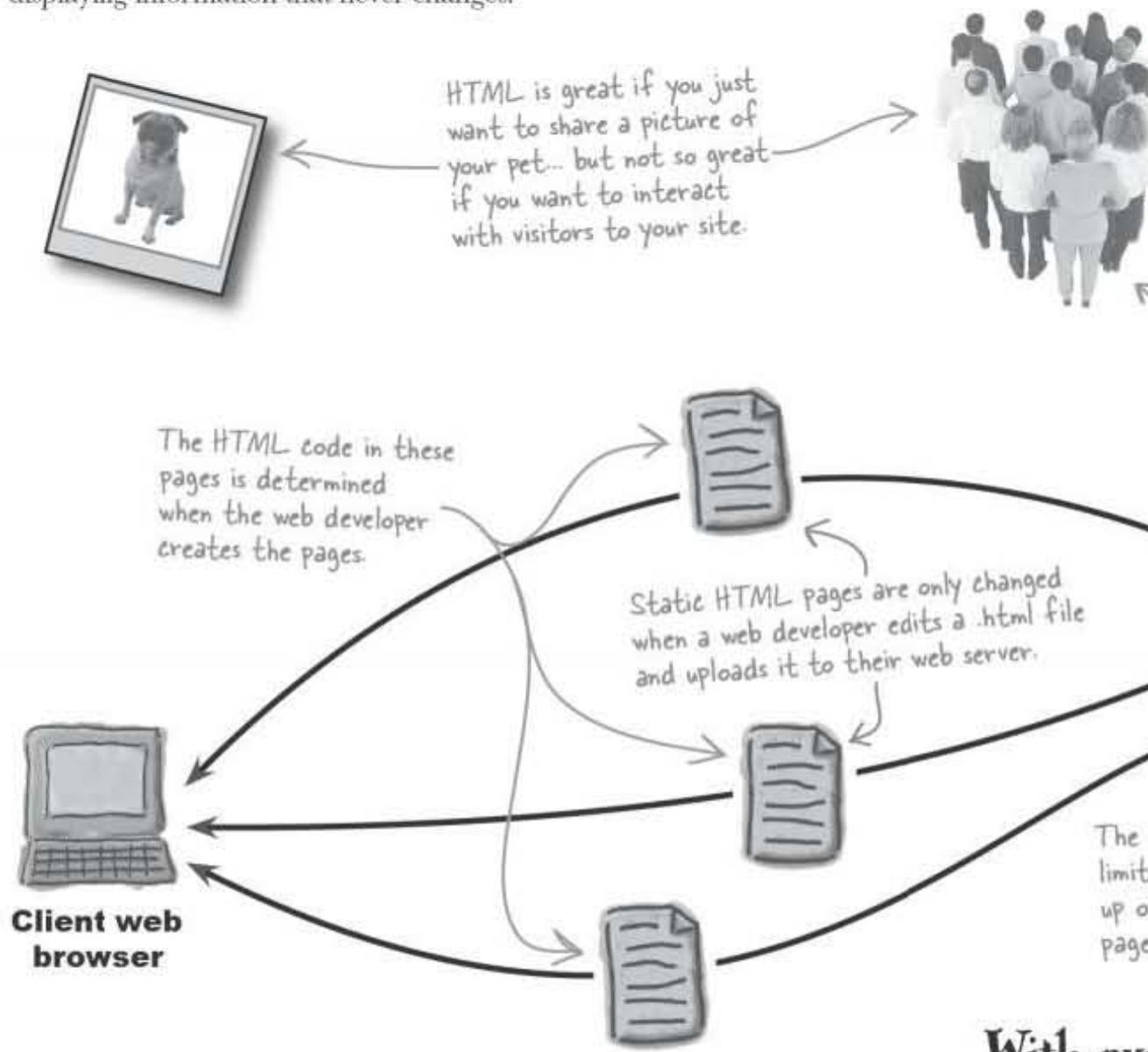
You've been creating great web pages with HTML, and a **sprinkling of CSS**. But you've noticed that visitors to your site can't do much other than passively look at the content on the pages. The communication's one-way, and you'd like to change that. In fact, you'd really like to **know what your audience is thinking**. But you need to be able to allow users to **enter information into a web form** so that you can find out what's on their minds. And you need to be able to **process the information** and **have it delivered to you**. It sounds as if you're going to need more than HTML to take your site to the next level.

sometimes just HTML isn't enough

HTML is static and boring

HTML's great for creating web pages, that much we already know. But what about when you need web pages that actually **do** something?

Suppose you need to search a database or send an email... what then? HTML falls short because it's a pretty lifeless language, designed for displaying information that never changes.



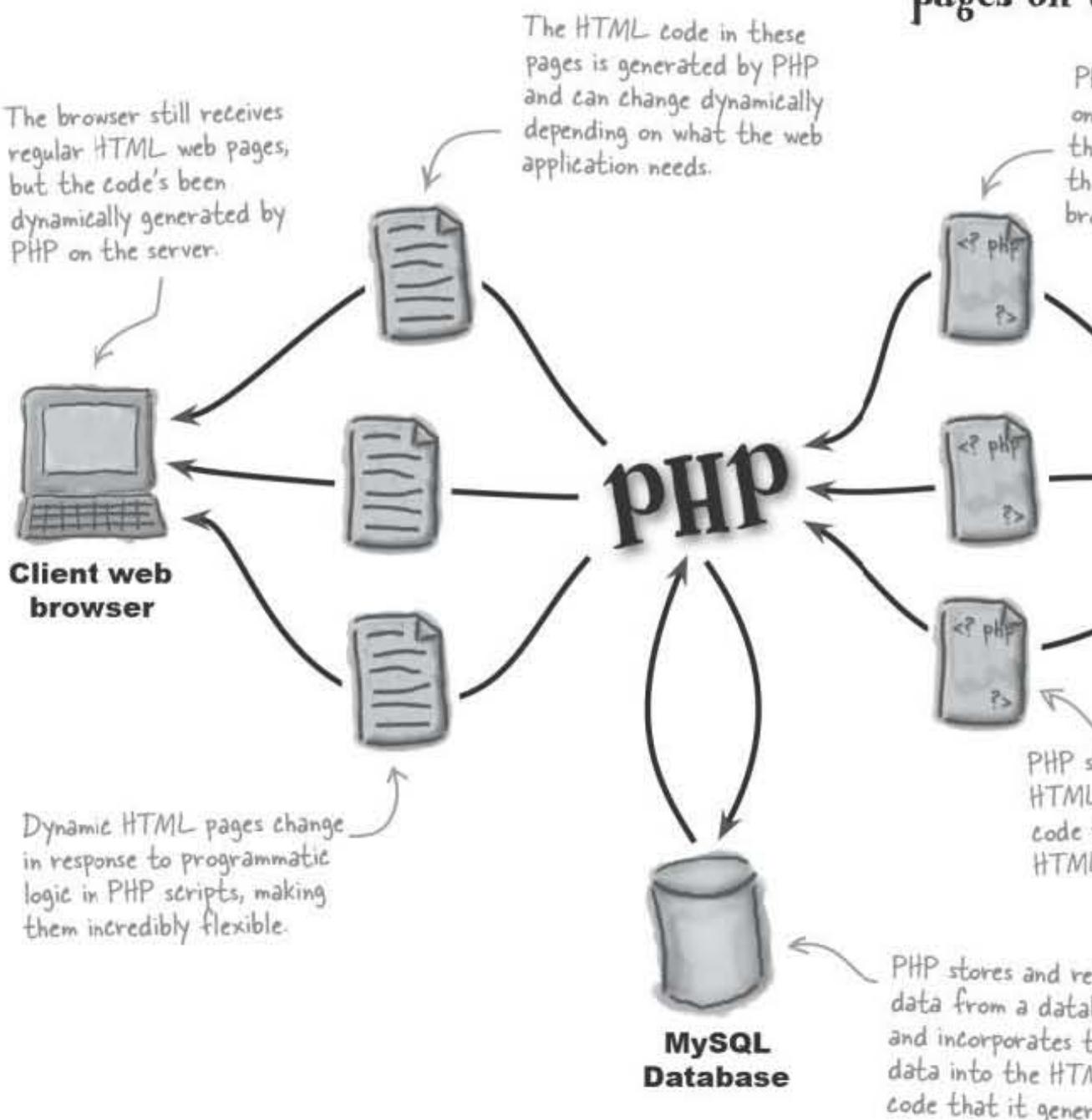
The web server's a big part of the problem with lifeless HTML since it serves as nothing more than a boring delivery mechanism. A browser requests a page, the server responds with HTML, end of story. To turn web sites into interactive web **applications**, the web server has to take on a new, more dynamic role... a role made possible by **PHP**.

With pure static HTML, web pages simply serve static HTML only displayed

PHP brings web pages to life

PHP allows you to manipulate web page content *on the server* just before a page is delivered to the client browser. It works like this: A PHP script runs on the server and can alter or generate HTML code at will. An HTML web page is still delivered to the browser, which doesn't know or care that PHP is involved in tweaking the HTML on the server.

**With PHP mix, the w
is able to
generate l
pages on t**



sending out an (internet) sos

Dogs in space

Meet Owen. Owen's lost his dog, Fang. But finding his dog isn't just a matter of searching the neighborhood. You see, Fang was abducted by aliens, which expands Owen's search to the entire galaxy. Owen knows some HTML and CSS, and he thinks a custom web site may help solve his problem by allowing other people to share their own alien abduction experiences.

But to get information from others, Owen's going to need a web form that's capable of receiving user input, lots of it, and notifying him about it. Not a problem—HTML has plenty of tags for whipping together web forms.



Details are sketchy, but we do know that Fang was whisked into the sky in a beam of light.

Owen knows some HTML and CSS and thinks he might be able to use the web to help track down his dog, Fang.

A form helps Owen get the whole story

Owen's new web site, AliensAbductedMe.com, aims to connect Owen with alien abductees who might be able to shed some light on Fang's disappearance. Owen knows he needs an HTML form to solicit abduction stories from visitors and that it must find out if they've run into Fang during their interstellar journeys. But he needs your help getting it up and running. Here's what he has in mind for the form.

Aliens Abducted Me - Report an Abduction

Share your story of alien abduction:

First name: _____
 Last name: _____
 What is your email address? _____
 When did it happen? _____
 How long were you gone? _____
 How many did you see? _____
 Describe them:
 What did they do to you?
 Have you seen my dog Fang?

Yes No

Anything else you want to add? _____

Report Abduction

This form is pure 100% Grade A HTML!

Here's the visitor's e-mail address.

Owen wants to receive an email message when the user submits the form.

Any comments go here.

What do you think of Owen's form?

Can you think of any problems Owen has with collecting alien abduction data using this form?

owen's form html

Forms are made of HTML

Owen's Report an Abduction form is built entirely out of HTML tags and attributes. There are text fields for most of the questions, radio buttons to find out if his visitor saw Fang, and a text area for additional comments. And the form is set up to deliver form data to Owen's email address.

*"mailto" is a protocol that allows
form data to be delivered via email.*

```

<p>Share your story of alien abduction:</p>
<form method="post" action="mailto:owen@aliensabductedme.com">
    <label for="firstname">First name:</label>
    <input type="text" id="firstname" name="firstname" /><br />
    <label for="lastname">Last name:</label>
    <input type="text" id="lastname" name="lastname" /><br />
    <label for="email">What is your email address?</label>
    <input type="text" id="email" name="email" /><br />
    <label for="whenithappened">When did it happen?</label>
    <input type="text" id="whenithappened" name="whenithappened" /><br />
    <label for="howlong">How long were you gone?</label>
    <input type="text" id="howlong" name="howlong" /><br />
    <label for="howmany">How many did you see?</label>
    <input type="text" id="howmany" name="howmany" /><br />
    <label for="aliendescription">Describe them:</label>
    <input type="text" id="aliendescription" name="aliendescription" />
    <label for="whattheydid">What did they do to you?</label>
    <input type="text" id="whattheydid" name="whattheydid" size="32" />
    <label for="fangspotted">Have you seen my dog Fang?</label>
    Yes <input id="fangspotted" name="fangspotted" type="radio" value="yes" />
    No <input id="fangspotted" name="fangspotted" type="radio" value="no" />
    <br />
    <label for="other">Anything else you want to add?</label>
    <textarea id="other" name="other"></textarea><br />
    <input type="submit" value="Report Abduction" name="submit" />
</form>

```

The form is bracketed with
open and close `<form>` tags.

No surprises here – the form
is pure, 100% HTML code!

If you need
creating Hi
out Chapt
HTML wit

Owe
of t
this
Owe
Your

This
to
"pos
the

Input ta
to expre

The type a
form actio

the

the

the

the

the

the



— Test DRIVE —

Try out the Report an Abduction form.

Download the code for the Report an Abduction web page from the Head First Labs web site at

www.headfirstlabs.com/books/hfphp. It's in the **chapter01** folder. The folder contains Owen's web form in **report.html**, as well as a style sheet (**style.css**) and an image of Fang (**fang.jpg**).

Open the **report.html** page in a text editor and change Owen's email address to yours. Then open the page in a web browser, enter some alien abduction information in the form, and click the Report Abduction button.

Aliens Abducted Me - Report an Abduction

Share your story of alien abduction:

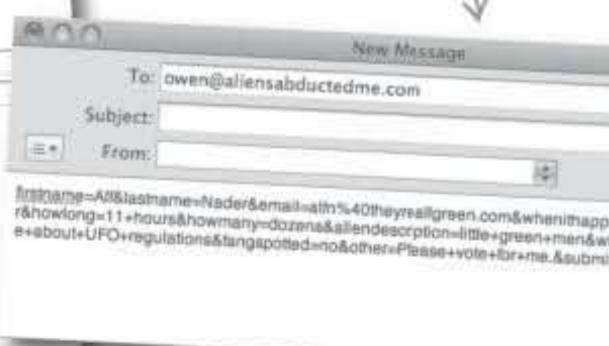
First name:	Alf
Last name:	Nader
What is your email address?	alfn@theyreallgreen.com
When did it happen?	last November
How long were you gone?	11 hours
How many did you see?	dozens
Describe them:	little green men
What did they do to you?	asked me about UFO regulations
Have you seen my dog Fang?	Yes <input type="radio"/> No <input checked="" type="radio"/>

Please vote for me.

Anything else you want to add?

Report Abduction

Submitting the form results in the form data getting emailed...sort of.



firstname=Alf&lastname=Nader&email=alfn%40theyreallgreen.com&whenithap...&howlong=11+hours&howmany=dozens&aliendescription=little+green+men&whattheydid=asked+me+about+UFO+regulations&fangspotted=no&other>Please+vote+for+me.&submit

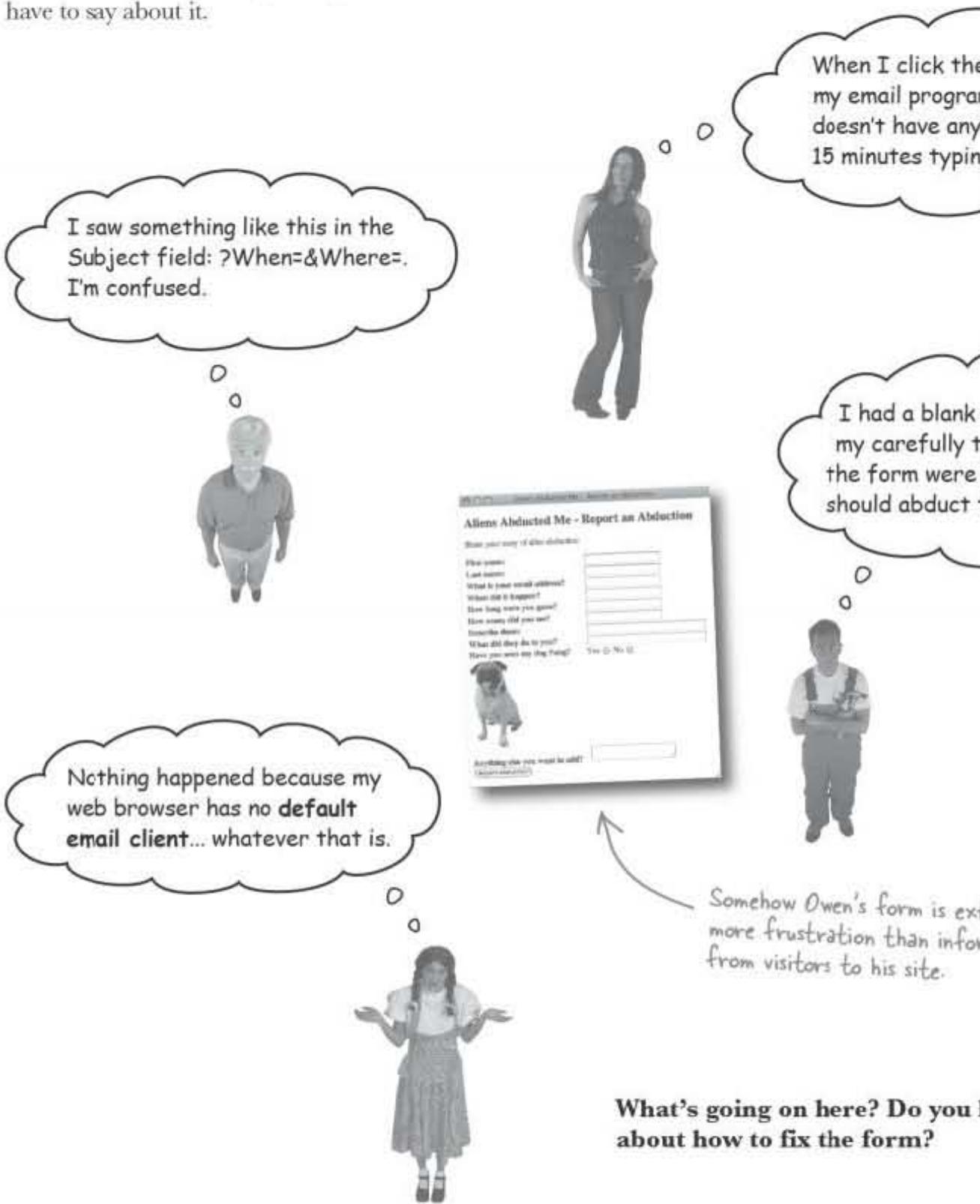
The form data isn't sent to Owen unless the user manually sends the weird looking email.

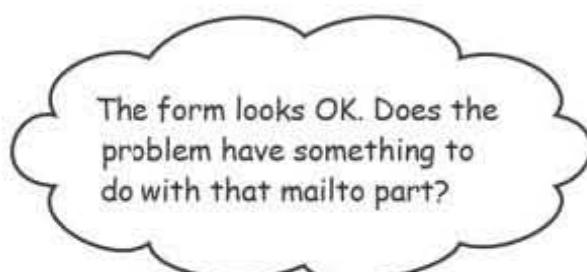
So, what do you think? Did you receive an email message in your Inbox?

mailto = bad idea

The HTML form has problems

Owen's Report an Abduction form is up and running, but he doesn't get much information from users. Is Fang's abduction really such an isolated incident... or is something wrong with his form? Let's see what the users have to say about it.





Yes. The HTML form code is fine, but mailto isn't a good way to deliver form data.

Owen's form is perfectly fine until the user clicks the Report Abduction button. At that point you rely on `mailto` to package up the form data into an email. But this email doesn't get sent automatically—it's created in the default email program on the user's computer instead. And the real kicker is that the user has to **send the email themselves** in order for the data to get sent to you! So you have no control over the email delivery, meaning that it may or may not successfully make the trip from your web form through the browser to their email client and back to you as an email message. Not cool.

You need a way to take control of the delivery of the web form. More specifically, you need PHP to package the form data into an email message and then make sure it gets sent. This involves shifting your attention from the **client** (HTML, `mailto`, etc.) to the **server** (PHP).

The form's wonderful until you click Report Abduction – then all bets are off!

Aliens Abducted Me - Report an Abduction

Share your story of alien abduction:

First name: _____
 Last name: _____
 What is your email address? _____
 When did it happen? _____
 How long were you gone? _____
 How many did you see? _____
 Describe them:
 What did they do to you?
 Have you seen my dog Fang?

Yes No

A small image of a dog is shown below the form.

Anything else you want to add? _____

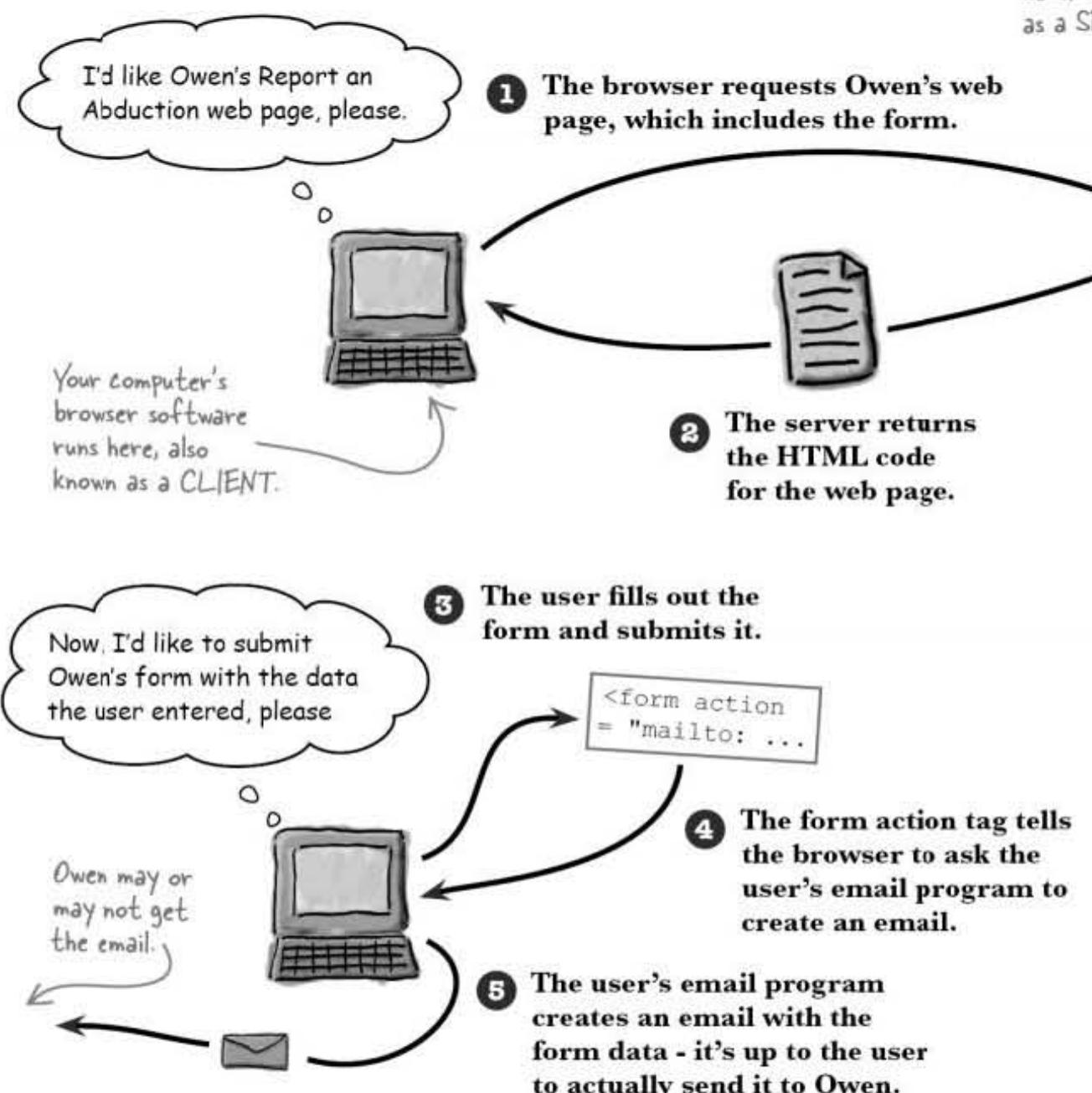
Submit

client-side versus server-side

HTML acts on the CLIENT

Owen's form is written in pure HTML with a `mailto` form action that attempts to send the form data via email. Although the `report.html` web page comes from a web server, it's filled out and processed entirely on the user's web browser.

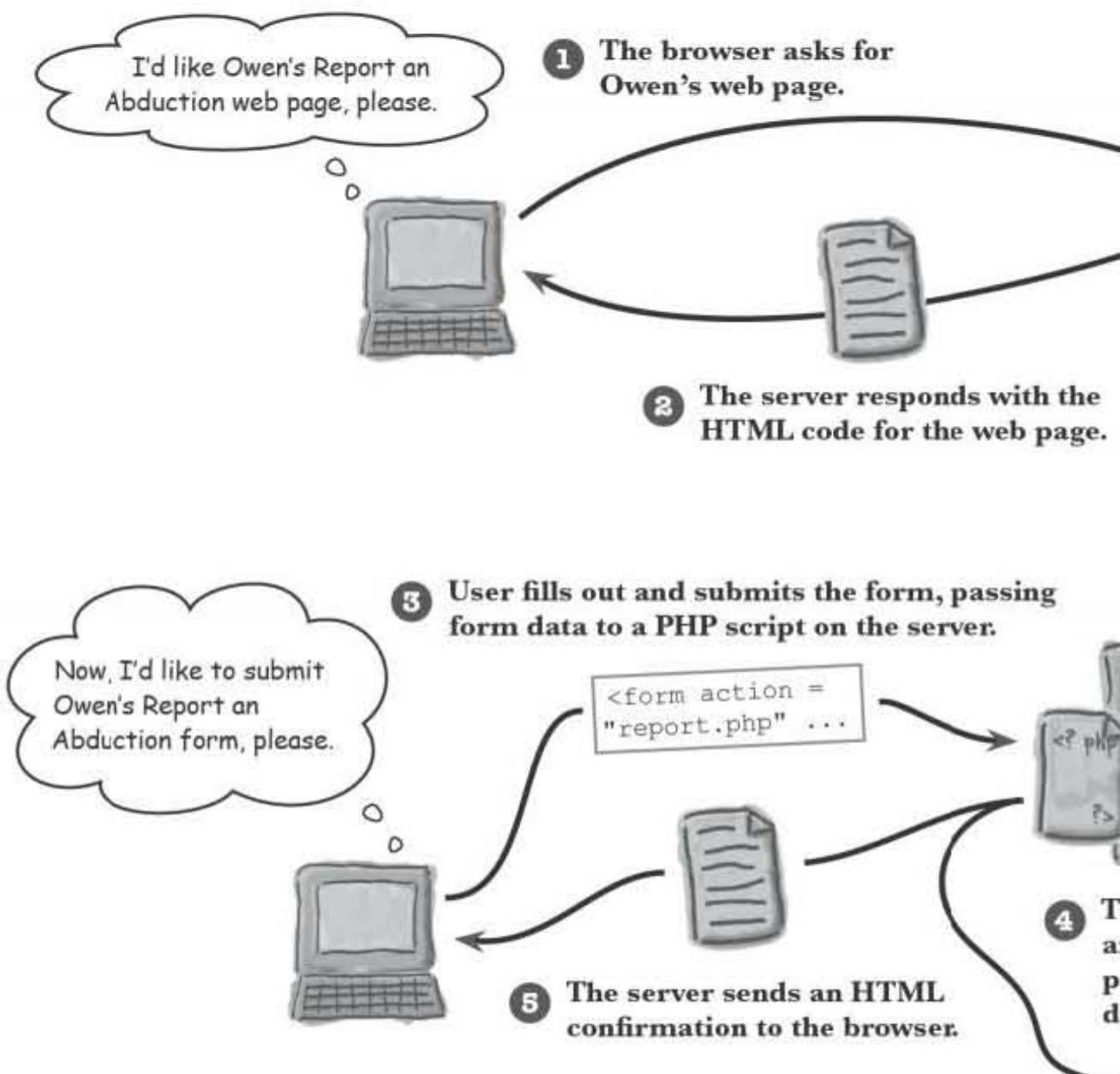
Owen's software here, also known as a **CLIENT**.



The server's role here is limited to just **delivering** the web page to the browser. When the user submits the form, the browser (client!) is left to its own devices to work out how to get the form data sent via email. The client isn't equipped to deliver form data—that's a job for the server.

PHP acts on the SERVER

PHP lets you take control of the data a user types into the form by emailing it to you **transparently**. The user types his abduction story into the form, hits the Report Abduction button, and he's done! The PHP code creates the email message, sends it to you, and then generates a web page confirmation for the user.



Check the boxes for where you think a PHP script belongs:

- Client Server Both Neither

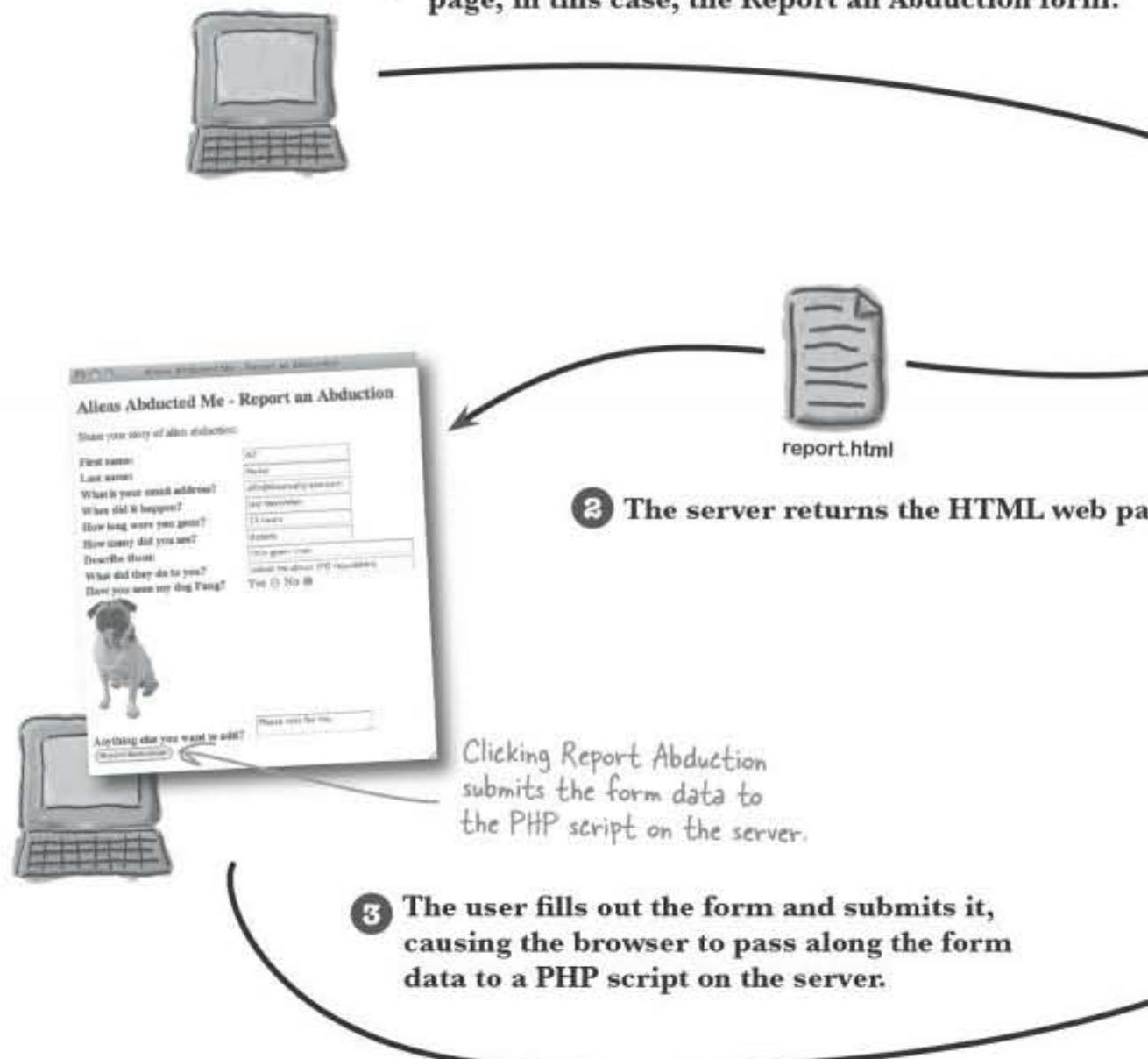
php is a server-side language

PHP scripts run on the server

PHP code runs on the server and is stored in **PHP scripts** that usually have a .php file extension. PHP scripts often look a lot like normal HTML web pages because they can contain both HTML code and CSS code. In fact, when the server runs a PHP script the end result is always pure HTML and CSS. So every PHP script ultimately gets turned into HTML and CSS once it's finished running on the server.

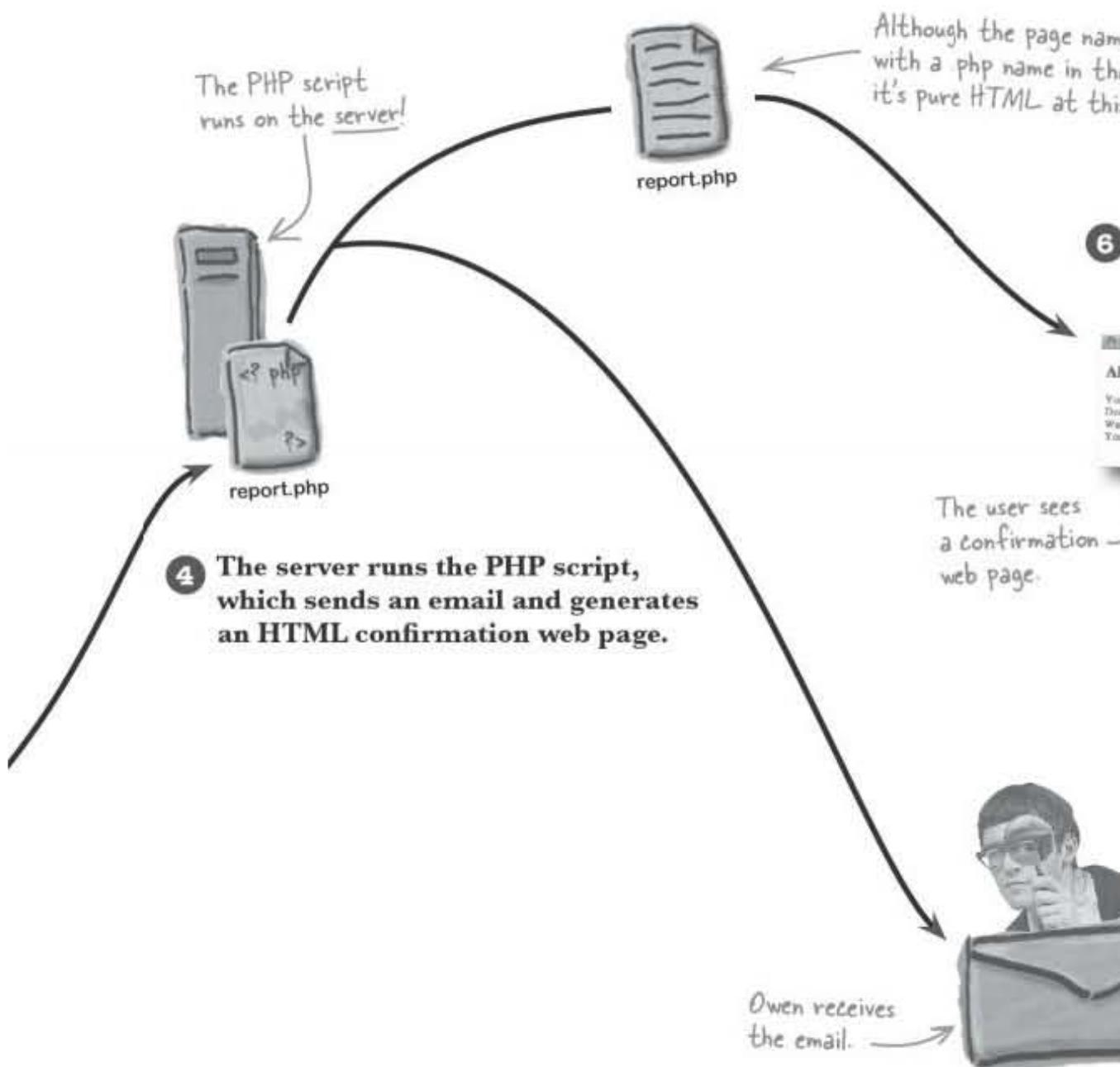
Let's take a closer look at how a PHP script changes the flow of Owen's web form.

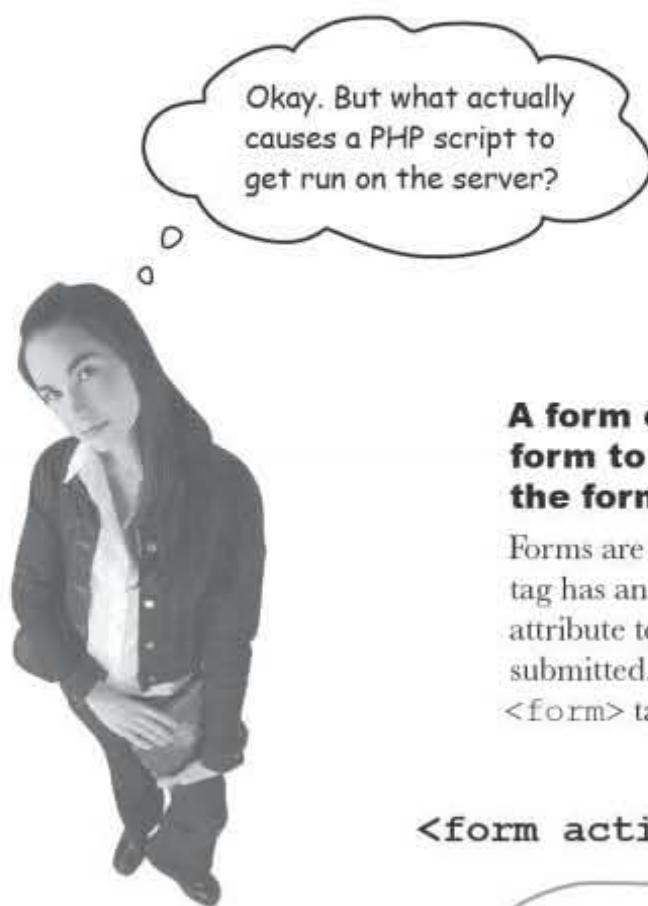
- 1 The client web browser requests an HTML web page, in this case, the Report an Abduction form.



PHP is a server-side programming language - it runs on a web server.

- 5 The server returns a pure HTML web page that was generated by the PHP script.



the form action attribute

A form element's action attribute is what connects the form to a PHP script, causing the script to run when the form is submitted.

Forms are created using the HTML <form> tag, and each <form> tag has an action attribute. Whatever filename you set the action attribute to is used by the web server to process the form when it is submitted. So if Owen's PHP script is named report.php, the <form> tag that connects it to the form looks like this:

<form action = "report.php" method="post">

This is the
filename of your
PHP script

When the user clicks the Report Abduction button in the form, the action attribute causes the report.php script to be run **on the server**, which then processes the form data.

The diagram illustrates the relationship between a computer system and its corresponding HTML files. On the left, there is a drawing of a computer monitor displaying a simple interface and a keyboard. Next to it is a drawing of a document with horizontal lines, labeled "report.html". A line connects this document to a large speech bubble on the right, which contains the actual HTML code. The code is as follows:

```

<html>
<head>
  <title>Aliens Abducted Me - Report an Abduction</title>
</head>
<body>
  <h2>Aliens Abducted Me - Report an Abduction</h2>
  <link rel="stylesheet" type="text/css" href="style.css" />
</body>
<h2>Aliens Abducted Me - Report an Abduction</h2>

<p>Share your story of alien abduction:</p>
<form method="post" action="report.php">
  <label for="firstname">First name:</label>
  <input type="text" id="firstname" name="firstname" /><br />

```

A callout arrow points from the "action='report.php'" part of the code back towards the "report.html" document, indicating the connection between the file name and the action attribute.

there are no
Dumb Questions

Q: What does PHP stand for?

A: PHP is an acronym that originally stood for Personal Home Pages. Somewhere along the way the acronym was changed to mean PHP: Hypertext Processor. The latter is considered a recursive acronym because it references itself—the acronym (PHP) is inside the acronym. Clever? Confusing? You decide!

Q: Even though my web browser shows that a web page has a name that ends in .php, it's still pure HTML? How is that?

A: It's possible because the page **originates** as PHP code on the server but is **transformed** into HTML code before making its way to the browser. So the server runs the PHP code and converts it into HTML code before sending it along to the browser for viewing. This means that even though a .php file contains PHP code, the browser never sees it—it only sees the HTML code that results from running the PHP code on the server.

Q: But don't all web pages originate on the server, even pure HTML pages in .html files?

A: Yes. All of the files for a web site are stored on the server—.html, .css, .php, etc. But they aren't all **processed** by the server. HTML and CSS files, as well as image files, are sent directly to the client browser without worrying about what's actually inside them. PHP files are different because they contain code that's processed and **run** on the web server. It's not the PHP code that's sent to the browser, it's the **results** of running the PHP code that are sent, and these results are pure HTML and CSS.

your first php script

Use PHP to access the form data

So Owen needs a PHP script that can get the alien abduction form information to him more reliably than the mailto technique. Let's create it. Don't worry about understanding everything yet—we'll get to that:

```

<html>
  <head>
    <title>Aliens Abducted Me - Report an Abduction</title>
  </head>
  <body>
    <h2>Aliens Abducted Me - Report an Abduction</h2>

    <?php
      $when_it_happened = $_POST['whenithappened'];
      $how_long = $_POST['howlong'];
      $alien_description = $_POST['description'];
      $fang_spotted = $_POST['fangspotted'];
      $email = $_POST['email'];

      echo 'Thanks for submitting the form.<br />';
      echo 'You were abducted ' . $when_it_happened;
      echo ' and were gone for ' . $how_long . '<br />';
      echo 'Describe them: ' . $alien_description . '<br />';
      echo 'Was Fang there? ' . $fang_spotted . '<br />';
      echo 'Your email address is ' . $email;

    ?>

  </body>
</html>

```

PHP scripts often start out looking a lot like HTML web pages.

Ah, here's where things get interesting – this is the beginning of the actual PHP code.

It's for including tags

Just like a normal web page, this PHP script finishes up by closing out open HTML tags.



— Test Drive —

Change Owen's form to use a PHP script to process the form data.

Create a new text file called **report.php**, and enter all of the code on the facing page. This is the script that will process Owen's web form.

The PHP script isn't connected to the form yet, so open the **report.html** page in a text editor and change the form action to **report.php** instead of **mailto**.

```
<form action = "report.php" method = "post">
```

Open the **report.html** page in a web browser, enter some alien abduction information in the form, and click Report Abduction.

Aliens Abducted Me – Report an Abduction

Aliens Abducted Me - Report an Abduction

Share your story of alien abduction:

First name:	All
Last name:	Nader
What is your email address?	ali@theyreallgreen.com
When did it happen?	last November
How long were you gone?	11 hours
How many did you see?	dozens
Describe them:	little green men
What did they do to you?	asked me about UFO regulations
Have you seen my dog Fang?	<input type="radio"/> Yes <input checked="" type="radio"/> No <input type="radio"/>



Anything else you want to add?

Aliens Abducted Me - Report an Abduction

```
report.php
Transitional//EN
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
<html xmlns="http://www.w3.org/1999/xhtml"
lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>Aliens Abducted Me - Report an Abduction</title>
</head>
<body>
<h2>Aliens Abducted Me - Report an Abduction</h2>

```

Do you think this is how the PHP script works?
Write down why or why not, and what you think.

.....
.....
.....

putting php scripts on the server

PHP scripts **MUST** live on a server!

Unless you happen to have a web server running on your local computer, the `report.php` script can't run when you submit the Report an Abduction form. Remember, PHP is a programming language, and it needs an environment to run in. This environment is a web server with PHP support. PHP scripts and web pages that rely on the scripts **must be placed on a real web server**, as opposed to just opening a script directly from a local file system.

If you do have an installed local web server with PHP support, you can test out PHP scripts on your local machine.

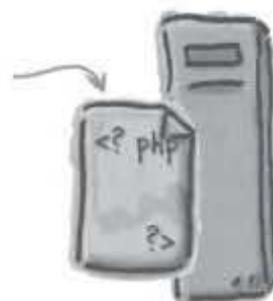
**Web browsers
know nothing about
PHP and, therefore,
have no ability to
run PHP scripts.**



Unlike HTML web pages, which can be opened locally in a web browser, PHP scripts must always be run through a URL from a web server.

This PHP script is just a bunch of meaningless code to the web browser.

The web server
understands this
PHP code and
runs the script!



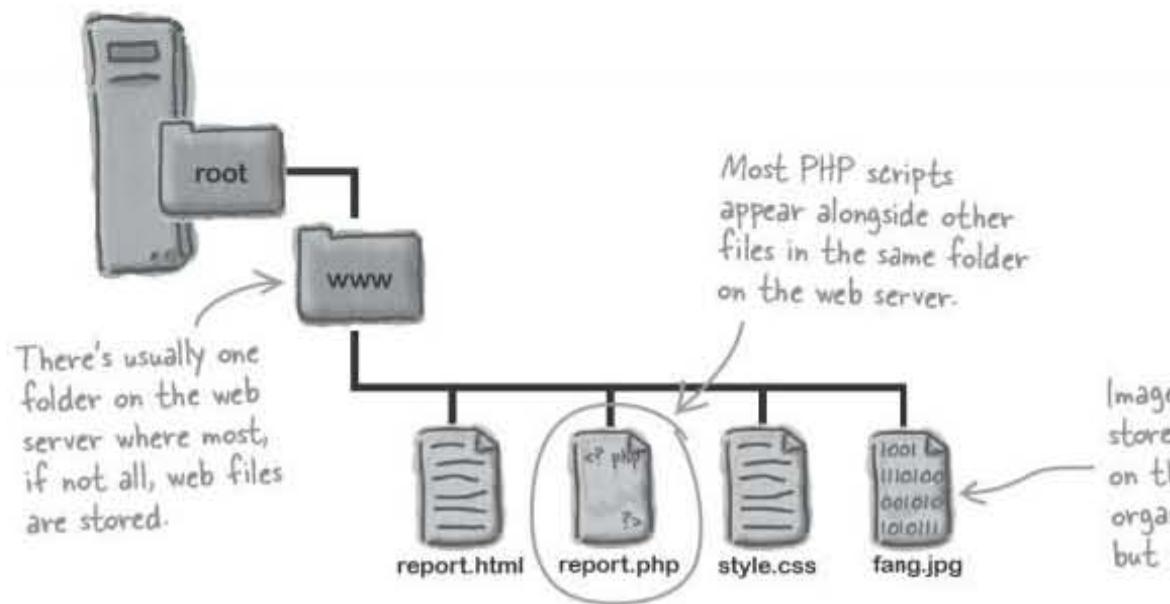
Web servers with PHP support are equipped to run PHP scripts and turn them into HTML web pages that web browsers can understand.

A quick way to tell if a web page is being delivered by a web server is to look for the URL starting with "http:". Web pages opened as local files always start with "file:".

PHP scripts must be run on a web server or they won't work.

Get your PHP scripts to the server

It's perfectly fine to create and edit PHP scripts on your local computer. But you need to put the files on a web server to run them. PHP files are often placed alongside HTML files on a web server. There's nothing magical about putting PHP scripts on a web server—just upload them to a place where your web pages can access them. Uploading files to a web server requires the help of a utility, such as an FTP (File Transfer Protocol) utility.



Uploading your PHP scripts to a web server isn't enough—that web server must also have PHP installed on it. Some web servers include PHP by default, some don't.

there are no
Dumb Questions

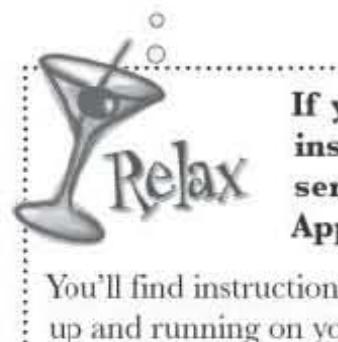
Q: How do I know if my web server has PHP installed?

A: You could ask your web administrator or web hosting company, or you could just perform a little test yourself. Create a text file called `test.php` and enter the following code into it:

```
<?php  
phpinfo();  
?>
```

This code asks PHP to display information about itself.

Now upload `test.php` to your web server, and then enter its URL into a web browser. If PHP is installed on your server, you'll see lots of detailed information about PHP, including its version. Bingo!



Remember to delete `phpinfo()` script when you're done, so...

test drive your php script

Test Drive

Upload the Report an Abduction files to a web server, and try out the form...again.

Upload report.html, report.php, style.css, and fang.jpg to a web server that has PHP installed. Enter the URL of the report.html page into your browser, fill out the form with alien abduction information, and click the Report Abduction button.

Aliens Abducted Me - Report an Abduction

Share your story of alien abduction:

First name:	Aif
Last name:	Nader
What is your email address?	aifn@theyreallygreen.com
When did it happen?	last November
How long were you gone?	11 hours
How many did you see?	dozens
Describe them:	little green men
What did they do to you?	asked me about UFO regulations
Have you seen my dog Fang?	Yes <input type="radio"/> No <input checked="" type="radio"/>

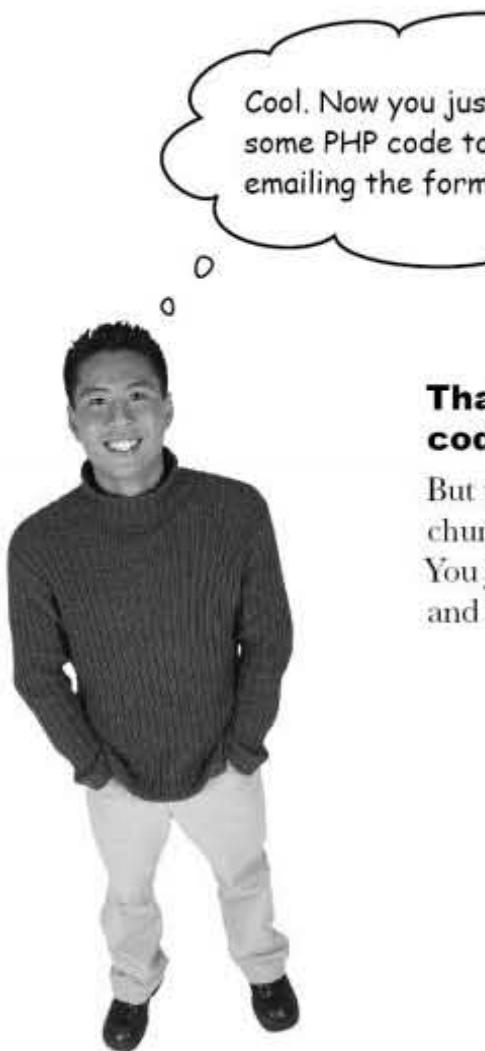


Anything else you want to add?

The PHP script works
displays form data
confirmation web page

Aliens Abducted Me - Report an Abduction

You were abducted last November and were gone for 11 hours.
Describe them:
Was Fang there? no
Your email address is aifn@theyreallygreen.com



Cool. Now you just need to add some PHP code to take care of emailing the form data.

That's right. The report.php script's still missing code to email the alien abduction data to Owen.

But that's not a problem because PHP offers a function, a pre-built chunk of reusable code, that you can use to send email messages. You just need to figure out what the email message needs to say and then use PHP to create and send it.

Time out! We don't even know how the original report.php script works, and now we're charging ahead into sending emails. This is like majorly overwhelming...hello?

It's true. Doing more with PHP requires knowing more about PHP.

So in order to add email functionality to Owen's report.php script, you're going to have to dig a little deeper into PHP and get a solid handle on how the script works up to this point.

how php code turns into html

The server turns PHP into HTML

A big part of understanding how a PHP script works is getting a handle on what happens to the script when it runs on the server. Most PHP scripts contain both PHP code and HTML code, and the PHP's run and turned into HTML before the server passes the whole thing off as HTML to the client web browser. In Owen's report.php script, PHP code generates most of the HTML content in the body of the confirmation page. The HTML code surrounding it is delivered unchanged.

```
<html>
<head>
    <title>Aliens Abducted Me - Report an Abduction</title>
</head>
<body>
    <h2>Aliens Abducted Me - Report an Abduction</h2>
```



```
<?php
$when_it_happened = $_POST['whenithappened'];
$how_long = $_POST['howlong'];
$alien_description = $_POST['aliendescription'];
$fang_spotted = $_POST['fangspotted'];
$email = $_POST['email'];

echo 'Thanks for submitting the form.<br />';
echo 'You were abducted ' . $when_it_happened;
echo ' and were gone for ' . $how_long . '<br />';
echo 'Describe them: ' . $alien_description . '<br />';
echo 'Was Fang there? ' . $fang_spotted . '<br />';
echo 'Your email address is ' . $email;

?>
```



```
</body>
</html>
```

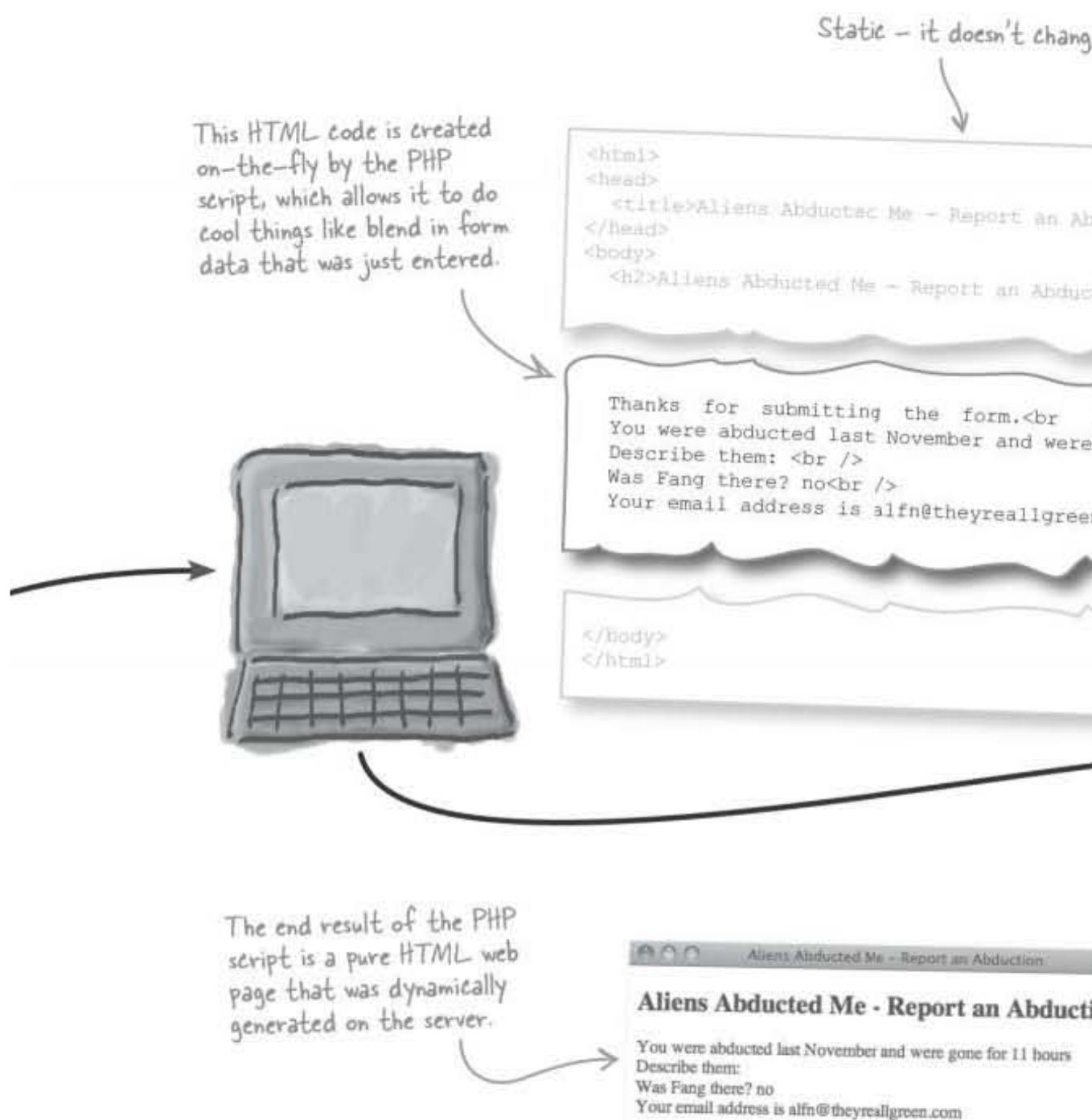


report.php

More static HTML code, which the server passes along to the browser with no changes.

This HTML code is passed unchanged to the browser.

This PHP code is processed by the server and generates HTML code containing dynamic content that was entered into the form.



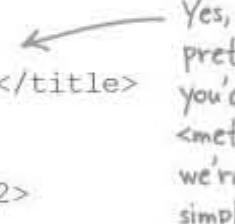
anatomy of owen's php script

Deconstructing Owen's PHP script

The `report.php` script is triggered by the Report an Abduction form, and its job (at the moment) is to take the form data and generate a confirmation web page. Let's see how.

The first chunk of code is pure HTML. It just sets up the page we're building, including a few HTML tags required of all pages.

```
<html>
<head>
    <title>Aliens Abducted Me - Report an Abduction</title>
</head>
<body>
    <h2>Aliens Abducted Me - Report an Abduction</h2>
```



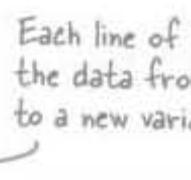
Here's where things start to get interesting. We're ready to break out of HTML code and into PHP code. The `<?php` tag opens a section of PHP code—everything following this tag is pure PHP.

`<?php`

From here on, PHP code...at least to the closing ?

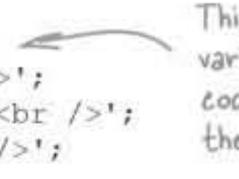
This code grabs the form data and stores it away in individual variables so that we can easily access it later. **PHP variables** allow you to store values, be they numbers, text, or other kinds of data.

```
$when_it_happened = $_POST['whenithappened'];
$how_long = $_POST['howlong'];
$alien_description = $_POST['description'];
$fang_spotted = $_POST['fangspotted'];
$email = $_POST['email'];
```



Now we're talking! Here the variables we just created are put to work by inserting them into dynamically generated HTML code. The `echo` command outputs HTML code that gets returned directly to the web browser.

```
echo 'Thanks for submitting the form.<br />';
echo 'You were abducted ' . $when_it_happened;
echo ' and were gone for ' . $how_long . '<br />';
echo 'Describe them: ' . $alien_description . '<br />';
echo 'Was Fang there? ' . $fang_spotted . '<br />';
echo 'Your email address is ' . $email;
```



The `?>` tag matches up with `<?php` and closes up a section of PHP code. From here on, we're back to normal HTML code.

`?>`

This ends the PHP code, so we're back to normal HTML code.

Now wrap up the page by closing out the HTML tags we opened earlier.

```
</body>
</html>
```

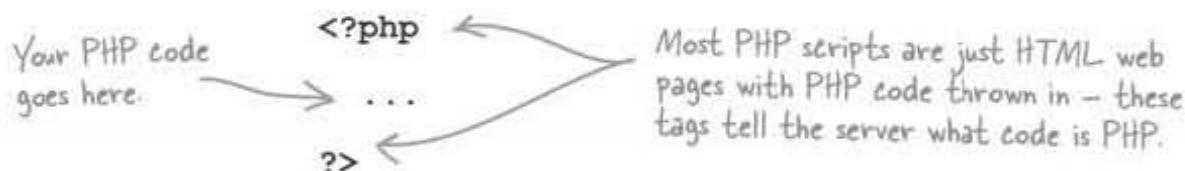
Don't forget, we're generating an HTML web page, so wrap up the HTML code.

code

A few PHP rules to live by

Owen's report.php script reveals a few fundamental rules of the PHP language that apply to all PHP scripts. Let's take a look at them.

- PHP code is always enclosed by <?php and ?>.



- Every PHP statement must end with a semicolon (;).

echo 'Thanks for submitting the form.
';

If your code ever breaks, check to make sure you haven't forgotten a semicolon. It happens more often than you'd think.

The semicolons know that end of a statement.

- If there is any PHP code in a web page, it's a good idea to name the file on the web server with .php, not .html.



This isn't a deal breaker, but it's a good idea to name PHP scripts with a .php file extension.

- PHP variable names must begin with a dollar sign (\$).

\$email = \$_POST['email'];

The dollar sign clearly identifies a PHP variable, which stores information within a PHP script.

Given the variables used in the report.php script, are there any other PHP rules pertaining to variables?

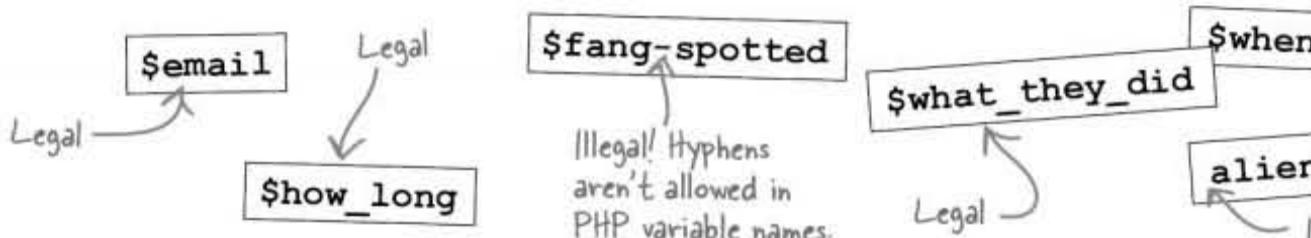
variable naming rules

Finding the perfect variable name

In addition to starting with a \$, PHP variable names are also case-sensitive. But that's not all—there are other important rules governing how you name variables. Some of these rules are syntax rules, meaning your code will break if you ignore them, while other rules are just good ideas passed down from wise old PHP coders.

Let's start with the official rules that will absolutely cause problems if you ignore them when naming variables. Follow these rules to create **legal** variable names.

- The first character must be a dollar sign (\$).** *Got it!*
- A variable name must be at least one character in length.**
- The first character after the dollar sign can be a letter or an underscore (_), and characters after that can be a letter, an underscore, or a number.**
- Spaces and special characters other than _ and \$ are not allowed in any part of a variable name.**



These rules will stop your code working if you don't follow them, but there are a couple more rules that are good to follow as more of a coding convention. These rules help make PHP code a little more consistent and easier to read.

- Use all lowercase for variable names.**
- Separate words in a multi-word variable name with underscores.**

These last two rules won't break your code if you ignore them, and you'll certainly run across PHP code that doesn't adhere to them yet works just fine. This is because they are just a stylistic convention—but they will serve you well as you begin creating and naming variables of your own.

there are no
Dumb Questions

Q: Does it matter whether I put PHP commands in uppercase or lowercase?

A: Yes and no. For the most part, PHP isn't case-sensitive, so you can get away with mixing the case of most commands. That means you can use `echo`, `ECHO`, or `EchC` when echoing content. However, as a matter of convention, it's a very good idea to be consistent with case in your scripts. Most PHP coders prefer lowercase for the vast majority of PHP code, which is why you'll see `echo` used throughout the example code in the book.

Q: So even if it's a bad coding convention, I can mix and match the case of PHP code?

A: No, not entirely. The huge exception to the case insensitivity of PHP is variable names, which apply to data storage locations that you create. So let's take the `$email` variable used in the Report an Abduction script as an example. This variable name is case-sensitive, so you can't refer to it as `$EMAIL` or `$eMail`. All variable names in PHP are case-sensitive like this, so it's important to name variables carefully and then reference them consistently in your code. More on variable names in just a moment.

Q: Is it really OK to put both PHP and HTML code in the same file?

A: Absolutely. In fact, in many cases it's absolutely necessary to do so.

Q: Why would I want to do that?

A: Because the whole idea behind a web server is to serve up HTML web pages to browsers. PHP doesn't change that fact. What PHP allows you to do is change the HTML content on the fly with things like today's date, data pulled from a database, or even calculated values such as the order total in a shopping cart. So PHP allows you to manipulate the HTML that goes into web pages, as opposed to them just being created statically at design time. It's very common to have HTML code for a page with PHP code sprinkled throughout to plug in important data or otherwise alter the HTML programmatically.

Q: Does PHP code embedded in an HTML file have to be on its own line, or can I embed it in an HTML line, like as part of an HTML tag attribute?

A: Other than needing to place your PHP code within the `<?php` and `?>` tags, there are no restrictions in how you embed it in HTML code. In fact, it's often necessary to wedge a piece of PHP code into the middle of HTML code, like when you're setting the attribute of an HTML tag. This is a perfectly legitimate usage of PHP.

Q: I've seen PHP code that's enclosed by `<?` as the start tag instead of `<?php`. Is that right?

A: Not really. Technically speaking, it's legal, but it isn't recommended. A server setting must be enabled for the short open tag (`<?`) to work. The usual `<?php` tag always works, so it's better to use that and know that your code will just work.

Q: If a web server receives an HTML code to a URL, what URLs show the webpage.php?

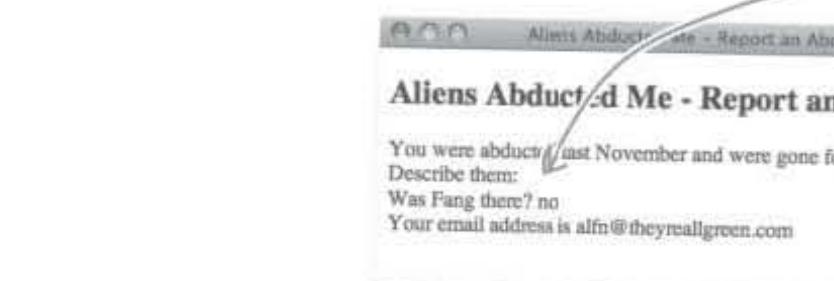
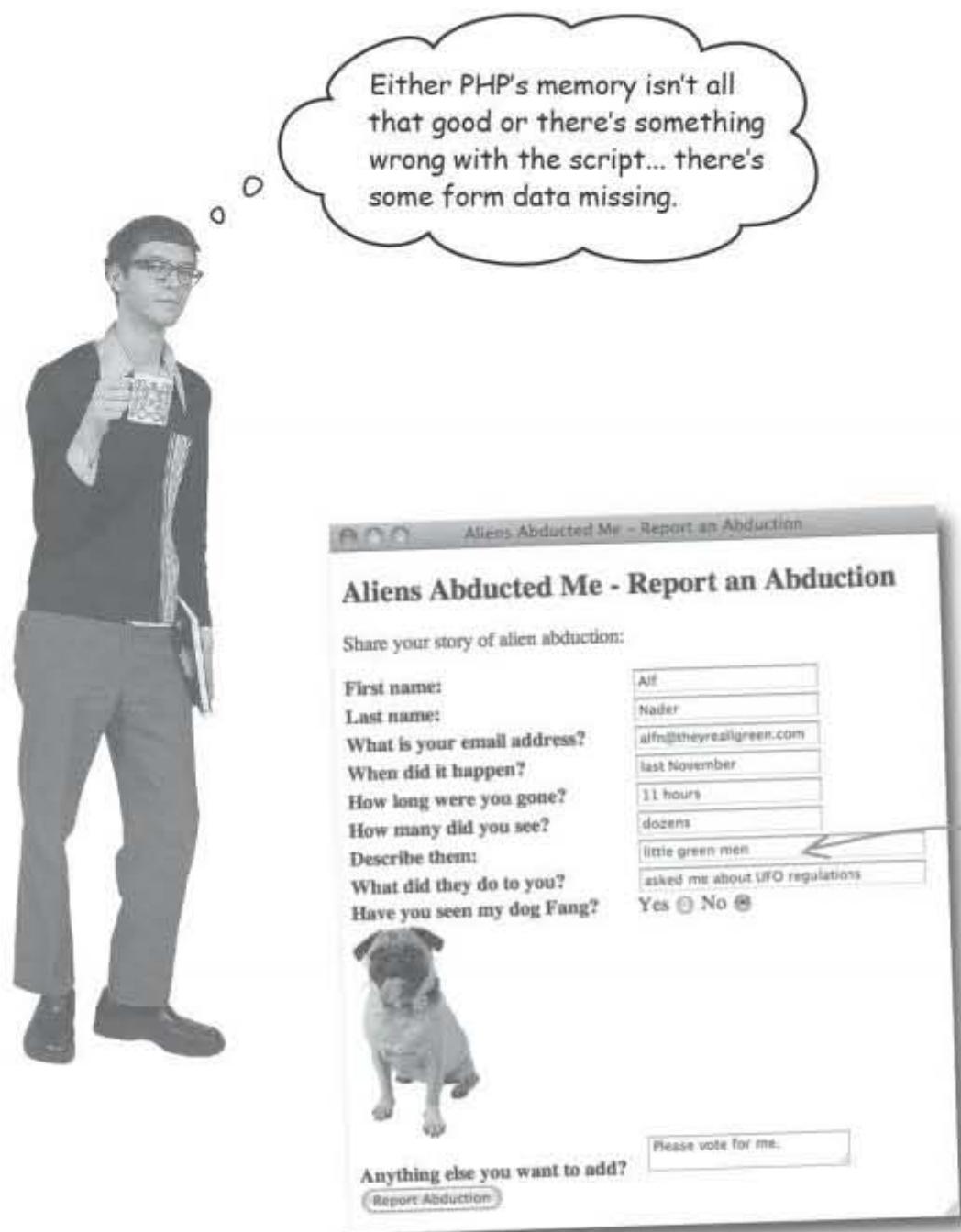
A: Remember, the result of a request involving a request and a response. URL is the basis of content returned in response. PHP is like normal HTML entered into the pages, or as form data in the URL for a PHP file of the PHP script.

The other half of the response from the server is the resulting code that the script. Since most of the HTML code, it means that a URL references which causes PHP to run on the server, ultimately outputting content being rendered.

Q: Can PHP handle all kinds of data?

A: Absolutely. PHP can store Boolean (true/false) data, where data can be either integer (decimal). There is also a collection of data types that associate a collection of objects. An object is used to manipulate objects covered a little later. Objects are tackled here. An object is also a special kind of variable that has been considered NULL.

add owen's missing data





Sharpen your pencil

There's a problem with the alien description form report.php script. Circle the lines of code that you think are causing the problem, and write down what they do. Any ideas?

```

<html>
<head>
    <title>Aliens Abducted Me - Report an Abduction</title>
</head>
<body>
    <h2>Aliens Abducted Me - Report an Abduction</h2>

    <?php
        $when_it_happened = $_POST['whenithappened'];
        $how_long = $_POST['howlong'];
        $alien_description = $_POST['description'];
        $fang_spotted = $_POST['fangspotted'];
        $email = $_POST['email'];

        echo 'Thanks for submitting the form.<br />';
        echo 'You were abducted ' . $when_it_happened;
        echo ' and were gone for ' . $how_long . '<br />';
        echo 'Describe them: ' . $alien_description . '<br />';
        echo 'Was Fang there? ' . $fang_spotted . '<br />';
        echo 'Your email address is ' . $email;
    ?>

    </body>
</html>

```



report.php

sharpen solution

Sharpen your pencil Solution

There's a problem with the alien description form report.php script. Circle the lines of code that you think might be causing the problem, and write down what they do. Any ideas?

This line of code grabs the alien description from the HTML form field and stores it in a PHP variable named \$alien_description.

This code combines the alien description with some other text and HTML code, and outputs all of it to the browser.

```
<html>
<head>
    <title>Aliens Abducted Me - Report an Abduction</title>
</head>
<body>
    <h2>Aliens Abducted Me - Report an Abduction</h2>
    <?php
        $when_it_happened = $_POST['whenithappened'];
        $how_long = $_POST['howlong'];
        $alien_description = $_POST['description'];
        $fang_spotted = $_POST['fangspotted'];
        $email = $_POST['email'];

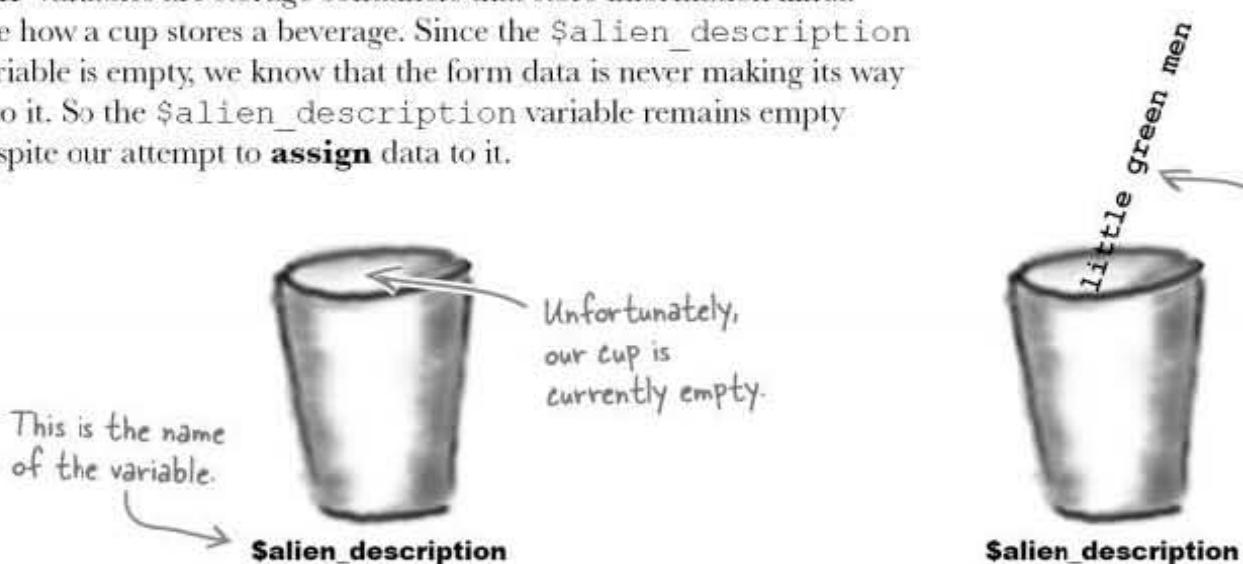
        echo 'Thanks for submitting the form.<br />';
        echo 'You were abducted ' . $when_it_happened . '<br />';
        echo 'and were gone for ' . $how_long . '<br />';
        echo 'Describe them: ' . $alien_description . '<br />';
        echo 'Was Fang there? ' . $fang_spotted . '<br />';
        echo 'Your email address is ' . $email;
    ?>

    </body>
</html>
```

For some reason the \$alien_description variable appears to be empty...not good.

Variables are for storing script data

PHP variables are storage containers that store information kinda like how a cup stores a beverage. Since the `$alien_description` variable is empty, we know that the form data is never making its way into it. So the `$alien_description` variable remains empty despite our attempt to **assign** data to it.



One way to fix the script would be to just assign the exact string we're expecting to the `$alien_description` variable, like this:

```
$alien_description = 'little green men';
```

The equal sign tells PHP to assign the value on the right to the variable on the left.

Pieces of text in PHP, also known as strings, must always be enclosed by quotes, either single quotes or double quotes.

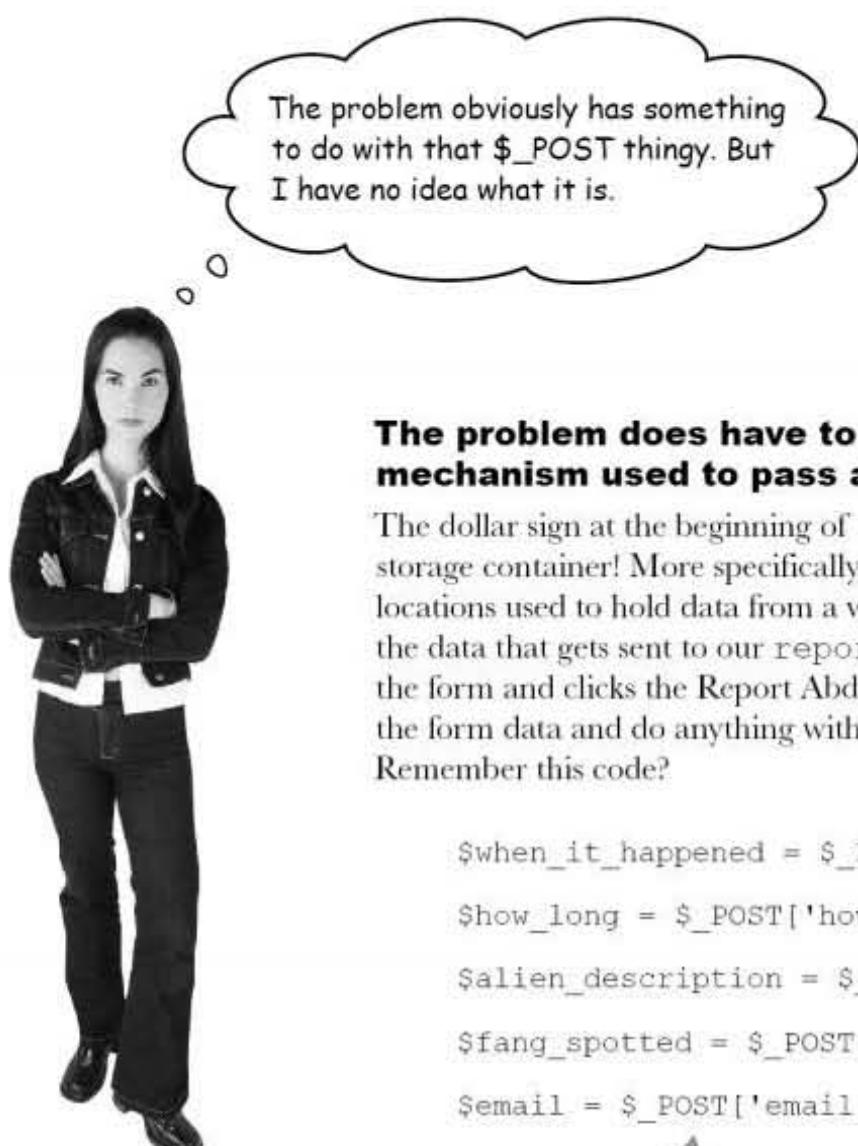
This code works in that it most definitely stores the text 'little green men' in the `$alien_description` variable. But we solved one problem by creating another one—this code causes the alien description to always be the same regardless of what the user enters into the form.



Somehow the assignment of alien description to the `$alien_description` variable is comin

`$alien_description = $_POST['desc']`

What do you think this code is doing wrong?

all about \$_POST

The problem does have to do with \$_POST, which is the mechanism used to pass along form data to a script.

The dollar sign at the beginning of `$_POST` is a clue... `$_POST` is a storage container! More specifically, `$_POST` is a **collection** of locations used to hold data from a web form. In Owen's case, it's the data that gets sent to our `report.php` script when someone fills out the form and clicks the Report Abduction button. So in order to get the form data and do anything with it, we have to go through `$_POST`. Remember this code?

```
$when_it_happened = $_POST['whenithappened'];
$how_long = $_POST['howlong']; ←
$alien_description = $_POST['description'];
$fang_spotted = $_POST['fangspotted'];
$email = $_POST['email'];
```

↑
Same deal here, except
the email form data is
being grabbed and stored
away in the \$email variable.

The pi
holding
the ab
to the

So the data in each field of the Report an Abduction form is accessed using `$_POST`. But what exactly is `$_POST`... a variable?

`$_POST` is a special variable that holds form data

`$_POST` is a special variable that is known as a **superglobal** because it is built into PHP and is available throughout an entire script. `$_POST` already exists when your script runs—you don't create it like you do other PHP variables.

The `$_POST` superglobal holds each piece of data entered into the form.

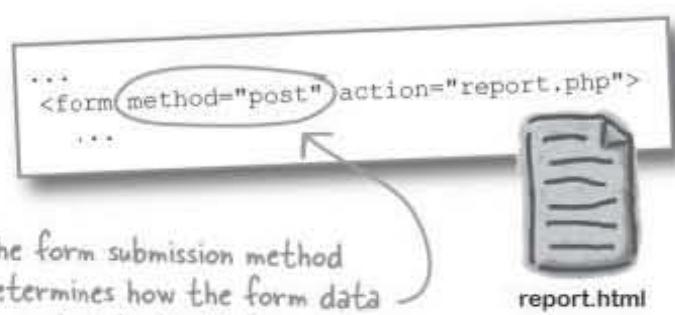
```
<html>
<head>
<title>Aliens Abducted Me - Report an Abduction</title>
</head>
<body>
<h2>Aliens Abducted Me - Report an Abduction</h2>
<?php
$when_it_happened = $_POST['whenithappened'];
$how_long = $_POST['howlong'];
$alien_description = $_POST['description'];
$fang_spotted = $_POST['fangspotted'];
$email = $_POST['email'];

echo 'Thanks for submitting the form.<br/>';
echo 'You were abducted ' . $when_it_happened . '<br/>';
echo 'and were gone for ' . $how_long . '<br/>';
echo 'Describe them: ' . $alien_description . '<br/>';
echo 'Was Fang there? ' . $fang_spotted . '<br/>';
echo 'Your email address is ' . $email;

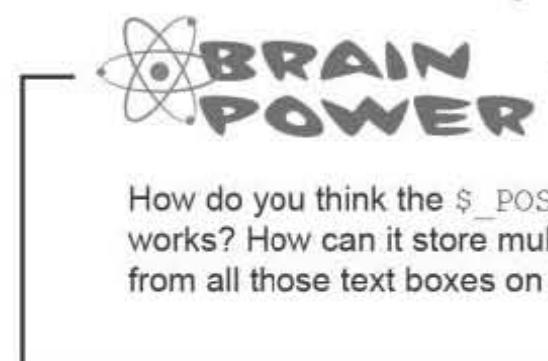
?>
</body>
</html>
```

The `$_POST` superglobal is directly tied to the form submission method used by the HTML form. If the method's set to post, then all of the form data gets packaged into the `$_POST` superglobal, where each piece of data can be plucked out and used as needed.

The name "has" the name attribute
<input> tag



The form submission method determines how the form data is supplied to the PHP script.



`$_POST` is an array

\$_POST transports form data to your script

`$_POST` is a special kind of PHP storage container known as an **array**, which stores a collection of variables under a single name. When someone submits Owen's form, the data they've typed into the form fields is stored in the `$_POST` array, whose job is to pass the data along to the script.

Each element in the `$_POST` array corresponds to a piece of data entered into a form field. To access the data for a specific form field, you use the name of the field with `$_POST`. So the duration of an abduction is stored in `$_POST['howlong']`. The HTML code for Owen's form reveals how form names relate to data stored in `$_POST`.

```
<p>Share your story of alien abduction:</p>
<form method="post" action="report.php">
  <label for="firstname">First name:</label>
  <input type="text" id="firstname" name="firstname" /><br />
  <label for="lastname">Last name:</label>
  <input type="text" id="lastname" name="lastname" /><br />
  <label for="email">What is your email address?</label>
  <input type="text" id="email" name="email" /><br />
  <label for="whenithappened">When did it happen?</label>
  <input type="text" id="whenithappened" name="whenithappened" /><br />
  <label for="howlong">How long were you gone?</label>
  <input type="text" id="howlong" name="howlong" /><br />
  <label for="howmany">How many did you see?</label>
  <input type="text" id="howmany" name="howmany" /><br />
  <label for="aliendescription">Describe them:</label>
  <input type="text" id="aliendescription" name="aliendescription" size="32" /><br />
  <label for="whattheydid">What did they do to you?</label>
  <input type="text" id="whattheydid" name="whattheydid" size="32" /><br />
  <label for="fangspotted">Have you seen my dog Fang?</label>
  Yes <input id="fangspotted" name="fangspotted" type="radio" value="yes" />
  No <input id="fangspotted" name="fangspotted" type="radio" value="no" /><br />
  <br />
  <label for="other">Anything else you want to add?</label>
  <textarea name="other"></textarea><br />
  <input type="submit" value="Report Abduction" name="submit" />
</form>
```



`$_POST`

All of the
is accessible
the `$_POST`



Sharpen your pencil

Scratch through the code in report.php that is description to come up blank, and then write down Hint: Use the HTML form code on the facing page problem.

```

<html>
<head>
    <title>Aliens Abducted Me - Report an Abduction</title>
</head>
<body>
    <h2>Aliens Abducted Me - Report an Abduction</h2>

    <?php
        $when_it_happened = $_POST['whenithappened'];
        $how_long = $_POST['howlong'];
        $alien_description = $_POST['description'];
        $fang_spotted = $_POST['fangspotted'];
        $email = $_POST['email'];

        echo 'Thanks for submitting the form.<br />';
        echo 'You were abducted ' . $when_it_happened;
        echo ' and were gone for ' . $how_long . '<br />';
        echo 'Describe them: ' . $alien_description . '<br />';
        echo 'Was Fang there? ' . $fang_spotted . '<br />';
        echo 'Your email address is ' . $email;
    ?>

    </body>
</html>

```

Remember, earlier we isolated the problem down to these two lines of code.



report.php

sharpen solution

Sharpen your pencil Solution

Scratch through the code in `report.php` that is description to come up blank, and then write down Hint: Use the HTML form code on the facing page problem.

```
...  
<input type="text" id="aliendescription" name="aliendescription" size="32" />  
...
```



The name of the form field in `report.html` is "aliendescription", which doesn't match the name used in `$_POST`.

We need to change `$_POST` so that the form field name is correct: 'aliendescription'.

```
<?php  
  
$when_it_happened = $_POST['whenithappened'];  
$how_long = $_POST['howlong'];  
$alien_description = $_POST['aliendescription'];  
$fang_spotted = $_POST['fangspotted'];  
$email = $_POST['email'];  
  
echo 'Thanks for submitting the form.<br />';  
echo 'You were abducted ' . $when_it_happened .  
echo ' and were gone for ' . $how_long . '<br />';  
echo 'Describe them: ' . $alien_description .  
echo 'Was Fang there? ' . $fang_spotted . '<br />';  
echo 'Your email address is ' . $email;  
?  
  
</body>  
</html>
```



— Test Drive —

Fix the script and test it out.

Change the broken line of code in `report.php`, and then upload it to your web server. Open the `report.html` page in your browser, fill out the form with alien abduction information, and click the Report Abduction button to submit it to the newly repaired script.

Aliens Abducted Me - Report an Abduction

Share your story of alien abduction:

First name:	Alf
Last name:	Nader
What is your email address?	alfn@theyrealgreen.com
When did it happen?	last November
How long were you gone?	11 hours
How many did you see?	dozens
Describe them:	little green men
What did they do to you?	asked me about UFO regulations
Have you seen my dog Fang?	Yes <input type="radio"/> No <input checked="" type="radio"/>



Aliens Abducted Me - Report an Abduction

You were abducted last November and were gone for 11 hours
 Describe them: little green men
 Was Fang there? no
 Your email address is alfn@theyrealgreen.com

The confirmation page now correctly shows the form data for the alien description!

revise owen's php script



There's some data entered into Owen's Report an Abduction form that we aren't currently using. Remember, this data contains vital information about an alien abduction that could lead Owen back to his lost dog, Fang. So we need to grab all of the abduction data and store it away in PHP variables.

The report.php script currently ignores five different pieces of form data. Shocking!

```
***<form method="post" action="report.php">
<label for="firstname">First name:</label>
<input type="text" id="firstname" name="firstname" /><br />
<label for="lastname">Last name:</label>
<input type="text" id="lastname" name="lastname" /><br />
<label for="email">What is your email address?</label>
<input type="text" id="email" name="email" /><br />
<label for="whenithappened">When did it happen?</label>
<input type="text" id="whenithappened" name="whenithappened" /><br />
<label for="howlong">How long were you gone?</label>
<input type="text" id="howlong" name="howlong" /><br />
<label for="howmany">How many did you see?</label>
<input type="text" id="howmany" name="howmany" /><br />
<label for="aliendescription">Describe them:</label>
<input type="text" id="aliendescription" name="aliendescription" size="32" /><br />
<label for="whattheydid">What did they do to you?</label>
<input type="text" id="whattheydid" name="whattheydid" size="32" /><br />
<label for="fangspotted">Have you seen my dog Fang?</label>
Yes <input id="fangspotted" name="fangspotted" type="radio" value="yes" />
No <input id="fangspotted" name="fangspotted" type="radio" value="no" /><br />

<label for="other">Anything else you want to add?</label>
<textarea id="other" name="other"></textarea><br />
<input type="submit" value="Report Abduction" name="submit" />
</form>
</body>
</html>
```



report.html

Aliens Abducted Me - Re

Aliens Abducted Me - Re

Share your story of alien abduction:

First name:

Last name:

~~What is your email address?~~

~~When did it happen?~~

~~How long were you gone?~~

~~How many did you see?~~

~~Describe them:~~

~~What did they do to you?~~

~~Have you seen my dog Fang?~~



Anything else you want to add?

All
Nade
alnig
last N
11 ho
dozen
little g
asked
Yes

Please v

Write PHP code to create four new variables that store the form data: \$name, \$how_many, \$what_they_did, a
Hint: Create the \$name variable so that it stores the user'

Your work is not quite done. The confirmation web page generated by the PHP script needs to use those new variables to display more information about the alien abduction.

We need to go from this...

Aliens Abducted Me - Report an Abduction

You were abducted last November and were gone for 11 hours
Describe them: little green men
Was Fang there? no
Your email address is alfn@theyreallgreen.com

...to this! Notice more information

Aliens Abducted Me - Report an Abduction

Thanks for submitting the form.
You were abducted last November and were gone for 11 hours.
Describe them: little green men.
The aliens did this: asked me about UFOs.
Was Fang there? no.
Other comments: Please vote for me.
Your email address is alfn@theyreallgreen.com

The user's name isn't critical to the confirmation page, although we'll need it later when we send an abduction email to Owen.

Using all of the variables you just created except \$name, finish the missing code below that generates a more informative confirmation page.

```

echo 'Thanks for submitting the form.<br>
echo 'You were abducted ' . $when_it_happened . '<br>
echo ' and were gone for ' . $how_long . '<br>
.....<br>
echo 'Describe them: ' . $alien_descripti...
.....<br>
echo 'Was Fang there? ' . $fang_spotted . '<br>
.....<br>
echo 'Your email address is ' . $email;

```

owen's revised php script

Sharpen your pencil
Solution

There's some data entered into Owen's Report an Abduction form that we aren't currently using. Remember, this data contains vital information about an alien abduction that could lead Owen back to his lost dog, Fang. So we need to grab all of the abduction data and store it away in PHP variables.

The report.php script currently ignores five different pieces of form data. Shocking!

```
***  
<form method="post" action="report.php">  
  <label for="firstname">First name:</label>  
  <input type="text" id="firstname" name="firstname" /><br />  
  <label for="lastname">Last name:</label>  
  <input type="text" id="lastname" name="lastname" /><br />  
  <label for="email">What is your email address?</label>  
  <input type="text" id="email" name="email" /><br />  
  <label for="whenithappened">When did it happen?</label>  
  <input type="text" id="whenithappened" name="whenithappened" /><br />  
  <label for="howlong">How long were you gone?</label>  
  <input type="text" id="howlong" name="howlong" /><br />  
  <label for="howmany">How many did you see?</label>  
  <input type="text" id="howmany" name="howmany" /><br />  
  <label for="aliendescription">Describe them:</label>  
  <input type="text" id="aliendescription" name="aliendescription" size="32" /><br />  
  <label for="whattheydid">What did they do to you?</label>  
  <input type="text" id="whattheydid" name="whattheydid" size="32" /><br />  
  <label for="fangspotted">Have you seen my dog Fang?</label>  
  Yes <input id="fangspotted" name="fangspotted" type="radio" value="yes" />  
  No <input id="fangspotted" name="fangspotted" type="radio" value="no" /><br />  
    
  <label for="other">Anything else you want to add?</label>  
  <textarea id="other" name="other"></textarea><br />  
  <input type="submit" value="Report Abduction" name="submit" />  
</form>  
</body>  
</html>
```



report.html

The period allows you to stick multiple strings of text together as one – a process known as concatenation.

Aliens Abducted Me - Re

Aliens Abducted Me - Re

Share your story of alien abduction:

First name:

Last name:

~~What is your email address?~~

When did it happen?

How long were you gone?

How many did you see?

Describe them:

What did they do to you?

Have you seen my dog Fang?



Anything else you want to add?

All
Nade
alnig
last N
11 ho
dozen
little g
asked
Yes

Please v

Write PHP code to create four new variables that store the form data: \$name, \$how_many, \$what_they_did, and \$other. Hint: Create the \$name variable so that it stores the user's first and last names concatenated together.

```
$name = $_POST['firstname'] . ' ' . $_POST['lastname'];
$how_many = $_POST['howmany'];
$what_they_did = $_POST['whattheydid'];
$other = $_POST['other'];
```

Your work is not quite done. The confirmation web page generated by the PHP script needs to use those new variables to display more information about the alien abduction.

We need to go from this...

Aliens Abducted Me - Report an Abduction

You were abducted last November and were gone for 11 hours
Describe them: little green men
Was Fang there? no
Your email address is alfn@theyreallgreen.com

...to this! Notice more information

Aliens Abducted Me - Report an Abduction

Thanks for submitting the form.
You were abducted last November and were gone for 11 hours.
Describe them: little green men.
The aliens did this: asked me about UFOs.
Was Fang there? no.
Other comments: Please vote for me.
Your email address is alfn@theyreallgreen.com

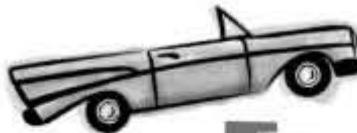
The user's name isn't critical to the confirmation page, although we'll need it later when we send an abduction email to Owen.

Using all of the variables you just created except \$name, finish the missing code below that generates a more informative confirmation page.

The echo command is used to output the additional information to the browser as HTML content.

Again, periods are used to concatenate strings and variables together.

```
echo 'Thanks for submitting the form.<br />'  
echo 'You were abducted ' . $when_it_happened .  
echo ' and were gone for ' . $how_long .  
echo 'Number of aliens: ' . $how_many . '<br />';  
echo 'Describe them: ' . $alien_descriptions .  
echo 'The aliens did this: ' . $what_they_did . '<br />';  
echo 'Was Fang there? ' . $fang_spotted .  
echo 'Other comments: ' . $other . '<br />';  
echo 'Your email address is ' . $email;
```

test drive owen's php script

Test Drive

Tweak Owen's script and try out the changes.

Add the code for the new variables to `report.php`, as well as the code that echoes the variables to the browser as formatted HTML. Then upload the script to your web server, open the `report.html` page in your browser, and fill out the form with alien abduction information. Finally, click the Report Abduction button to submit the form and see the results.

*there are no
Dumb Questions*

Q: What actually happens when I concatenate multiple strings together using periods?

A: Concatenation involves sticking more than one string together to form a completely new string. The end result of concatenating strings is always a single string, no matter how many strings you started with. So when you concatenate strings as part of an `echo` command, PHP combines the strings together into one first, and then echoes that string to the browser.

Q: When I concatenate a variable with a string, does the variable have to contain text?

A: No. Although concatenation always results in a string, variables don't have to contain strings in order for you to concatenate them. So say a variable contains a number, PHP converts the number to a string first and then concatenates it.

Q: What happens to PHP code on the browser?

A: Nothing. And that's because PHP code is never seen by a browser. PHP code runs on the server and gets turned into HTML code that's sent along to the browser. So the browser is completely unaware of PHP's existence—web pages arrive as pure HTML and CSS.

Q: OK, so how exactly does the code get turned into HTML and CSS code?

A: First off, remember that by default, PHP code is assumed to be HTML code. You identify a PHP script by placing it between `<?php` and `?>`. The browser sees those tags and knows to run the code between them and all of the code outside of those tags gets turned into HTML.

Q: Right. But that still doesn't explain how the code gets turned into HTML/CSS code.

A: Ah, that's where the `echo` command comes in. You can think of the `echo` command as a way to break beyond the confines of the `<?php` and `?>` tags. The `echo` command is the key to PHP's ability to output plain old HTML/CSS code. By concatenating strings and variables, you can construct HTML code and then use the `echo` to output it to the browser as part of the page. A good example of this is in Owen's `report.php` script when the `
` tag is tack onto the end of a string of text to generate a line break in the output.



The PHP script still needs to email the form data

As it stands, the `report.php` script is grabbing the data from an Abduction form and generating an HTML confirmation page. But it's not yet solving the original problem of emailing a message when the form is submitted. He just wants to receive a simple text message that looks something like this:

Similar to the confirmation web page, this email message consists of static text combined with form data.

Alf Nader was abducted last November and was gone for 11 days.
Number of aliens: dozens
Alien description: little green men
What they did: asked me about UFO regulations
Fang spotted: no
Other comments: Please vote for me.

This email message can be generated from PHP code by putting a string that combines static text such as "Other comments:" with field data stored in variables.

Write down how you'd put together an email message from static text and PHP variables.

.....
.....

building the message body in php

Creating the email message body with PHP

You've already seen how a period can be used in PHP code to concatenate multiple strings of text together into a single string. Now you need to use concatenation again to build an email message string with variables sprinkled in among static text.

Variables and static text are concatenated into a single email message string using periods.

```
$msg = $name . ' was abducted ' . $when_it_happened . ' and was gone for ' . $when_it_gone_for .  
'Number of aliens:' . $how_many . 'Alien description:' . $alien_description .  
$what_they_did . 'Fang spotted:' . $fang_spotted . 'Other comments:' . $other_comments;
```

Most text editors automatically add code to the line if you don't own line break

One problem with building such a large string is that it requires a huge line of PHP code that's difficult to read and understand. You can break the PHP code across multiple lines to make it easier to follow. Just make sure to separate the code in spots where the spacing doesn't matter, like **between** two concatenated strings, not in the middle of a string. Then put a semicolon at the end of the last line of the code to finish the PHP statement.

This is really just one big line of code divided across multiple lines.

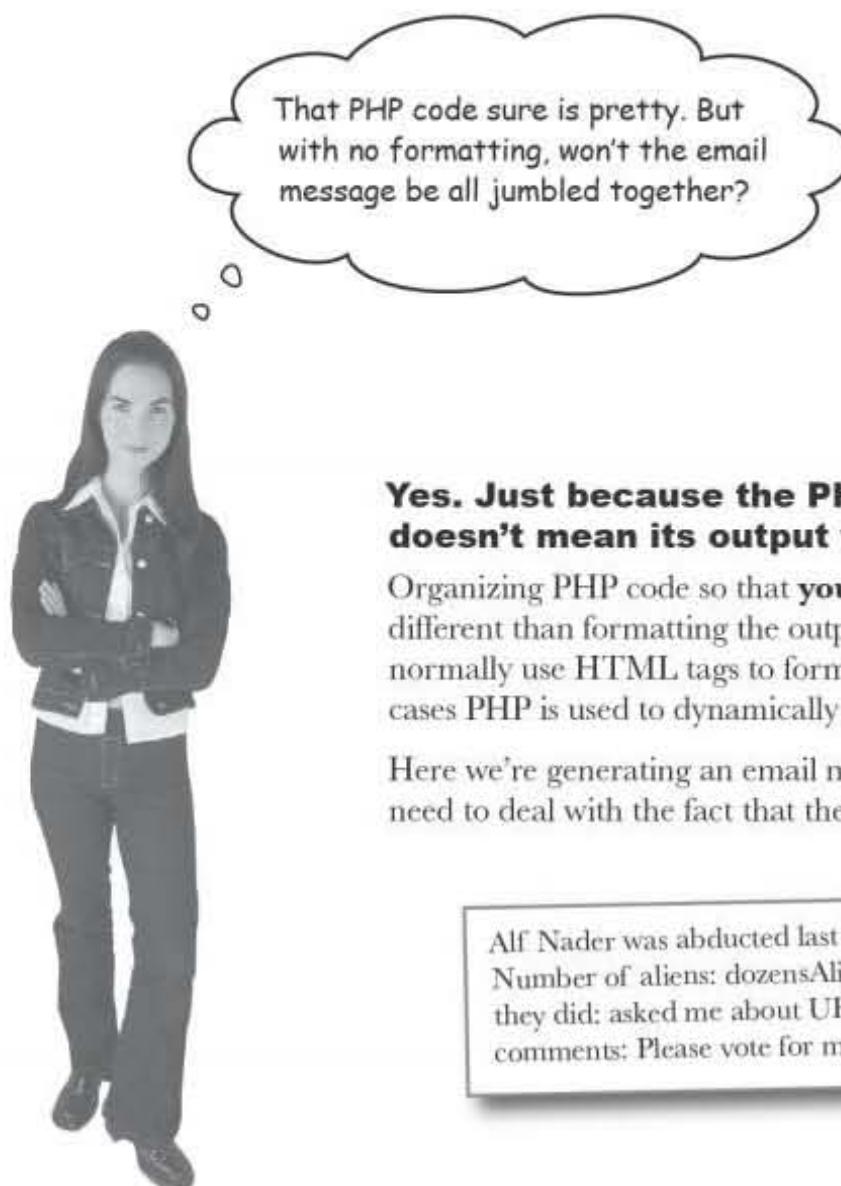
```
$msg = $name . ' was abducted ' . $when_it_happened . ' and was gone for ' . $when_it_gone_for .  
'Number of aliens:' . $how_many .  
'Alien description:' . $alien_description .  
$what_they_did . 'Fang spotted:' . $fang_spotted .  
$other_comments;
```

The line of code is carefully extended by not breaking in the middle of a string.

When a line of PHP code is deliberately extended across multiple lines, it's customary to indent the lines after the first one to help you see which lines belong together in your code.

You still have to finish the entire statement with a semicolon.

A long line of PHP code can be broken across multiple lines as long as you're careful about how you break it.



Yes. Just because the PHP code is organized doesn't mean its output will automatically look good.

Organizing PHP code so that **you** can better understand it is very different than formatting the output of PHP code that users normally use HTML tags to format the output of PHP code. In some cases PHP is used to dynamically generate a web page. But in others,

Here we're generating an email message, which is plain text, so we need to deal with the fact that the message currently looks like this:

Alf Nader was abducted last November and was gone for 10 months. Number of aliens: dozens Alien description: little green men What they did: asked me about UFO regulations Fang spotted: none Comments: Please vote for me.

Ouch! This is NOT what Owen had in mind for his Abduction Report email message.

there are no Dumb Questions

Q: Is there a way to use HTML formatting in emails you send from a PHP script?

A: There is. But it requires an additional step that involves setting the content type header for the message. Headers and content types are a bit beyond the scope of this discussion, which is why we're sticking with pure text email messages for Owen's email responses. You'll learn more about headers in Chapter 6, so you'll definitely gain the knowledge to revisit HTML emails later.



How would you make plain text emails look better so that it is easier to read?

formatting text with php

Even plain text can be formatted... a little

Since Owen's sending email messages as plain text with no HTML formatting, he can't just stick in `
` tags to add line breaks where the content's running together. But he *can* use newline characters, which are **escaped** as `\n`. So wherever `\n` appears in the email text, a newline will be inserted, causing any content after it to start on the next line. Here's the new email message code with newlines added:

```
$msg = $name . ' was abducted ' . $when_it_happened . ' and was gone for ' . $how_long . ' ' . $when_it_happened . ' ' . $what_they_did . ' ' . $other;

'Number of aliens: ' . $how_many . '\n' .
'Alien description: ' . $alien_description . '\n' .
'What they did: ' . $what_they_did . '\n' .
'Fang spotted: ' . $fang_spotted . '\n' .
'Other comments: ' . $other;
```

`\n` is used to place newline characters throughout the email message.

Newlines sound like a great idea... too bad that code doesn't

Alf Nader was abducted last November and was gone for 11 hours.
 \nNumber of aliens: dozens\nAlien description: little green men
 \nWhat they did: asked me about UFO regulations\nFang spotted:
 no\nOther comments: Please vote for me.

there are no
Dumb Questions

The `\n` is appearing as normal text instead of a new line character... not good!

Q: What exactly is an escape character?

A: An escape character is a character that's either difficult to type or would otherwise cause confusion in PHP code. You may be familiar with escape characters from HTML, where they're coded a little differently, like `©` or `©` for the copyright symbol. PHP has a very small set of escape characters that are helpful for escaping things that might be confused with the PHP language itself, such as single quotes (`\ '`), double quotes (`\ "`), and of course, newlines (`\n`).

Escape
in **PHP**
a **backslash**

Newlines need double-quoted strings

The problem with Owen's code is that PHP handles strings differently depending on whether they're enclosed by single or double quotes. More specifically, newline characters (`\n`) can only be escaped in double-quoted strings. So the Abduction Report email message must be constructed using double-quoted strings in order for the newlines to work.

But there's more to the single vs. double quote story than that. Single-quoted strings are considered raw text, whereas PHP processes double-quoted strings looking for variables. When a variable is encountered within a double-quoted string, PHP inserts its value into the string as if the strings had been concatenated. So not only is a double-quoted string necessary to make the newlines work in the email message, but it also allows us to simplify the code by sticking the variables directly in the string.

```
$msg = "$name was abducted $when_it_happened and was gone for $h  
"Number of aliens: $how_many\n" .  
"Alien description: $alien_description\n" .  
"What they did: $what_they_did\n" .  
"Fang spotted: $fang_spotted\n" .  
"Other comments: $other";
```

There's no need for a newline at the very end since this is the last line of the email message.

Newline characters are interpreted properly thanks to the double-quoted strings.

there are no Dumb Questions

Q: If double-quoted strings are so cool, why have we used mostly single-quoted strings up until now?

A: Well, keep in mind that single-quoted strings are not processed by PHP in any way, which makes them ideal for strings that are pure text with no embedded variables. So we'll continue to use single-quoted strings throughout the book unless there is a compelling reason to use a double-quoted string instead. The most important thing about using single vs. double quotes around strings is to try and be as consistent as possible.

Q: What happens if I need to use a single quote (apostrophe) within a single-quoted string, as in 'He's lost! '?

A: This is where escape characters come in. You can put a single quote inside of a single-quoted string like this: 'He\''s lost'. You can also put a double quote inside of a double-quoted string like this: "He said, \"I'm lost\"." Note that you don't need to escape quotes when they don't contain other characters.

Q: So single-quoted strings can't contain apostrophes? How do I know what escape characters to use?

A: Single-quoted strings can't contain apostrophes, but they can contain other escape characters—all other escape characters are valid in single-quoted strings.

assemble owen's email

Assemble an email message for Owen

With the body of the email message generated as a string, you can move on to assembling the rest of Owen's email. An email message is more than just a message body—there are several different parts. Although some are optional, the following pieces of information are used in pretty much all emails:

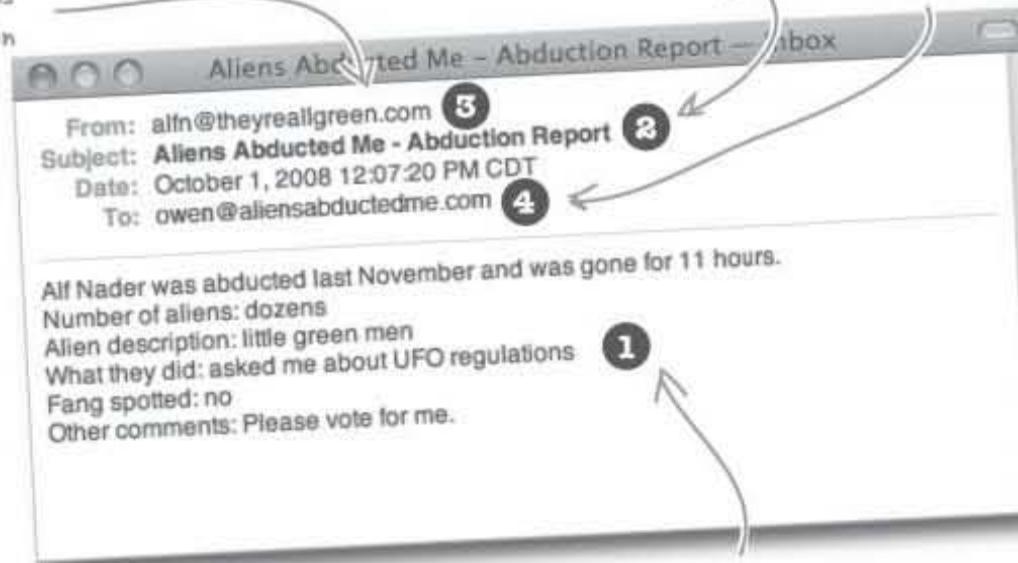
- ➊ **The message body.** Already done!
- ➋ **The message subject.**
- ➌ **The sender's email address (who the message is FROM).**
- ➍ **The recipient's email address (who the message is TO).**

This is the kind of email message Owen hopes to receive upon someone submitting an alien abduction report.

This is the user's email address, which is already stored away in the `$email` variable.

This can be a static string.

This is Owen's address, which will be a static string.

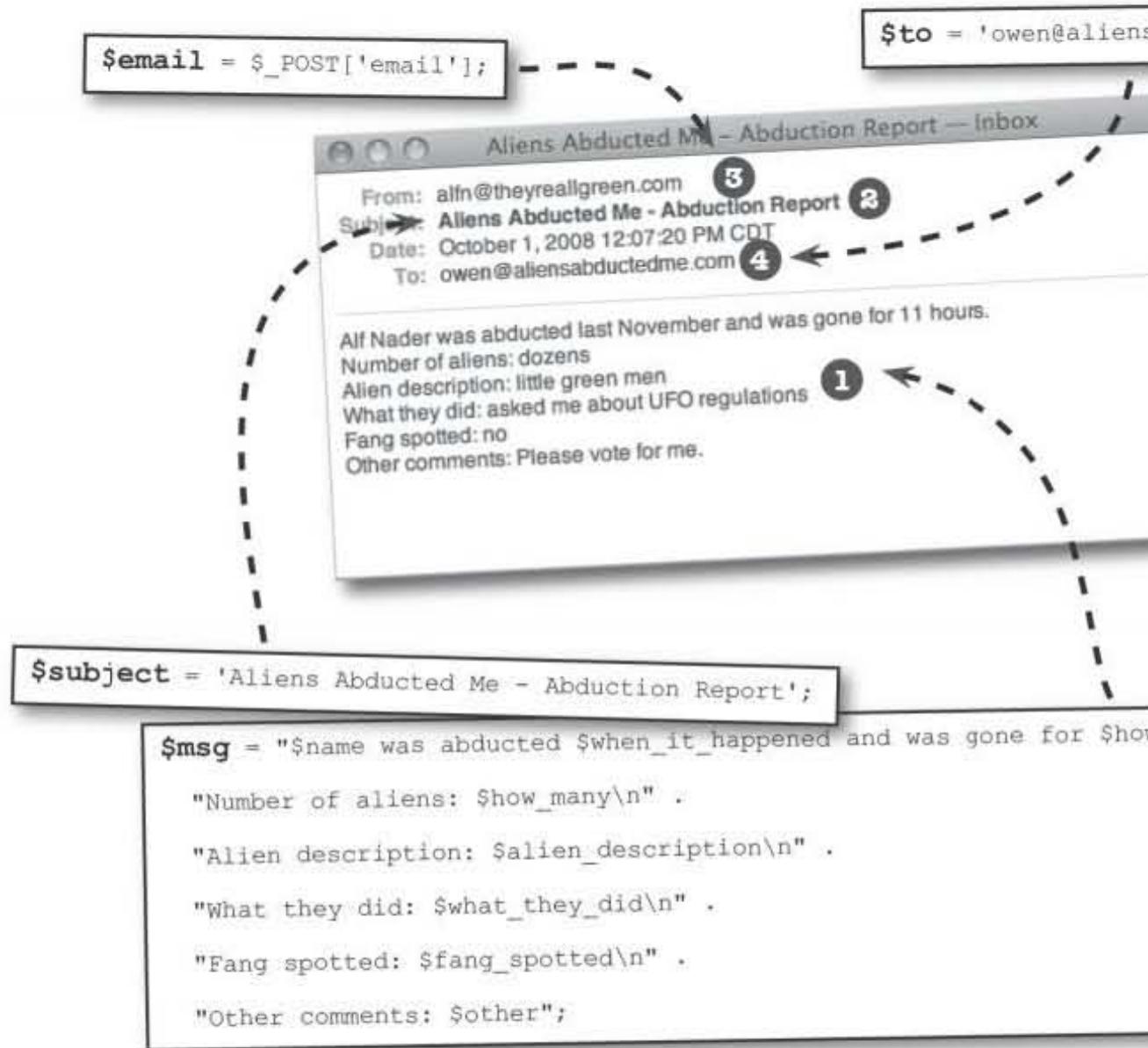


We've already constructed a string for the email body, which is stored in the `$msg` variable.

This sample email message reveals that most of the components of an email message, which you've already finished. All that's left is the message subject, "from" and "to" email addresses... and then you'll learn how to use PHP to actually send the message!

Variables store the email pieces and parts

We already have the message body stored in \$msg, but we're still missing the message subject and "from" and "to" email addresses. The subject and the "to" email address can just be set as static text in new variables, while the "from" email address is already stored away in the \$email variable thanks to the form-handling code we wrote earlier in the chapter.



All the email information's gathered and ready to go!

- ➊ The message body.
- ➋ The message subject.
- ➌ The sender's email address (who the message came from).
- ➍ The recipient's email address (who the message went to).

send the email with php

Sending an email message with PHP

So you're ready to write the PHP code to actually *send* the email message to Owen. This requires PHP's built-in `mail()` function, which sends a message based on information you provide it.

```
mail($to, $subject, $msg);
```

The "to" email address
The subject of the message
The body of the message

These three pieces of information are required by the `mail()` function, so you always need to provide them. The "from" email address isn't required but it's still a good idea to include it. To specify the "from" field when calling the `mail()` function, an additional function argument's required, along with some string concatenation.

```
mail($to, $subject, $msg, 'From:' . $email);
```

```
$to = 'owen@aliensabductedme.com';
$subject = 'Aliens Abducted Me - Abduction Report';
$msg = "$name was abducted $when_it_happened and was gone for $show_long.\n";
        "Number of aliens: $how_many\n";
        "Alien description: $alien_description\n";
        "What they did: $what_they_did\n";
        "Fang spotted: $fang_spotted\n";
        "Other comments: $other";
```

```
$email = $_POST['email'];
```

The period's handy for concatenating "From:" with Owen's email address

Each piece of the email message is provided to the `mail()` function by a variable.

That's right, two escape characters back-to-back!

The `PHP` function `email` message within a

The text 'From:' prepended to the address when specifying the address of the

there a Dumb

Q: Is there any specified as part of addition to the "from"

A: Yes. You can "blind copy" recipient "from" recipient—just instead of 'From:' use both a "from" and a separate them with a character combination

"From:" . \$from

We have to use double quotes



Just add the code that calls mail() to your script

The line of code that calls the `mail()` function is all you need to send an email message. Make sure this code appears in the script **after** it creates the email variables, and you're good to go. Here's the code for Owen's `report.php` script, including the call to the `mail()`

```

<html>
<head>
  <title>Aliens Abducted Me - Report an Abduction</title>
</head>
<body>
  <h2>Aliens Abducted Me - Report an Abduction</h2>

<?php
  $name = $_POST['firstname'] . ' ' . $_POST['lastname'];
  $when_it_happened = $_POST['whenithappened'];
  $how_long = $_POST['howlong'];
  $how_many = $_POST['howmany'];
  $alien_description = $_POST['aliendescription'];
  $what_they_did = $_POST['whattheydid'];
  $fang_spotted = $_POST['fangspotted'];
  $email = $_POST['email'];
  $other = $_POST['other'];

  $to = 'owen@aliensabductedme.com';
  $subject = 'Aliens Abducted Me - Abduction Report';
  $msg = "$name was abducted $when_it_happened and was gone for $how_long\n";
  $msg .= "Number of aliens: $how_many\n";
  $msg .= "Alien description: $alien_description\n";
  $msg .= "What they did: $what_they_did\n";
  $msg .= "Fang spotted: $fang_spotted\n";
  $msg .= "Other comments: $other";

  mail($to, $subject, $msg, 'From:' . $email);

  echo 'Thanks for submitting the form.<br />';
  echo 'You were abducted ' . $when_it_happened;
  echo ' and were gone for ' . $how_long . '<br />';
  echo 'Number of aliens: ' . $how_many . '<br />';
  echo 'Describe them: ' . $alien_description . '<br />';
  echo 'The aliens did this: ' . $what_they_did . '<br />';
  echo 'Was Fang there? ' . $fang_spotted . '<br />';
  echo 'Other comments: ' . $other . '<br />';
  echo 'Your email address is ' . $email;
?>

</body>
</html>
```

Send the email message.

Generate an HTML web page on the fly that confirms the successful form submission.

Grab all from the array and individual

Make sure this email is your own in the script

Assemble the different parts of the email message to be sent

the final test drive

Test Drive

Finish up Owen's script and then try it out.

Add the three new email variables (`$to`, `$subject`, and `$msg`) to the `report.php` script, as well as the call to the `mail()` function. Make sure the `$to` variable is set to **your** email address, not Owen's! Upload the script to your web server, open it in your browser, and fill out the form with alien abduction information. Click the Report Abduction button to submit the form. Wait a few seconds and then go check your email Inbox for the message.

The screenshot shows a web application for reporting alien abductions. On the left, the 'Report an Abduction' page has fields for First name (Ali), Last name (Hadie), Email (ali@theyallgreen.com), Date (last November), Duration (11 hours), Description (little green men), What they did (asked me about UFO regulations), Was Fang there? (no), and Other comments (Please vote for me). A dog named Fang is shown below the form. On the right, a confirmation page shows the same information with a success message: 'Thanks for submitting the form. You were abducted last November and were gone for 11 hours. Number of aliens: eleven. Describe them: little green men. What they did: asked me about UFO regulations. Was Fang there? no. Other comments: Please vote for me. Your email address is ali@theyallgreen.com'. An arrow points from the text 'The dynamically generated confirmation page still confirms the form submission.' to the confirmation page.



Watch it!

You may need to **watch** your web server settings to send email. If the `mail()` function does not work, the problem may be that your web server is not properly configured for your PHP installation. See www.php.net/mail for details on how to enable the email features on your web server.



Owen starts getting emails

Owen is thrilled that he's reliably receiving alien abduction reports via a web form directly to his email Inbox. Now he doesn't have to worry about missing any reports because he'll receive them from everyone who contacts him. And even better, he'll be able to respond to those reports at his leisure.



Aliens Abducted Me - Report an Abduction	
Share your story of alien abduction!	
First name:	John
Last name:	Johnson
What is your email address?	johnjohnson@john.com
Where did it happen?	2nd floor
How long were you gone?	1 hr
How many did you see?	Two
Describe them:	grey-skinned with large heads and long fingers, one very hairy and another very slimy.
What did they do to you?	Take @ No ☺
Were you given any dog tags?	
	
Aliens Abducted Me - Report an Abduction	
Thanks for submitting this form.	
You were abducted! I always had been positive I was.	
Describe the process you were involved in.	
The aliens did this to me yesterday and I have no idea what's going on here!	
Other questions? I may have more just for you. Contact me.	
Your email address is sallyjohnson@john.com .	

Sally submits
the form.

```
<form action  
= "report.php"  
...  
...
```



The PHP script dynamically generates a confirmation HTML page.



This is awesome!
With email abduction
reports like this, I
know I'll find Fang.



Owen is one happy camper now that he's receiving alien abduction emails through his form.

email overload

Owen starts losing emails

The good news is that Owen's getting emails now. The bad news is that he's getting lots and lots of emails. So many that he's having difficulty keeping track of them. His Inbox is packed, and he's already accidentally deleted some... Owen needs a better way to store the alien abduction data.





WHO DOES WHAT?

Got aliens on the brain? Shake them loose by matching each HTML and PHP component to what you think it does.

HTML

A software application for viewing and interacting with web pages that acts as the client side of web communication.

PHP

A PHP command that is used to output content as either pure text or HTML code.

web form

These tags are used to enclose PHP code so that the web server knows to process it and run it.

browser

A built-in PHP array that stores data that has been submitted using the “post” method.

<?php ?>

A programming language used to create scripts that run on a web server.

variable

All strings must be enclosed within these.

quotes

A software application for delivering web pages that acts as the server side of web communication.

echo

A markup language used to describe the structure of a web page content that is viewed in a web browser.

\$_POST

A name used to describe built-in PHP variables that are accessible to all scripts.

web server

A series of input fields on a web page that is used to get information from users.

array

A built-in PHP function that sends an email message.

superglobal

A storage location in a PHP script that has its own unique name and data type.

mail()

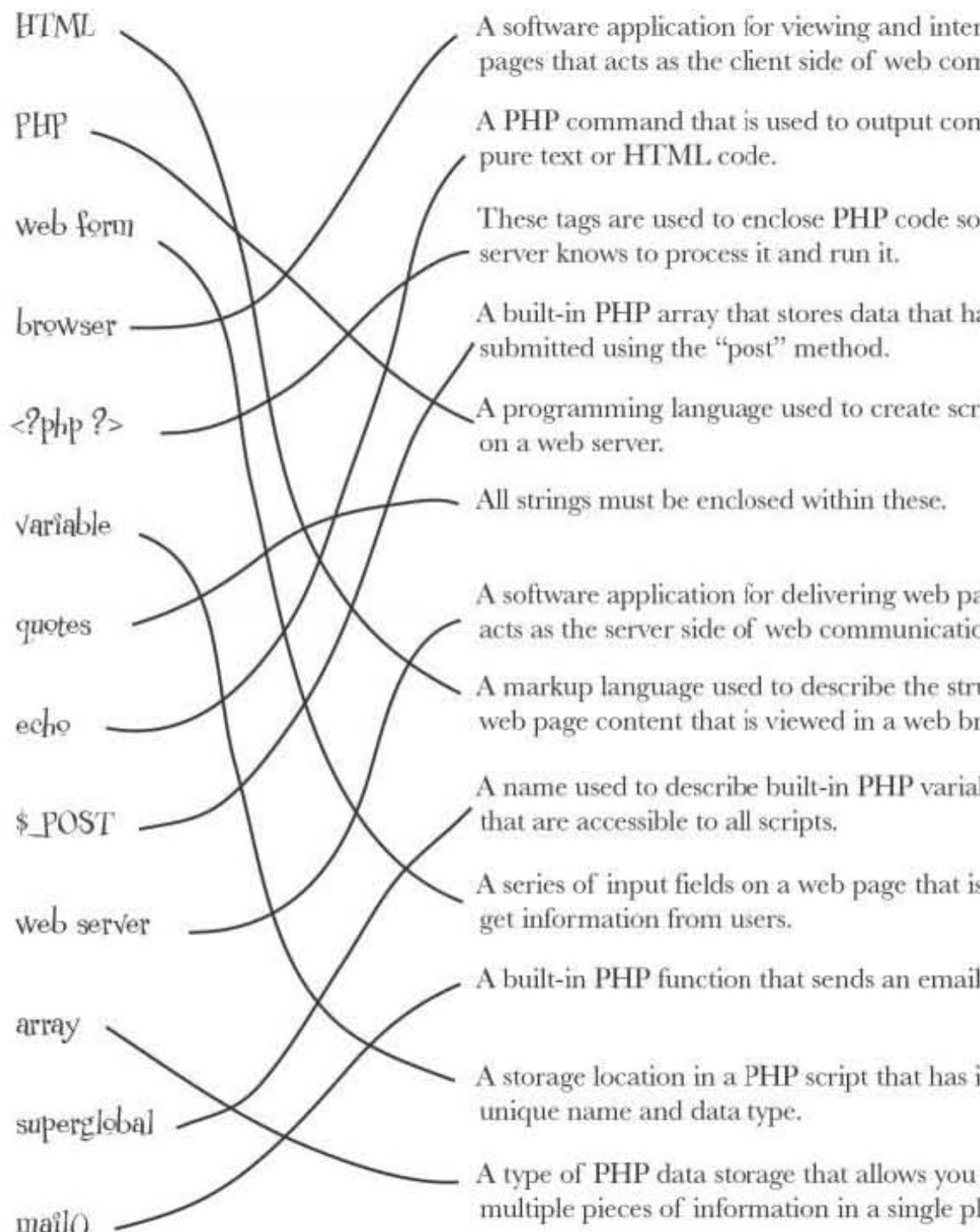
A type of PHP data storage that allows you to store multiple pieces of information in a single place.

who does what solution

WHO DOES WHAT?

SOLUTION

Got aliens on the brain? Shake them loose by matching each HTML and PHP component to what you think it does.





Your PHP & MySQL Toolbox

In Chapter 1, you learned how to harness PHP to bring life to Owen's web form. Look at everything you've learned already...

PHP

A server-side scripting language that lets you manipulate web page content on the server before a page is delivered to the client browser.

PHP script

A text file that contains PHP code to carry out tasks on a web server.

MySQL

An application that lets you store data in databases and tables and insert and retrieve information using the SQL language.

SQL

A query language for interacting with database applications like MySQL.

variable

A storage container for data. In PHP, variables start with a dollar sign: `$variable_name`.

`$_POST`

A special variable that holds data.

`echo`

The PHP command to output to the browser. Its syntax is:

```
echo 'Hello'
```

`<?php ?>`

These tags must surround all PHP code in your PHP scripts.

`mail()`

The PHP function for sending an email. It takes the email subject, email body text, and the destination email address as parameters (you can optionally specify a From address too).

client-side

Interpreted solely by the client web browser.

server-side

Interpreted by a web server, not a client machine.

array

A data structure of values. Each value can have an escape character.

Used to represent PHP code that has the same type or that other code.

2 Connecting to MySQL



How it fits together

We have to plug in
the interweb **before**
we can connect the web
site configurator.

I'm not letting her
anywhere near my
web application.



Knowing how things fit together before you start building is a good idea. You've created your first PHP script, and it's working well. But getting your form results in an email isn't good enough anymore. You need a way to **save the results** of your form, so you can **keep them** as long as you need them and **retrieve them** when you want them. A **MySQL database** can store your data for safe keeping. But you need to hook up your PHP script to the MySQL database to make it happen.

the pitfalls of emailing form data

Owen's PHP form works well. Too well...



Owen's email script was fine when he was getting a few responses, but now he's getting lots of emails, far more than he can manage.

He's accidentally deleted some without reading them, and others are getting stuffed in his spam folder, which he never checks. In fact, an email he'd be very interested in seeing is hidden deep in his spam folder right this moment... Owen needs a way to easily find ones related to Fang.

It will take more than a coffee buzz for Owen to keep up with all the alien abduction reports arriving in his inbox.



Owen needs messages like this safely stored in one place where he can sift through them for possible Fang sightings.

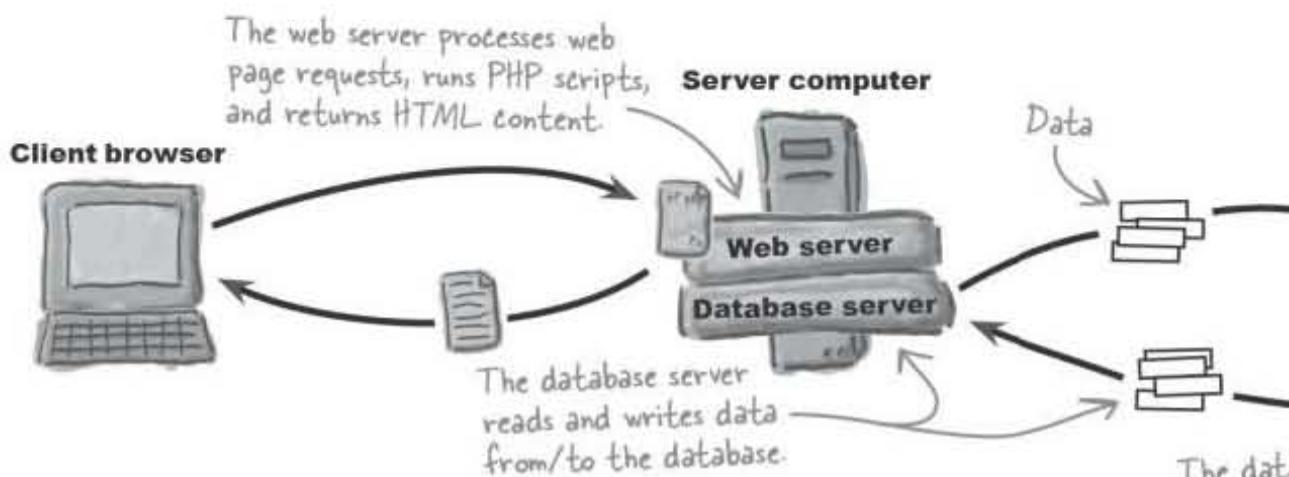
Just in case you didn't know, most people pronounce MySQL by spelling out the last three letters, as in "my-ess-que-el".

This is where a MySQL database comes in.

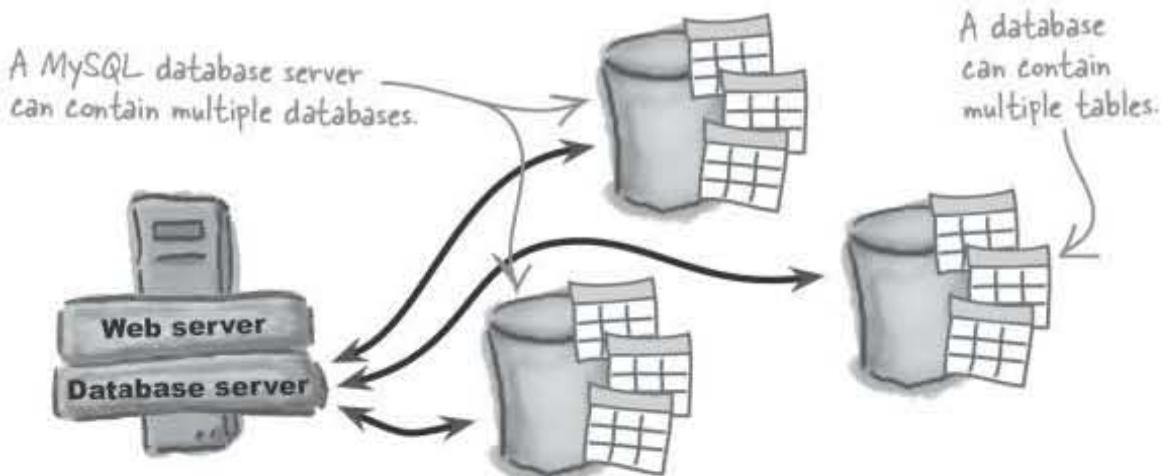
MySQL excels at storing data

Owen really needs a way to store the alien abduction report data in a safe place other than his email Inbox. What he needs is a **database**, which is kinda like a fancy, ultra-organized electronic file cabinet. Since the information in a database is extremely organized, you can pull out precisely the information you need when you need it.

Databases are managed by a special program called a **database server**, in our case a **MySQL** database server. You communicate with a database server in a language it can understand, which in our case is **SQL**. A database server typically runs alongside a web server on the same server computer, working together in concert reading and writing data, and delivering web pages.



MySQL databases are organized into **tables**, which store information as rows and columns of related data. Most web applications use one or more tables inside a single database, sort of like different file folders within a file cabinet.



With alien abduction data safely stored in a MySQL database, Owen can analyze the reports from everyone who answered “yes” to the Fang question at his convenience. He just needs to use a little SQL code to talk to the database server.

mysql can help owen

Owen needs a MySQL database

So it's decided: MySQL databases are good, and Owen needs one to store alien abduction data. He can then modify the `report.php` script to store data in the table instead of emailing it to himself. The table will keep the data safe and sound as it pours in from abductees, giving Owen time to sift through it and isolate potential Fang sightings. But first things first... a database!

Creating a MySQL database requires a MySQL database server and a special software tool. The reason is because, unlike a web server, a database server has to be communicated with using SQL commands.

I've always heard the tool makes all the difference in getting a job done right. How do I know what MySQL tool to use to create a database and table?



MySQL terminal

```
File Edit Window Help MySQL> CREATE TABLE aliens_abduction (
    first_name varchar(30),
    last_name varchar(30),
    when_it_happened varchar(30),
    how_long varchar(30),
    how_many varchar(30),
    alien_description varchar(100),
    what_they_did varchar(100),
    thing_spotted varchar(10),
    other varchar(100),
    email varchar(100)
);
Query OK, 0 rows affected (0.14 sec)
```

phpMyAdmin graphical tool



Two popular MySQL tools are the MySQL terminal and phpMyAdmin. Both tools let you issue SQL commands to create databases, insert data, select data, etc., but phpMyAdmin goes a step further by providing a point-and-click web-based interface. Some web hosting services include phpMyAdmin as part of their standard MySQL service. You can also use the MySQL terminal to access most MySQL installations.

**Creatin
databa
tables
commu
with a
databa**

MySQL
line wind
access to
You can c

ph
tool
crea
thre



You must have a MySQL database server installed before turning on your website.

It's impossible to help Owen without one! If you already have a MySQL database server installed and working, read on. If not, turn to Appendix ii and follow the instructions for getting it up and running. If you're using a web hosting service that offers MySQL, go ahead and ask them to install it for you. All the pieces of information required to access a MySQL database server. You'll need them to set up your website so now is a good time to figure out what they are. Check off each one after you write it down.

- { **My MySQL server location (IP address or hostname):**
- My database user name:**
- My database password:**
- You need to check all of these.*
- If you might have to type in your password by hand, write it down here.*

With your MySQL database server information in hand, all that's left is confirming that it's up and running. Check one of the boxes below to confirm that you can successfully access your MySQL server.

- { **I can successfully access MySQL server using the MySQL terminal:**
- I can successfully access MySQL server using phpMyAdmin:**
- I can successfully access MySQL server using ...:**
- You only need to check one of these.*
- If you've found some other MySQL tool that works, write it down here.*

creating databases and tables in mysql

Create a MySQL database and table

Some MySQL installations already include a database. If yours doesn't, you'll need to create one using the CREATE DATABASE SQL command in the MySQL terminal. But first you need to open the MySQL terminal in a command-line window—just typing **mysql** will often work. You'll know you've successfully entered the terminal when the command prompt changes to **mysql>**.

To create the new alien abduction database, type

`CREATE DATABASE aliendatabase;` like this:

```
File Edit Window Help PhoneHome
mysql> CREATE DATABASE aliendatabase;
Query OK, 1 row affected (0.01 sec)
```

The MySQL server usually responds to let you know that a command was successful.

When you use the `CREATE DATABASE` command, you must put a semicolon at the end of the command.

Before you can create the table inside the database, you need to make sure you've got our new database selected. Enter the command

`USE aliendatabase;`

```
File Edit Window Help PhoneHome
mysql> USE aliendatabase;
Database changed
```

The SQL code to create a table is a little more involved since it has to spell out exactly what kind of data's being stored. Let's take a look at the SQL command before entering it into the terminal:

```
CREATE TABLE aliens_abduction (
    first_name varchar(30),
    last_name varchar(30),
    when_it_happened varchar(30),
    how_long varchar(30),
    how_many varchar(30),
    alien_description varchar(100),
    what_they_did varchar(100),
    fang_spotted varchar(10),
    other varchar(100),
    email varchar(50)
);
```

This is an SQL command that creates a new table.

All the other stuff is detailed information about what kinds of data can be stored in the table.

All SQL commands entered into the MySQL terminal must end with a semicolon.

To actually create the new table, type the big CREATE TABLE command into the MySQL terminal. (You can find the code for the command on the web at www.headfirstlabs.com/books/hfphp.) After successfully entering this command, you'll have a shiny new `aliens_abduction` table.

The "Query OK" response from the MySQL server lets you know the table was created without any problems.

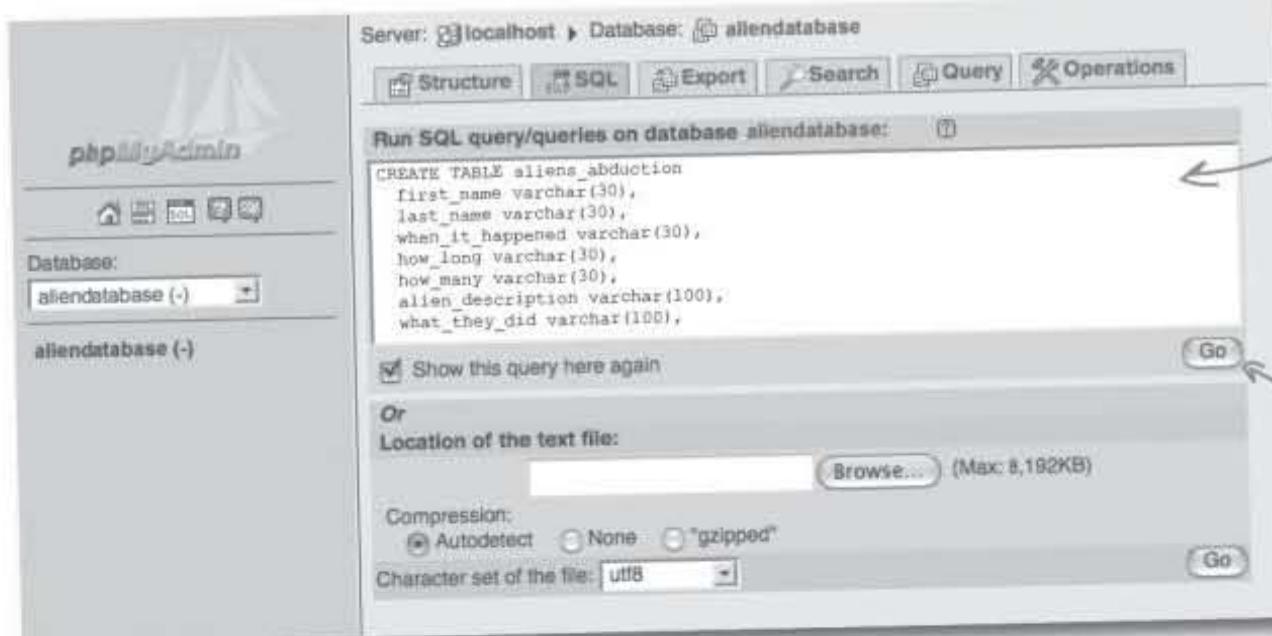


```

File Edit Window Help PhoneHome
mysql> CREATE TABLE aliens_abduction (
    first_name varchar(30),
    last_name varchar(30),
    when_it_happened varchar(30),
    how_long varchar(30),
    how_many varchar(30),
    alien_description varchar(100),
    what_they_did varchar(100),
    fang_spotted varchar(10),
    other varchar(100),
    email varchar(50)
);
Query OK, 0 rows affected (0.14 sec)

```

Your MySQL installation may offer the phpMyAdmin web-based tool, which lets you access your databases and tables graphically. You can use the phpMyAdmin user interface to click your way through the creation of a database and table, or enter SQL commands directly just as if you're in the MySQL terminal. Click the SQL tab in phpMyAdmin to access a text box that acts like the MySQL terminal.



Server: localhost > Database: aliendatabase

Structure SQL Export Search Query Operations

Run SQL query/queries on database aliendatabase:

```

CREATE TABLE aliens_abduction
    first_name varchar(30),
    last_name varchar(30),
    when_it_happened varchar(30),
    how_long varchar(30),
    how_many varchar(30),
    alien_description varchar(100),
    what_they_did varchar(100),

```

Show this query here again

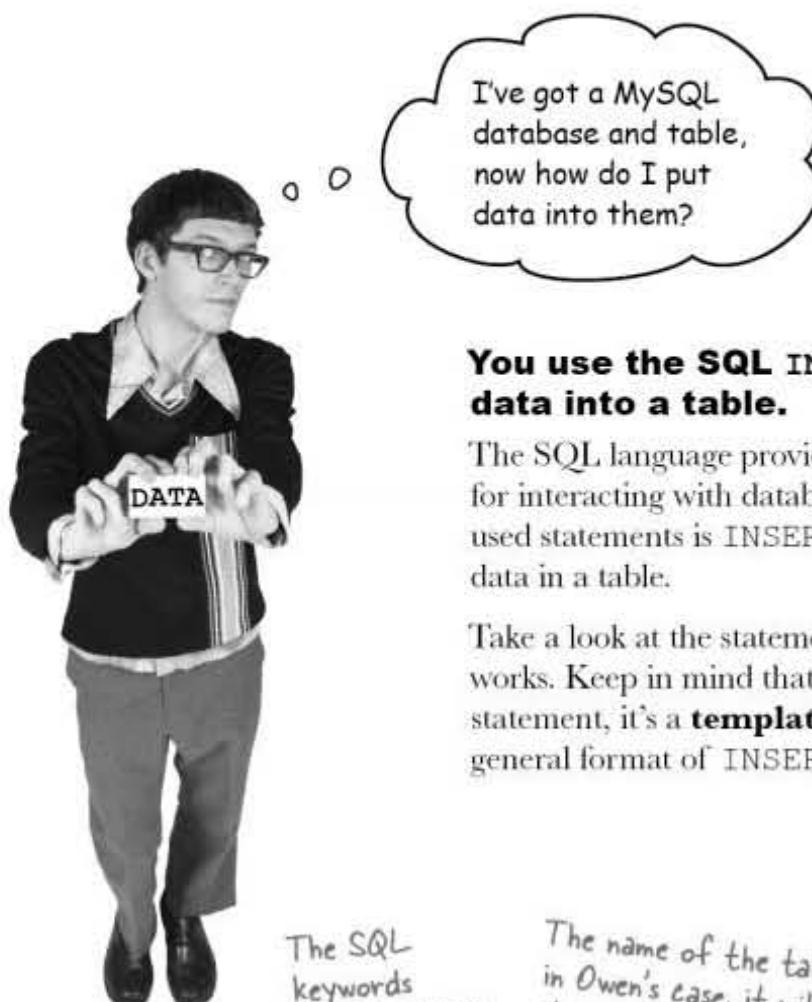
Or
Location of the text file: (Max: 8,192KB)

Compression: Autodetect None "gzipped"

Character set of the file: utf8

So the SQL tab of the phpMyAdmin application provides a way to issue SQL commands just as if you were using the MySQL terminal.

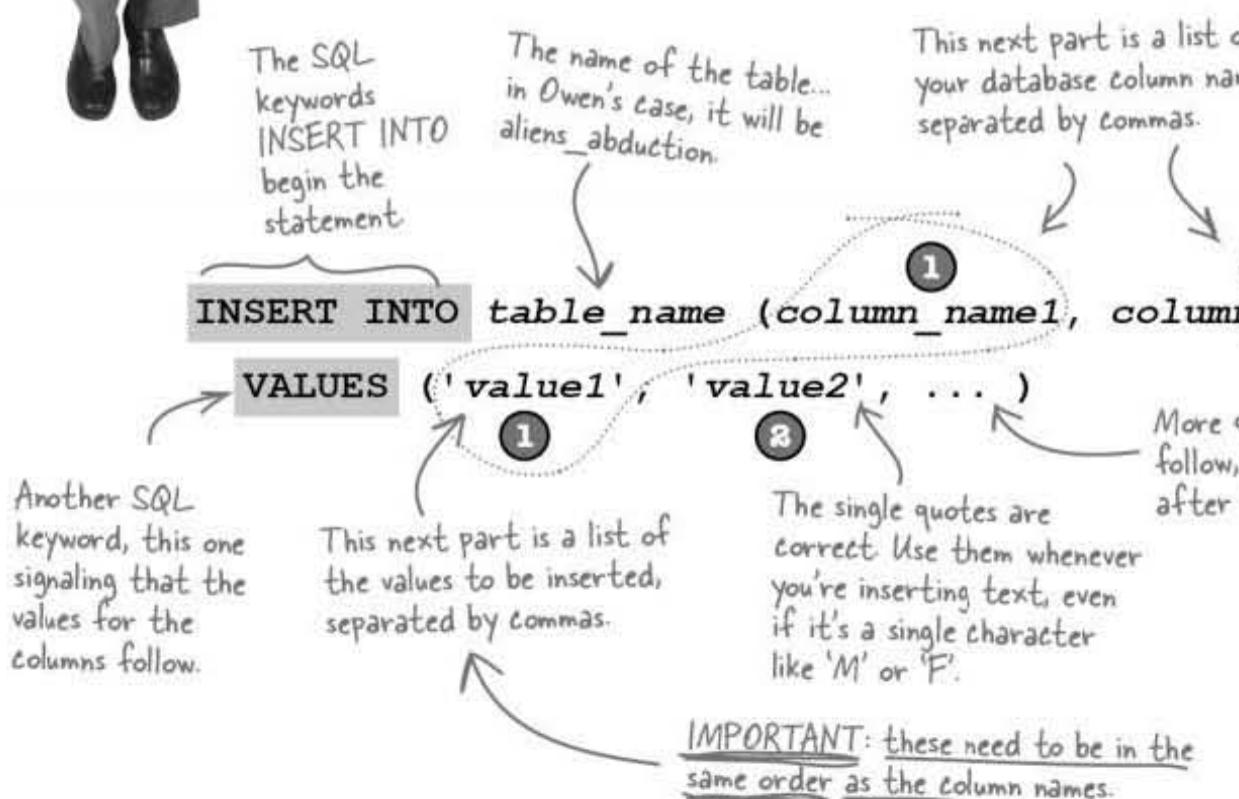
introducing the `INSERT` statement



You use the `SQL INSERT statement` to insert data into a table.

The SQL language provides all kinds of cool statements for interacting with databases. One of the more commonly used statements is `INSERT`, which does the work of storing data in a table.

Take a look at the statement below to see how the `INSERT` works. Keep in mind that this statement isn't an actual SQL statement, it's a **template** of a statement to show you the general format of `INSERT`.



One of the most important things to note in this statement is that the values in the second set of parentheses have to be in the **order as the database column names**. This is how the statement matches values to columns when it inserts the data.

solutions and no dumb questions



Sharpen your pencil Solution

The aliens_abduction table is shown below, but it doesn't have any data yet. Write Sally's alien abduction data into the table. It's OK to write some of the data above the table, and use arrows if you don't have room.

first_name	last_name	when_it_happened	how_long	how_many	alien_description	what_they_did	funny_things
Sally	Jones	3 days ago	1 day	four	green with six tentacles	We just talked and played with a dog.	I may have seen a dog. Contact me.

there are no
Dumb Questions

Q: I'm not sure I understand the difference between a database and a table. Don't they both just store data?

A: Yes. Tables serve as a way to divide up the data in a database into related groups so that you don't just have one huge mass of data. It's sort of like the difference between throwing a bunch of shoes into a huge box, as opposed to first placing each pair in a smaller box—the big box is the database, the smaller shoeboxes are the tables. So data is stored in tables, and tables are stored in databases.

Q: What exactly is the MySQL terminal? How do I find it on my computer?

A: The MySQL terminal is a **technique** for accessing a MySQL database server through a command-line interface. In many cases the MySQL terminal is not a unique program, but instead a connection you establish using the command line from a "generic" terminal program, such as the terminal application in Mac OS X. How you access the MySQL terminal varies widely depending on what operating system you are using and whether the MySQL server is local or remote (located somewhere other than your computer). Appendix ii has more details about how to go about accessing the MySQL terminal.

Q: What about phpMyAdmin? Where can I find it?

A: Unlike the MySQL terminal, phpMyAdmin is a web-based application that allows access to a MySQL database. It is actually a PHP application, which is why you access it from a web server, as opposed to installing it as a separate application. Many web hosting companies offer phpMyAdmin as part of their standard MySQL hosting package. You may already be installed for you. If not, you can download and install phpMyAdmin yourself. It is available for download from www.phpmyadmin.net. Just make sure that your web server supports PHP, and that phpMyAdmin is installed on a web server and configured correctly. Then you will have access to your MySQL databases, just like you would any other MySQL application.

Q: I have both the MySQL terminal and phpMyAdmin available. Which one should I use to access my database?

A: It's totally a personal preference. The main advantage of phpMyAdmin is that you can explore your database and its tables visually without having to enter SQL commands. This can be very handy once you get comfortable with MySQL, but if you don't want to manually enter commands to interact with your database, then the MySQL terminal is a good choice. However, for now it's a good idea to learn how to use both, understanding how to interact with your MySQL database using either tool.



Test Drive

Store an alien abduction sighting in your database with an SQL INSERT statement.

Using a MySQL tool such as the MySQL terminal or the SQL tab of phpMyAdmin, enter an INSERT statement for an alien abduction. As an example, here's the INSERT statement for Sally Jones' abduction:

```
INSERT INTO aliens_abduction (first_name, last_name,
when_it_happened, how_long, how_many, alien_description,
what_they_did, fang_spotted, other, email)
VALUES ('Sally', 'Jones', '3 days ago', '1 day', 'four',
'green with six tentacles', 'We just talked and played with',
'yes', 'I may have seen your dog. Contact me.',

'sally@gregs-list.net')
```

```
File Edit Window Help PugsInSpace
mysql> INSERT INTO aliens_abduction (first_name, last_name,
when_it_happened, how_long, how_many, alien_description,
what_they_did, fang_spotted, other, email)
VALUES ('Sally', 'Jones', '3 days ago', '1 day', 'four',
'green with six tentacles', 'We just talked and played with',
'yes', 'I may have seen your dog. Contact me.',

'sally@gregs-list.net');
Query OK, 1 rows affected (0.0005 sec)
```

Executing the INSERT statement in the MySQL terminal results in a new row of data being added to the aliens_abduction table.

The INSERT statement appears to have worked, but how can we confirm that the data was added?

introducing the SELECT statement

Use SELECT to get table data

Inserting data into a table is handy and all, but it's hard not to feel a certain sense of unease at the fact that you haven't confirmed that the data actually made its way into the table. It's kind of like depositing money into a savings account but never being able to get a balance. The SELECT statement is how you "get the balance" of a table in a database. Or more accurately, SELECT allows you to request columns of data from a table.

Follow SELECT with a list of the columns you want data for.

`SELECT columns FROM table_name`

A SELECT always takes place with respect to a specific table, not a database in general.

The FROM part of a SELECT statement is how SELECT knows what table we'll be selecting data from.

The columns supplied to a SELECT statement must be separated by commas. Regardless of how many columns a table has, only data in the columns specified in SELECT is returned. This SELECT statement grabs all of the first and last names of alien abductees from the aliens_abduction table:

Only the data for these two columns is returned by this SELECT statement.

`SELECT first_name, last_name FROM aliens_abduction`

The SELECT statement only retrieves data from the aliens_abduction table.

To check an INSERT, you need a quick way to look at **all of the data** in a table, not just a few columns. The SELECT statement offers a shortcut for just this thing:

The asterisk, or "star," tells the SELECT statement to get the data for all of the columns in the table.

`SELECT * FROM aliens_abduction`

No list of columns is necessary because * means "get them all!"



Test Drive

Make sure the alien abduction INSERT statement worked by SELECTing the table data.

Execute a SELECT query using a MySQL tool to view all of the contents of the aliens_abduction table. Make sure the new row of data you just inserted appears in the results.

```
SELECT * FROM aliens_abduction
```

These are the columns.

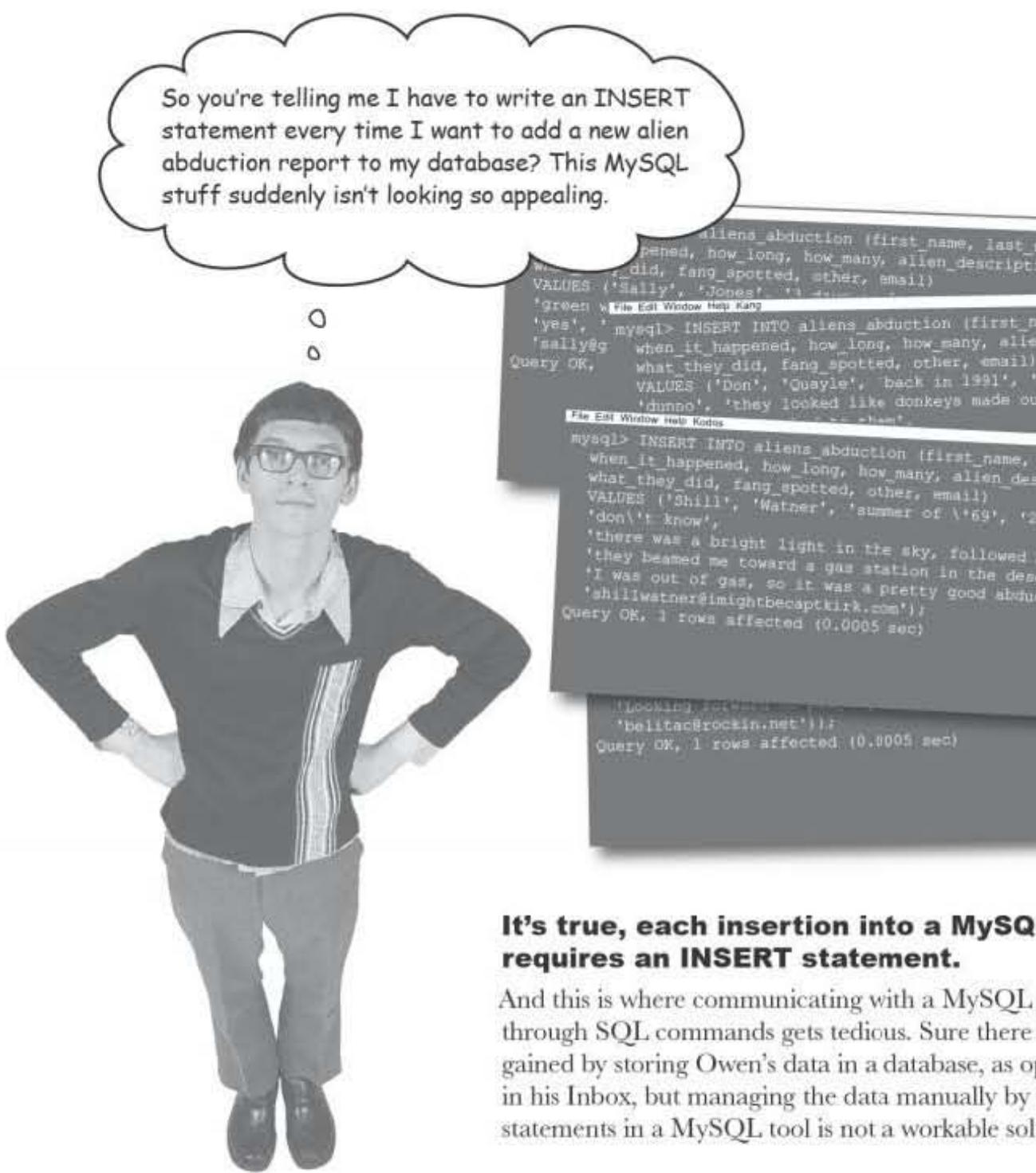
first_name	last_name	when_it_happened	how_long	how_many
Sally	Jones	3 days ago	1 day	four

1 row in set (0.0005 sec)

The SELECT query reveals a single row of data stored in the table.

Below each column name is the data for that column.

How many rows of data does your table have?

automating SQL commands with PHP

It's true, each insertion into a MySQL database requires an **INSERT statement.**

And this is where communicating with a MySQL database through SQL commands gets tedious. Sure there are benefits gained by storing Owen's data in a database, as opposed to his Inbox, but managing the data manually by running **INSERT** statements in a MySQL tool is not a workable solution.



How do you think Owen's MySQL insertion problem can be solved?

Let PHP handle the tedious SQL stuff

The solution to Owen's problem lies not in avoiding SQL but in **automating** SQL with the help of PHP. PHP makes it possible to issue SQL statements in script code that runs on the server, so you don't need to use a MySQL tool at all. This means Owen's HTML form can call a PHP script to handle inserting data into the database whenever it's submitted—no emails, no SQL tools, no hassle!

The HTML form generates an email that Owen receives and must then manually add to the database.

Aliens Abducted Me - Report an Abduction

Show your story of alien abduction:

First name:

Last name:

What is your email address?:

When did it happen?:
2 Aug 1997
12:00 PM

How long were you gone?:
1 day

How many did you see?:
1 alien

Describe them:
green with six tentacles

What did they do to you?:
They ate my dog!

Have you seen my dog back?:
Yes No (Yes)

Anything else you want to add? (Leave blank)

report.html

The HTML form calls a PHP script and asks it to add the form data to the database.

```
mysql> INSERT INTO aliens_abduction (first_name, last_name, when_it_happened, how_long, how_many, alien_description, what_they_did, fang_spotted, other, email)
VALUES ('Sally', 'Jones', '3 days ago', '1 day', 'green with six tentacles', 'We just talked and played with a dog', 'seen your dog. Contact me.', 'no');
Query OK, 1 row affected (0.0005 sec)
```

Without PHP, a manual SQL INSERT statement is required to store each alien abduction report in the database.

With PHP, a PHP script automatically handles the INSERT when the form is submitted.

```
<?php
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensroot',
or die('Error connecting to MySQL server.'));
$query = "INSERT INTO aliens_abduction (first_name, last_name, "
"when_it_happened, how_long, how_many, alien_description, "
"what_they_did, fang_spotted, other, email) "
"VALUES ('Sally', 'Jones', '3 days ago', '1 day', 'green with six tentacles', 'We just talked and played with a dog', "
"seen your dog. Contact me.', 'no')";
$result = mysqli_query($dbc, $query)
or die('Error querying database.');
mysqli_close($dbc);
?>
```

The PHP script creates an **INSERT** statement that inserts the form data into the database... no Owen required!

Owen creates a statement from the

how owen's application can use php and mysql

PHP lets data drive Owen's web form

PHP improves Owen's alien abduction web form by letting a script **send the form data directly to a database**, instead of sending it to Owen's email address and Owen entering it manually. Let's take a closer look at exactly how the application works now that PHP is in the picture.

- 1 Sally fills out the alien abduction form and presses the Report Abduction button to submit it. The information gets sent to the `report.php` script on the web server.



- 2 Lots and lots and lots of other people continue to submit the form too.



3

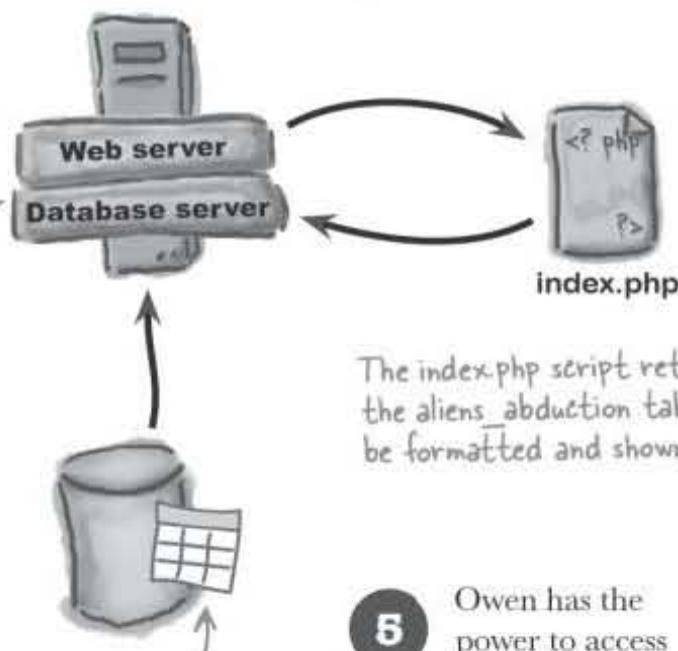
Owen's `report.php` script connects to a MySQL database and inserts the information from each submission using SQL INSERT statements.

**4**

Not only does Owen need a script to put the data in the database, he also needs a script to search and view the data. In fact, he needs a script for his main page for his web site. The `index.php` script connects to the MySQL database and retrieves alien abduction data, and shows it to Owen.

A database server is just a program running on a server computer, usually alongside the web server.

The aliens_abduction table serves as a data source for the index.php script.

4**5**

Owen has the power to access the data in many new ways, allowing him to really focus on finding his lost dog, Fang.

making a mysql connection

Connect to your database from PHP

Before a PHP script can insert or retrieve data from a MySQL database, it must connect to the database. Connecting to a MySQL database from PHP is similar in many ways to accessing a database from a MySQL tool, and it requires the same pieces of information. Remember the three checkboxes you filled out earlier in the chapter? Here they are again, along with a new one for the name of the database—go ahead and write them down one more time.

1 MySQL server location (IP address or hostname):

2 My database user name:

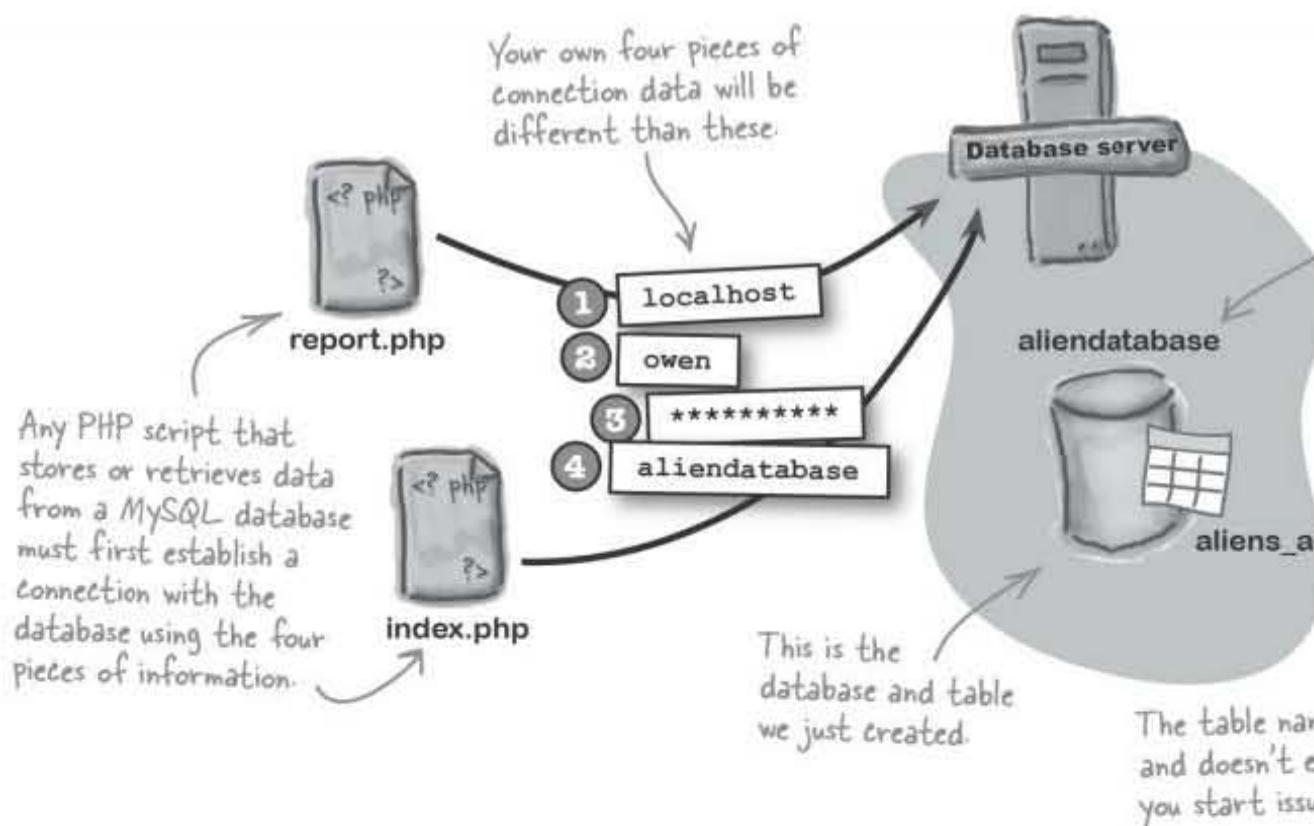
3 My database password:

4 My database name: ←

The database server host location, username, password, and database name are all required in order to establish a connection to a MySQL database from a PHP script. Once that connection is made, the script can carry out SQL commands just as if you were entering them manually in a MySQL tool.

Your web hosting provider may tell you that your server and MySQL are running on Linux. You can use the word "Linux" to impress your friends.

The name of the database you created earlier is aliendatabase. If for some reason you named it something else, or if you want to use a different database, when you created, use the



Insert data with a PHP script

Issuing a MySQL query from PHP code first requires you to establish a connection with the database. Then you build the query as a PHP string. The query isn't actually carried out until you pass along the query string to the database server. And finally, when you're finished querying the database, you close the connection. All of these tasks are carried out through PHP script code. Here's an example that inserts a new row of alien abduction data:

```
<?php
    Connect to the
    MySQL database.
    $dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensroot', 'aliens');
    or die('Error connecting to MySQL server.');

    You may be able to use 'localhost' instead
    of the database location instead.

    $query = "INSERT INTO aliens_abduction (first_name, last_name, when_it_happened,
        "how_many, alien_description, what_they_did, fang_spotted, other, email) "
        . "VALUES ('Sally', 'Jones', '3 days ago', '1 day', 'four', 'green with six tentacles',
        '"We just talked and played with a dog', 'yes', 'I may have seen your dog. Can I have one?',
        "'sally@gregs-list.net')";

    Build the INSERT query
    as a string in PHP code.

    $result = mysqli_query($dbc, $query)
    or die('Error querying database.');

    Issue the INSERT query
    on the MySQL database.

    mysqli_close($dbc);
?>
```

These functions require
your web server to have
PHP version 4.1 or greater.



What do you think each of these PHP functions is doing in the script?

`mysqli_connect()`
`mysqli_query()`
`mysqli_close()`

php's three mysql connection functions

Use PHP functions to talk to the database

There are three main PHP functions used to communicate with a MySQL database: `mysqli_connect()`, `mysqli_query()`, and `mysqli_close()`. If you see a pattern it's no accident—all of the modern PHP functions that interact with MySQL begin with `mysqli_`.

An older
that int
begin wi
the "i".
"improved
function

`mysqli_connect()`

Connect to a MySQL database using the four pieces of information you already learned about.

`mysqli_query()`

Issue a query on a MySQL database, which often involves storing or retrieving data from a table.

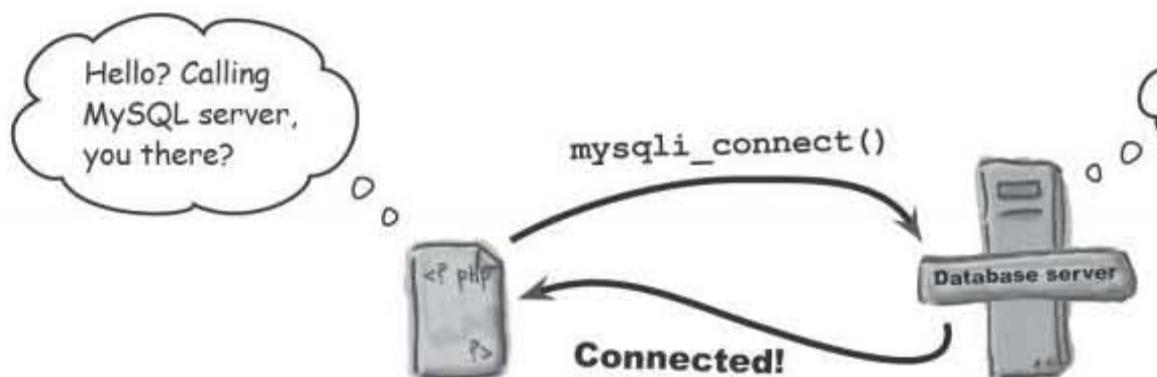
`mysqli_close()`

Close a connection to a MySQL database.

Using these three functions typically involves a predictable sequence of steps.

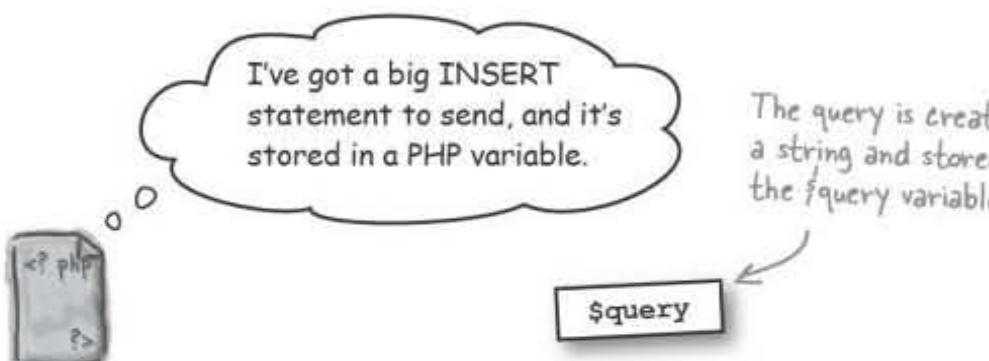
① Connect to a database with the `mysqli_connect()` function.

Provide the server location, username, and password to get permission to interact with the server. Also specify the database name since this is a connection to a specific database.



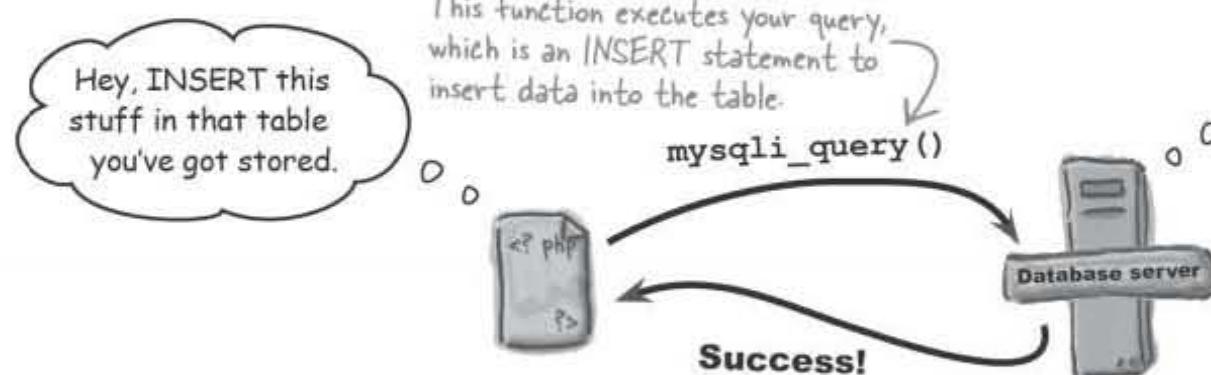
② Create an SQL query and store it as a string in a PHP variable.

To communicate with the database server, you have to use SQL commands. For example, an `INSERT` statement is needed to add data to the `aliens_abduction` table. There's nothing special about the variable name we chose, but a straightforward name like `$query` works just fine.



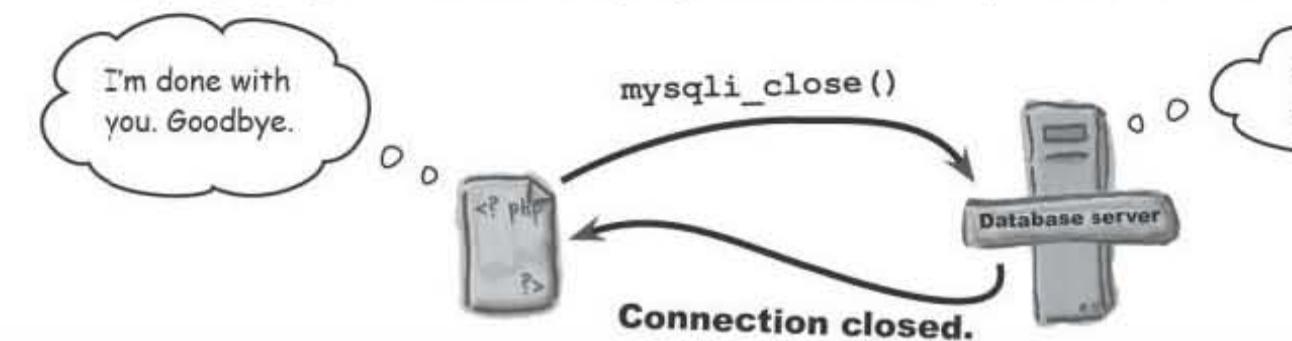
3 Issue the query with the mysqli_query() function.

Use the \$query variable with the mysqli_query() function to talk to the MySQL database server and add data to the aliens_abduction table. You have to tell mysqli_query() both the connection you created back in Step 1 and the name of the variable that holds your query.



4 Close the database connection with the mysqli_close() function.

Finally, mysqli_close() tells the MySQL database server that you are finished communicating.



```

This is the name of your
connection variable.

1 <?php
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensrool', 'aliens');
or die('Error connecting to MySQL server.');

2 $query = "INSERT INTO aliens_abduction (first_name, last_name, when_it_happened,
    "how_many, alien_description, what_they_did, fang_spotted, other, email) "
    "VALUES ('Sally', 'Jones', '3 days ago', '1 day', 'four', 'green with six teeth',
    "'We just talked and played with a dog', 'yes', 'I may have seen your dog. Can I have one?', 
    "'sally@gregs-list.net')";

3 $result = mysqli_query($dbc, $query)
or die('Error querying database.');

4 mysqli_close($dbc);
?>

```

Annotations for the code:

- Annotation 1: Points to the connection variable \$dbc with the text "This is the name of your connection variable."
- Annotation 2: Points to the error handling line with the text "If something goes wrong, this will send back a message to you and stop everything."
- Annotation 3: Points to the mysqli_query() call with the text "mysqli_query() is how PHP communicates with the MySQL server. The code stored in the \$query variable is SQL code, not PHP code."
- Annotation 4: Points to the mysqli_close() call with the text "Here's where we close the connection."

Let's take a closer look at each one of these functions, starting with mysqli_connect().

using mysqli_connect()

Get connected with mysqli_connect()

For our PHP script to be able to create a connection to the database with the `mysqli_connect()` function, you'll need a few pieces of information that you're starting to get very familiar with. Yes, it's the same information you used earlier when working with the MySQL terminal, plus the name of the database.

- 1 Connect with my...
- 2 Assemble the qu...
- 3 Execute the que...
- 4 Close the connec...

Who?

Your username and password

You'll need your own username and password for your own database server. These will either be set up by you or given to you by your web hosting company when MySQL is first installed. If you set up your own MySQL, follow the instructions to give yourself a secure username and password.

What?

The name of your database

In our example, we've named the database `aliendatabase`. Yours will be whatever name you decided to give it when you set it up earlier, or if your web hosting company created your database for you, you'll be using that name.

Where?

The location of the database (a domain name, an IP address or localhost)

In our example, we're using the location of Owen's (fictional) database. You need to use the location of your own MySQL server. Often, this is `localhost` if the database server is on the same machine as your web server. Your web hosting company will be able to tell you this. It may also be an IP address or a domain name like Owen's, such as `yourserver.yourisp.com`.

The location, username, password, and name of the MySQL database in the `mysqli_connect()` function must all have quotes around them.

```
$dbc = mysqli_connect(
    'data.aliensabductedme.com',
    'owen',
    'aliensrool',
    'aliendatabase');
```

The diagram shows the `mysqli_connect()` function call with several annotations:

- An arrow points from the text "Use this variable to perform other actions on the database." to the opening parenthesis of the function call.
- An arrow labeled "Username" points to the string "owen".
- An arrow labeled "Password" points to the string "aliensrool".
- An arrow labeled "Database name" points to the string "aliendatabase".
- An arrow labeled "Location of the database" points to the string "data.aliensabductedme.com".

The result of calling the function is a database connection and a PHP variable that you can use to interact with the database. The variable is named `$dbc` in the example, but you can name it anything you like.

The my...
function...
location...
password...
database...
strings,...
quote t...



Sharpen your pencil

Here are some examples of PHP database connection code. Look at each one and then write down whether or not it is correct. If it is wrong, write down what is wrong and how to fix it. Also circle any of the code you find suspicious.

```
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensrool',
    'aliendatabase');
```

.....

```
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensrool',
    "aliendatabase")
```

.....

```
$fangisgone = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensrool',
    'aliendatabase');
```

.....

```
$dbc = mysqli_connect('localhost', 'owen', 'aliensrool', 'aliendatabase');
```

.....

```
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', '', 'aliendatabase');
```

.....

```
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensrool',
    mysqli_select_db($dbc, 'aliendatabase'));
```

.....

sharpen solution

Sharpen your pencil Solution

Here are some examples of PHP database connection code. Look at each one and then write down whether or not it will work, and how to fix it. Also circle any of the code you find confusing.

```
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensrool',
    'aliendatabase');
```

This connection string will work.

You need a semicolon here to terminate the PHP statement and reserving double quotes for strings.

```
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensrool',
    "aliendatabase");
```

This won't work because it's missing a semicolon. The double quotes will work just like the single quotes.

Not a very descriptive name for a database connection.

```
$fangisgone = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensrool',
    'aliendatabase');
```

This will work, although it's not a very good name for a database connection.

This assumes the database server is located on the same server computer as the web server.

```
$dbc = mysqli_connect('localhost', 'owen', 'aliensrool', 'aliendatabase');
```

This will work, assuming the web server and database server are on the same machine.

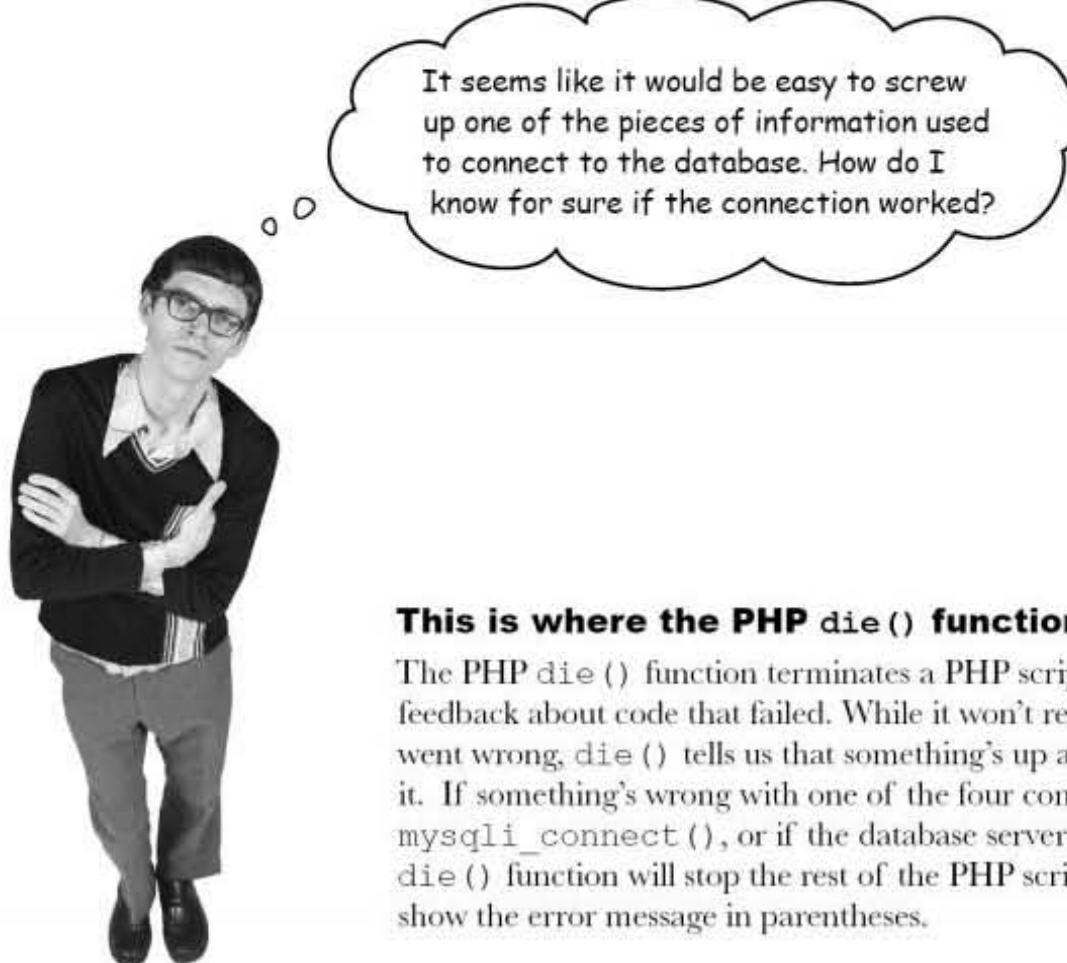
An empty database password is not a good idea.

```
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', '', 'aliendatabase');
```

This will work only if you set a blank password for the database. Not a good idea, though. You should always have a password set for each database.

```
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensrool',
    mysqli_select_db($dbc, 'aliendatabase'));
```

Sorry, this is a trick question. In `mysqli_connect()`, that fourth item, the name of the database, is optional. You can leave it out of the function and use `mysqli_select_db()` to specify the name of the database instead. So this code is the same as if you had passed all four arguments to `mysqli_connect()`.



This is where the PHP die() function comes in handy.

The PHP `die()` function terminates a PHP script and provides feedback about code that failed. While it won't reveal precisely what went wrong, `die()` tells us that something's up and that we need to fix it. If something's wrong with one of the four connection variables in `mysqli_connect()`, or if the database server can't be located, the `die()` function will stop the rest of the PHP script from running and show the error message in parentheses.

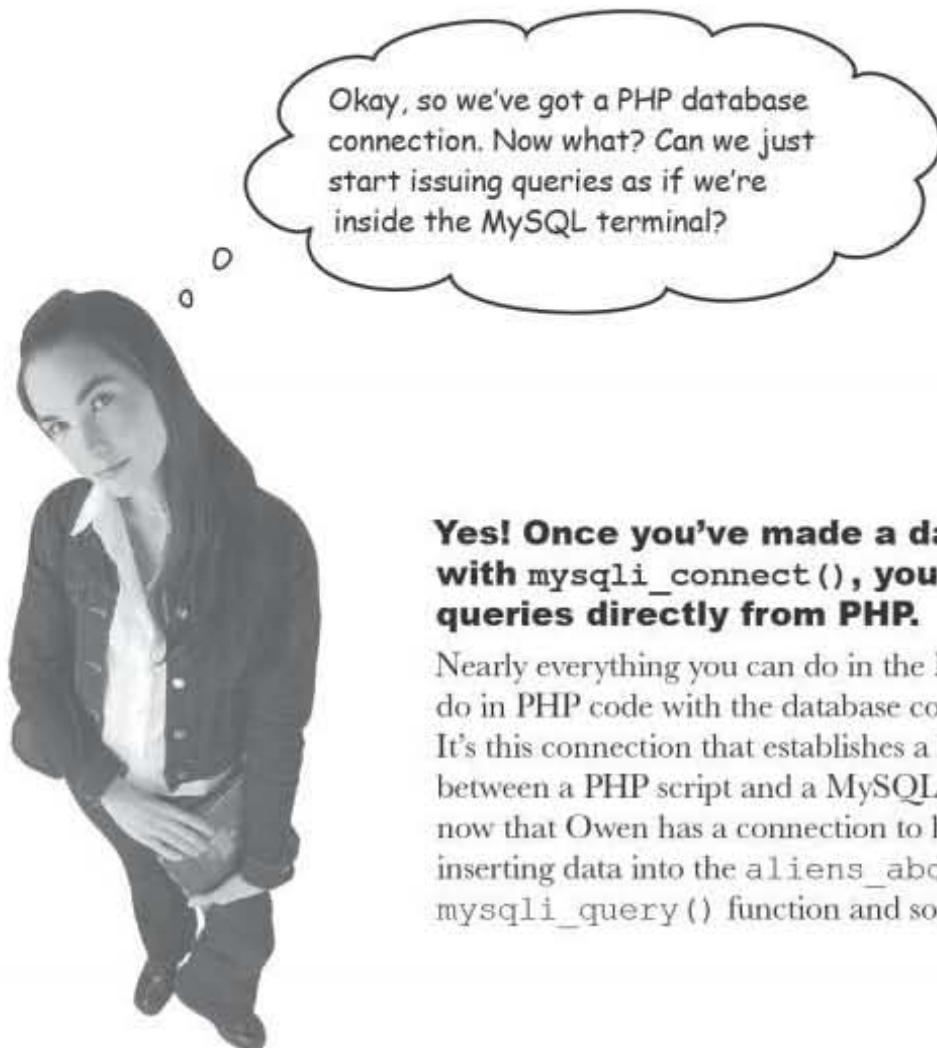
```
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'alienschool', 'alienschool');
or die('Error connecting to MySQL server.');
```

The `die()` function is called if the connection isn't created.

If one of our four strings in the `mysqli_connect()` function isn't right, we'll get feedback.

This message is echoed to the web page if the connection fails.

A semicolon isn't required since "or die(...)" is a continuation of a statement.

building queries in php

Yes! Once you've made a database connection with `mysqli_connect()`, you can issue SQL queries directly from PHP.

Nearly everything you can do in the MySQL terminal you can do in PHP code with the database connection you've now established. It's this connection that establishes a line of communication between a PHP script and a MySQL database. For example, now that Owen has a connection to his database, he can insert data into the `aliens_abduction` table with the `mysqli_query()` function and some SQL query code:

```
File Edit Windows Help UFO
mysql> INSERT INTO aliens_abduction (first_name, last_name,
   when_it_happened, how_long, how_many, alien_description,
   what_they_did, fang_spotted, other, email)
VALUES ('Sally', 'Jones', '3 days ago', '1 day', 'four',
'green with six tentacles', 'we just talked and played with a dog',
'yes', 'I may have seen your dog. Contact me',
'sally@gregs-list.net');
Query OK, 1 rows affected (0.0005 sec)
```

`mysqli_query($dbc, $query)`

The SQL query is passed to `mysqli_query()` as a PHP string.

The `mysqli_query()` function needs an SQL query string (`$query`) in order to carry out the insertion of alien abduction data.

Remember
automate
query usin



Build the INSERT query in PHP

SQL queries in PHP are represented as strings, and it's customary to store a query in a string before passing it along to the `mysqli_query()` function. Since SQL queries can be fairly long, it's often necessary to construct a query string from smaller strings that span multiple lines of code. Owen's INSERT query is a good example of this:

```
$query = "INSERT INTO aliens_abduction (first_name, last_name,
    when_it_happened, how_long, how_many, alien_description,
    what_they_did, fang_spotted, other, email) "
VALUES ('Sally', 'Jones', '3 days ago', '1 day', 'four',
    'green with six tentacles', 'We just talked and played with',
    "'yes', 'I may have seen your dog. Contact me.', "
    "'sally@gregs-list.net')";
```

This is a PHP string variable that now holds the INSERT query.

Since this entire piece of code is PHP code, it must be terminated with a semicolon.

With the INSERT query stored in a string, you're ready to pass it to the `mysqli_query()` function and actually carry out the query.

The query string is broken into multiple lines to make the query more readable. You can tell PHP to turn this into one single string by using the `\n` character.

there are no
Dumb Questions

Q: Why is an INSERT into a database called a query? Doesn't "query" mean we're asking the database for something?

A: Yes, "query" does mean you're asking for something...you're asking the database to do something. In MySQL database applications, the word "query" is quite general, referring to any SQL command you perform on a database, including both storing and retrieving data.

Q: Why isn't the INSERT statement just created as one big string?

A: Keep in mind that the INSERT statement is stored as one big string, even though it is created from multiple smaller strings. Ideally, the INSERT statement would be coded as a single string. But like many SQL statements, the INSERT statement is quite long and doesn't fit on a "normal" line of code. So it's easier to read the query string if it's coded as smaller strings that are glued together with periods.

- 1 Connect with my database
- 2 Assemble the query string
- 3 Execute the query
- 4 Close the connection

The process
tackles the
string

Q: Is it really necessary to use column names in the INSERT statement?

A: No. You can use column names in the INSERT statement. In fact, in some cases, you must use column names. If the columns in the table don't appear in the same order as they appear in the query, you must specify the column names.

the `mysqli_query()` function

Query the MySQL database with PHP

The `mysqli_query()` function needs two pieces of information to carry out a query: a database connection and an SQL query string.

- 1 Connect with my...
- 2 Assemble the que...
- 3 Execute the que...
- 4 Close the connec...

`mysqli_query(database_connection, query);`

This is a database connection that's already been established via the `mysqli_connect()` function.

This is the SQL query that will be performed and stored in a variable.

The database connection required by the `mysqli_query()` function was returned to you by the `mysqli_connect()` function. Just in case that's a bit fuzzy, here's the code that established that connection:

```
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensroot');
or die('Error connecting to MySQL server.');
```

The connection to the database was stored away earlier in the \$dbc variable.

Remember variables for you.

So you have a database connection (`$dbc`) and an SQL query (`$query`). All that's missing is passing them to the `mysqli_query()` function.

```
$result = mysqli_query($dbc, $query);
or die('Error querying database.');
The result of the query
```

The query

The database connection.

This code shows that calling the `mysqli_query()` function isn't just a one-way communication. The function talks back to you by returning a piece of information that's stored in the `$result` variable. But no actual data is returned from the `INSERT` query—the `$result` variable just stores whether or not the query issued by `mysqli_query()` was successful.

The `mysqli_query()` function requires a database connection and a query string in order to carry out an SQL query.

An SQL query is a command written in code that's sent to a database.

Close your connection with `mysqli_close()`

Since we're only interested in executing the single INSERT query, the database interaction is over, at least as far as the script is concerned. And when you're done with a database connection, you should close it. Database connections will close by themselves when the user navigates away from the page but, just like closing a door, it's a good habit to close them when you're finished. The PHP `mysqli_close()` function closes a MySQL database connection.

- 1 Connect with `mysqli_connect()`
- 2 Assemble the query
- 3 Execute the query
- 4 Close the connection with `mysqli_close()`

It's a good habit to close database connections when finished.

```
mysqli_close(database_connection);
```

This is where you pass the database connection variable that we've been using to interact with the database.

In the case of Owen's script, we need to pass `mysqli_close()` the actual database connection, which is stored in the `$dbc` variable.

```
mysqli_close($dbc);
```

This variable holds a reference to the database connection, which was created by `mysqli_connect()` back when the connection was first opened.

But if database connections are closed automatically, why bother?



Database servers only have a certain number of connections available at a time, so they are preserved whenever possible.

And when you close one connection, it frees that connection so that a new one can be created. If you are on a shared host, you might only have five connections allocated to you, for example. As you create new database-driven applications, you'll want to keep your supply of available connections open as much as possible.

no dumb questions and bullet points

there are no
Dumb Questions

Q: Couldn't you just put all the SQL code directly in the `mysqli_query()` function in place of the `$query` variable?

A: You could, but it gets messy. It's just a bit easier to manage your code when you store your queries in variables, and then use those variables in the `mysqli_query()` function.

Q: Should the code that issues the query do anything with the result?

A: Perhaps, yes. So far we've been using the `die()` function to stop a script and send a message to the browser if a query fails. Eventually you may want to provide more feedback to the user, such as when a query's unsuccessful, in which case you might want to use the `echo` command to output the query to determine the query's success or failure.

BULLET POINTS

- 
- Database connections need a location, a username, a password, and a database name.
 - The `mysqli_connect()` function creates a connection between your PHP script and the MySQL database server.
 - The `die()` function exits the script and returns feedback if your connection fails.
 - Issuing an SQL query from PHP code involves assembling the query in a string and then executing it with a call to `mysqli_query()`.
 - Call the `mysqli_close()` function to close a MySQL database connection from PHP when you're finished with it.



Test Drive

Replace the email code in Owen's report.php script so that it inserts data into the MySQL database, and then try it out.

Remove the code in the report.php script that emails form data to Owen. In its place, enter the code that connects to your MySQL database, builds a SQL query as a PHP string, executes the query on the database, and then closes the connection.

Here's the new PHP database code you've been working on. Don't enter the <?php ?> tags in report.php since you're adding this code to a spot in the script that's already inside the tags.

```
<?php
$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensrocks');
or die('Error connecting to MySQL server.');

$query = "INSERT INTO aliens_abduction (first_name, last_name, "
        . "when_it_happened, how_long, how_many, alien_description, "
        . "what_they_did, fang_spotted, other, email";
$query .= "VALUES ('Sally', 'Jones', '3 days ago', '1 day', 'four', "
        . "'green with six tentacles', 'We just talked and played with a dog",
        . "'yes', 'I may have seen your dog. Contact me.', "
        . "'sally@gregs-list.net')";

$result = mysqli_query($dbc, $query);
or die('Error querying database.');

mysqli_close($dbc);
?>
```

Upload the new report.php file to your web server, and then open the report.html page in a browser to access the Report an Abduction form. Fill out the form and click Report Abduction to store the data in the database. Now fire up your MySQL tool and perform a SELECT query to view any changes in the database.

```
File Edit Window Help Administration
mysql> SELECT * FROM aliens_abduction;
+-----+-----+-----+-----+-----+-----+
| first_name | last_name | when_it_happened | how_long | how_many | alien_description |
+-----+-----+-----+-----+-----+-----+
| Sally      | Jones     | 3 days ago       | 1 day    | four     | green with six tentacles |
| Sally      | Jones     | 3 days ago       | 1 day    | four     | green with six tentacles |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.0005 sec)
```

Is this correct? Write down if you think you should be doing, and why.

.....
.....
.....

use `$_POST` in the `INSERT` query

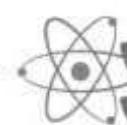


Hang on a second. Isn't the whole point here to take data from a **form** and store it in a database? It looks like the query's inserting the same data no matter what gets entered into the form. I don't see how this PHP script automates anything.

This is a big problem. The `INSERT` query needs to insert the form data, not static strings.

The query we've built consists of hard coded strings, as opposed to data driven from text data that was entered into the alien abduction form. In order for the script to work with the form, we need to feed the data from the fields into the query string.



 **BRAIN POWER**

What PHP code can help us get the data from Owen's form into the `INSERT` query?

\$_POST provides the form data

The good news is that the `report.php` script already has the form data stored away in variables thanks to the `$_POST` superglobal. Remember this PHP code?

```
$name = $_POST['firstname'] . ' ' . $_POST['lastname'];
$when_it_happened = $_POST['whenithappened'];
$how_long = $_POST['howlong'];
$how_many = $_POST['howmany'];
$alien_description = $_POST['aliendescription'];
$what_they_did = $_POST['whattheydid'];
$fang_spotted = $_POST['fangspotted'];
$email = $_POST['email'];
$other = $_POST['other'];
```

The `$_POST` superglobal is being used to extract data from each of Owen's form fields and store it in variables.

Remember, the name you use for `$_POST` needs to match up with the name of an HTML form field.

So you already have the form data in hand, you just need to incorporate it into the alien abduction `INSERT` statement. But you need to make a small change first. Now that you're no longer emailing the form data, you don't need the `$name` variable. You *do* still need the first and last name of the user so that they can be added to the database—but you need the names in separate variables.

```
$first_name = $_POST['firstname'];
$last_name = $_POST['lastname'];
```

The user's name is now stored in separate variables so that it can be inserted into distinct columns of the `aliens_abduction` table.



Write the PHP code to create Owen's `INSERT` query string that is stored in the `$query` variable, making sure that it stores actual form data in the `aliens_abduction` table being executed.

.....
.....
.....
.....

single and double quotes in php

Write the PHP code to create Owen's INSERT query string that is stored in a variable, making sure that it stores actual form data in the `aliens_abduction` table being executed.

The column names appear
in the SQL statement
exactly as they did before.

```
$query = "INSERT INTO aliens_abduction (first_name, last_name, when_it_happened, how_long,
    how_many, alien_description, what_they_did, fang_spotted, other, email) "
VALUES ('$first_name', '$last_name', '$when_it_happened', '$how_long', '$how_many',
    '$alien_description', '$what_they_did', '$fang_spotted', '$other', '$email')";
```

Instead of static data about
Sally Jones' abduction, now we
insert whatever data the user
entered into the form.

The order of the variables
match the order of the
column names for the data to
be inserted into the correct columns.

there are no Dumb Questions

Q: Do I have to create all those variables to store the `$_POST` data? Can't I just reference the `$_POST` data directly into the `$query` string?

A: Yes, you can. There's nothing stopping you from putting `$_POST` directly in a query. However, it's a good coding habit to isolate form data before doing anything with it. This is because it's fairly common to process form data to some degree before inserting it into a database. For example, there are clever ways for hackers to try and hijack your queries by entering dangerous form data. You'll learn how to thwart such attempts in Chapter 6. To keep things simple, this chapter doesn't do any processing on form data, but that doesn't mean you shouldn't go ahead and get in the habit of storing form data in your own variables first before sticking it in a query.

Q: OK, so does it matter where you use single quotes versus double quotes? Can I use single quotes around the whole query and double quotes around each variable?

A: Yes, it matters. And no, you can't use single quotes around the whole query with double quotes around the variables. That's because PHP treats strings differently depending on whether they appear inside single quotes or double quotes. The difference between the two is that single quotes represent the characters contained within them, while some additional processing is done on the text within double quotes. This processing is done inside of double quotes getting processed as part of the string in lieu of the variable name. This is why single quotes are generally preferred for database queries.

Q: Couldn't you just build query strings by concatenating variables with the SQL code?

A: Yes, and if you went the concatenation route, you could certainly use single quotes instead of double quotes. Single quotes strings tend to be messy as it is, so anything that makes them more readable is a good thing—especially when you're working in a double-quoted string instead of concatenating strings together. Single quotes definitely makes query strings easier to read.



Let's use everything we've learned to finish Owen's form-handling PHP script. You'll successfully store alien abduction data in a database. Finish the PHP code for the report.php script.

```
<?php

.....
$when_it_happened = $_POST['whenithappened'];
>Show_long = $_POST['howlong'];
>Show_many = $_POST['howmany'];
$alien_description = $_POST['aliendescription'];
$what_they_did = $_POST['whattheydid'];
$fang_spotted = $_POST['fangspotted'];
$email = $_POST['email'];
$other = $_POST['other'];

$dbc =
.....
.....
$query = "INSERT INTO aliens_abduction (first_name, last_name, when_it_happened, how_many, alien_description, what_they_did, fang_spotted, other, email) "
VALUES ('$first_name', '$last_name', '$when_it_happened', '$show_long', '$show_many',
'$alien_description', '$what_they_did', '$fang_spotted', '$other', '$email'
);

$result =
.....
.....
echo 'Thanks for submitting the form.<br />';
echo 'You were abducted ' . $when_it_happened;
echo ' and were gone for ' . $show_long . '<br />';
echo 'Number of aliens: ' . $show_many . '<br />';
echo 'Describe them: ' . $alien_description . '<br />';
echo 'The aliens did this: ' . $what_they_did . '<br />';
echo 'Was Fang there? ' . $fang_spotted . '<br />';
echo 'Other comments: ' . $other . '<br />';
echo 'Your email address is ' . $email;
?>
```

exercise solution

Let's use everything we've learned to finish Owen's form-handling PHP script. We'll successfully store alien abduction data in a database. Finish the code below for report.php.

```
<?php
$first_name = $_POST['firstname'];
$last_name = $_POST['lastname'];
$when_it_happened = $_POST['whenithappened'];
>Show_long = $_POST['howlong'];
>Show_many = $_POST['howmany'];
$alien_description = $_POST['aliendescription'];
$what_they_did = $_POST['whattheydid'];
$fang_spotted = $_POST['fangspotted'];
$email = $_POST['email'];
$other = $_POST['other'];

$dbc = mysqli_connect('data.aliensabductedme.com', 'owen', 'aliensrool', 'aliendatabase')
      or die('Error connecting to MySQL server.');

$query = "INSERT INTO aliens_abduction (first_name, last_name, when_it_happened,
                                             how_many, alien_description, what_they_did, fang_spotted, other, email) "
      . "VALUES ('$first_name', '$last_name', '$when_it_happened', '$Show_long', '$Show_many',
              '$alien_description', '$what_they_did', '$fang_spotted', '$other', '$email')";

$result = mysqli_query($dbc, $query)
      or die('Error querying database.');

mysqli_close($dbc);
?>
```

The new name variables hold the first and last name of the user, as entered into the form.

You must connect to the database and provide the proper connection information before performing SQL queries from PHP.

The query is a string, making sure to extract the variables from the form.

Execute the query on the database - this inserts the data!

Close the database connection.

Confirm the successful form submission, just like you did in the old script.



Test Drive

Change Owen's script to use actual form data when you do.

Remove the \$name variable in the report.php script, add the \$first_name and \$last_name variables, and modify the \$query variable to use form variables instead of static text in the INSERT statement. Upload the new version of the script and test it by submitting the form in the report.html page a few times, making sure to change the data each time.

Aliens Abducted Me - Report an Abduction

Show your story of alien abduction:

First name: Last name: What is your email address? When did it happen? How long were you gone? How many did you see? Describe them: What did they do to you? Yes No Have you seen my dog Fang? Yes No

Anything else you want to add?

Aliens Abducted Me - Report an Abduction

Show your story of alien abduction:

First name: Last name: What is your email address? When did it happen? How long were you gone? How many did you see? Describe them: What did they do to you? Yes No Have you seen my dog Fang? Yes No

Anything else you want to add?

Aliens Abducted Me - Report an Abduction

Show your story of alien abduction:

First name: Last name: What is your email address? When did it happen? How long were you gone? How many did you see? Describe them: What did they do to you? Yes No Have you seen my dog Fang? Yes No

Anything else you want to add?

Now use your MySQL tool to carry out a SELECT and view the contents of the aliens_abductions table.

The new alien
abduction
reports appear in
the table just as
you would expect!

```
File Edit Window Help BeaM@BeaM: ~
mysql> SELECT * FROM aliens_abductions;
+-----+-----+-----+-----+-----+-----+-----+
| first_name | last_name | when_it_happened | how_long | how_many | alien_description |
+-----+-----+-----+-----+-----+-----+
| Sally | Jones | 3 days ago | 1 day | four | green with six tentacles |
| Sally | Jones | 3 days ago | 1 day | four | green with six tentacles |
| Dan | Quayle | back in 1991 | 37 seconds | dunno | they looked like dorks |
| Bill | Watson | summer of '99 | 2 hours | don't know | there was a bright light |
| Alf | Radec | last November | 11 boxes | doesn't care | little green men |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.0005 sec)
```

adding WHERE to your SELECT

Owen needs help sifting through his data

The new and improved `report.php` script is doing its job and automating the process of adding alien abduction reports to the database. Owen can just sit back and let the reports roll in... except that there's a new problem. More data isn't exactly making it any easier to hone in on alien abduction reports involving a potential Fang sighting.

I'm really stoked that I've now got a database automatically filled with alien abduction reports submitted by users. But it doesn't help me isolate the reports that might help me find Fang.

Owen needs a way to find specific data, such as alien abductions where Fang was spotted.

You know what column of the database contains the information in question: `fang_spotted`. This column contains either yes or no depending on whether the abductee reported that they saw Fang. So what you need is a way to select only the reports in the `aliens_abduction` table that have a value of `yes` in the `fang_spotted` column.

You know that the following SQL query returns all of the data in the table:

```
SELECT * FROM aliens_abduction
```

The SQL `SELECT` statement lets you tack on a clause to control the data returned by the query. It's called `WHERE`, and you tell it exactly how you want to filter the query results. In Owen's case, this means only selecting alien abduction reports where the `fang_spotted` column equals yes.

```
SELECT * FROM aliens_abduction WHERE fang_spotted
```

This part of the `SELECT` query stays the same – the `WHERE` clause takes care of whittling down the results.

The name of the column

Remember, without the `WHERE` clause, this causes all of the data in the table to be selected.

This clause reduces the data returned by the query to the data where the `fang_spotted` column is set to yes.

The value column must be in order for data to be selected.



Test Drive

Try out the SELECT query with a WHERE clause to find specific abduction data.

Use a SELECT query with a WHERE clause in your MySQL tool to search for all abduction data that specifically involves Fang sightings.

```

File Edit Window Help HaveYouSeenHim
mysql> SELECT * FROM aliens_abduction WHERE fang_spotted = 'yes';

+-----+-----+-----+-----+
| first_name | last_name | when_it_happened | how_long |
+-----+-----+-----+-----+
| Sally      | Jones     | 3 days ago       | 1 day    |
| Sally      | Jones     | 3 days ago       | 1 day    |
| Don        | Quayle   | back in 1991     | 37 seconds |
| Shill      | Watner   | summer of '69     | 2 hours  |
| Mickey     | Mikens   | just now         | 45 minutes...and counting |
+-----+-----+-----+-----+
5 rows in set (0.0005 sec)

```

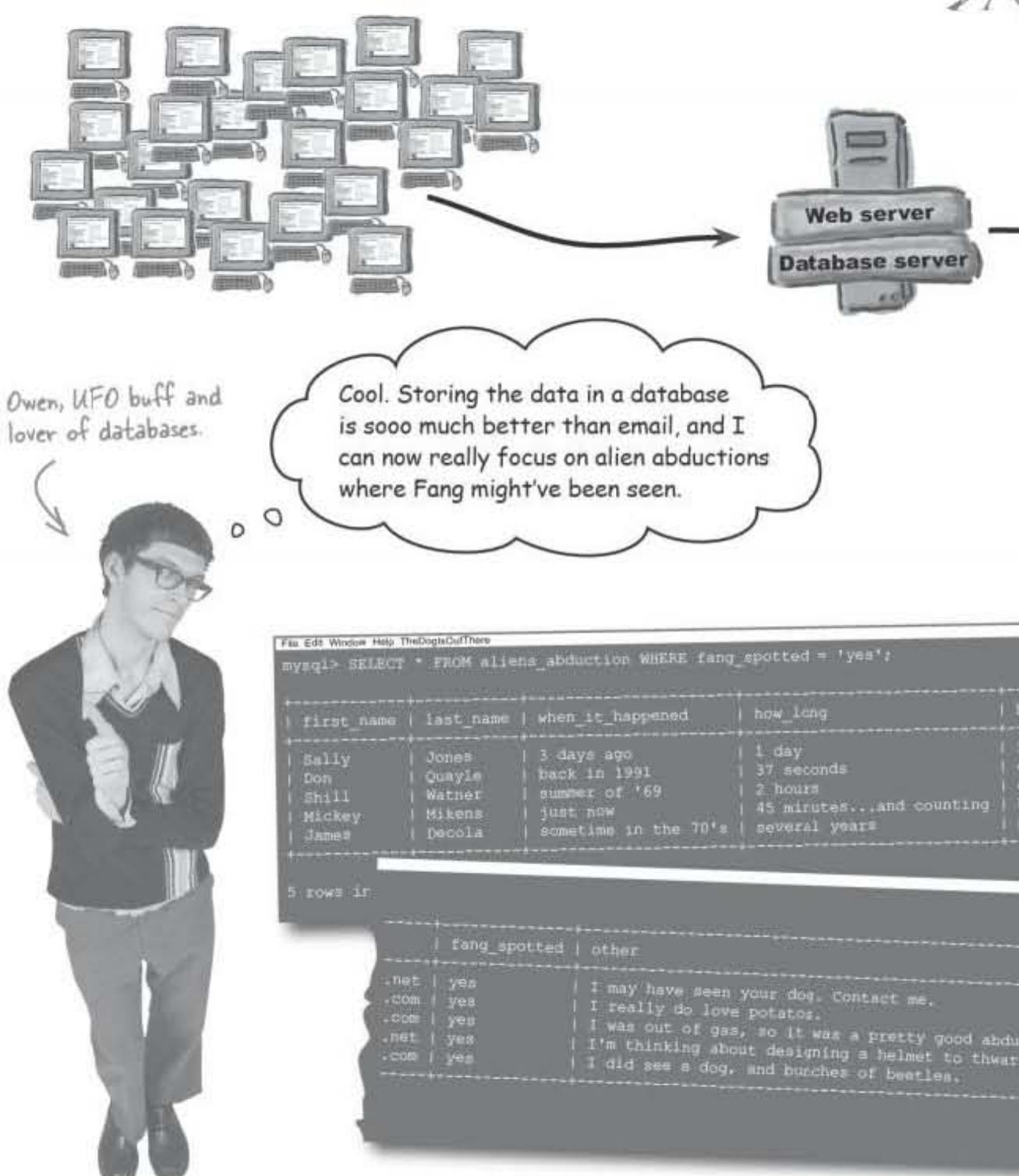
	fang_spotted	other
.net	yes	I may have seen your dog. Contact me.
.net	yes	I may have seen your dog. Contact me.
.com	yes	I really do love potatos.
.com	yes	I was out of gas, so it was a pretty good abduction.
.net	yes	I'm thinking about designing a helmet to thwart...

All of these records
have the fang_spotted
column set to yes.

owen's mysql database is up and running

Owen's on his way to finding Fang

Thanks to PHP and its functions that interface to MySQL, Owen's MySQL database server receives the alien abduction data from an HTML form and stores it in a database table. The data waits there safely in the table until Owen gets a chance to sift through it. And when he's ready, a simple SELECT query is all it takes to isolate abduction reports that potentially involve Fang.





WHO DOES WHAT?

Even though you haven't seen it all put together yet, match each HTML, PHP, and MySQL component to what you think it does.

alien database

This is the SQL code the PHP script passes to the MySQL server.

aliens_abduction table

This runs PHP scripts and returns HTML pages, often communicating with a database along the way.

report.html

The name of the database that contains the aliens_abduction table.

report.php

The HTML form uses this request method to pass data in the form to a PHP script.

POST

This is where the data from the report.html form will eventually end up being stored.

web server

This is where Owen collects data from the user.

MySQL database server

This PHP function closes a connection to the MySQL database.

Submit button

This is the name of Owen's PHP script that processes the data users enter into his report.html form.

query

This PHP function sends a query to the MySQL database.

`mysqli_connect()`

This HTML element is used by visitors to the site when they finish filling out the form.

`mysqli_close()`

This is another name for the software that runs MySQL and all the databases and tables it contains.

`mysqli_query()`

This optional PHP function tells the database which database to use.

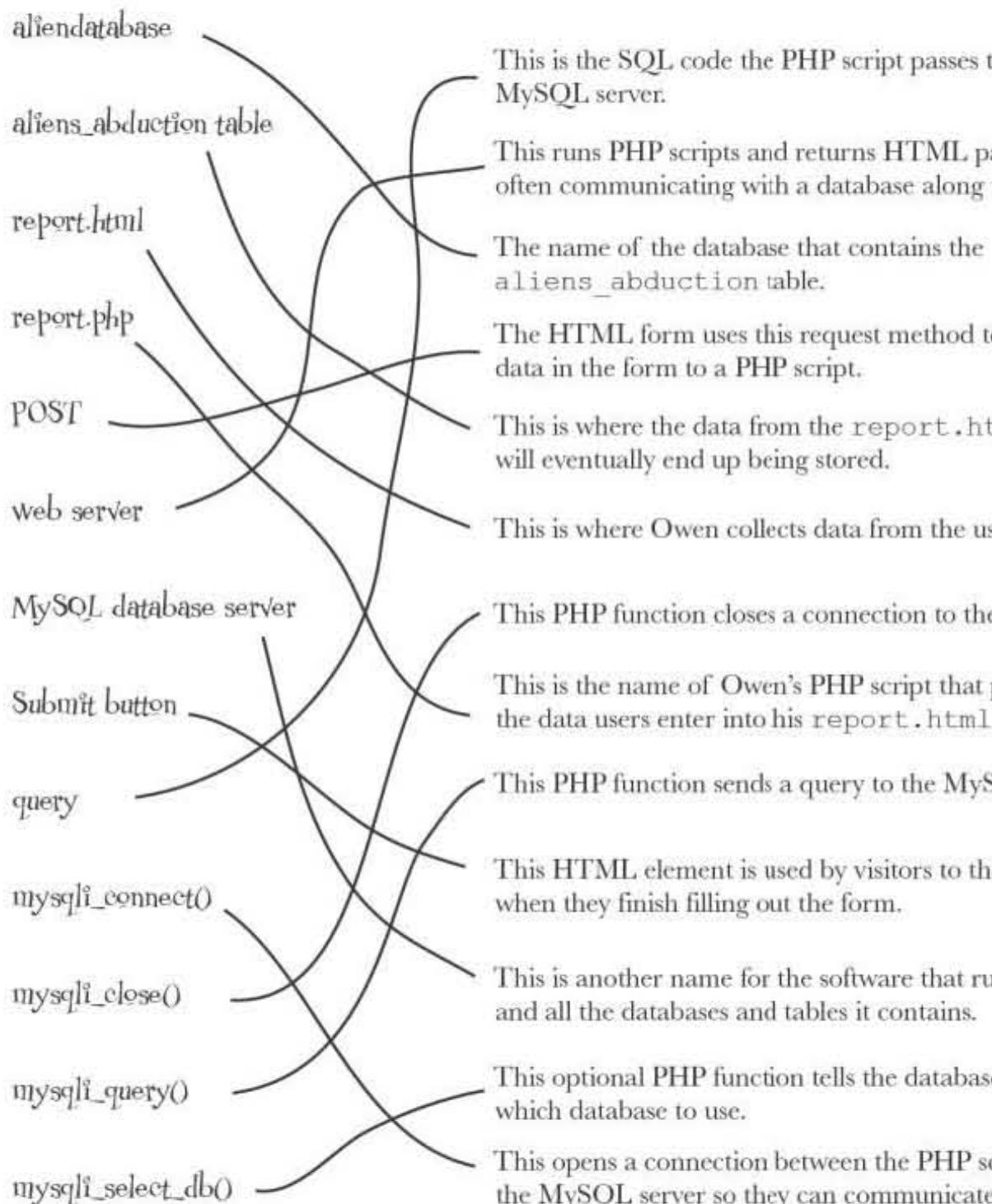
`mysqli_select_db()`

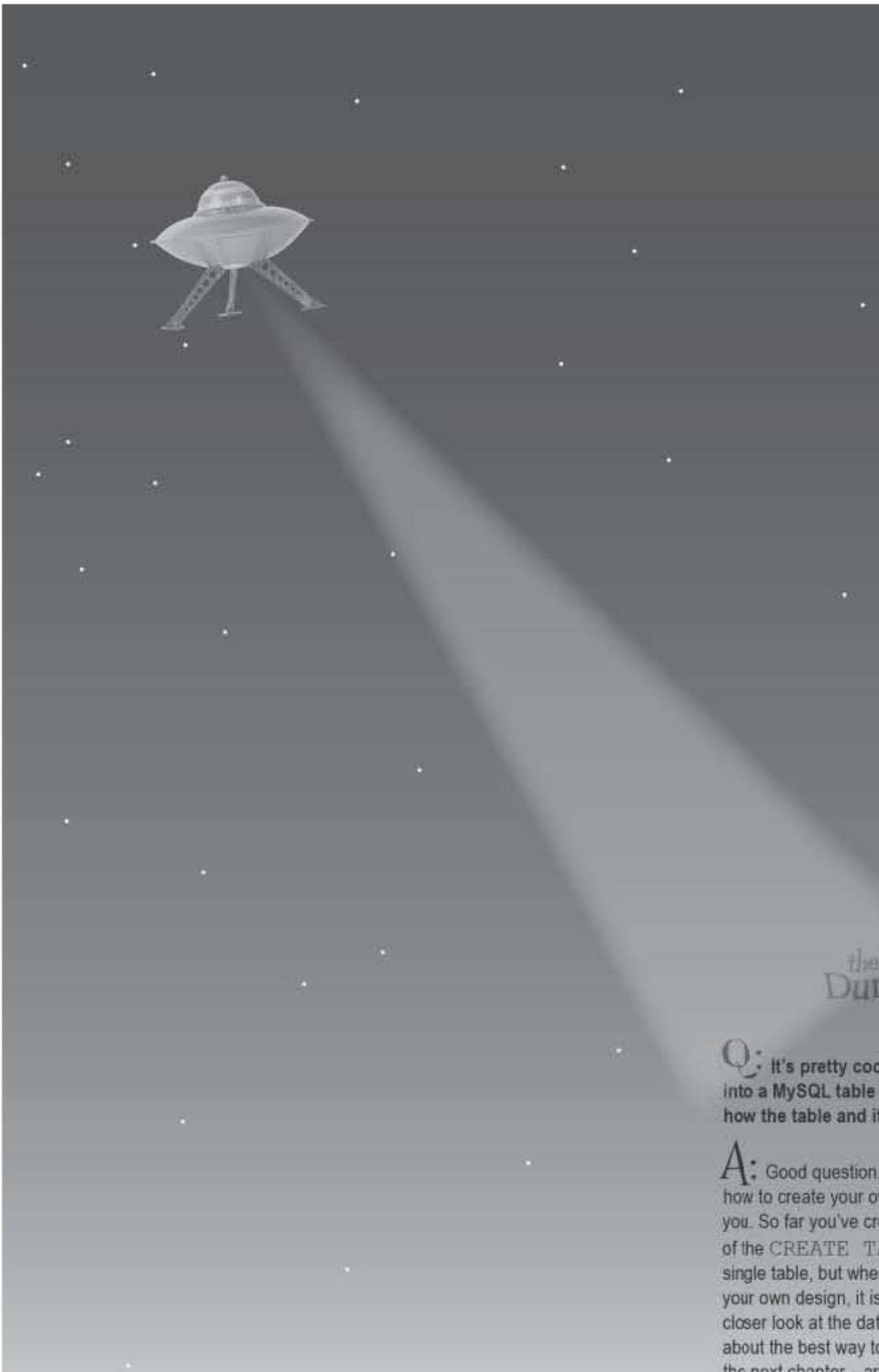
This opens a connection between the PHP script and the MySQL server so they can communicate.

who does what solution

WHO DOES WHAT? — SOLUTION

Even though you haven't seen it all put together yet, match each HTML, PHP, and MySQL component to what you think it does.





the
Dut

Q: It's pretty cool to store data into a MySQL table. But how do you know if the table and its

A: Good question. When you're creating your own database, you need to know how to create your own tables. So far you've created a single table, but when you're creating your own design, it's important to take a closer look at the data you have and think about the best way to store it. In the next chapter, we'll cover how to create multiple tables and how to relate them to each other.

3 create and populate a database

Creating your own da



You don't always have the data you need.

Sometimes you have to *create the data* before you can use it. And sometimes you have to *create tables* to hold that data. And sometimes you have to *create the database*. The database holds the data that you need to create before you can use it. Confused? You won't be after this chapter. Get ready to learn how to create databases and tables of your very own. And if that's not enough, along the way, you'll build your very first PHP & MySQL application.

mailing-list app needed!

The Elvis store is open for business

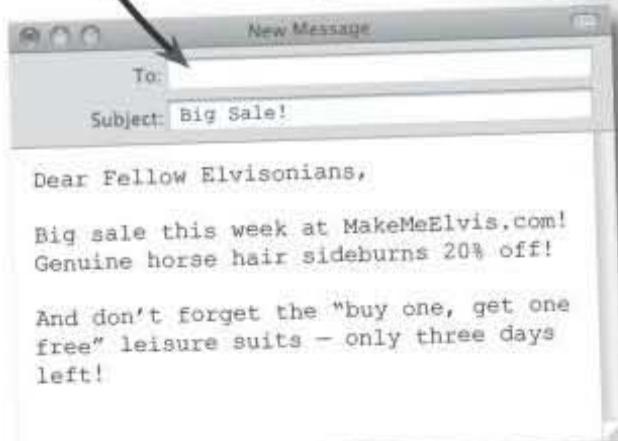
Elmer Priestley has opened his Elvis store, MakeMeElvis.com. Demand has been huge. He's sold a number of studded polyester jump suits, many fake sideburns, and hundreds of pairs of sunglasses.

Each time someone buys something, Elmer collects a new email address. He uses these to send out newsletters about sales at his store. Right now Elmer has to manually go through each email address in his list and copy and paste to send out his email advertising sales. It works, but it takes a lot of time and effort.

Elmer, the undisputed King of online Elvis goods.

This is taking too long. I'd rather be spending my time imitating Elvis, not sending out emails manually.

Elmer writes this email and copies and pastes each email address in the "To" field.



El
add
Poin

Elmer's customer mailing list

Anderson	Jillian	jill_anderson@simudude.com
Joffe	Kevin	joffe@simudude.com
Newsome	Amanda	aman210@objectville.net
Garcia	Ed	ed99@b0tt0m.com
Roundtree	Jo-Ann	jojoround@cbnggs.board.com
Briggs	Chris	cbriggs@board.com
Harte	Lloyd	hovercraft@breathless.com
Toth	Anne	AnneToth@leap.com
Wiley	Andrew	andrewwiley@objectville.net
Palumbo	Tom	palomine@angrypirate.com
Ryan	Alanna	clay@starbuzzcoffee.com
McKinney	Clay	annmeeker@chickenbp@honey-dot.com
Meeker	Ann	debm0nster@breakedesco.com
Powers	Brian	janistedesco@starbuzzcoffee.com
Marison	Anne	szwedjoe@objectville.net
Mandel	Debra	sheridi@mightygun.snowman@tikibeans.com
Tedesco	Janis	glenno098@objectville.net
Talwar	Vikram	anneh@b0tt0msup.com
Szwej	Joe	nobigdeal@starbuzzcoffee.com
Shердан	Diana	dreamgirl@breakneckpizza.com
Snow	Edward	dmelfi@b0tt0msup.com
Otto	Glenn	leeoliver@weatherorama.com
Hardy	Anne	ricciman@tikibeanolounge.com
Deal	Mary	grace23@objectville.net
Jagel	Ann	zelda@weatherorama.com
Melfi	James	clifnight@breakneckpizza.com
Oliver	Lee	joyce@chocoholic-inc.com
Parker	Anne	anneblunt@breakneckpizza.com
Ricci	Peter	lindy@tikibeanolounge.com
Reno	Grace	fgares@objectville.net
Moss	Zelda	anneegg@objectville.net
Day	Clifford	
Bolger	Joyce	
Blunt	Anne	
Bolling	Lindy	
Gares	Fred	
Jacobs	Anne	

Elmer spends far too much time copying and pasting emails into the "To" field of his client email application. He wants to simplify the task of adding new email addresses and sending out mass emails.

Elmer needs an application

An **application** is a software program designed to fulfill a particular purpose for its users. Elmer needs an application that will keep track of his email address list and allow him to send out email to the people on the list by clicking a single form button. Here's how he wants it to work:

- Go to a web page and enter an email message.
- Click a Submit button on the page, and the message gets sent to the entire MakeMeElvis.com email list.
- Let the email list build itself by allowing new customers to sign up through a web form.

With this laundry list of application needs, it's possible for Elmer to visualize his application in all its glory...



The MakeMeElvis.com web application consists of two main components: a form to send email messages to people on Elmer's email list and a form to allow new customers to sign up to Elmer's email list. With these two forms in mind, sketch a design of Elmer's application.

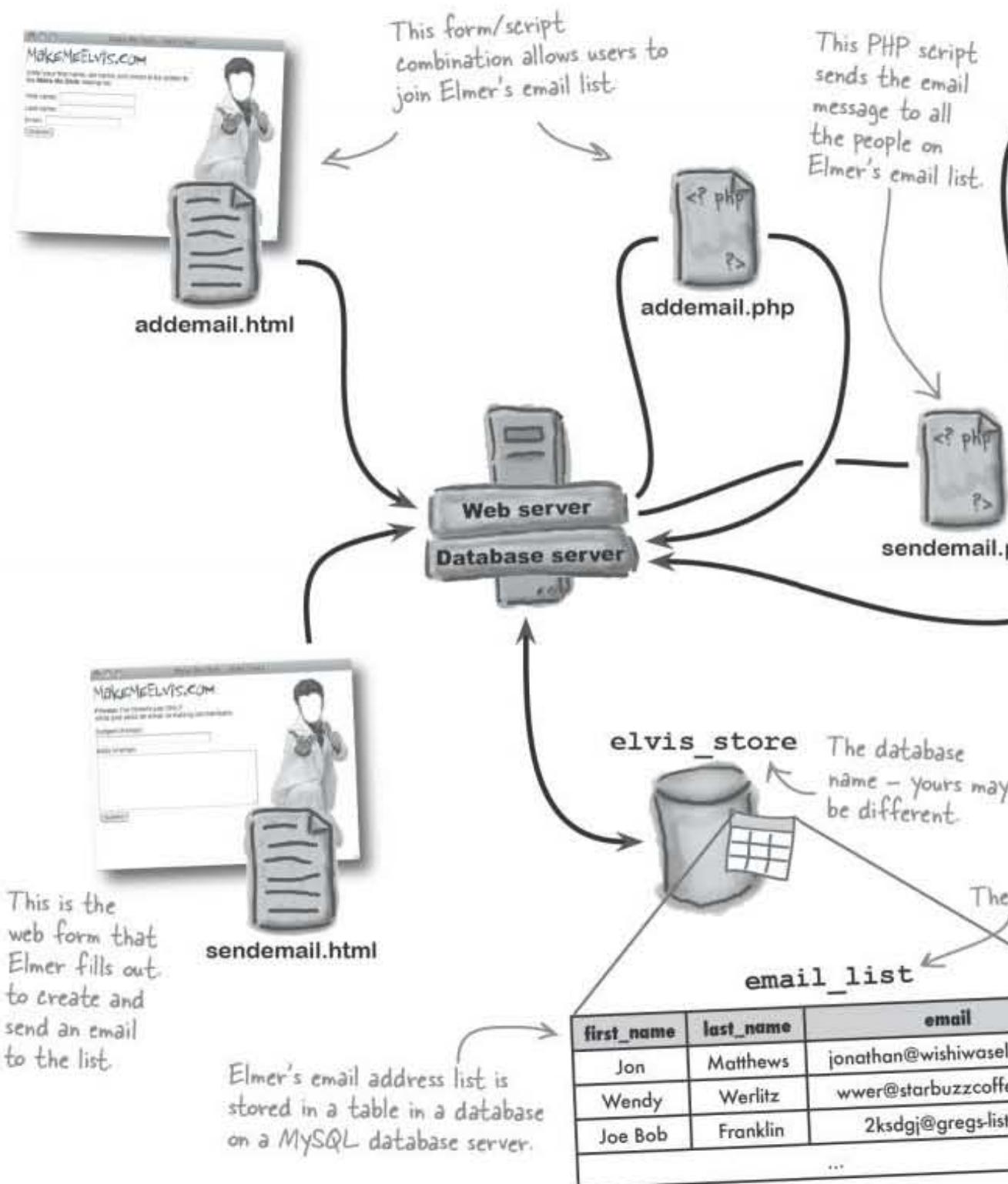
A web application
a dynamic application
that is designed to
fulfill a particular
purpose

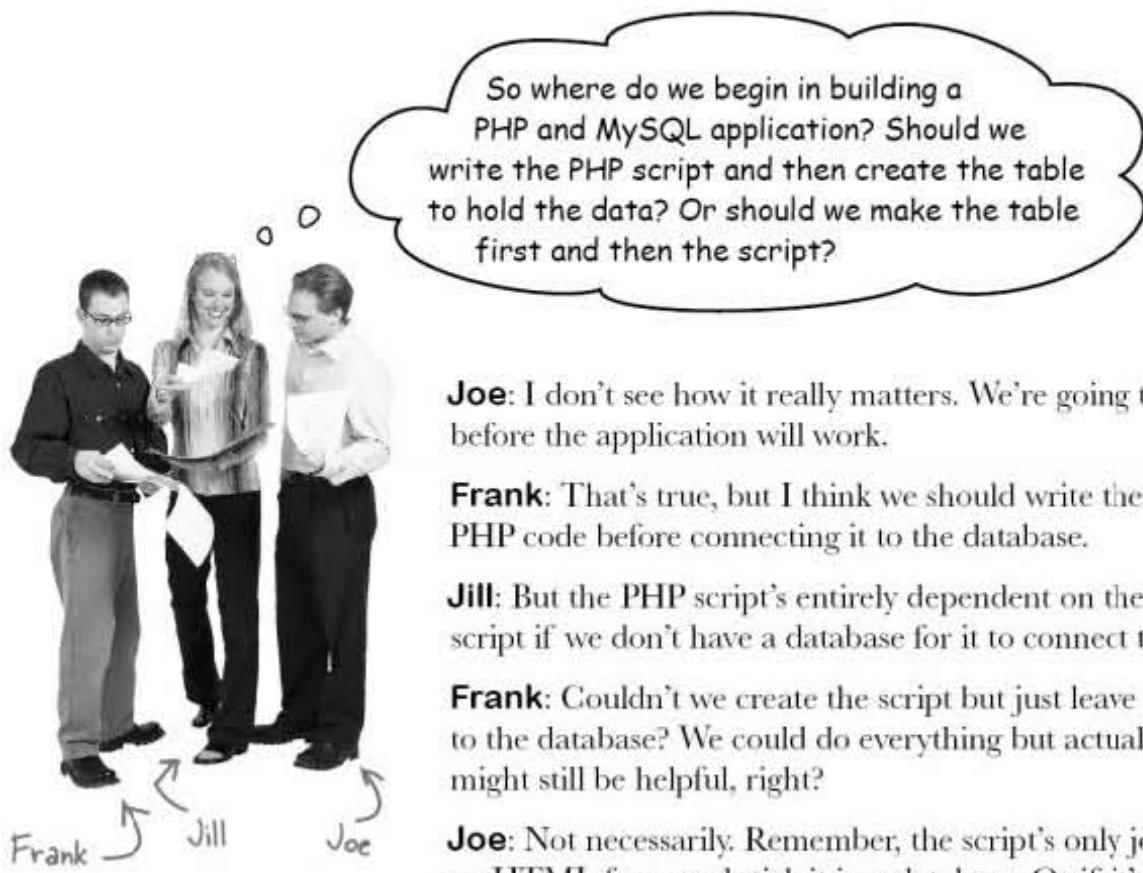
This email stuff
Owen's Alien Abduction
the difference between
email list will be
messages will go
Elmer's app is a

our mailing-list app design

Visualize Elmer's application design

It always helps to visualize the design of an application before diving into the development details. This means figuring out what web pages and scripts will be involved, how they connect together, and perhaps most importantly, how you'll store the data in a MySQL database.





Joe: I don't see how it really matters. We're going to need the table before the application will work.

Frank: That's true, but I think we should write the script first so the PHP code before connecting it to the database.

Jill: But the PHP script's entirely dependent on the database. It needs the database to run. We can't have a PHP script if we don't have a database for it to connect to.

Frank: Couldn't we create the script but just leave out the specific connection to the database? We could do everything but actually interact with the database. That might still be helpful, right?

Joe: Not necessarily. Remember, the script's only job is to take data from an HTML form and stick it in a database. Or if it's sending an email or generating a list, the script reads from the database and generates an email message. Either way, the database is critical to the script.

Jill: True, but we didn't even think about the HTML form. Where does that fit into all of this? We can't have a form until we create the database before we can even think about writing the script.

Frank: That's it! First we create the HTML form, then we figure out what data goes in the database, then we create the database, then we tie it all together with the script.

Joe: I'm not sure if that really makes sense. How can we create an HTML form when we aren't sure what data we need to get from the user?

Jill: Joe's right. The HTML form still leads back to us needing to have the data for the application. The data drives everything, so we should probably build the database and table first, then the HTML form, then the script that reacts to the form submission.

Frank: I'm sold. Let's do it!

Joe: I still think we probably need to come up with specific steps of how this application is going to work.

Write down the specific steps you think are needed to build this application from design to implementation with MakeMeDoIt!

.....
.....
.....

planning the application

PLAN AHEAD

We really need a plan of attack for putting together Elmer's application. By breaking it down into steps, we can focus on one thing at a time and not get overwhelmed.

1 Create a database and table for the email list.

This table will hold the first names, last names, and email addresses of everyone on Elmer's mailing list.

elvis_st



2 Create an Add Email web form and PHP script for adding a new customer to the list.

Here's where we'll build a form and script that will allow a customer to easily enter their first name, last name, and email address, and then add them to the email list.



adde

3 Create a Send Email web form and PHP script for sending an email to the list.

Finally, we'll build a web form that will allow Elmer to compose an email message and, more importantly, a script that will take that message and send it to everyone stored in his email list table.



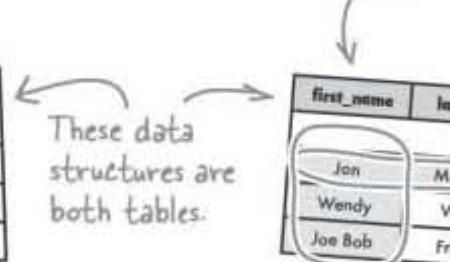
sendemail.html
sender

It all starts with a table

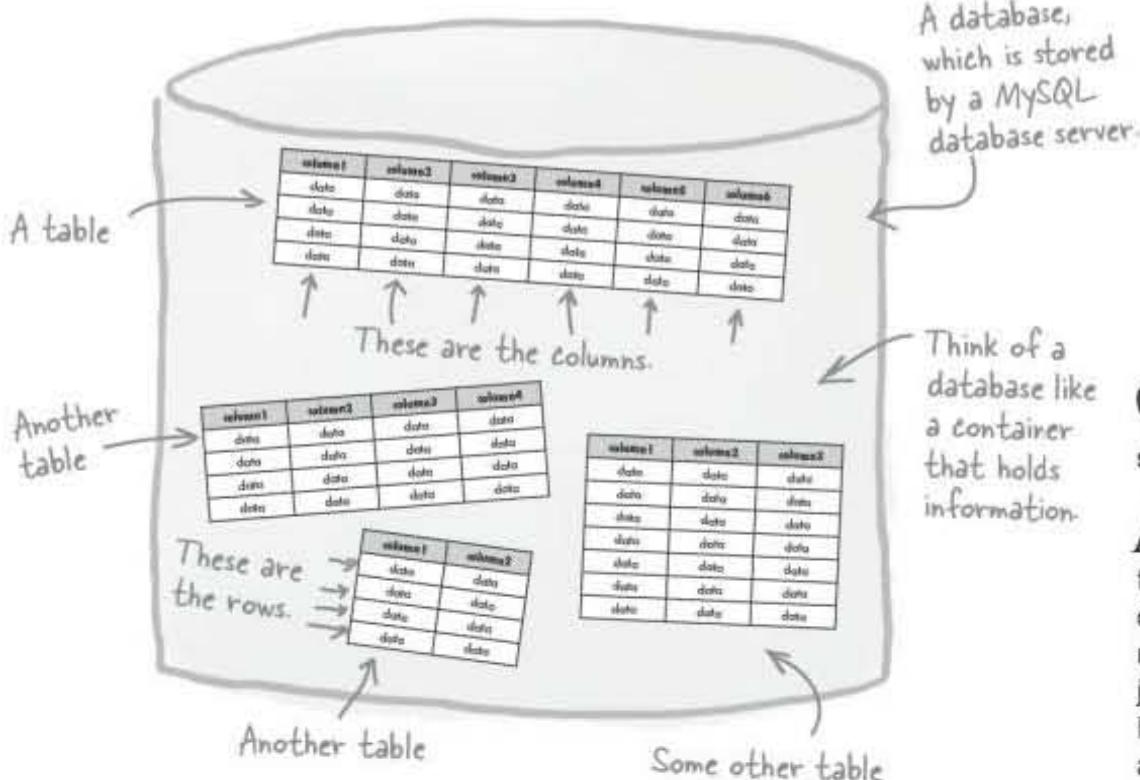
Actually, it all starts with a database, which is basically a container for storing data. Remember, in the last chapter, how databases are divided internally into more containers called **tables**.

Like days and weeks in a calendar, a table's made up of columns and rows of data. **Columns** consist of one specific type of data, such as "first name," "last name," and "email." **Rows** are collections of columns where a single row consists of one of each column. An example of a row is "Wency, Werlitz, wwer@starbuzzcoffee.com."

calendar						
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
...						



Generally, all the tables in a database have some relationship to each other, even if that affiliation is sometimes loose. It's common for a web application to consist of multiple tables that are connected to one another through their data. But all the tables are still made up of columns and rows.



Tables
a grid
of colu

there
Dumb

Q: Where's
stored? Can I

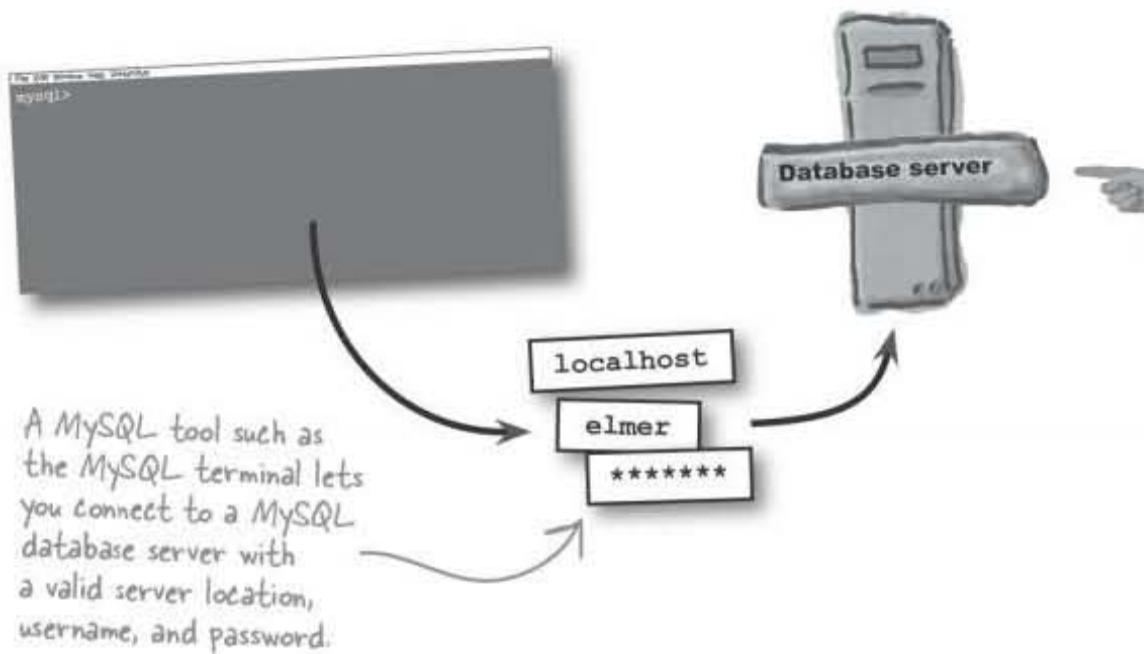
A: Database
files on a hard
certainly look at
much. Database
just be opened
have SQL—to a
and interact with

CREATE your database

Make contact with the MySQL server

Elmer's application design needs a database and a table. Most of the day-to-day work of dealing with a database involves interacting with tables, but you can't just jump in and start creating tables without creating a database to hold them first.

The `CREATE DATABASE` command is the SQL command used to create a database. Once that's done, you can move on to creating a table with the `CREATE TABLE` command. **But before you can use either of those commands, you have to connect to your MySQL database server.** You did this back in the last chapter, and it required a few pieces of important information.



As well as letting a PHP script make a connection to a database and perform database actions, the database server location, username, and password are the key to using the MySQL terminal or phpMyAdmin. And these tools are pretty helpful for getting a database application off the ground with the initial database and table creation.

Since creating a database and table for Elmer's application only has to happen once, it makes sense to use an SQL query to create them manually. So fire up your MySQL tool of choice, and get ready to knock out the first step of Elmer's application, creating a database and table for the email list.

- 1 Create a the email
- 2 Create a and PHP customer
- 3 Create a and PHP email to

Create a database for Elmer's emails

To create a new table and database for Elmer's email list, first we need to create the `elvis_store` database, which will hold the `email_list` table. We'll use SQL commands to create both. The SQL command used to create a database is `CREATE DATABASE`, which you used briefly in the previous chapter. Let's look a bit closer at how it works.

`CREATE DATABASE database_name`

The name of the new database to be created

You need to specify the name of the new database after the command `CREATE DATABASE`. Here's the SQL statement to create Elmer's database:

`CREATE DATABASE elvis_store`

When you execute this statement on a MySQL database server, the database will be created.

File Edit Window Help Don'tBeCruel

```
mysql> CREATE DATABASE elvis_store;
Query OK, 1 row affected (0.01 sec)
```

When you run SQL terminal, you always end...but not when through the PHP m

Creating the `elvis_store` database with the `CREATE DATABASE` command results in a shiny new database but no table to actually store data in yet...



Watch it!

SQL statements only end with semicolons when you use the terminal.

In your PHP code, your SQL statements don't need to end with a semicolon. The MySQL terminal is different, however, and requires a semicolon at the end of every SQL statement. This is because the terminal is capable of running multiple SQL statements, whereas in PHP, you only submit one statement at a time.

now **CREATE** your table

Create a table inside the database

You have to know what kind of data you want to store in a table before you can create the table. Elmer wants to use the first and last names of people on his email list to make the email messages he sends out a bit more personal. Add that information to the email address, and Elmer's `email_list` table needs to store three pieces of data for each entry.

Each piece of data in a table goes in a column, which needs a name that describes the data. Let's use `first_name`, `last_name`, and `email` as our **column names**. Each **row** in the table consists of a single piece of data for each of these columns, and constitutes a single entry in Elmer's email list.

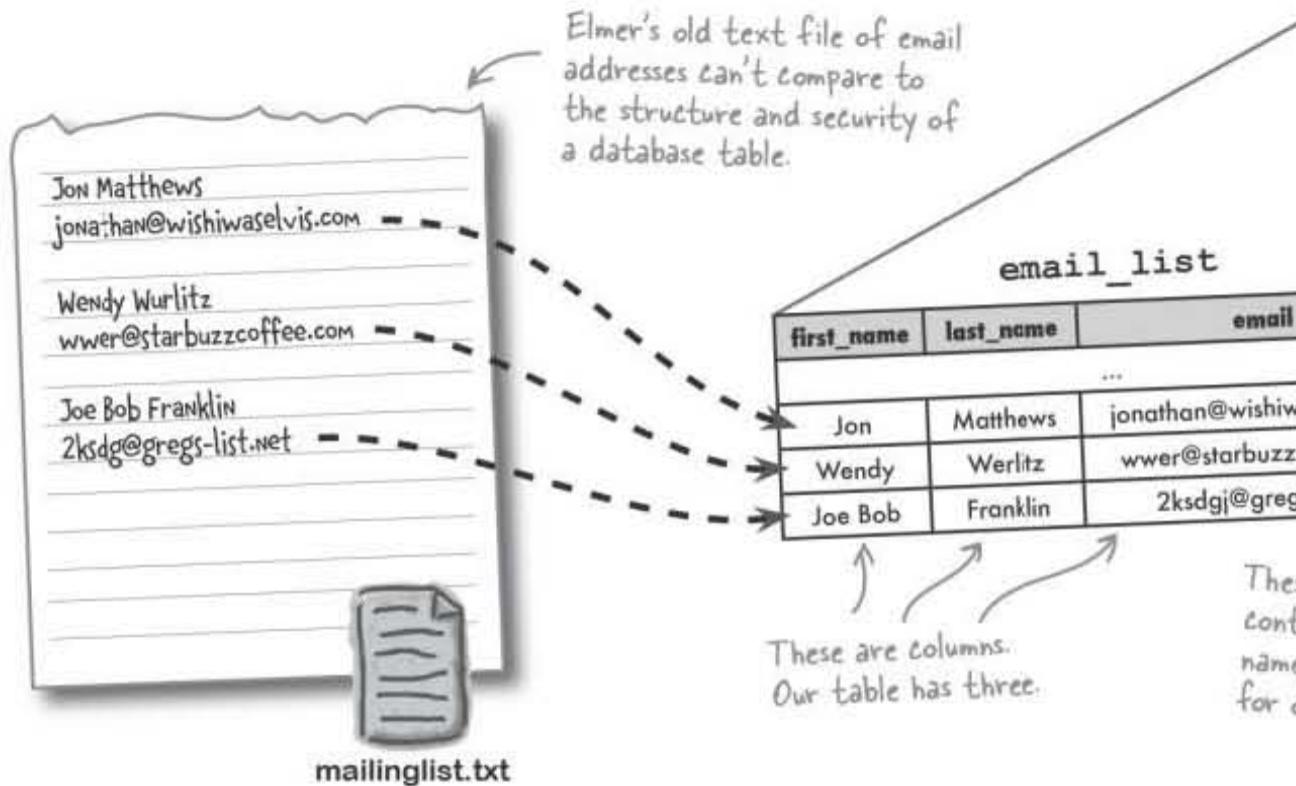
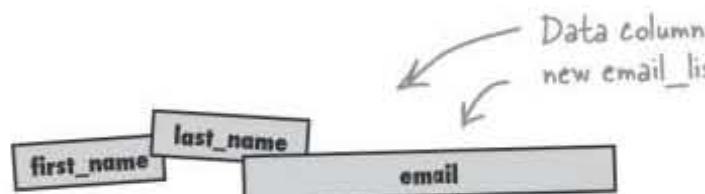


Table rows are horizontal, and table columns are vertical.

So now we know that the first name, last name, and email address for each customer must be created as columns in the `email_list` table. MySQL tables are highly structured and expect you to tell it exactly what kind of data you intend to store in the column.



We need to define our data

When you create a table, you have to tell the MySQL server what type of data each column will hold. Data types are required for *all* MySQL columns, and each column in a table holds a particular type of data. This means some columns may hold text, some may hold numeric values, some may hold time or dates, and so on. MySQL has a variety of data types, and you need to know which one suits your particular data. Let's suppose Elmer has a table named `products` that keeps track of the items for sale at his store:

The diagram shows a table named `products` with four columns: `id`, `product`, `inventory`, and `price`. The `product` column contains text descriptions of products. The `inventory` column contains integer values for stock levels. The `id` column contains unique ID values. The `price` column contains decimal values.

<code>id</code>	<code>product</code>	<code>inventory</code>	<code>price</code>
1	Blue Suede Shoes	24	59.00
2	Polyester Pants with Sequins	16	23.50
3	Stick-On Sideburns	93	1.99
4	Elvis wig	7	48.00
...			

This column contains text descriptions of each product in Elmer's store.

The inventory column contains an integer value for how many of each item are in stock.

The id column contains unique ID values for each product in Elmer's store.

The price column contains decimal values.

product

- Blue Suede Sh
- Polyester Pants with
- Stick-On Sideb
- Elvis wig

Text

price

- 59.00
- 23.50
- 1.99
- 48.00

Decimal Nu

Notice that `product` is the only text column in the `products` table. There are also decimal numbers for `price` and integer numbers for `inventory` and `id`. MySQL has its own names for each one of these types of data, as well as a few more such as types for dates and times.

It's important to use the appropriate data types when you create table columns so that your tables are accurate and efficient. For example, **text data takes more room to store than integer data**, so if a column only needs to hold integers, it's a smart practice to use an integer data type for it. Also, if it knows what kind of data a column holds, the web server won't allow you to accidentally insert the wrong type of data. So if you have a column that holds a date, you will receive an error if you try to insert anything except a date in that column.

To create a table, you need to know what type of data is stored in each table column.

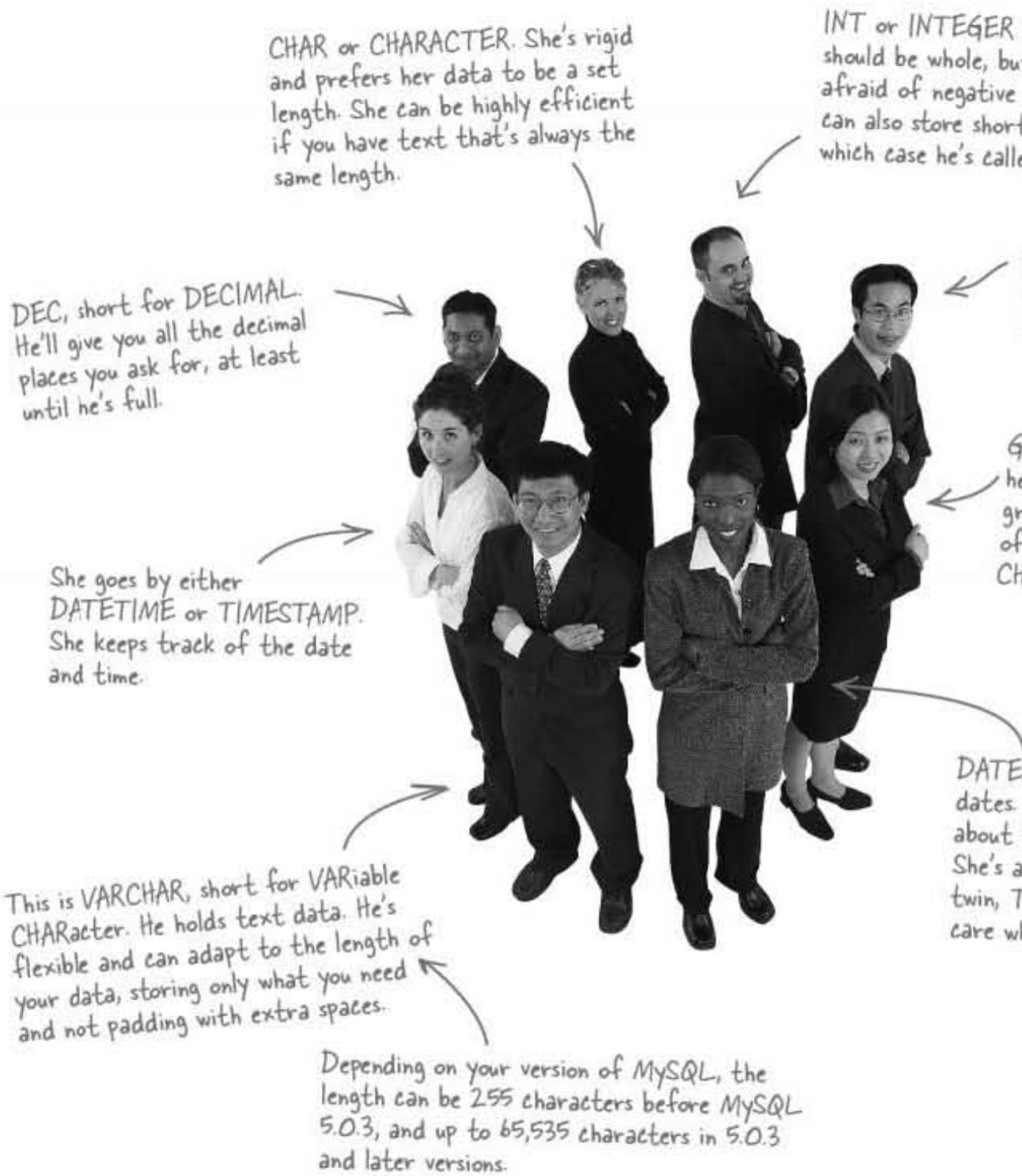
Why do you think data types is better than just text to store?



frequently used mysql data types

Take a meeting with some MySQL data types

These are a few of the most useful MySQL data types. Remember, you can use any of them to describe the data stored within a particular column of table data. It's their job to store your data for you without mucking it up.



there are no
Dumb Questions

Q: Why would I ever use a CHAR when a VARCHAR does the same thing with more flexibility?

A: The answer is accuracy and efficiency. From a design perspective, you should always design your tables to model your data as rigidly as possible. If you know without a shadow of a doubt that a state column will always hold exactly a two-character abbreviation, then it makes sense to only allot two characters of storage for it with CHAR(2). However, if a password column can contain up to 10 characters, then VARCHAR(10) makes more sense. That's the design side of things. So CHAR is a little more efficient than VARCHAR because it doesn't have to keep track of a variable length. Therefore, it's more desirable when you know for certain a text column has an exact length.

Q: Why do I need these numeric types?

A: It all comes down to database storage. Pick the best matching data type for each column based on the size of the table and make operations faster. Storing a number as an actual number (INT, DECIMAL) instead of characters is usually more efficient.

Q: Is this it? Are these all the types?

A: No, but these are the most common and running with these for now, rather than looking at data types you may never need.

WHAT'S MY PURPOSE?

Match each MySQL data type to each description of some data you might store in a table.

Data Type

INT

CHAR(1)

DATE

TIME

VARCHAR(2)

DEC(4,2)

VARCHAR(60)

CHAR(2)

DATETIME

DEC(10,2)

Description

Your full name

A two letter state abbreviation

Cost of an Elvis wig: 48.99

How much money Elvis's best-selling book cost

Date of alien abduction: 2/19/2009

Number of Elvis sideburns in stock

Did you see Owen's dog? Y or N

Your email address

When you eat dinner

How many aliens you saw when you were born

When Elvis was born

what's my purpose? solution

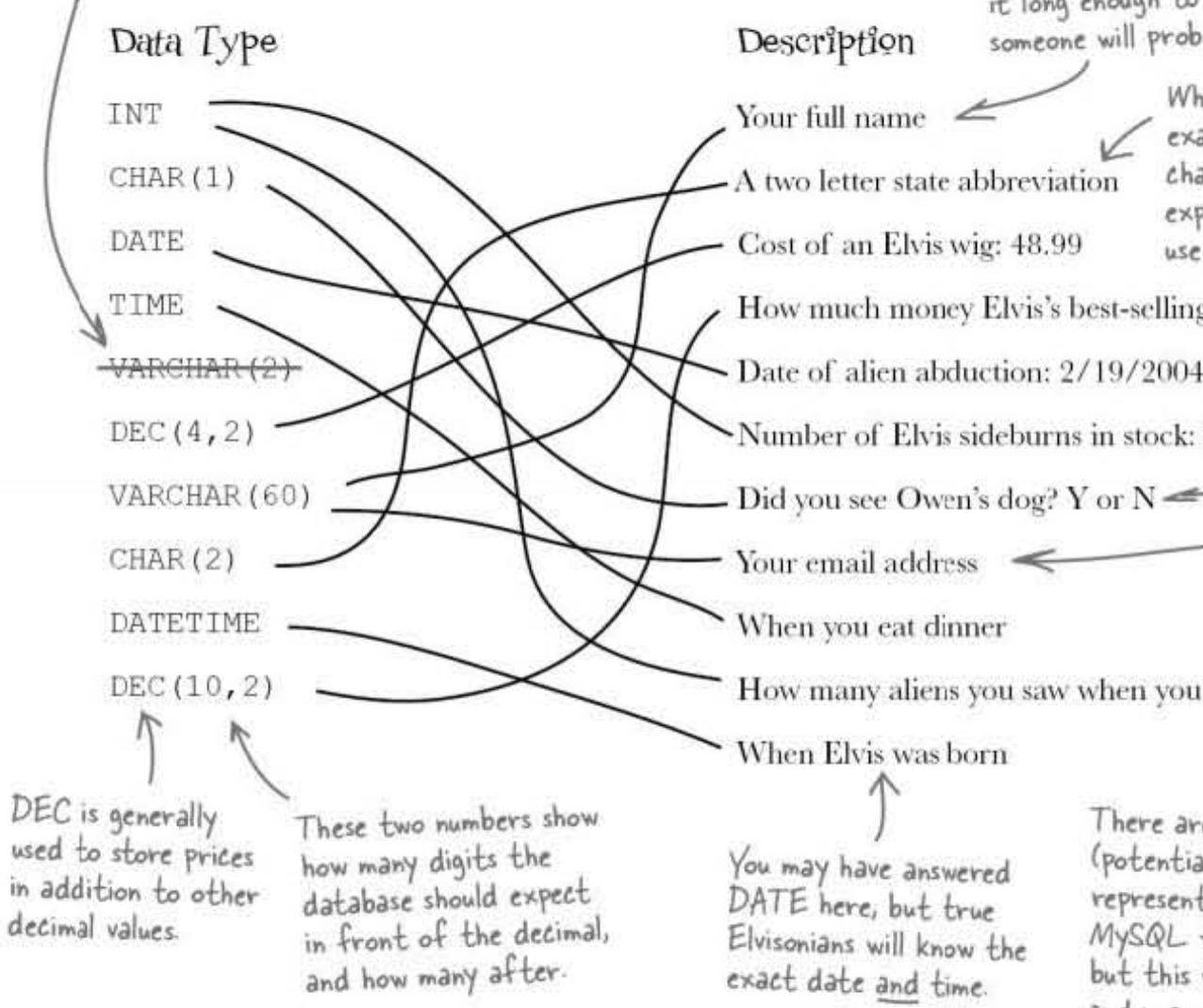
* WHAT'S MY PURPOSE? *

Match each MySQL data type to each description of some data you might store in a table.

Not needed. Although it would work for the state abbreviation, CHAR(2) is a better choice because it's usually a little more efficient.

When the length vary, VARCHAR is it long enough to someone will prob

Wh
exa
cha
exp
use



Create your table with a query

We've got all the pieces that we need to create our table, even a good name (`email_list`). We also have names for the columns of data: `first_name`, `last_name`, and `email`. All that's missing is the data type for each column and an SQL statement to tie it all together and create the table. The SQL command to create your table is `CREATE TABLE`.

It begins with `CREATE TABLE` then your table name. Two parentheses hold a comma separated list of all the column names, each one followed by a data type. Here's what the command looks like:

```
CREATE TABLE table_name
(
    column_name1 column_type1,
    column_name2 column_type2,
    ...
)
```

The table name
The column name
The data type of the column
More columns, if needed
You don't have to name your tables and columns with an underscore separating words but it's a good idea to be consistent with naming.

Sharpen your pencil



Write an SQL query to create Elmer's `email_list` required columns of data: `first_name`, `last_name`

.....
.....
.....
.....
.....

Yep, we're

- 1 Create a the email
- 2 Create an and PHP customer
- 3 Create a and PHP email to

The
TA
com
used
new
data

test-drive your CREATE queries



Sharpen your pencil Solution

Write an SQL query to create Elmer's email list table. Use the required columns of data: first_name, last_name, and email.

Here's the SQL command to create the table, notice the caps.

The opening parenthesis opens the list of columns to create.

The closing parenthesis closes the list of columns.

CREATE TABLE email_list

first_name VARCHAR(20),

last_name VARCHAR(20),

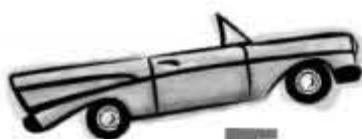
email VARCHAR(60)

The name of the column that stores the email address.

Your table names can have underscores and have a maximum of 63 characters.

The column names can have underscores and have a maximum of 63 characters.

This tells MySQL that the email column can store up to 60 characters.



— Test Drive —

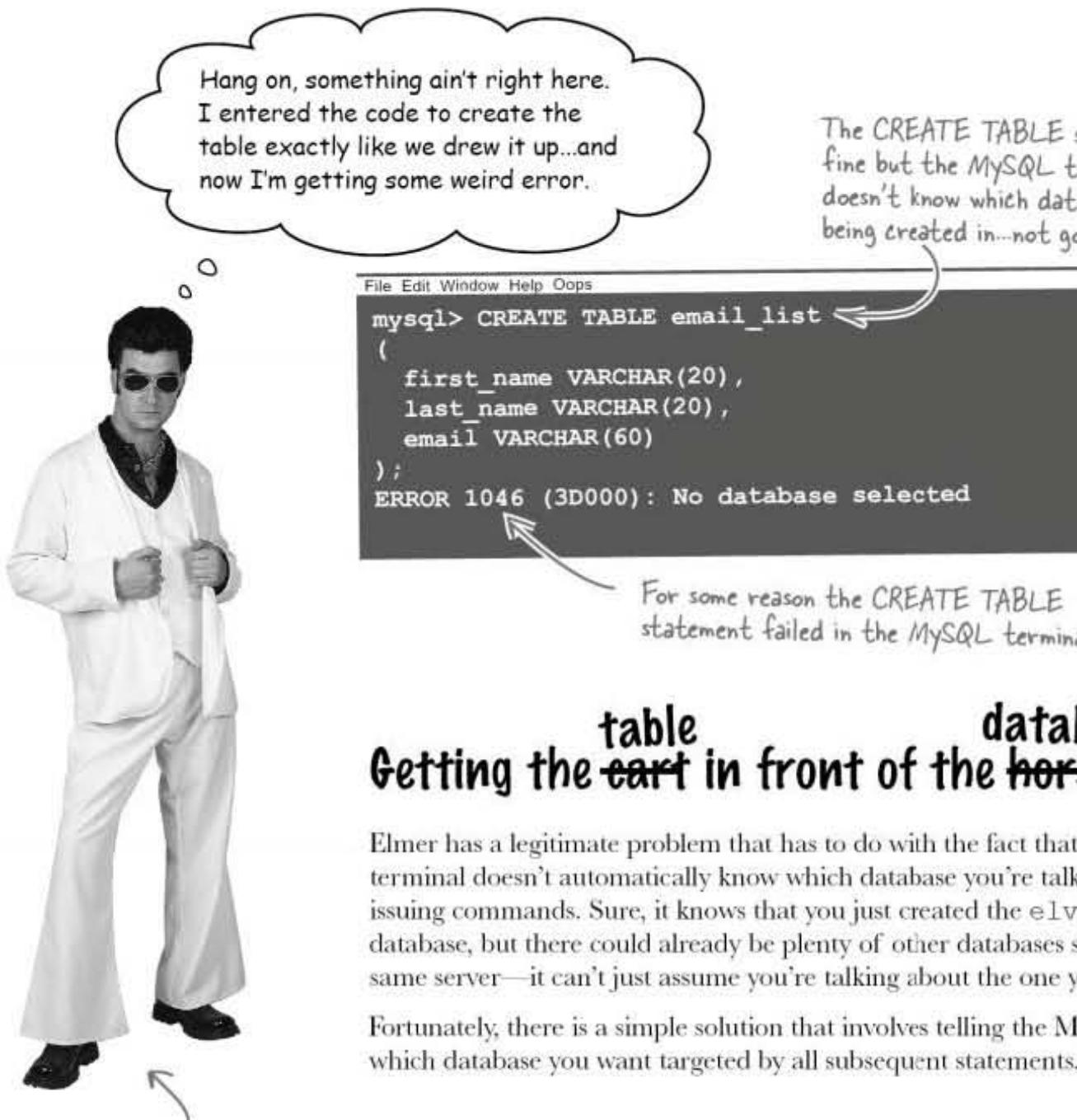
Create Elmer's database and table.

Execute the CREATE DATABASE and CREATE TABLE queries using a MySQL tool to create the elvis_store database and the email_list table within it.

```
CREATE DATABASE elvis_store
```

```
CREATE TABLE email_list(first_name VARCHAR(20), last_name VARCI
```

Did both queries execute without a hitch? If so, what you think might have gone wrong?



The CREATE TABLE statement is fine but the MySQL terminal doesn't know which database is being created in—not go

For some reason the CREATE TABLE statement failed in the MySQL terminal

table data Getting the ~~cart~~ in front of the horse

Elmer has a legitimate problem that has to do with the fact that the terminal doesn't automatically know which database you're talking about when issuing commands. Sure, it knows that you just created the elvish database, but there could already be plenty of other databases stored on the same server—it can't just assume you're talking about the one you want.

Fortunately, there is a simple solution that involves telling the MySQL terminal which database you want targeted by all subsequent statements.

Elmer's all shook up because his CREATE TABLE statement is flawless, yet the MySQL terminal's reporting an error.

there are no Dumb Questions

Q: What's up with the weird -> prompt I get sometimes in the MySQL terminal?

A: The -> prompt indicates that you're entering a single statement across multiple lines—MySQL is basically telling you that it knows you're still entering the same statement, even though you've hit Return to break it out across more than one line. Once you finish the statement and put the semicolon on the end, MySQL

don't forget the USE command!

USE the database before you use it

So that the CREATE TABLE statement will work, Elmer needs to **select the database** in the MySQL terminal so that it knows what database the new table belongs to. The USE command chooses a database as the default database in the terminal, meaning that all subsequent commands apply to that database. Here's how it works:

The USE command tells MySQL what database you intend to use.

```
USE database_name
```

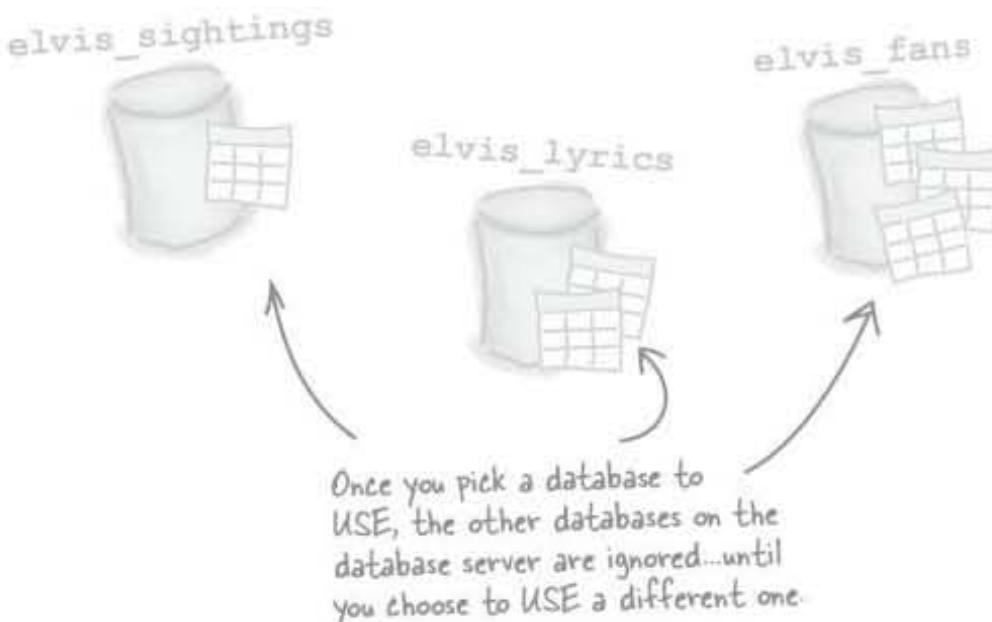
Elmer should specify his database name (`elvis_store`) in a USE statement to select the database and access his new table.

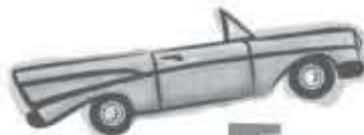
The name of the database you'd like to USE.

```
USE elvis_store
```

The USE command
selects a database
as the default
for subsequent
SQL statements.

The USE command
chooses the database
you want to use.





— Test Drive —

First USE Elmer's database, then create the table.

Execute the USE query to select Elmer's elvis_store database in a MySQL tool, and then execute the CREATE TABLE query to create the email_list table inside the database.

```
USE elvis_store
```

```
CREATE TABLE email_list(first_name VARCHAR(20), last_name VARC
```

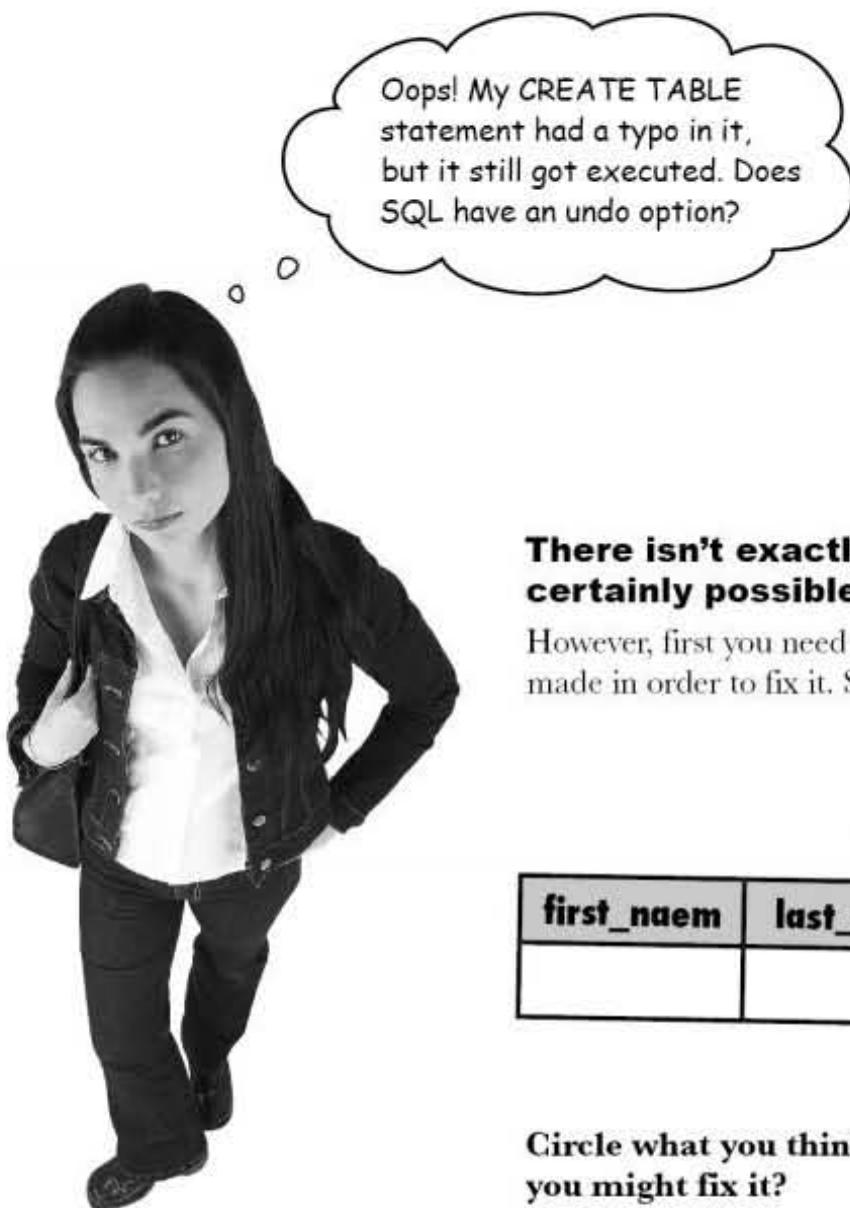
The USE statement isn't necessary if you're using a graphical SQL tool such as phpMyAdmin – it requires you to select the database graphically before issuing SQL commands.

```
File Edit Window Help LisaMarie
mysql> USE elvis_store;
Database changed
mysql> CREATE TABLE email_list
(
    first_name VARCHAR(20),
    last_name VARCHAR(20),
    email VARCHAR(60)
);
Your SQL query has been executed successfully (Query took 0.000 sec)
```

The table creation code is the same as before – it just needed the database selected before it would work.

With the database selected, thanks to the USE command, the table creation now works with no problem.

the DESCRIBE command



There isn't exactly an undo option in SQL, but it is certainly possible to fix mistakes.

However, first you need to find out exactly what kind of mistake was made in order to fix it. Suppose the `email_list` table has the following structure:

email_list

first_name	last_name	email

Circle what you think is wrong with this table. Then you might fix it?

DESCRIBE reveals the structure of tables

Repairing a mistake in a table first involves finding the mistake. Even if you don't suspect a mistake, it's never a bad idea to check your work. The SQL DESCRIBE command analyzes the structure of a table, displaying a list of column names, data types, and other information.

DESCRIBE *table_name*

Plugging in Elmer's table name gives us the following SQL statement:

DESCRIBE email_list ← This is the name of the table we want to see described.

File Edit Window Help Graceland

```
mysql> DESCRIBE email_list;
+-----+-----+-----+-----+
| Field | Type | Null | Key | Default |
+-----+-----+-----+-----+
| first_name | varchar(30) | YES | | NULL |
| last_name | varchar(30) | YES | | NULL |
| email | varchar(60) | YES | | NULL |
+-----+-----+-----+-----+
3 rows in set (0.02 sec)
```

Under "Field" you find the names of each column.

Under "Type" you see the data types we set for each column.

MySQL is not case sensitive when it comes to reserved words like data types, which is why you sometimes see them in lowercase.

there are no
Dumb Questions

Q: What's up with those other columns: Null, Key, Default, and Extra?

A: MySQL lets you set a number of options for each column in your table. These options control things like whether a column can be left empty or if it has a default value. We'll explore these a bit later when they become more critical to the application.

Q: So if I actually had data stored in my table, would it show up here?

A: No. DESCRIBE only shows you the table structure, not the data stored in it. But don't worry, you'll see the data in your table very soon... but first we have to learn how to actually put data into the table.

Q: Can I look at the structure of a table using phpMyAdmin?

A: Yes. Graphically, phpMyAdmin allows you to view the structure of tables by issuing a 'Structure' statement or by clicking on a table. It's entirely up to you how you use to analyze the table.

DROP your table

The `first_name` column
was accidentally misspelled
`first_naem...oops!`

I fixed the typo and tried to run the `CREATE TABLE` query again. It didn't work. Surely I don't have to delete the typo'd table first... do I?

```
File Edit Window Help Typo?
mysql> DESCRIBE email_list;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| first_naem | varchar(30) | YES |   | NULL |
| last_name  | varchar(30) | YES |   | NULL |
| email      | varchar(60)  | YES |   | NULL |
+-----+-----+-----+-----+
3 rows in set (0.02 sec)
```

Actually, you do. You can't recreate a table again with `CREATE TABLE` once it's been created.

Once you've created a table, it exists and **can't be overwritten** by a new `CREATE TABLE` query. If you want to recreate a table from scratch you'll have to delete the existing one first, and then start over again.

In SQL, the `DROP TABLE` command is used to delete a table from a database. It deletes the table and anything you've stored in it. Since no data exists in a new table, we won't lose anything by dropping it and creating a new one with the `first_name` correction.

DROP TABLE email_list

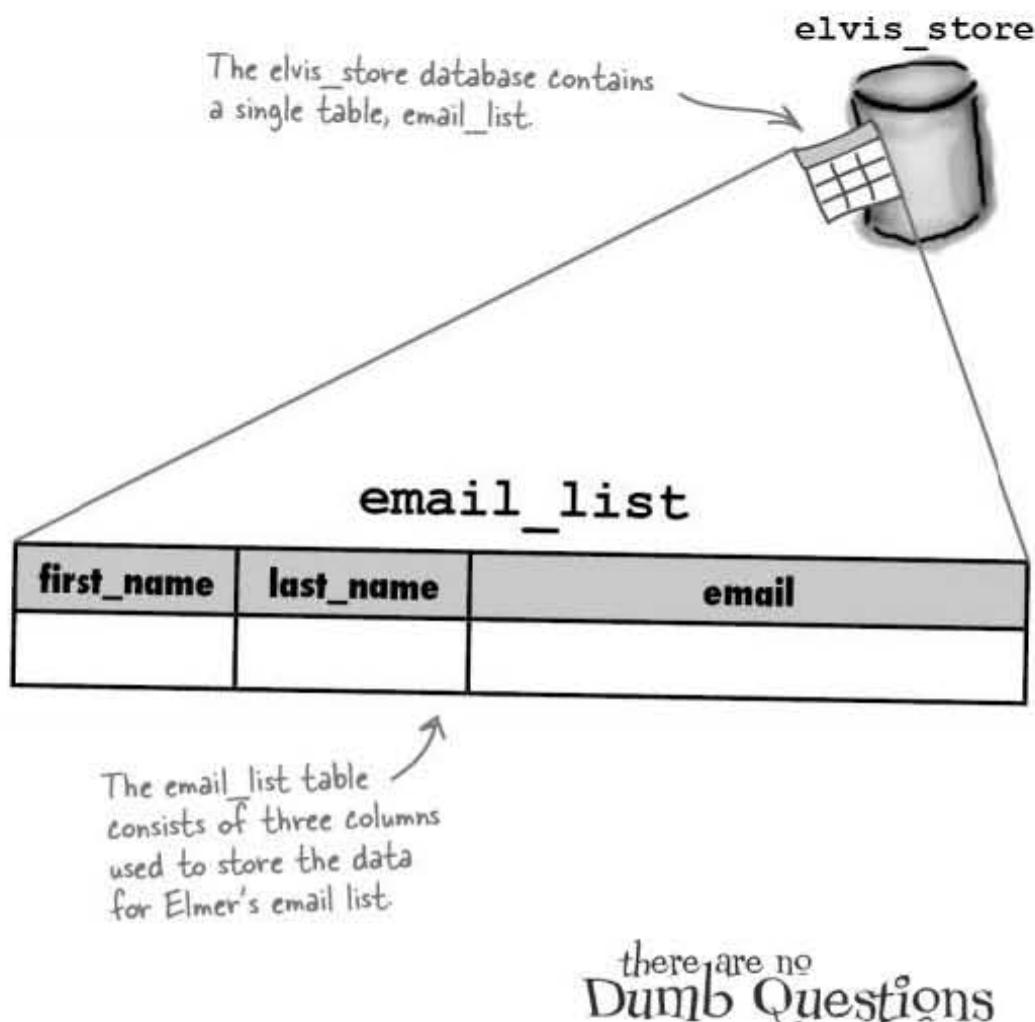
The name of the
you'd like to delete
from the database

The `DROP TABLE` command
deletes a table AND all its
data from the database.

Elmer's ready to store data

The CREATE DATABASE, USE, and CREATE TABLE SQL commands were successfully used to create Elmer's email list database and table. Elmer couldn't be more pleased, unless maybe the table was already filled with eager customers. That's a job for PHP...

Nice. We
and table
ready to
serious r



there are no
Dumb Questions

Q: Hey, I have a copy of Head First SQL (great book, by the way). In that book, every time you show the code for an SQL statement, you put a semicolon after it. Why not here?

A: We're glad you enjoyed *Head First SQL*. The difference is that when you talk to MySQL directly, you need a semicolon so it knows where the end of the statement is. That's because it's possible to issue multiple statements to MySQL directly. In PHP, when you use the `mysqli_query()` function, you only execute a single SQL command at a time, so no semicolon is needed. But don't forget that you **do** still need a semicolon at the end of each **PHP statement**!

Q: So if my table has data in it and I drop it, will it be deleted too?

A: That is true. So drop tables with care.

Q: So if I need to change a table will I have to drop it first?

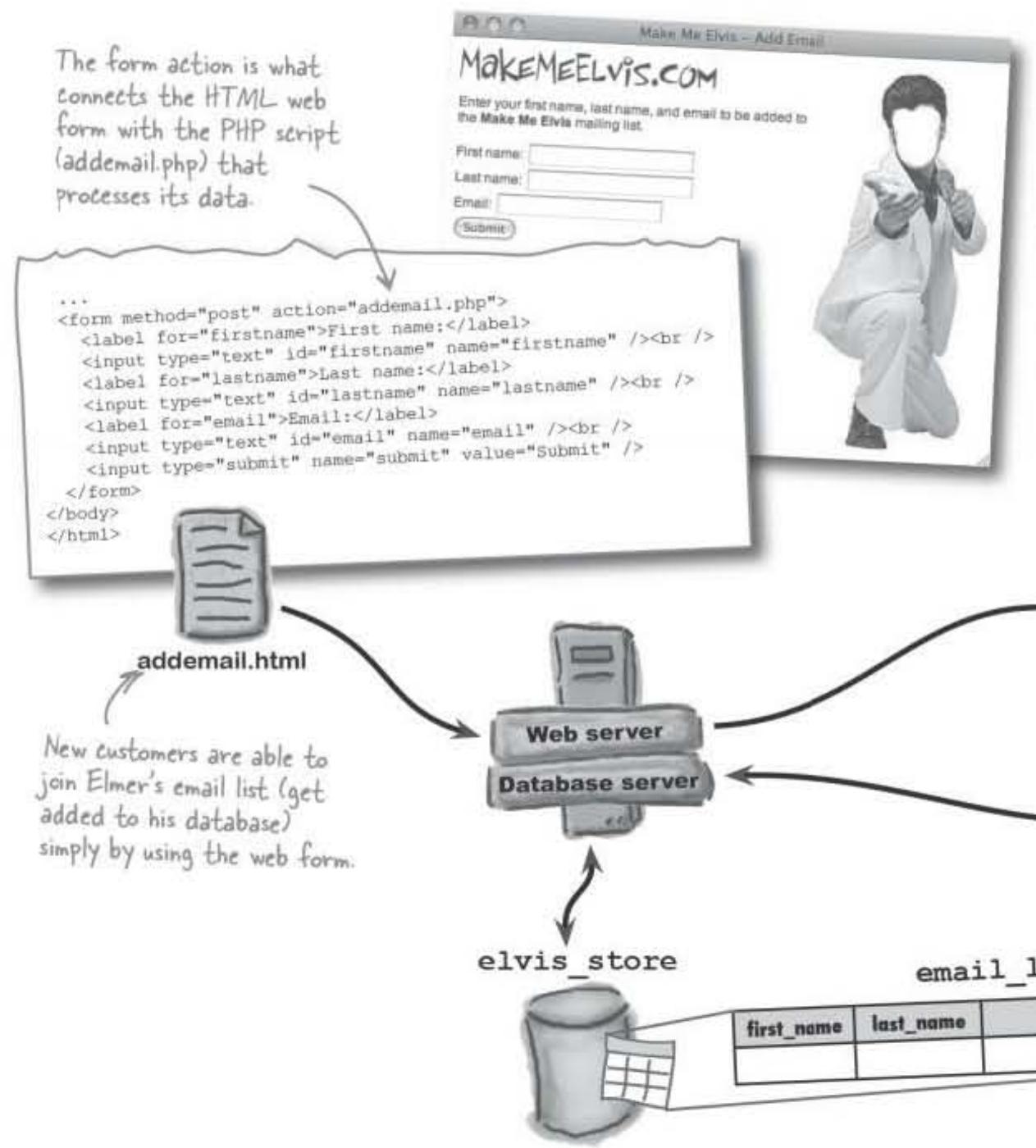
A: Hey, no one is perfect. Everyone makes mistakes. MySQL offers the `ALTER` statement to help us fix them. We'll talk about this command a bit later on in the chapter.

the addemail.php script

Create the Add Email script

Elmer needs an HTML form that can collect names and email addresses from customers. Once he has those, he can grab them with a PHP script and store them in the `email_list` table. The web form (`addemail.html`) requires three input fields and a button. The form action is the most important code in the form since its job is to pass along the form data to the `addemail.php` script we're about to create.

- 1 Create a new file named `addemail.php`
- 2 Create an HTML form and PHP script to add a customer
- 3 Create a new table and PHP script to store the email to the database





The addemail.php script processes data from the Add Email form. The script takes the data from the form, connect to the elvis_store database, and INSERTS the email_list table. Help Elmer by first writing an example SQL query for a customer, and then use that query to finish the PHP script code.

Write an example query
here that inserts data
into Elmer's table.

```
<?php
$dbc = .....  

.....  

$first_name = $_POST['firstname'];
.....  

$query = .....  

.....  

mysqli_query(.....)
.....  

echo 'Customer added.';  

.....  

?>
```

exercise solution

The addemail.php script is called upon to process data from the Add Email form. It should take the data from the form, connect to the elvis_store database, and insert the data into the email_list table. Help Elmer by first writing an example SQL query to add a new customer, and then use that query to finish the PHP script code.

`INSERT INTO email_list (first_name, last_name, email)
VALUES ('Julian', 'Oates', 'julian@breakneckpizza.com')`

Here are the `$_POST` array values that contain the submitted information.

The example INSERT query rewritten as a PHP script using the submitted form data for the values.

```
<?php  
$dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking', 'elvis_store')  
or die('Error connecting to MySQL server.');  
  
$first_name = $_POST['firstname'];  
$last_name = $_POST['lastname'];  
$email = $_POST['email'];  
  
$query = "INSERT INTO email_list (first_name, last_name, email) "  
."VALUES ('$first_name', '$last_name', '$email');  
  
mysqli_query($dbc, $query)  
or die('Error querying database.');  
  
echo 'Customer added.';  
  
mysqli_close($dbc);  
?>
```

If we wanted to be fancy here, we could put a link back to our form with an HTML `<a>` tag.



— Test Drive —

Try out the Add Email form.

Download the code for the Add Email web page from the Head First Labs web site at www.headfirstlabs.com/books/hfphp. It's in the **chapter03** folder. This code consists of Elmer's web form in **addemail.html**, a style sheet (**style.css**), and two images (**elvislogo.gif** and **blankface.jpg**).

Now create a new text file called **addemail.php**, and enter all of the code on the facing page. This is the script that will process Elmer's web form and add new customers to the **email_list** table.

Upload all of these files to your web server and open the **addemail.html** page in a web browser. Enter a new customer in the form, and click Submit.

Check to see that the customer was added to the database by issuing a SELECT query in a MySQL tool.

```
File Edit Window Help BlueSuedeShoes
mysql> SELECT * FROM email_list;
+-----+-----+-----+
| first_name | last_name | email
+-----+-----+-----+
| Julian     | Oates    | julian@breakneckpizza.com
+-----+-----+-----+
1 row in set (0.0005 sec)
```

test your SELECT skills

there are no
Dumb Questions

Q: Is the star in the SQL SELECT command the same thing as an asterisk?

A: Yes, it's the same character on your keyboard, located above the 8 key. Hit SHIFT at the same time as the 8 to type one. But although it's exactly the same character as asterisk, in SQL lingo, it's always referred to as a **star**. This is a good thing, since saying "SELECT asterisk FROM..." is not as easy as saying "SELECT star FROM...".

Q: Are there other characters in SQL with meaning like the star does?

A: While SQL does have other special characters, the star is the only one you need to worry about right now. More importantly for our immediate needs, it's the only one used in the SELECT part of the statement.



Sharpen your pencil

With Elmer's email list starting to fill up, help him write some queries that he can use to find specific customer data.

Select all of the data for customers with a first name of Martin:

.....

Select only the last name for customers with a first name of Bubba:

.....

Select the first name and last name for the customer with an email address of ls@objectville.net.

.....

Select all of the columns for customers with a first name of Amber and a last name of McCarthy:

.....



sharpen your pencil solution

Sharpen your pencil Solution

With Elmer's email list starting to fill up, help him write queries that he can use to find specific customer data.

Select all of the data for customers with a first name of Martin:

```
SELECT * FROM email_list WHERE first_name = 'Martin'
```

The star selects all the columns in the table.

This WHERE clause limits the query results to only those rows with a first name of Martin.

Select only the last name for customers with a first name of Bubba:

```
SELECT last_name FROM email_list WHERE first_name = 'Bubba'
```

Only the last_name column is returned in the query results.

Select the first name and last name for the customer with an email address of ls@objectville.net.

```
SELECT first_name, last_name FROM email_list WHERE email = 'ls@objectville.net'
```

You specify multiple columns of result data by separating the column names with commas.

Select all of the columns for customers with a first name of Amber and a last name of McCarthy:

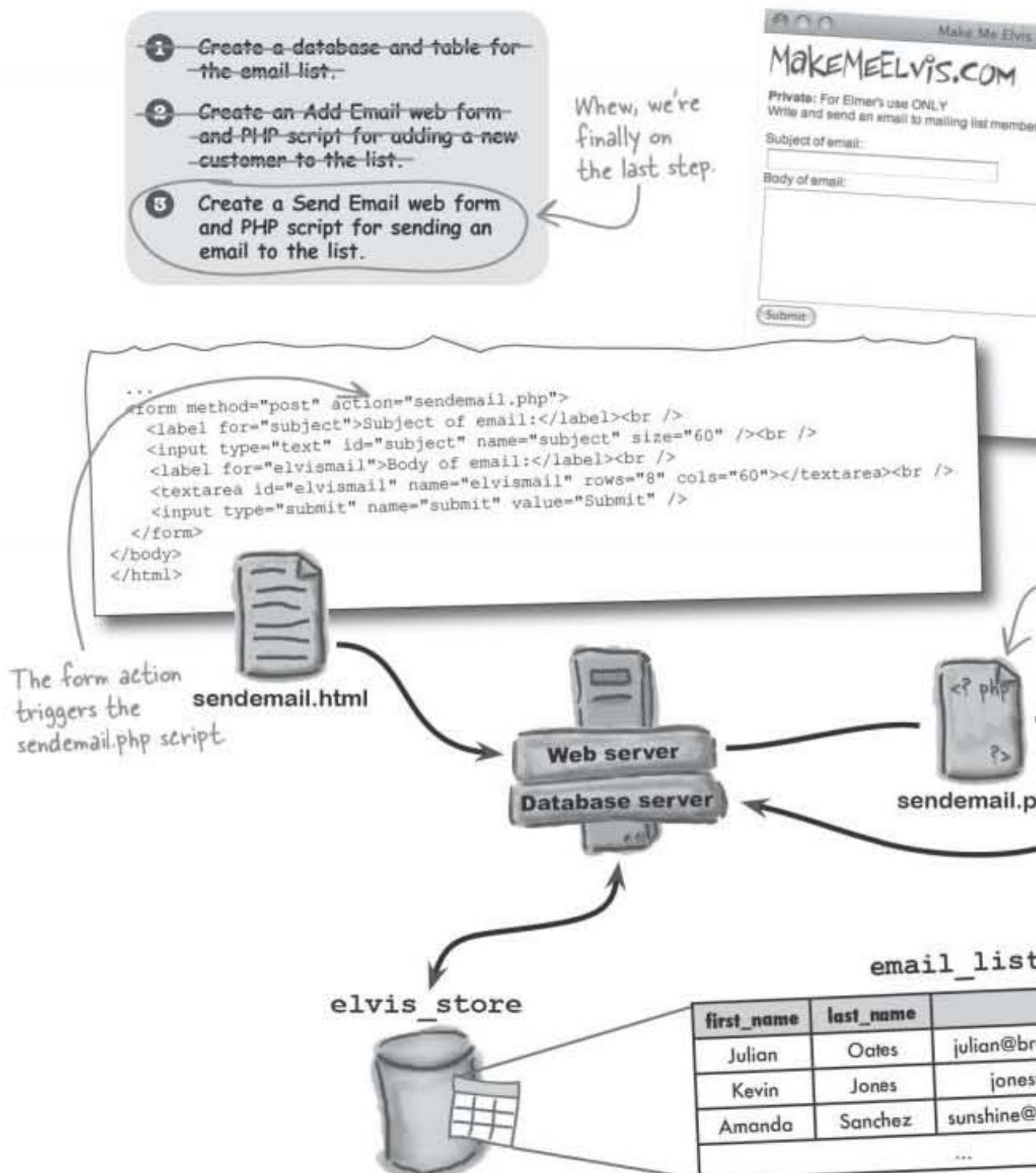
```
SELECT * FROM email_list WHERE first_name = 'Amber' AND last_name = 'McCarthy'
```

The WHERE clause can be made dependent on multiple pieces of information, in this case a match for both a first name AND a last name.

The other side of Elmer's application

Sending email messages to people on Elmer's email list is similar in some ways to adding people to the list because it involves an HTML web form and a PHP script. The big difference, is that sending an email message to the mailing list involves dealing with the *entire contents* of the `email_list` table, whereas the `addemail.php` script only deals with one row of data.

The Server
allows Elmer
subject
email
it to h



the sendemail.php script

The nuts and bolts of the Send Email script

The `sendemail.php` script must combine data from two different sources to generate and send email messages. On the one hand, the script needs to pull the names and email addresses of the email recipients from the `email_list` table in the `elvis_store` database. But it also has to grab the email subject and message body entered by Elmer into the Send Email web form (`sendemail.html`). Let's break down the steps involved.

1 Use the `$_POST` array to get the email subject and message body from the user.

There's nothing new here. Clicking the Submit button in the `sendemail.html` form sends data to `sendemail.php`, where we capture it in variables with a little help from the `$_POST` superglobal array.

2 Run a SELECT query on the `email_list` table.

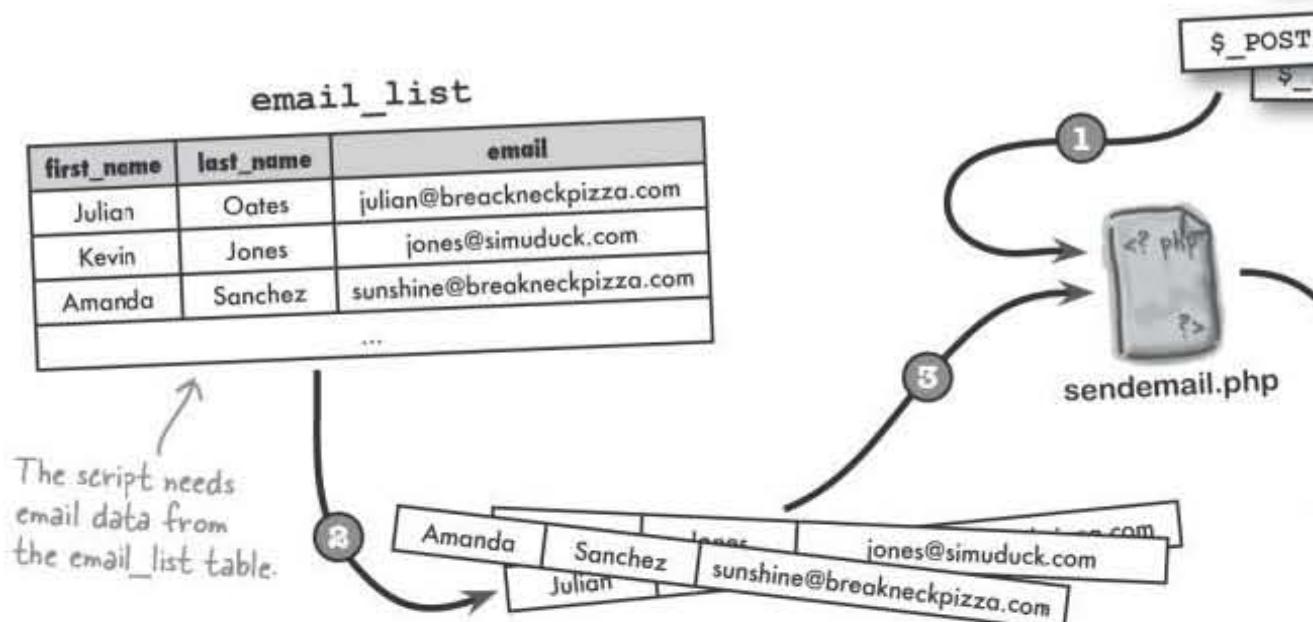
The PHP `mysqli_query()` function runs a SELECT query to get the data for the email list. Since we want all of the data in the table, we can use `SELECT *`.

3 Fetch the email data from the query result.

Running a query alone doesn't provide access to data. We need to grab each row of data from the results in order to have access to the first name, last name, and email address of each customer.

4 Call the `mail()` function to send an email message to each customer.

Sending the emails involves looping through each customer in the email list, which corresponds to each row of data in the query results. The loop we create here starts at the first row of data, then moves on to the second row, and loops through the remaining rows of the data obtained by the SELECT query. We stop when we reach the end of the data.



First things first, grab the data

We're already pretty well versed in extracting data from forms in PHP, so the first step is nothing new, just use the `$_POST` superglobal to store away the email subject and message body in variables. While we're at it, let's go ahead and store Elmer's email address in a variable since we'll need it later when sending the emails.

```
$from = 'elmer@makemeelvis.com';
$subject = $_POST['subject'];
$text = $_POST['elvismail'];
```

Elmer's email address variable so that we know it is in case it ever needs to be used.

The email message form stored in variables, too.

The remaining data required by the `sendemail.php` script comes from Elmer's MySQL database. Pulling customer data from the `email_list` table data into the script requires a `SELECT` query. Unlike before when we've used the MySQL terminal to issue a `SELECT` and look at table data, this time we're doing it in the `sendemail.php` script and issuing the query with `mysqli_query()`.

```
$query = "SELECT * FROM email_list";
$result = mysqli_query($dbc, $query);
```

`mysqli_query` executes the query using a connection variable (`$dbc`) and a query string (`$query`).

Here's our query, which selects all of the columns from the `email_list` table.

A database connection is required in order to submit a query – the details of the connection are stored in the `$dbc` variable.

So all we have to do is go through the query results in the `$result` variable, right?



No, the `$result` variable doesn't actually hold any query results.

If you try to echo the `$result` variable directly, you'll see something like this:

Resource id #3

The `$result` variable stores an ID number for a MySQL **resource**, returned by the query. What happens is the MySQL server temporarily stores your query and gives them a resource number to identify them. You then use this ID with the PHP `mysqli_fetch_array()` function to grab the data.

use mysqli_fetch_array() to get the query results

mysqli_fetch_array() fetches query results

Once our query executes, we can grab the results with the \$result variable. This variable's used with the `mysqli_fetch_array()` function to get the data in the table one row at a time. Each row of data is returned as an array, which we can store in a new variable named \$row.

```
$row = mysqli_fetch_array($result);
```

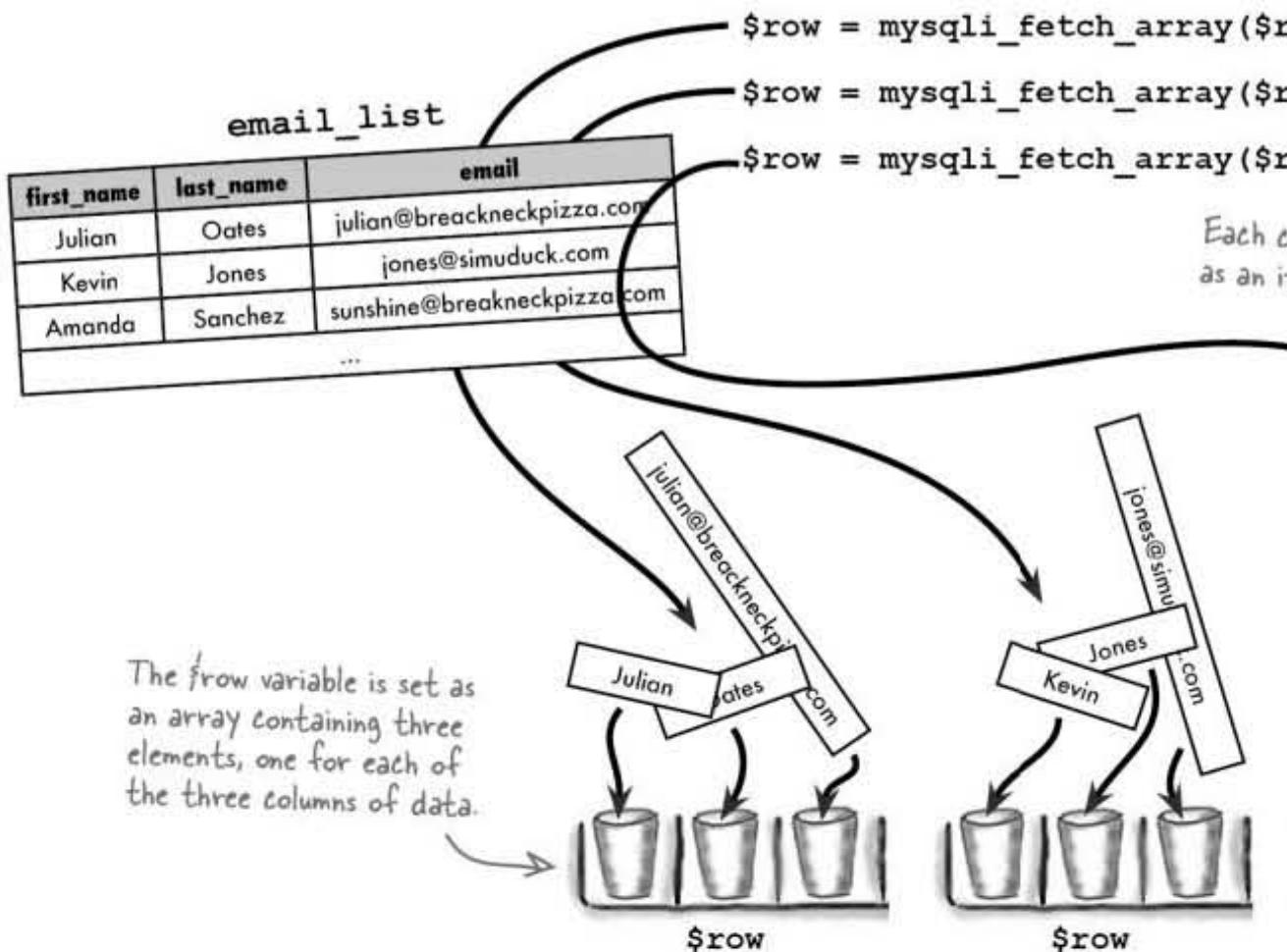
The variable \$row is an array that initially stores the first row of data from our results.

This function retrieves a row of data from the query result and stores it in an array.

Each SQL query has its own unique number that is used to identify the row associated with its results.

Each time this code is executed by the web server, a row of data from the query results gets stored in the \$row array. You repeatedly call the `mysqli_fetch_array()` function to step through each row of the query results. So the first three calls to the `mysqli_fetch_array()` function retrieve the first three rows of data from the table, storing each column of the row as an item in the \$row array.

The mysqli_fetch_array() function stores data in arrays





Sharpen your pencil

As a test to make sure we can actually get the customer's time, finish writing the PHP code to echo the first name and email address of each customer in the email.

```
$query = "SELECT * FROM email_list";
$result = mysqli_query($dbc, $query);
$row = mysqli_fetch_array($result);
```

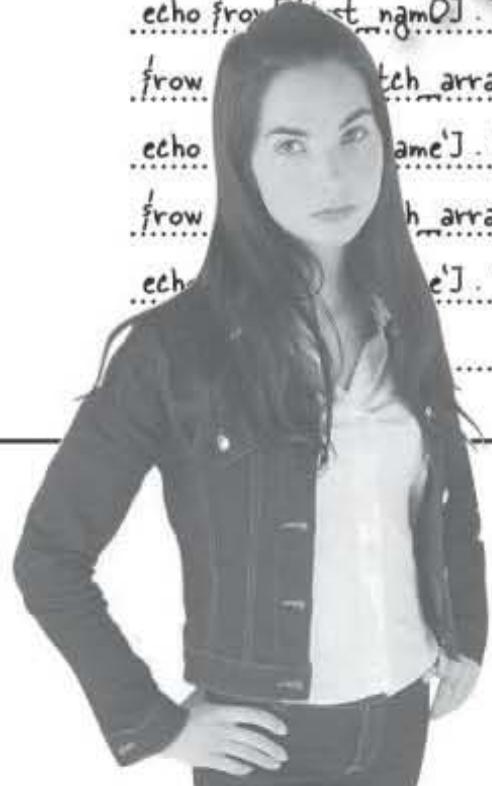
not the best solution


Sharpen your pencil Solution

As a test to make sure we can actually get the customer's name and email address at one time, finish writing the PHP code to echo the first name and email address of each customer in the `email_list` table.

```
$query = "SELECT * FROM email_list";
$result = mysqli_query($dbc, $query);
$row = mysqli_fetch_array($result);
echo $row['first_name'] . ' ' . $row['last_name'] . ':' . $row['email'] . '<br />';
$row = mysqli_fetch_array($result);
echo $row['first_name'] . ' ' . $row['last_name'] . ':' . $row['email'] . '<br />';
$row = mysqli_fetch_array($result);
echo $row['first_name'] . ' ' . $row['last_name'] . ':' . $row['email'] . '<br />';
$row = mysqli_fetch_array($result);
echo $row['first_name'] . ' ' . $row['last_name'] . ':' . $row['email'] . '<br />';
echo $row['first_name'];
$row = mysqli_fetch_array($result);
echo $row['first_name'] . ' ' . $row['last_name'] . ':' . $row['email'] . '<br />';
echo $row['first_name'] . ' ' . $row['last_name'] . ':' . $row['email'] . '<br />';
echo $row['first_name'] . ' ' . $row['last_name'] . ':' . $row['email'] . '<br />';
echo $row['first_name'] . ' ' . $row['last_name'] . ':' . $row['email'] . '<br />';
```

You have got to be kidding me. Repeating the same two lines of code over and over is about the dumbest thing I've ever seen. Surely there's a better way.



There is a better way—we need a loop!

A **loop** is a mechanism in the PHP language that runs code until a certain condition's been met, like running a race. So a loop can **cycle through each row of data**, taking any action we want to each row along the way.

Looping for a WHILE

A while loop is a loop specifically geared toward repeating code **while a certain condition is met**. For example, you might have a variable in a customer service application named `$got_customers` that keeps up with whether or not customers are waiting to be helped. If `$got_customers` is set to `true`, you know there are more customers, so you might call the `next_customer()` function to get the next customer and help them. Here's how this scenario could be coded using a `while` loop:

As long as we still have customers, keep on looping.

```
while ($got_customers) {
```

```
    next_customer();
```

```
    ...
```

This is the code that gets executed each time through the loop.

```
}
```

Enclosing the loop code within curly braces lets you execute as many lines of code as you want.



When we look to see if there are more customers, we're **testing a condition**. The **condition** is the code in the parentheses, and it always poses a question that results in a yes/no answer. If it's yes, or **true**, then the action is performed. If it's no, or **false**, then we quit the loop.

When we call `next_customer()` and proceed to help them, we're **performing an action**. The action is the code inside the curly braces, which is repeated as long as the condition remains `true`. If the condition ever goes `false`, the loop exits and the action is not repeated again. Here's the general format of a `while` loop:

The test condition always results in true or false... keep looping (true) or stop looping (false).

```
while (test_condition) {
```

```
    action
```

```
}
```

The loop action takes place once each time through the loop.



How do you think a loop could be used to go through the contents of Elmer's email?

how while() works

Looping through data with while

Applying a `while` loop to Elmer's email data lets us access the data a row at a time without duplicating any code. We know that `mysqli_fetch_array()` can take a table row and put the column values in the `$row` array, but the function by itself won't get through all of our data—it will store the first row and then stop. A `while` loop can call `mysqli_fetch_array()` to go through each row of result data, one at a time, until it reaches the end.

The while loop condition value of the `mysqli_fetch_array()` function, which is interpreted as true if data is available and false if we're all out of data.

```
while ($row = mysqli_fetch_array($result)) {
    echo $row['first_name'] . ' ' . $row['last_name'] .
        ' : ' . $row['email'] . '<br />';
```

The loop action gets run each time through the loop.

The loop action consists of an echo statement that sticks the row data together with a line break at the end.

The first time through the loop the `$row` array holds the first row of the `email_list` table.

email_list		
first_name	last_name	email
Julian	Oates	julian@breackneckpizza.com
Kevin	Jones	jones@simuduck.com
Amanda	Sanchez	sunshine@breakneckpizza.com
...		

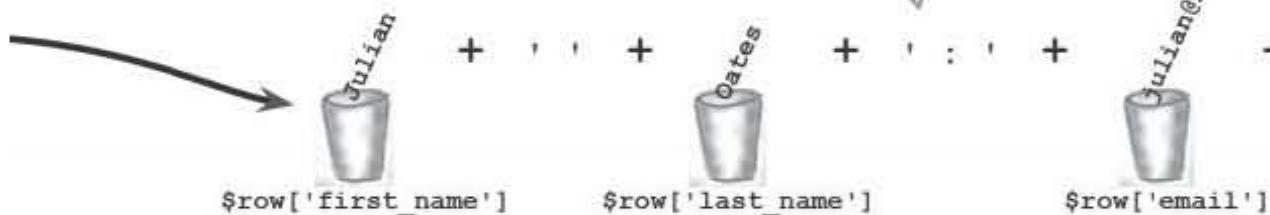
1st loop!

2nd loop!

More loops...

The second time through the `$row` array holds the second row of the `email_list` table...see?

The echo statement inside the while loop takes the data in the \$row array and outputs formatted HTML content.

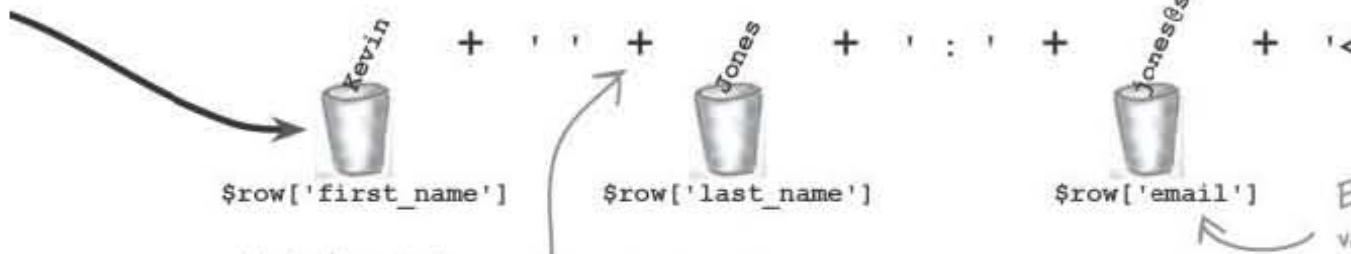


The key used to access the array element must match a column name.

The while loop goes through the table data, row by row. When it runs out of rows of data, it ends.

Julian Oates : julian@breakneckpizza.com
 Kevin Jones : jones@simuduck.com
 Amanda Sanchez : sunshine@breakneckpizza.com
 Bo Wallace : bo@b0tt(msup.com
 Amber McCarthy : amber@breakneckpizza.com
 Cormac Hurst : churst@boards-r-us.co
 Joyce Harper : joyceharper@breakneckpizza.com
 Stephen Meyer : meyers@leapinlimos.co
 Martin Wilson : martybaby@objectville.co
 Walt Peralta : walt@mightygumball.net
 Shannon Munyon : crafisman@breakneckpizza.com
 Joe Milano : joe_m@starbuzzcoffee.co

The second time through the loop the echo statements output another sequence of text, but this time the data in the second row of the table is used.



We don't actually use a plus sign to add strings together - we use the dot operator.

no dumb questions about while()

there are no Dumb Questions

Q: How exactly does the `while` loop know to keep looping? I mean, a `while` loop's controlled by a `true/false` condition, and `mysqli_fetch_array()` returns some kind of resource ID, which is stored in `$row`... That sure doesn't look like a `true/false` test condition.

A: Good observation. As it turns out, PHP is fairly liberal when it comes to how it interprets the "true" condition. In short, any value that is not zero (0) or `false` is considered `true` for the sake of a test condition. So when the `mysqli_fetch_array()` function returns a row of data, the `$row` array is interpreted as `true` since it isn't set to 0 or `false`. And since the test condition is `true`, the loop keeps on chugging. What's interesting is what happens when no more data's available—the `mysqli_fetch_array()` returns `false`, which terminates the loop.

Q: So I can control a `while` loop with any kind of data, not just `true/false` values?

A: That's correct. But keep in mind that ultimately the `while` loop's interpreting the data as `true` or `false`. So the important thing to understand is what constitutes `true` or `false` when it comes to the interpretation of other types of data. And the simple answer is that anything other than 0 or `false` is always interpreted as `true`.

Q: What happens to the `while` loop if no data is returned by the `mysqli_fetch_array()` function?

A: If the query doesn't result in any data, then the `mysqli_fetch_array()` function returns `false`. And this causes the `while` loop to never make it into the action code, not even once.

Q: So it's possible to have a loop that never loops?

A: Indeed it is. It's also possible to have a loop that never stops looping. Consider this `while` loop:

```
while (true) {
```

This is known as an **infinite loop** because the test condition never causes the loop to exit. Infinite loops are a very bad thing.

BULLET POI

- A database is a container for structured data, such as a highly structured map of information.
- Tables store data in a structured way, organized into columns and rows with specific column names.
- The `CREATE DATABASE` statement is used to create a new database.
- The `CREATE TABLE` statement creates a table within a database. It requires detailed information about the columns of data within the table.
- You can delete a table using the `DROP TABLE` statement.
- The `mysqli_fetch_array()` function retrieves a row of results from a database.
- A `while` loop repeatedly executes code while a test condition is true.

- ➊ Create a database named `email_list` and add an `email` table to it.
- ➋ Create an `Add Email` and `PHP` script for adding a customer to the list.
- ➌ Create a `Send Email` and `PHP` script for sending email to the list.

Don't forget that last step!



PHP & MySQL Magnets

Use the magnets below to finish the code for the Send Email script so that Elmer can email to his customer list. As a refresher, here's how the `mail()` function works:

```
mail(to, subject, msg, 'From:' . from);
```

```
<?php
$from = 'elmer@makemeelvis.com';

$subject =
.....;

$text =
.....;

$dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking', 'elvis_st
    or die('Error connecting to MySQL server.');

$query = "SELECT * FROM email_list";
$result = mysqli_query($dbc, $query)
    or die('Error querying database.');

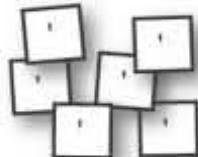
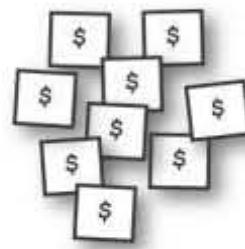
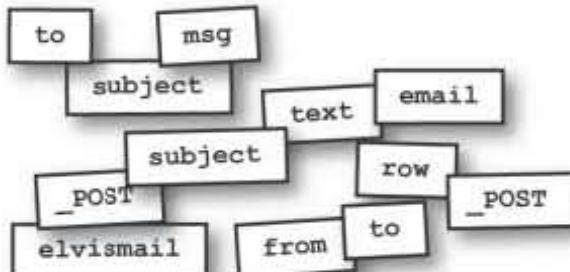
while($row = mysqli_fetch_array($result)) {
    $first_name = $row['first_name'];
    $last_name = $row['last_name'];

    $msg = "Dear $first_name $last_name,\n
.....";

    $to =
.....;
    mail(
.....,
.....,
.....,
.....);
    echo 'Email sent to: ' .
.....;

}

mysqli_close($dbc);
?>
```



the finished sendemail.php script



PHP & MySQL Magnets

Use the magnets below to finish the code for the Send Email script so that Elmer can email his customer list. As a refresher, here's how the mail() function works:

`mail(to, subject, msg, 'From:' . from)`

Make sure to change this to your own email address.

```
<?php
    $from = 'elmer@makemeelvis.com';
    $subject = '$_POST['subject']';
    $text = '$_POST['elvismail']';

    $dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking', 'elvismail');
    or die('Error connecting to MySQL server.');

    $query = "SELECT * FROM email_list";
    $result = mysqli_query($dbc, $query);
    or die('Error querying database.');

    while($row = mysqli_fetch_array($result)) {
        $first_name = $row['first_name'];
        $last_name = $row['last_name'];

        $msg = "Dear $first_name $last_name,\n $text";
        $to = '$row['email']';

        mail($to, $subject, $msg, 'From:' . $from);

        echo 'Email sent to: ' . $to . '<br />';
    }

    mysqli_close($dbc);
?>
```

The Subject form "subject", which is used to access it in the code.

The email message body is entered into the field.

The email message body is taken from the customer's form field email text.

The email recipient, message subject, and message body, are passed into the mail() function, along with Elmer's "from" address.

The "email" column in the database holds the email address of the customer, which the message should be addressed to.

A confirmation message is echoed to the page with the email address of each customer who is mailed.

It's generally good security practice to directly echo user input into a database query.



— Test Drive —

Send an email to the mailing list using the Send Email form

Download the code for the Send Email web page from the Head First Labs website at www.headfirstlabs.com/books/hfphp. It's in the **chapter03** folder. Similar to the Add Email page you saw earlier, this code consists of a web form in **sendemail.html**, a style sheet (**style.css**), and a couple of images (**elvislogo.gif** and **blankface.jpg**).

Create a new text file called **sendemail.php**, and enter all of the code on the facing page. Upload all of these files to your web server and open the **sendemail.html** page in a web browser. Enter an email message in the form and click Submit.

You've got mail...from Elmer!

At last, Elmer can send out his MakeMeElvis.com sale emails to everyone on his mailing list by using his new Send Email web form and PHP script. He can also use the output from the script to confirm that each message is successfully being sent. Each time the code in the script's `while` loop executes, he sees "Email sent to `someone@somewhere.com`" with the email address of the person in his database. The end result is more exposure for his products, and for better or worse, more Elvis look-alikes!

The illustration shows Elmer Fudd, a character from Looney Tunes, wearing a white lab coat and sunglasses, standing next to a computer monitor. A thought bubble above him contains the text: "I've sold out of blue suede shoes...I'm rich!" The computer screen displays a web form titled "ELVIS - Send Email". The form includes fields for "Subject of email:" (containing "Big Sale!") and "Body of email:" (containing "Big sale this week at MakeMeElvis.com! Genuine horse hair sideburns 20% off! And don't forget the 'buy one, get one free' leisure suits — only three days left!"). Below the body field is a "Submit" button. To the right of the form, a scroll-down list shows a series of email addresses that have been sent:

- Email sent to: julian@breakneckpizza.net
- Email sent to: jones@simuduck.com
- Email sent to: sunshine@breakneckpizza.net
- Email sent to: bo@b0t0msup.com
- Email sent to: amber@breakneckpizza.net
- Email sent to: churst@boards.r-us.co
- Email sent to: joycecharpen@breakneckpizza.net
- Email sent to: meyers@leapinlimos.com
- Email sent to: martybaby@objectville.net
- Email sent to: wahl@mightygumball.net
- Email sent to: craftsman@breakneckpizza.net
- Email sent to: joe_m@starbuzzcoffee.com
- Email sent to: bruce@chocholic-inc.com
- Email sent to: pr@honey-dot.com
- Email sent to: bertieh@objectville.net
- Email sent to: gregcek@breakneckpizza.net
- Email sent to: wilmauw@starbuzzcoffee.com
- Email sent to: samjaffe@starbuzzcoffee.com
- Email sent to: ls@objectville.net
- Email sent to: bshakes@mightygumball.net

our app needs delete functionality

Sometimes people want out

As with any blossoming new business, there are bumps in the road. It seems some Elvis fans have jumped ship on the King and want off Elmer's mailing list. Elmer wants to oblige, but that means he needs to remove the customers from his database.

Dear Elmer,

I do not wish to receive any more sales emails for the Elvis Store. I'm still a fan of Elvis, but I can no longer look the part. Please take me off of your list. My email is cbriggs@boards-r-us.com.

Thanks,

An Ex-Impersonator

Dear Fellow Hip Swiveler,

While I still enjoy Elvis's spirited moves, I'm just not into him so much anymore. I now prefer Liberace's understated showmanship and deft piano skills. Here's my email address (please remove me):

lindy@tikibeanlounge.com

Yours Truly,

Liberace Lindy

Dear Sir,

After several allergic reactions to your authentic horse hair sideburns, I've decided that maybe looking like Elvis isn't my "thing." I do love a good cape but the sideburns are just too much. Please remove me from your email list.

Yours Truly,

Brian Powers

bp@honey-doit.com

I suppose n
out to emula
to get these
so I can foo

Elmer's not too happy about losing customers, but he wants to honor their requests to be removed from his mailing list.

It's a fact of MySQL life—sometimes you need to remove data from your database. Elmer needs to expand his application to delete users from the `email_list` table.

Write down the new application components you think Elmer is going to need to implement the Remove Email feature:

.....

.....

Removing data with DELETE

To delete data from a table, we need a new SQL command, DELETE. We'll use DELETE in a new Remove Email script that deletes customers' data from Elmer's mailing list. In fact, we need a new script and a new web form to drive it... but first we need DELETE.

The DELETE SQL command removes rows of data from a table. This makes it a command you should use very carefully since it's capable of wiping out a table full of data in the blink of an eye. Knowing this, here's the most dangerous form of DELETE, which deletes every row from a table.

DELETE FROM *table_name*

Without any other qualifiers, the DELETE command completely empties a table of all its data.

This is the name of the table you want to delete rows from.

So we can never delete anything from a table without deleting **everything**?

No, not at all. DELETE can be used to pinpoint a specific row or rows for deletion.

To precisely target the row or rows you want to delete with DELETE, you need to tack on a WHERE clause. If you recall from using it with the SELECT command, WHERE allows you to isolate specific rows in a query.

Sharpen your pencil

Suppose Elmer had 23 customers with a first name of Anne, a last name of Parker, and one customer with the name Anne. Fill in the blanks below to count down how many rows of data are deleted by each of these queries:

```
DELETE FROM email_list WHERE first_name = 'Anne';
```

```
DELETE FROM email_list WHERE first_name = 'Anne' OR last_name = 'Parker';
```

```
DELETE FROM email_list WHERE last_name = Parker;
```

- 1 Create a new script to handle the email removal.
- 2 Create an email list and PHP script to add a customer.
- 3 Create a new email list and PHP script to remove a customer.
- 4 Create a new email list and PHP script to remove multiple customers.

Look
new
desi

DELETE with WHERE

Sharpen your pencil Solution

Suppose Elmer had 23 customers with a first name of Anne and a last name of Parker, and one customer with the name Anne Parker. Write down how many rows of data are deleted by each of these queries:

```
DELETE FROM email_list WHERE first_name = 'Anne';
```

```
DELETE FROM email_list WHERE first_name = 'Anne' OR last_name = 'Parker';
```

```
DELETE FROM email_list WHERE last_name = Parker;
```

Trick question! The last name Parker is not quoted, so no rows are deleted. Both the first and last names must be quoted.

Use WHERE to DELETE specific data

By using a WHERE clause with the DELETE command, we target specific rows of data for deletion, instead of emptying an entire table. The WHERE clause lets us focus on just the row we want to remove, in this case one of Elmer's customers who wants to be removed from the mailing list.

`DELETE FROM email_list`

`WHERE email = 'pr@honey-doit.com'`

The value to match

The name of a table column

This part of the WHERE clause performs a test on every row to see what rows match.

The actual test within a WHERE clause performs a comparison that is carried out against every row in the table. In this example, the equal sign (=) tests each value in the email column to see which rows are equal to "pr@honey-doit.com". If the value in the email column of a row matches, then that row will be deleted.

Write down why you think the email column is used in the WHERE clause, as opposed to first_name or last_name:

.....

.....

Minimize the risk of accidental deletions

It's important to understand that although any column name can be used in a WHERE clause to match rows, there's a *very* good reason why we chose the email column for Elmer's DELETE query. Consider that if more than one row matches a WHERE clause, all of the matching rows will be deleted. So it's important for Elmer's WHERE clause to pinpoint ***exactly*** the row you want to delete.

What we're really talking about is uniqueness. It's fairly safe to assume that email addresses are unique within the email_list table, whereas first names and last names are not. You don't want to create a WHERE clause matching the first_name column to "Pat" just to delete a single customer—you'll end up deleting every customer named Pat! That's why Elmer's WHERE clause is carefully crafted to look for a specific match with the email column.

```
DELETE FROM email_list
WHERE email = 'pr@honey-doit.com'
```

Using the email column in the WHERE clause helps to establish uniqueness and reduce the risk of accidentally deleting a row.

If we used first_name in the WHERE clause instead of email, this user would accidentally get deleted.

first_name	last_name	email
Joe	Milano	joe_m@starbu...
Bruce	Spence	bruce@choco...
Pat	Riese	pr@honey-d...
Bertie	Henderson	bertieh@o...
Greg	Eckstein	gregeck@bre...
Wilma	Wu	wilmawu@sta...
Sam	Jaffe	samjaffe@sta...
Louis	Shaffer	ls@obj...
Bubba	Shakespeare	bshakes@mi...
John	Doe	johndoe@tik...
Pat	Grommet	grommet@...

File Edit Window Help ByeBye

```
mysql> DELETE FROM email_list WHERE email = 'pr@honey-doit.com'
1 row deleted (0.005 sec)
```

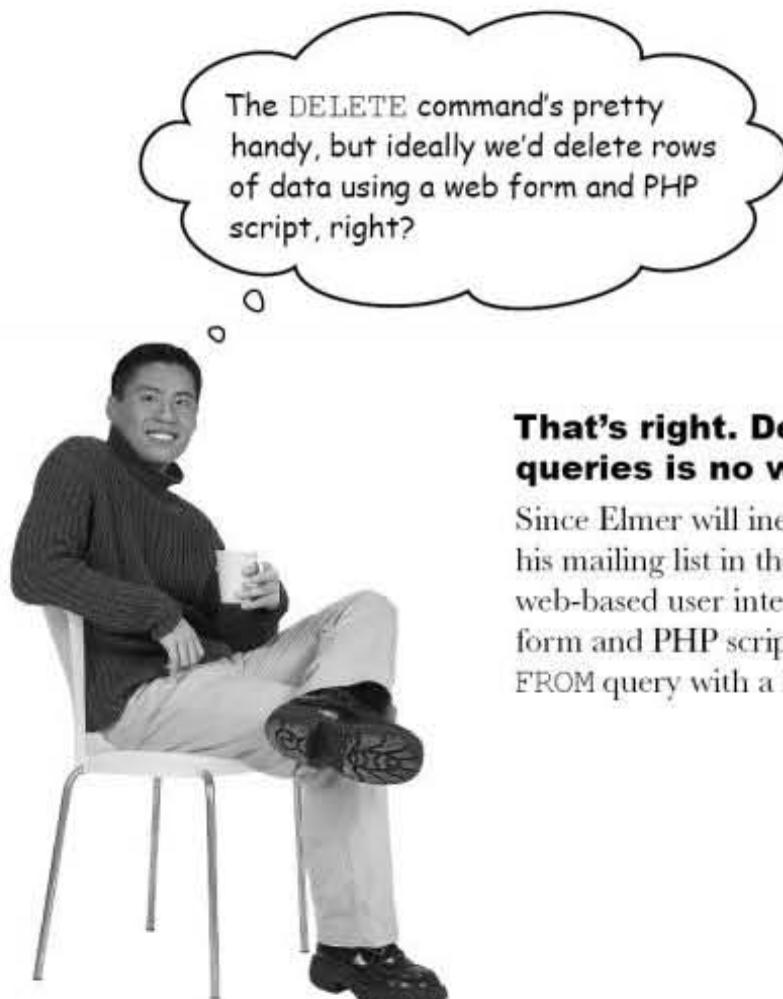
test-drive the DELETE command



—Test Drive

Try out the **DELETE** command on Elmer's database.

Fire up a MySQL tool and try a few **DELETE** commands to delete individual rows of data from the `email_list` table based on customers' email addresses. Just make sure to include a `WHERE` clause on each **DELETE** statement so that you don't accidentally wipe out the whole table!



The **DELETE** command's pretty handy, but ideally we'd delete rows of data using a web form and PHP script, right?

That's right. Deleting users by hand queries is no way to manage a mailing list.

Since Elmer will inevitably face users who want to manage their own entries in his mailing list in the future, it makes a lot of sense to provide a simple web-based user interface for removing customer entries. A simple web form and PHP script should do the trick, not to mention the added benefit of being able to use a `DELETE` query with a little help from a `WHERE` clause.



Elmer has created a web form (`removeemail.html`) for deleting a customer from a mailing list. All the form accepts is an email address, which is entered into a text input field named `email`. Finish the code for Elmer's `removeemail.php` script that processes the form to carry out each customer removal.

225b

```
$dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking', 'elvis'
    or die('Error connecting to MySQL server.'))
```

the finished removeemail.php script

Elmer has created a web form (`removeemail.html`) for deleting a customer mailing list. All the form accepts is an email address, which is entered into a text field named `email`. Finish the code for Elmer's `removeemail.php` script so that the form can carry out each customer removal.

This form field is named "email".

Clicking the Remove button submits the form as a POST request to the PHP script.

The email form data in `$_POST` is stored in a variable and then used in the DELETE query.

```

<?php
    $dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking', 'elvis');
    or die('Error connecting to MySQL server.');

    $email = $_POST['email'];

    $query = "DELETE FROM email_list WHERE email = '$email'";
    mysqli_query($dbc, $query);
    or die('Error querying database.');

    echo 'Customer removed: ' . $email;
}

mysqli_close($dbc);
?>
```

Don't forget to clean up by closing the database connection.

MAKEELVIS.COM

Enter an email address to remove.

Email address:

Remove



removeemail.html

Watch out for the double quotes here: quotes go around the query and the string for the email address.

It never hurts to confirm what happened, especially in the case of a database deletion.

Whew, we're
finally finished!

- ➊ Create a database and table for the email list.
- ➋ Create an Add Email web form and PHP script for adding a new customer to the list.
- ➌ Create a Send Email web form and PHP script for sending an email to the list.
- ➍ Create a Remove Email web form and PHP script for removing a customer from the list.



— Test Drive —

Remove a customer from the mailing list using the Remove Email form.

This is starting to feel a little familiar, eh? Download the code for the Remove Email web page from the Head First Labs web site at www.headfirstlabs.com/books/hfphp. It's in the `chapter03` folder. This code consists of a web form in `removeemail.html`, a style sheet (`style.css`), and a couple of images (`elvislogo.gif` and `blankface.jpg`).

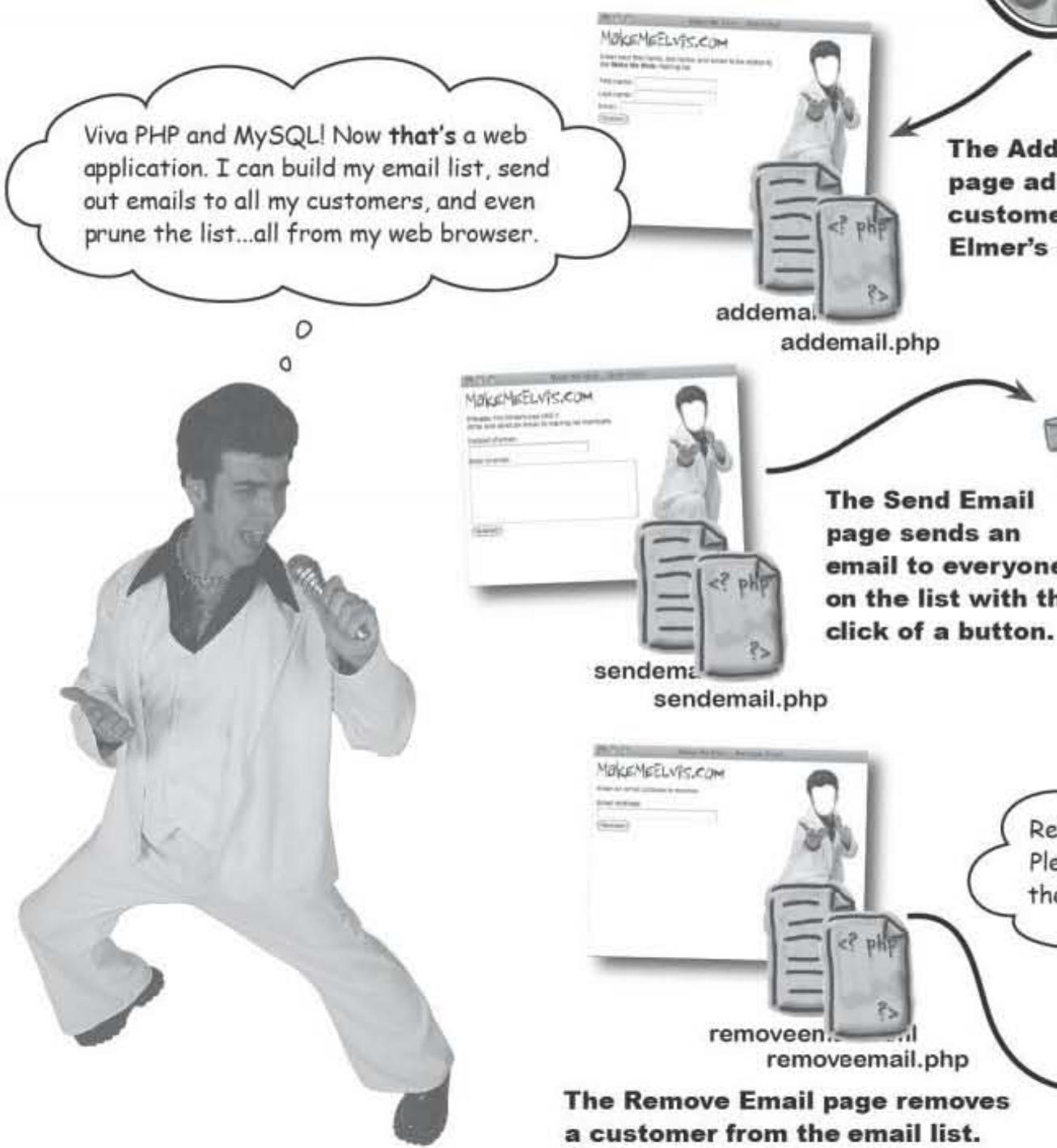
Create a new text file called `removeemail.php`, and enter all of the code on the facing page. Upload all of these files to your web server and open the `removeemail.html` page in a web browser. Enter the email address of a customer in the form, and click Remove to delete them from the database.



the mailing-list app is complete!

MakeMeElvis.com is a web application

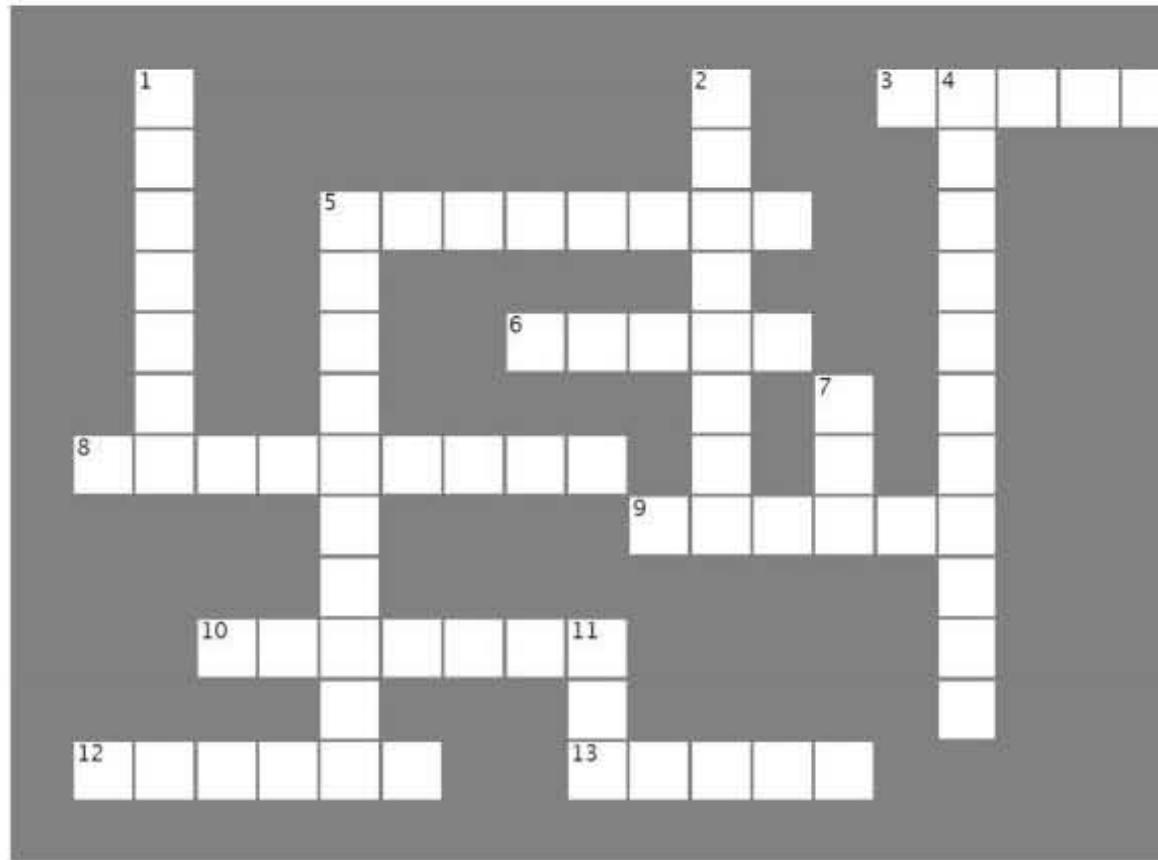
It's official. With the help of PHP and MySQL, Elmer's MakeMeElvis.com web site is now worthy of being called an application. Elmer can now store data persistently in a MySQL database, and also interact with that data through web forms. A combination of HTML pages, PHP scripts, and embedded SQL queries allow Elmer to add and remove customers to/from his email list (they can also add themselves), as well as send email messages to the entire list.





PHP&MySQLcross

When you're finished perfecting Elmer's dance moves, see if you can hum along and finish this crossword puzzle.



Across

3. A MySQL database is divided into these.
5. A persistent, highly organized, data structure that is typically stored in a file on a hard drive.
6. This conditional clause can be added to SQL statements to control which rows are targeted.
8. This SQL command removes an entire table from a database.
9. Use this SQL command to choose rows from a table.
10. Use this MySQL data type to store a varying amount of text.
12. Within a MySQL table, this holds a specific type of data.
13. Keep doing something as long as a certain test condition remains true.

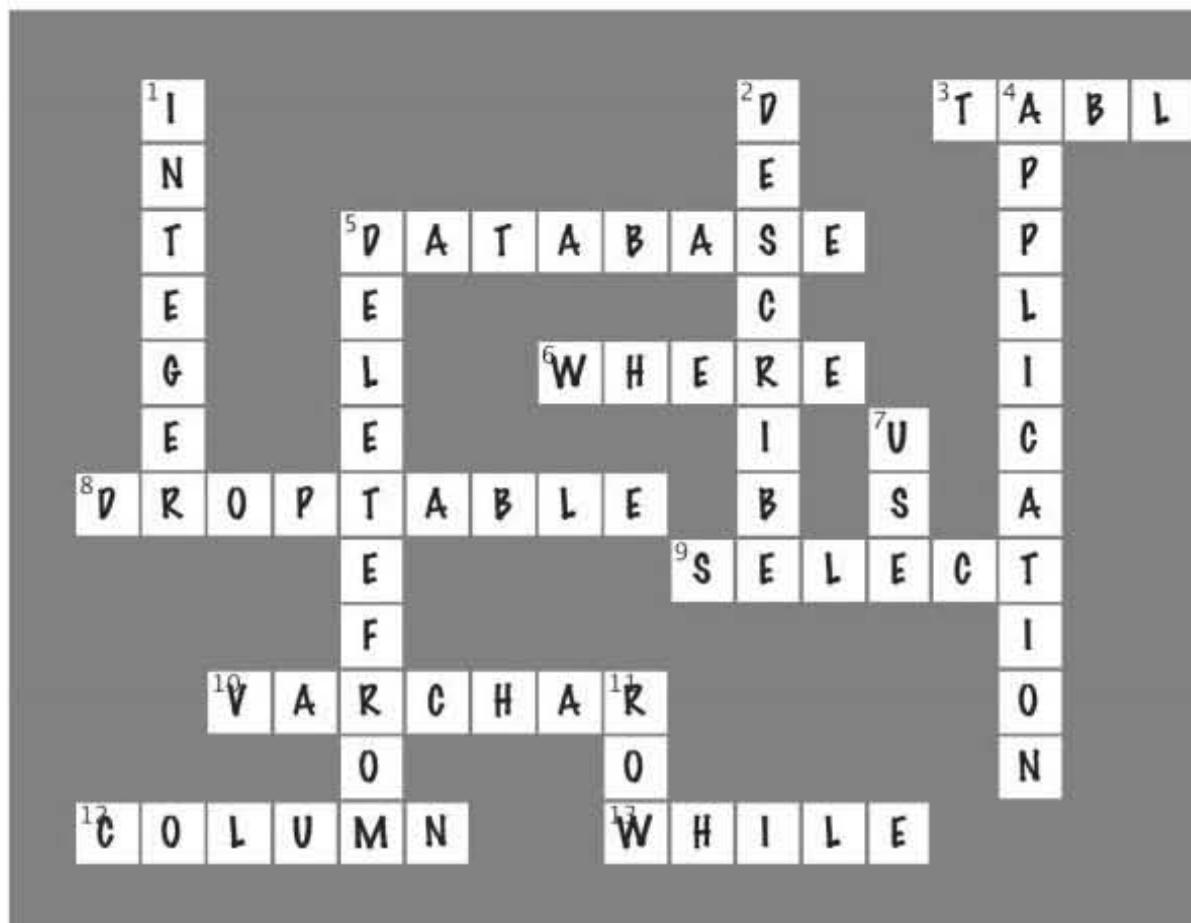
Down

1. A MySQL data type that stores numbers with decimal places.
2. Use this SQL command to look at the structure of a table.
4. When dynamic functionality is added to PHP and MySQL, it becomes an
5. Use this SQL command to destroy a table.
7. After creating a new database in a MySQL server, issue this command before you can do anything with it.
11. A single collection of data in a table or column.

php&mysqlcross solution



PHP&MySQL cross Solution





Your PHP & MySQL Toolbox

Not only did you help Elmer get his web application off the ground, but you also developed some valuable PHP and MySQL skills in this chapter. For instance...

`while`

A PHP looping construct that allows you to repeat a section of code as long as a certain condition remains true. One particularly handy usage of the while loop is in looping through rows of data in an SQL query result.

`DROP TABLE tableName`

This SQL statement drops an entire table from the database, meaning that the table is removed along with any and all data stored within it.

`DELETE FROM tableName`

Use this SQL statement to delete rows from a table. Depending on how you use the statement, you can delete individual rows or multiple rows.

`WHERE`

This SQL clause is used in conjunction with other SQL commands to build statements that target specific rows in a table. For example, you can isolate rows that have a column matching a specific value.

`mysqli_fetch`

This built-in PHP function retrieves a single row from the results of a query. You can call it repeatedly to read all of the data.

`DESCRIBE`

If you need to see the structure of a table, this statement is what you want. It doesn't reveal any data, but it does show the columns and their respective types.

`SELECT * FROM`

This SQL statement retrieves all data from a table. When the asterisk (*) is used (*), all of the rows in the table are returned. You can be more specific by selecting individual columns instead of the *. If you don't specify a column name, all of the column data will be included in the query.

4 realistic and practical applications

Your Application on the Web

If I put a banana in my teacher's tailpipe, her car won't start, so no exam. But then the substitute might give the test, so he gets a banana, too. But then the substitute's substitute...



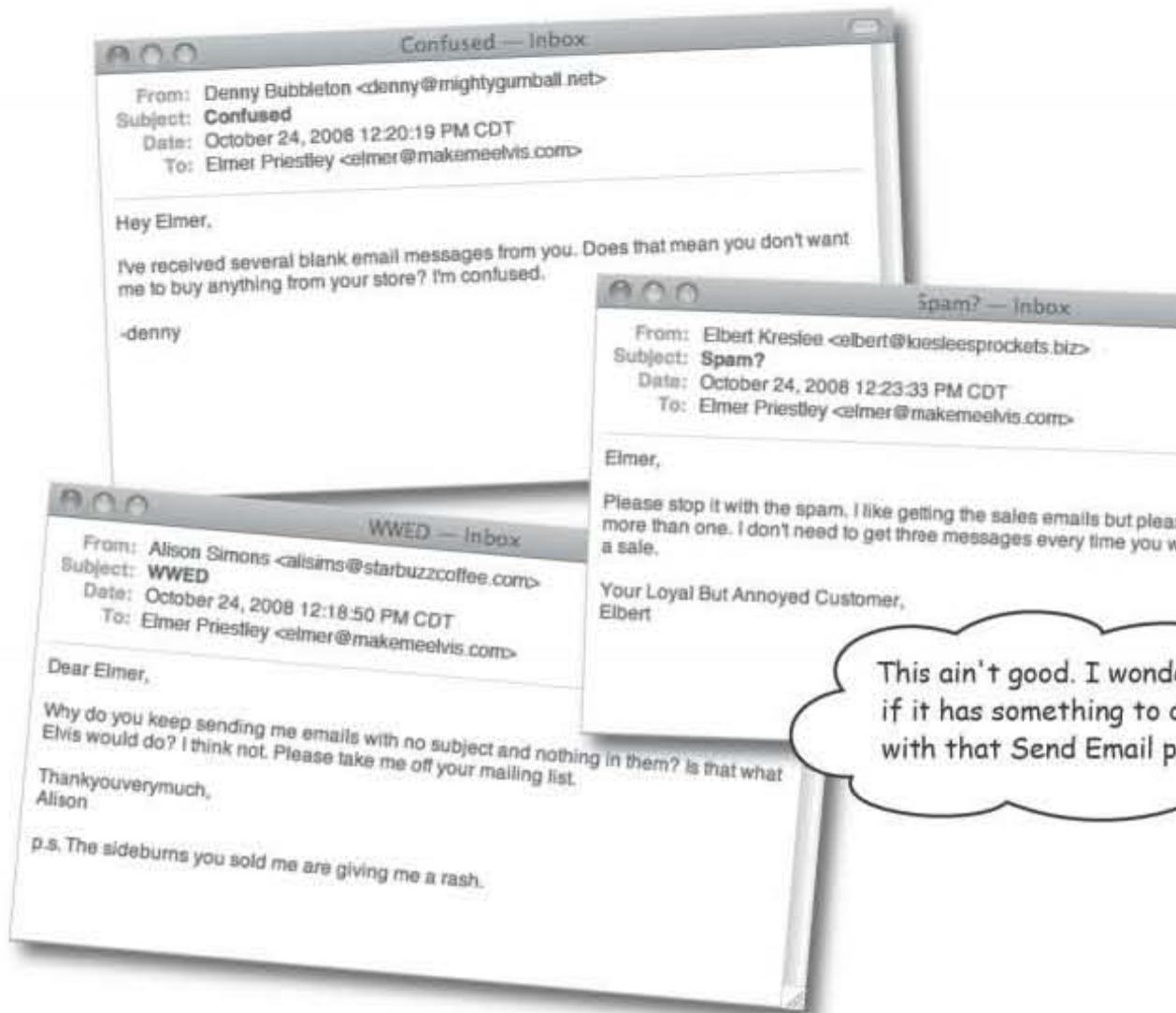
Sometimes you have to be realistic and rethink your plans.

Or plan more carefully in the first place. When your application's out there on the Web, you may discover that you haven't planned well enough. Things that you thought would work aren't good enough in the real world. This chapter takes a look at some *real-world problems* that can occur as you **move your application from testing to a live site**. Along the way, we'll show you more important PHP and SQL code.

elmer needs a better mailing-list app!

Elmer has some irritated customers

Elmer's customer mailing list has grown by leaps and bounds, but his emails have generated some complaints. The complaints vary, but they all seem to involve customers receiving blank email messages or multiple messages, neither of which is good. Elmer needs to figure out what's gone wrong and fix it. His business depends on it.



Elmer knows he has a problem, but he's going to need some help figuring out exactly what it is.

BE Elmer the email list manager

Your job is to play Elmer and figure out how those blank emails are getting sent. He suspects it has something to do with the sendemail.html form.



Write
thinks

Make Me Elvis - Send Email

MakEMEElvis.COM

Private: For Elmer's use ONLY
Write and send an email to mailing list members.

Subject of email:

Body of email:

Submit

 sendemail.html

A black and white illustration of a person wearing a white lab coat and holding a clipboard, standing next to a large, empty thought bubble. To the right of the thought bubble, a portion of another person's arm and shirt is visible.

be elmer solution

BE Elmer the email list manager Solution

Your job is to play Elmer and figure out how those blank emails are getting sent. He suspects it has something to do with the `sendemail.html` form.

Write
thinks

Make Me Elvis - Send Email

MakEMEElvis.COM

Private: For Elmer's use ONLY
Write and send an email to mailing list members.

Subject of email:

Body of email:

Submit

 sendemail.html

If I click the Submit button without filling out a message body, a blank email gets sent.



If the Submit button's pressed on the form with nothing in the Body field, a blank email gets sent. Come to think of it, an empty Subject field is a problem too.

Protecting Elmer from... Elmer

So “operator error” is really the problem here—Elmer inadvertently clicks Submit without entering the email information, and blank emails get sent to the entire list. It’s never safe to assume a web form will be used exactly the way it was intended. That’s why it’s up to you, the vigilant PHP scripter, to try and head off these kinds of problems by anticipating that some users will misuse your forms.

Let’s take a look at the code in our current `sendemail.php` script to see how Elmer’s empty email messages are getting created.

Our Send Email script uses the text from the form to build an email, even if the user didn’t enter anything.

```
<?php
    $from = 'elmer@makemeelvis.com';
    $subject = $_POST['subject'];
    $text = $_POST['elvismail'];

    $dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking',
        or die('Error connecting to MySQL server.');

    $query = "SELECT * FROM email_list";
    $result = mysqli_query($dbc, $query)
        or die('Error querying database.');

    while ($row = mysqli_fetch_array($result)) {
        $to = $row['email'];
        $first_name = $row['first_name'];
        $last_name = $row['last_name'];
        $msg = "Dear $first_name $last_name,\n$text";
        mail($to, $subject, $msg, 'From:' . $from);
        echo 'Email sent to: ' . $to . '<br />';
    }

    mysqli_close($dbc);
?>
```

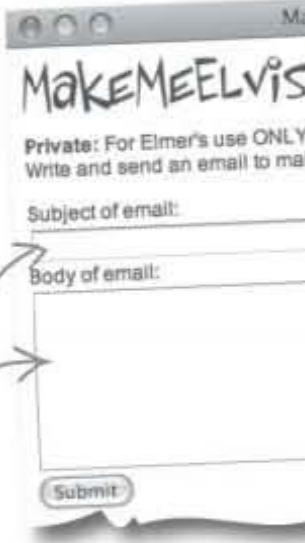
The text in the form is retrieved from `$_POST['subject']` and `$_POST['elvismail']`, and is saved in `$subject` and `$text`, respectively.

Problem is, we use `$text` message whether the value contains text or not...

...and we also use `$subject` whether there's text in it or not.

Write down what you think should be changed in the `sendemail.php` script code to fix the blank email problem.

.....
.....



sendemail.php needs validation

Demand good form data

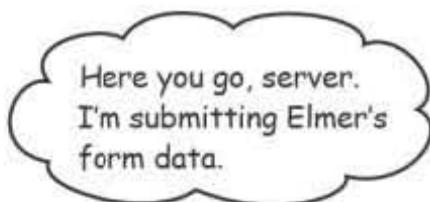
Elmer's Send Email form's in need of **validation**, which is the process of checking to make sure form data is OK before doing anything with it. Elmer already uses validation even though he doesn't call it that. Whenever he receives an order for Elvis gear, he doesn't just immediately fill it and send it out... he validates it first!

In the case of an order, Elmer first checks to see if the customer's credit card is **valid**. If so, he fills the order and gets it ready to ship. But then he has to check if the customer's shipping address is complete. If that checks out, then Elmer goes ahead and sends out the order. A successful order for Elmer's store always hinges on the validation of the order data.



To solve Elmer's blank email problem, we need to validate the form data delivered to the *sendemail.php* script. This means the form data is submitted from the client web page (*sendemail.html*) to the server, and the server (*sendemail.php*) checks to make sure all the data is present. We can add code to *sendemail.php* that examines the values in the text boxes and checks to make sure they aren't empty. If everything checks out OK, the script sends out the emails.

- 1 Elmer fills out and submits the Send Email form.



- 2 The form data is sent to the Send Email script on the server.

```
<form action = "sendemail.php"
...

```

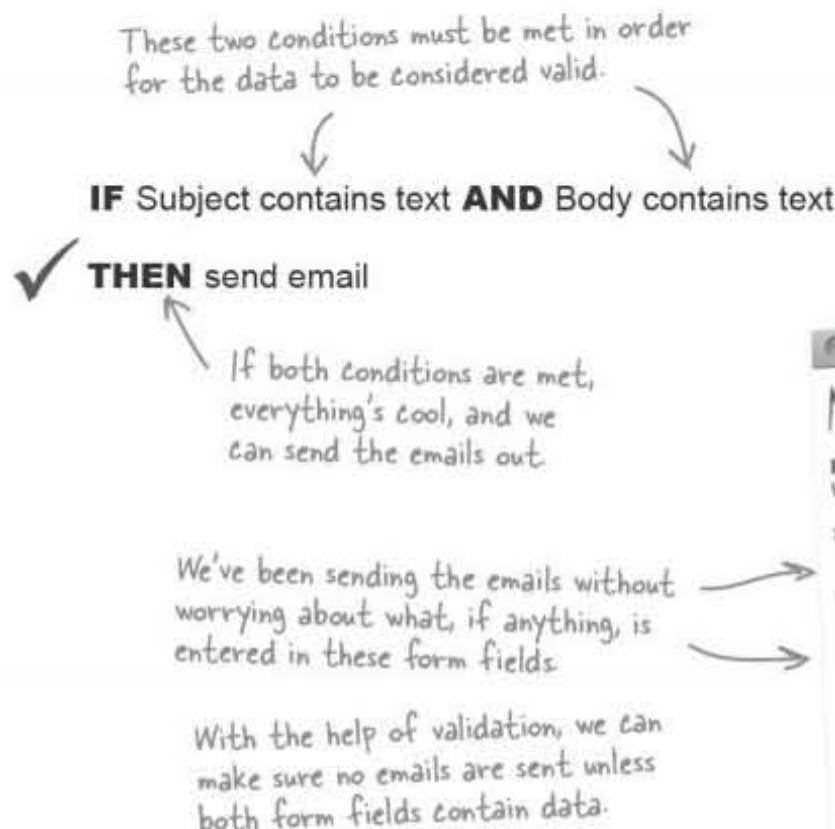


- 4 The server sends an HTML response back to the browser: either that the mail was sent or that the form data was invalid.



The logic behind Send Email validation

Elmer needs to **validate** the data he gets from the `sendemail.html` form before he sends any emails. In fact, sending the emails should completely hinge on the data validation. What we really need PHP to do is **make a decision** based on the validity of the form data received by the `sendemail.php` script. We need code that says, “**if** the data is **valid**, go ahead and send the emails.”



there are no
Dumb Questions

Q: I've also heard of validating data on the client instead of on the server. How does that work?

A: The web browser is considered the client, so client-side validation would be any checking that occurs before the data's sent to the PHP script. Languages like JavaScript can do client-side validation. If you're interested in learning more, check out *Head First JavaScript*, which discusses client-side validation in depth.

Q: So why use server-side validation?

A: If we validate on the client, only part of Elmer could potentially browse directly to send out a blank email. But if we validate on the client, there are still problems. Blank data in the form will be sent to the PHP script, which will then be sending data from the PHP script being directly to the server. It's wrong to validate on the client. In fact, the server is the last line of defense for client-side validation, so client-side validation can't be ignored.

the if statement

Your code can make decisions with IF

The PHP **if** statement lets your code make decisions **based on whether or not something is true**. Consider Elmer's orders again. Before filling an order, Elmer must get paid, which means charging the customer's credit card. If the customer gave Elmer the wrong card number, he can't fill the order. So Elmer performs a kind of real-world validation on every order that goes like this:

If the customer's credit card checks out, go ahead and fill the order.

We can translate this scenario to PHP code using the **if** statement, which is designed to handle just this kind of decision making.

The basic if statement has three parts:

1 The if keyword

This starts off the statement.

2 The test condition

The test condition, or **conditional expression**, is located in **parentheses** right after the **if** keyword. Here's where you put the statement that you want to determine the validity, or truth, of.

3 The action

The action of an **if** statement directly follows the test condition and is enclosed in **curly braces**. Here's where you put the PHP code you want to execute if the condition is, in fact, **true**.

```

if (isValid($credit_card_num)) {
    fillOrder();
}
  
```

The statement begins with if.

This is the condition. It's calling a function to check and see if what's stored in \$credit_card_num is valid.

This brace encloses the action.

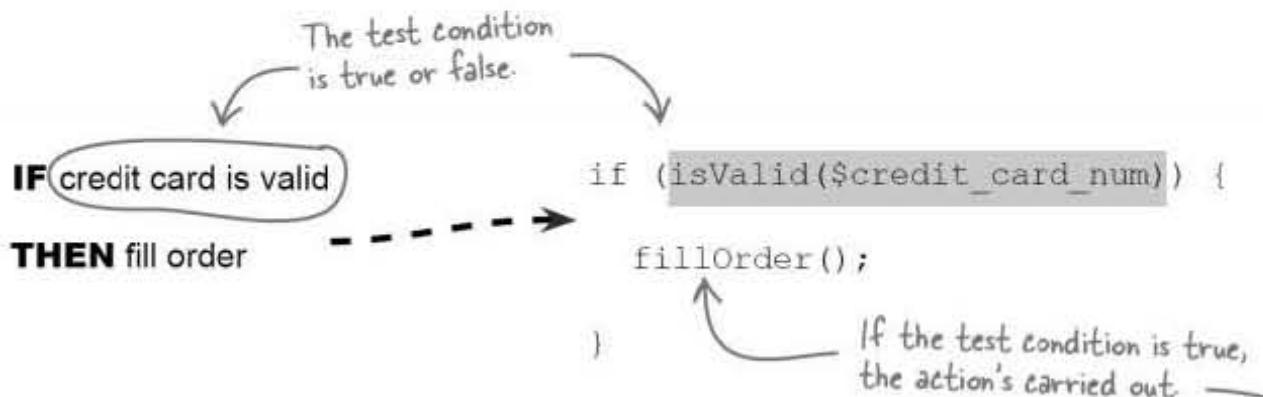
This ends the action and the if statement.

This is the action—what PHP will execute if the condition is true. You can have as many lines of code here as you wish.

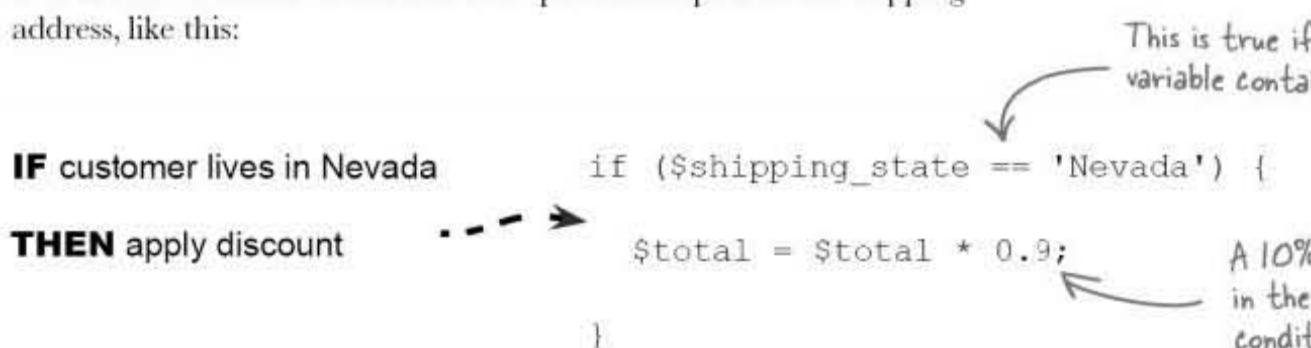
This hypothetical function returns true or false depending on the validity of the credit card.

Testing for truth

The heart of the `if` statement is its test condition, which is always interpreted as either `true` or `false`. The test condition can be a variable, a function call, or a comparison of one thing to another, as a few examples. Elmer's credit card validation relies on a function call as the test condition, which means the value returned by the function is either `true` or `false`.



It's quite common to use a comparison as a test condition, which typically involves comparing a variable to some value. For example, maybe Elmer wants to give a discount to customers who live in Nevada. He could create an `if` statement that carries out a comparison on part of the shipping address, like this:



This test condition performs a comparison for equality, which involves two equal signs (`==`). Equality comparisons aren't just for variables and strings. You can compare variables to numbers, variables to variables, and even perform calculations.

You can check to see if what is stored in one variable is equal to what is stored in another.

`($num_items == 10)`

Don't put quotes around numeric values.

`($shipping_address == $billing_address)`

`(2 + 2 == 4)`

You can carry out math operations in a test condition.

comparing values in php

IF checks for more than just equality

An if statement can check for more than just equality. The test condition in your if statement can also check to see if a value is **greater than** another one. If it is, the result of the condition is true, and the action code is executed. Here are a few more tests you can use to control the decision of an if statement.

Start out with these two variables.

\$small_number = 2;
\$big_number = 98065;

Both of these are true.

There are two ways to check if things are **not equal**: `<>` and `!=`. These give you the opposite results of an `==` equality test.

if (`$small_number <> $big_number`)
if (`$small_number != $big_number`)

The **greater than** sign (`>`) checks to see if the value on the left is greater than the value on the right. If so, the condition is true, otherwise it's false.

if (`$small_number > $big_number`) {
This condition is true}

The **less than** sign (`<`) compares the value on the left to the value on the right. If the left value is smaller than the right, the condition is true.

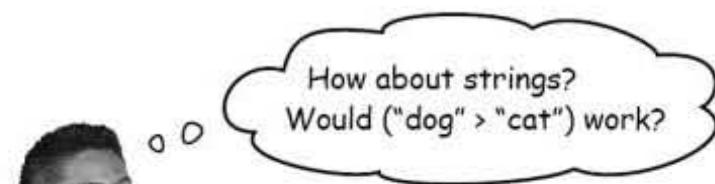
if (`$small_number < $big_number`) {
This condition is true}

Greater than or equal to (`>=`) is like greater than (`>`) except it also results in true if the two values are equal.

if (`$small_number >= $big_number`) {
This condition is true}

Less than or equal to (`<=`) is similar to less than, except it's also true if the values are equal.

if (`$small_number <= $big_number`) {
This condition is true}



Yes, you can compare strings in if

They work based on the alphabet, with *a* being less than (less than) *z*. Using greater than and less than, you need to present information in alphabetical order.



BE the test condition in the if statement

Your job is to play the if test condition and decide if you are true or false given the following variables.

```
$my_name = 'Buster';
$a_number = 3;
$a_decimal = 4.6;
$favorite_song = 'Trouble';
$another_number = 0;
$your_name = $my_name;
```

<code>(\$a_number == 3)</code>	true or false
<code>(\$another_number == "")</code>	true or false
<code>(\$favorite_song == "Trouble")</code>	true or false
<code>(\$my_name == '\$your_name')</code>	true or false
<code>(\$my_name == "\$your_name")</code>	true or false
<code>(\$your_name == \$my_name)</code>	true or false
<code>(\$favorite_song == 'Trouble')</code>	true or false
<code>(\$a_number > 9)</code>	true or false
<code>(\$favorite_food = 'hamburger')</code>	true or false

be the test condition solution



BE the test condition in the if statement

Your job is to play the if test condition and decide if you are true or false given the following variables.

```
$my_name = 'Buster';
$a_number = 3;
$a_decimal = 4.6;
$favorite_song = 'Trouble';
$another_number = 0;
$your_name = $my_name;
```

`($a_number == 3)`

true or false

O and an evaluate a

`($another_number == "")`

true or false

Because the if the the

`($favorite_song == "Trouble")`

true or false

if the the the

`($my_name == '$your_name')`

true or false

the the the

`($my_name == "$your_name")`

true or false

the the the

`($your_name == $my_name)`

true or false

the the the

`($favorite_song == 'Trouble')`

true or false

the the the

`($a_number > 9)`

true or false

OK, the only or it's ac

`($favorite_food == "hamburger")`

true or false

not a it end anythi

Should be == if we intended this to be a comparison.

there are no
Dumb Questions

Q: Okay, is a test condition the same thing we used to control while loops in Chapter 3?

A: It's exactly the same. And even though we used it to tell us when we had remaining rows of query data back in Chapter 3, we can devise more interesting test conditions for while loops by using different kinds of comparisons. You'll see that later in the book.

The logic behind Send Email validation

Elmer needs to **validate** the data he gets from the sendemail.html form before he sends any emails. In fact, sending the emails should completely hinge on the data validation. What we really need PHP to do is **make a decision** based on the validity of the form data received by the sendemail.php script. We need code that says, "**if** the data is **valid**, go ahead and send the emails."

But first we need to grab the form data and store it in a couple of variables:

```
$subject = $_POST['subject'];
$text = $_POST['elvismail'];
```

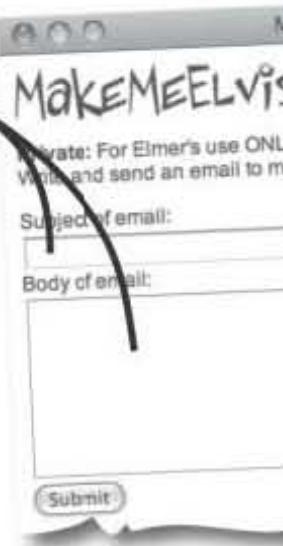
This form data is all we need to check and see if there is data in each of the form fields. The logic might look something like this:

```
IF $subject contains text AND $body contains text
THEN send email
```

Or we could take the opposite approach and check to see if the form fields are both empty, in which case we could display a warning to the user:

```
IF $subject is empty AND $body is empty
THEN echo error message
```

Both of these examples have a problem in that their logic requires us to make two comparisons in a single **if** statement. One possible solution is to use two **if** statements...



Sharpen your pencil

Write two **if** statements that check to see if both the subject and body of Elmer's Send Email form are empty. Echo a warning if they're empty.

`isset()` and `empty()` functions



our pencil
Solution

Write two `if` statements that check to see if both the subject and body of Elmer's Send Email form are empty. Echo a warning message if they're empty.

```
if (${subject == ''}) {  
    echo 'You forgot the email subject and body text.<br />';  
}  
}  
} ←
```

By nesting the second if statement inside of the first one, the code is saying that both must be true in order for the echo statement to run.

Indentation helps to show where the inner if statement ends, and where the outer if statement ends.

PHP functions for verifying variables

Using `==` to check for an empty string works, but there's a better way that involves built-in PHP functions. The `isset()` function tests to see if a variable exists, which means that it's been assigned a value. The `empty()` function takes things one step further and determines whether a variable contains an **empty value**, which PHP defines as 0, an empty string ('' or ""), or the values `false` or `NULL`. So `isset()` only returns `true` if a variable has been assigned a value, while `empty()` only returns `true` if a variable has been set to 0, an empty string, `false`, or `NULL`.

Let's take a look at how these functions work:

```
$v1 contains a value.          Both $v1 and $v2 are considered to be
                                set, even though only $v1 has a value.
                                } ←
$V2 is an empty string.      $v1 = 'aloha';
                            $v2 = '';
if (isset($v1)) { echo '$v1 is set<br />'; }
if ($v2) { echo '$v2 is set<br />'; }
if (empty($v1)) { echo '$v1 is empty<br />'; }
if (isset($v2)) { echo '$v2 is set<br />'; }
if (empty($v2)) { echo '$v2 is empty<br />'; }
if (isset($v3)) { echo '$v3 is set<br />'; }
if ($v3) { echo '$v3 is set<br />'; }
if (empty($v3)) { echo '$v3 is empty<br />'; }
```



I get it. We can use `isset()` and `empty()` to validate the `$subject` and `$text` form data.

That's half right. We're really just checking to see if the form data isn't empty, so `empty()` is what we want.

The `$subject` and `$text` variables are assigned values from `$_POST['subject']` and `$_POST['elvismail']` superglobal arrays. If you test these variables with `isset()`, it will always return `true`, regardless of whether or not they hold any actual text. In other words, `isset()` doesn't show you the difference between a blank form field and an empty variable. The `empty()` function checks to see if a variable is actually empty, which is what we need for form validation.

*there are no
Dumb Questions*

`isset()` checks that a variable exists and is set.

`empty()` checks to see if a variable has any contents.



Q: So what's the point of using `isset()`?

A: The `isset()` function is extremely useful when you need to know if a piece of data **exists**. For example, if a form has been submitted via a POST request, the `isset()` function `$_POST`. This ends up being a very handy technique, as you find out a little later in the chapter.

Rewrite the two `if` statements that check to see if both the `$subject` and `$text` message body of Elmer's Send Email form are empty, but use the `empty()` function instead of `==` in the test conditions.

.....
.....
.....
.....

the ! operator

Sharpen your pencil Solution

Rewrite the two `if` statements that check to see if both message body of Elmer's Send Email form are empty, but `empty()` function instead of `==` in the test conditions.

A call to the `empty()` function replaces the equality operator (`==`) in each of the `if` test conditions.

```
if (empty($subject)) {  
    if (empty($text)) {  
        echo 'You forgot the email subject and body text.<br />';  
    }  
}  
} The rest of the code is  
..... the same as before.
```



What if we need to only take a certain action if a form field is **not empty**? Is there a `notempty()` function?

No, but there's an easy way to reverse the logic condition... the negation operator.

We know the test condition that controls an `if` statement always results in either true or false. But what if our logic dictates that we need to do the reverse of what a condition gives us? For example, it would be helpful if Elmer's form fields are **not empty** before sending a bunch of emails with his data. Problem is, there is no `notempty()` function. The solution is the negation operator (`!`), which turns true into false, or false into true. It literally calls the `empty()` function and reverses its result, like this:

The NOT operator (!)
turns true into false,
or false into true.

```
if (!empty($subject)) {
```

...
}

This condition asks
"Is the \$subject field not empty?"
Is there data in it?



Fill in the blanks in Elmer's sendemail.php code so that email only goes out when both \$subject and \$text are **not** empty. Use if statements and empty() function.

All my fields need
to have values.

Make Me Elvis - Send Email

MAKEELVIS.COM

Private: For Elmer's use ONLY.
Write and send an email to mailing list members.

Subject of email:

Body of email:

Submit

```
<?php
    $from = 'elmer@makemeelvis.com';
    $subject = $_POST['subject'];
    $text = $_POST['elvismail'];

    if......
        if......
            $dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking');
            or die('Error connecting to MySQL server.');

            $query = "SELECT * FROM email_list";
            $result = mysqli_query($dbc, $query)
            or die('Error querying database.');

            while ($row = mysqli_fetch_array($result)) {
                $to = $row['email'];
                $first_name = $row['first_name'];
                $last_name = $row['last_name'];
                $msg = "Dear $first_name $last_name,\n$text";
                mail($to, $subject, $msg, 'From:' . $from);
                echo 'Email sent to ' . $to . '<br />';
            }
            mysqli_close($dbc);

.....
.....
?>
```

sendemail.php—now with validation!



Fill in the blanks in Elmer's sendemail.php code so that email only goes out when both \$subject and \$text are **not** empty. Use if statements and empty() function.

All my fields need to have values.

Make Me Elvis - Se

MAKEMEELVIS.COM

Private: For Elmer's use ONLY.
Write and send an email to mailing list members.

Subject of email:

Body of email:

Submit

send

The exclamation point reverses the logic of the empty() function.

The first condition checks to see if \$subject is not empty...

...if not, good! Now we check to see if \$text is not empty.

We had to put one if statement inside the other one to make this work. This is called nesting.

```
<?php
    $from = 'elmer@makemeelvis.com';
    $subject = $_POST['subject'];
    $text = $_POST['elvismail'];

    if (!empty($subject)) {
        if (!empty($text)) {
            $dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking');
            or die('Error connecting to MySQL server.');

            $query = "SELECT * FROM email_list";
            $result = mysqli_query($dbc, $query)
            or die('Error querying database.');

            while ($row = mysqli_fetch_array($result)) {
                $to = $row['email'];
                $first_name = $row['first_name'];
                $last_name = $row['last_name'];
                $msg = "Dear $first_name $last_name,\n$text";
                mail($to, $subject, $msg, 'From:' . $from);
                echo 'Email sent to ' . $to . '<br />';
            }
            mysqli_close($dbc);
        }
    }
?>
```

We have to close off the action part of both of the if statements. The first brace ends the inner if statement, and the second brace ends the outer if statement.

If either form data is empty, one of the conditions will be false, and the code here will run, which is exactly what we wanted!



Test Drive

See if the empty form field validation works.

Modify the code in sendemail.php to use if statements that check the form field data before sending email messages. Upload the new version of the script to your web server and open the sendemail.html page in a web browser. Make sure to leave at least one of the form fields blank, and click Submit.

The body of the message is empty, which makes the form data fail validation.

Make Me Elvis - Send Email

MakEMEElvis.COM

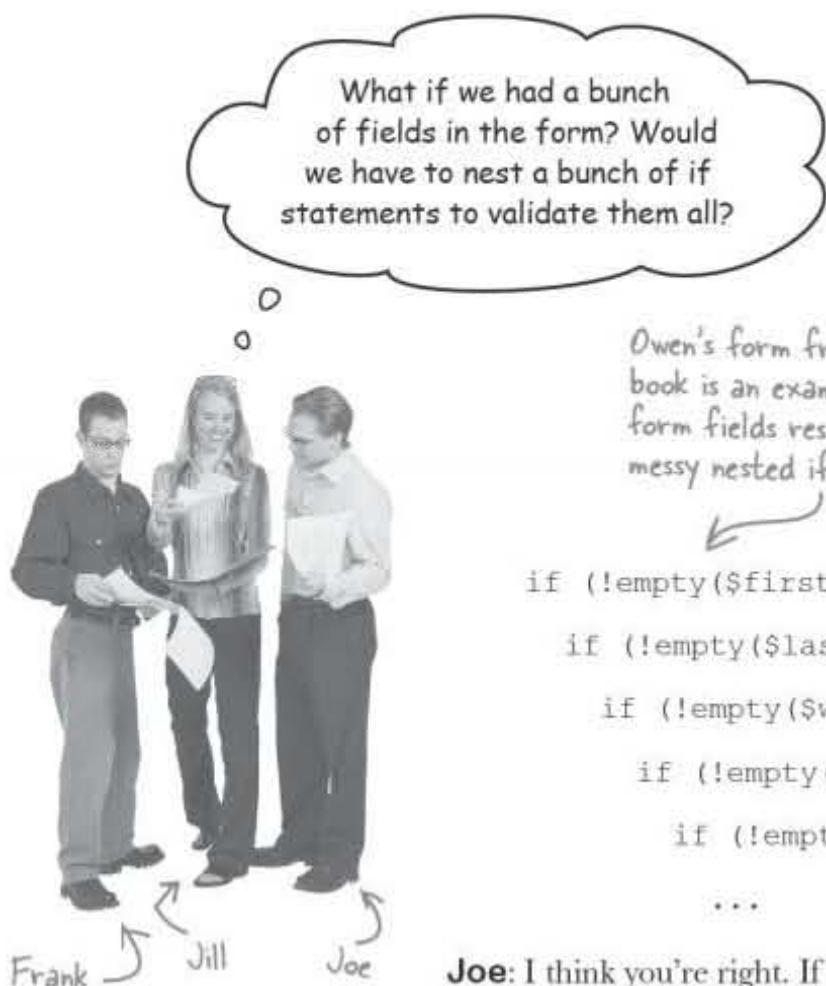
Private: For Elmer's use ONLY
Write and send an email to mailing list members.

Subject of email:
Blue Suede Clearance

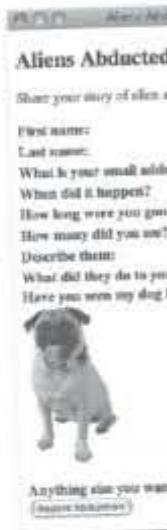
Body of email:

Submit

No email confirmations reveal that nothing was sent, which is what we want. But some kind of warning message would be more helpful than a blank page.

cubicle conversation

Owen's form from earlier in the book is an example of how more form fields result in a bunch of messy nested if statements.



Joe: I think you're right. If we want to make sure all those fields have to nest an if statement for each field.

Frank: As long as we indent each line of code for each if statement, aren't we OK?

Jill: Technically, yes. I mean, the code will certainly work no matter how many if's we nest, but getting hard to understand with so much nesting. Just matching up curly braces accurately could...

Frank: That's true. I think it'd also be a pain having to indent the action code so far... let's see... giving us ten nested ifs with ten levels of indentation. Even if we just indent each if two spaces before every line of action code. Yuck.

Joe: But if we indent with tabs, it cuts that in half—10 tabs versus 20 spaces isn't so bad.

Jill: Guys, the issue isn't really about the specific code used to indent the nested if's. It's just not good practice to nest if statements so deep. Think about it like this—we're really talking about one question: "are all our form fields non-empty?" The problem is, that test condition involves ten different pieces of code for us to have to break it into ten separate if statements.

Frank: Ah, I see. So what we need is a way to test all ten pieces of form data in a **single** test condition.

Jill: Yup.

Joe: Then we could write one big test condition that checks all the form fields at once. Awesome!

Jill: Yeah, but we're still missing the piece of the puzzle that lets us combine multiple comparisons in a single test condition...

Test multiple conditions with AND and OR

You can build a test condition for an `if` statement with multiple checks by connecting them with a **logical operator**. Let's look at how it works with two familiar conditions, `!empty($subject)` and `!empty($text)`. This first example involves two expressions joined by the logical AND operator, which is coded using `&&`.

```
if ((!empty($subject)) && (!empty($text))) {
```

The logical AND operator.

The extra parentheses help make it clear that the negation operator applies only to the `empty()` function.

This test condition is only true if both `$subject` AND `$text` are not empty.

The AND operator takes two `true/false` values and gives you `true` only if they are both `true`; otherwise the result is `false`. So in this case both form fields must be non-empty in order for the test condition to be `true` and the action code for the `if` statement to run.

The logical OR operator, coded as `||`, is similar to AND except that it results in `true` if either of the `true/false` values is `true`. Here's an example:

```
if ((!empty($subject)) || (!empty($text))) {
```

This test condition is true if either `$subject` OR `$text` are not empty.

So the action code for this `if` statement is executed if either one of the form fields is not empty. Things get even more interesting if you want to isolate one form field as being empty but the other having data, like this:

```
if (empty($subject) && (!empty($text))) {
```

`$subject` must be empty
`$text` must be non-empty
this test condition

Since this test condition uses AND, both expressions inside of the test condition must be `true` in order for the action code to be run. This means the Subject form field must be empty, but the Body field must have data. You can reverse this check by moving the `negation operator (!)` to the other `empty()` function:

```
if ((!empty($subject)) && empty($text)) {
```

This is true only if `$subject` isn't empty but `$text` is.

The AND (`&&`) and OR (`||`) logical operators make it possible to structure much more powerful test conditions that would otherwise require additional, often messy, `if` statements.

eliminate the nested if statements



Rewrite the highlighted sections of the `sendemail.php` script so that logical operators in a single `if` test condition instead of nested `if` statements.

```
<?php
$from = 'elmer@makemeelvis.com';
$subject = $_POST['subject'];
$text = $_POST['elvismail'];

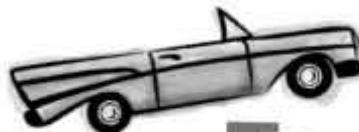
if (!empty($subject)) {
    if (!empty($text)) {
        .....
        .....
        $dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking'
            or die('Error connecting to MySQL server.');

        $query = "SELECT * FROM email_list";
        $result = mysqli_query($dbc, $query)
            or die('Error querying database.');

        while ($row = mysqli_fetch_array($result)) {
            $to = $row['email'];
            $first_name = $row['first_name'];
            $last_name = $row['last_name'];
            $msg = "Dear $first_name $last_name,\n$text";
            mail($to, $subject, $msg, 'From:' . $from);
            echo 'Email sent to ' . $to . '<br />';
        }
    }
    mysqli_close($dbc);
}
?>
```

Here are our nested if statements.
Rewrite them using a single if statement with logical operators.

These braces close the two if statements.



— Test Drive —

Make sure the logical operators in the Send Email script do the same job as the nested if statements.

Modify the code in `sendemail.php` to use a single `if` statement that takes advantage of logical operators to check the form field data before sending email messages. Double-check the exercise solution on the following page if you aren't sure about the changes to make.

Upload the new version of the script to your web server and open the `sendemail.html` page in a web browser. Make sure to leave at least one of the form fields blank, and click Submit. Does the script still prevent the email messages from being sent when a form field is blank?

*there are no
Dumb Questions*

Q: Does it matter what order you put two conditions joined by `&&` or `||` in an `if` statement?

A: Yes. The reason is because these two operators are **short-circuited** whenever possible. What this means is that if the first operand is enough to determine the outcome of the expression, the second operand is ignored. As an example, if the first operand in an AND expression is `false`, this is enough to cause the expression to be `false` regardless of the second operand, so the second operand is ignored. The same rule applies when the first operand in an OR expression is `true`.

Q: I've seen PHP code that uses `and` instead of `&&` and `or` instead of `||`. How do those differ?

A: They're virtually the same as `&&` and `||`. There's a slight difference in how they're relative to other operators, but if you use parentheses to make your test conditionals, there's essentially no difference.

sendemail.php—now without nested if statements!



Exercise Solution

Rewrite the highlighted sections of the sendemail.php script so that logical operators in a single if test condition instead of nested if statements.

```
<?php
$from = 'elmer@makemeelvis.com';
$subject = $_POST['subject'];
$text = $_POST['elvismail'];

if (!empty($subject)) {
    if (!empty($text)) {
        if (!!empty($subject) && !!empty($text))) {
```

The negation, or NOT operator (!), is used to check for non-empty form fields.

We can use AND to check for both conditions in one if statement.

Remember, && is how you actually specify the AND logical operator.

```
        $dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking'
            or die('Error connecting to MySQL server.');

        $query = "SELECT * FROM email_list";
        $result = mysqli_query($dbc, $query)
            or die('Error querying database.');

        while ($row = mysqli_fetch_array($result)) {
            $to = $row['email'];
            $first_name = $row['first_name'];
            $last_name = $row['last_name'];
            $msg = "Dear $first_name $last_name,\n$text";
            mail($to, $subject, $msg, 'From:' . $from);
            echo 'Email sent to ' . $to . '<br />';
        }

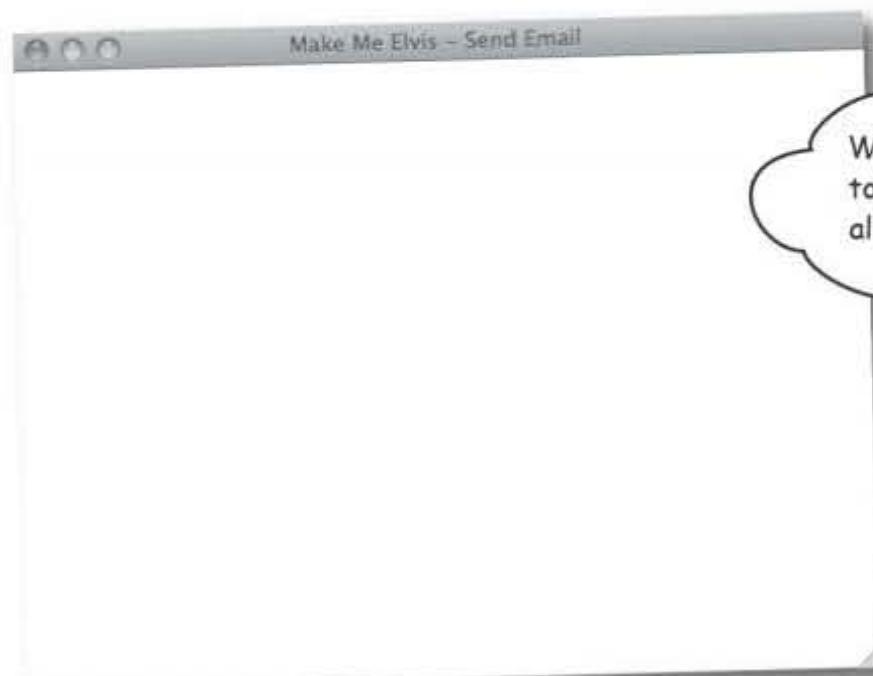
        mysqli_close($dbc);
    }
}
```

Our single if statement means we only need one closing curly brace.

```
?>
```

Form users need feedback

Our `sendemail.php` code does a great job of validating the form data so that no mail gets sent out if either the Subject or Body fields are left blank. But when the validation fails, and no emails are sent out, the script doesn't tell Elmer what happened. He just gets a blank web page.



The problem is that our code only reacts to a **successful** validation, in which case it sends the email messages. But if the `if` statement turns out being `false` (invalid form data), the code doesn't do anything, leaving Elmer in the dark about whether any emails were sent or what went wrong. Here's the abbreviated script code, which reveals the blank page problem:

```
<?php
$from = 'elmer@makemeelvis.com';
$subject = $_POST['subject'];
$text = $_POST['elvismail'];

if ((!empty($subject)) && (!empty($text))) {
    $dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking',
    ...
    mysqli_close($dbc);
}
?>
```

Nothing at all happens if the if statement fails to run the action code, which is why a blank page is generated when there is missing form data.

We need to let Elmer know that there was a problem, ideally telling him what form fields were blank so that he can try entering the message again.

the else clause

Not a problem. Just put an echo statement after the closing brace of the if statement.

That won't work because code after the if will always be executed.

Placing the echo statement after the if statement just runs it after the if statement, but it always runs regardless of the outcome. That's not what we need. We need the echo statement to only message **only** if the test condition of the if statement is true. To express our logic as this:

- IF** subject contains text **AND** body contains
- ✓ **THEN** send email
- ✗ **ELSE** echo error message

The if statement offers an optional else clause that runs when the test condition is false. So our error message ends up in an else clause, in which case it only gets run when one of the conditions is left empty. Just place the word else after the if statement and enclose the action code for it inside curly braces:

The else clause starts right after the closing curly brace for the if statement

```
if ((!empty($subject)) && (!empty($text))) {
    ...
}
else {
    echo 'You forgot the email subject and/or body.';
}
```

Just like the action code in an if, the code in an else is enclosed in curly braces.

This is a placeholder for the code that sends the email messages.

The code here only gets run if the if statement turns out false.

The else clause executes code when an if test condition is false.



Below is new code for Elmer's `sendemail.php` script that uses `if` statements to provide feedback, but some of the code has gotten misplaced. Use the `if` statements to replace the missing code.

exercise solution

Below is new code for Elmer's sendemail.php script that uses if statements to provide feedback, but some of the code has fallen off. Use the missing code.

```
<?php
    $from = 'elmer@makemeelvis.com';
    $subject = $_POST['subject'];
    $text = $_POST['elvismail'];

    if (empty($subject) && empty($text)) {
        // We know both $subject AND $text are blank
    }
    else {
        if (empty($subject) || empty($text)) {
            // We know we are missing $subject OR $text - let's find out which
            if (empty($subject)) {
                // $subject is empty
                echo 'You forgot the email subject.<br />';
            }
            else {
                // $text is empty
                echo 'You forgot the email body text.<br />';
            }
        }
        else {
            // Everything is fine, send email
            ...
            while ($row = mysqli_fetch_array($result)) {
                $to = $row['email'];
                $first_name = $row['first_name'];
                $last_name = $row['last_name'];
                $msg = "Dear $first_name $last_name,\n$text";
                mail($to, $subject, $msg, 'From:' . $from);
                echo 'Email sent to ' . $to . '<br />';
            }

            mysqli_close($dbc);
        }
    }
?>
```

The outer if statement checks if both \$subject and \$body are empty. If they are, it prints a message saying both are blank. If either one is empty, it prints a message saying the respective one is missing. Finally, if neither is empty, it prints a message saying everything is fine and sends the email.

At this point, the script has checked all possibilities through both form fields and database rows.



All those nested if's and else's are making the script hard to follow. I'd hate to ever have to work on that script! It needs to be simplified before someone gets hurt.

It's always a good idea to simplify code whenever possible, especially nested code that gets too deep.

Too many else clauses with nested if statements can make your code hard to follow. Maybe it wouldn't matter if we never had to look at it, but that's unlikely. If we ever needed to change the form and add another field, validating it would be trickier than it needed to be because it would be hard to read the code and figure out what changes need to go.

clean the if code

cleaner

BE the IF code

Your job is to play IF code and clean up the messy nested IF's and ELSE's. Rewrite the code to get rid of the nesting, but make sure it still works correctly.

Hint: You might not even need any elses!

```

if (empty($subject) && empty($text)) {
    echo 'You forgot the email subject and body text.<br />';
} else {
    if (empty($subject) || empty($text)) {
        if (empty($subject)) {
            echo 'You forgot the email subject.<br />';
        } else {
            echo 'You forgot the email body text.<br />';
        }
    } else {
        // Everything is fine. send the email
    }
}

```

Rew
that



— Test Drive —

Try out the cleaner if code to make sure it works as expected.

Modify the code in `sendemail.php` to use `if` statements similar to those you wrote that simplify the `if` nesting. Flip to the solution on the following page if you aren't sure about the changes to make.

Upload the new version of the script to your web server and open the `sendemail.html` page in a web browser. Experiment with the script by submitting the form with form fields both blank and filled. Does the script display error messages?

*there are no
Dumb Questions*

Q: Are a few levels of nesting really that big of a deal?

A: It depends. If you're writing some code that only you will ever see and you think you'll remember exactly what every next line does in six months time when you come back to it to tweak it, nest away.

If on the other hand, you'd like to keep your code as clean and logical as possible, you can use any of the several logic operators you've met so far.

Q: How does `else` work?

A: In an `if...else` statement, the `else` matches anything and everything that doesn't match the `if` part.

Q: Hmm. Okay. Does that mean I could nest `if` and `else` in existing `if...else` statements?

A: Well, you could, but with all that nesting things would get complex pretty fast and trying to avoid nesting here!

the cleaned-up if code

cleaner BE the IF code Solution

Your job is to play IF code and clean up the messy nested IF's and ELSE's. Rewrite the code to get rid of the nesting, but make sure it still works correctly.

```

if (empty($subject) && empty($text)) {
    echo 'You forgot the email subject and body text.<br />';
} else {
    if (empty($subject) || empty($text)) {
        if (empty($subject)) {
            echo 'You forgot the email subject.<br />';
        } else {
            echo 'You forgot the email body text.<br />';
        }
    } else {
        // Everything is fine. send the email
    }
}

```

Here, we're testing to see if both the \$subject and \$text variables are empty.

→ if (empty(\$subject) && empty(\$text)) {
 echo 'You forgot the email subject and body text.
';
}

Here we're testing to see if \$text is empty and \$subject is not empty.

→ if (empty(\$subject) && (!empty(\$text))) {
 echo 'You forgot the email subject.
';
}

This code checks to empty and \$text is

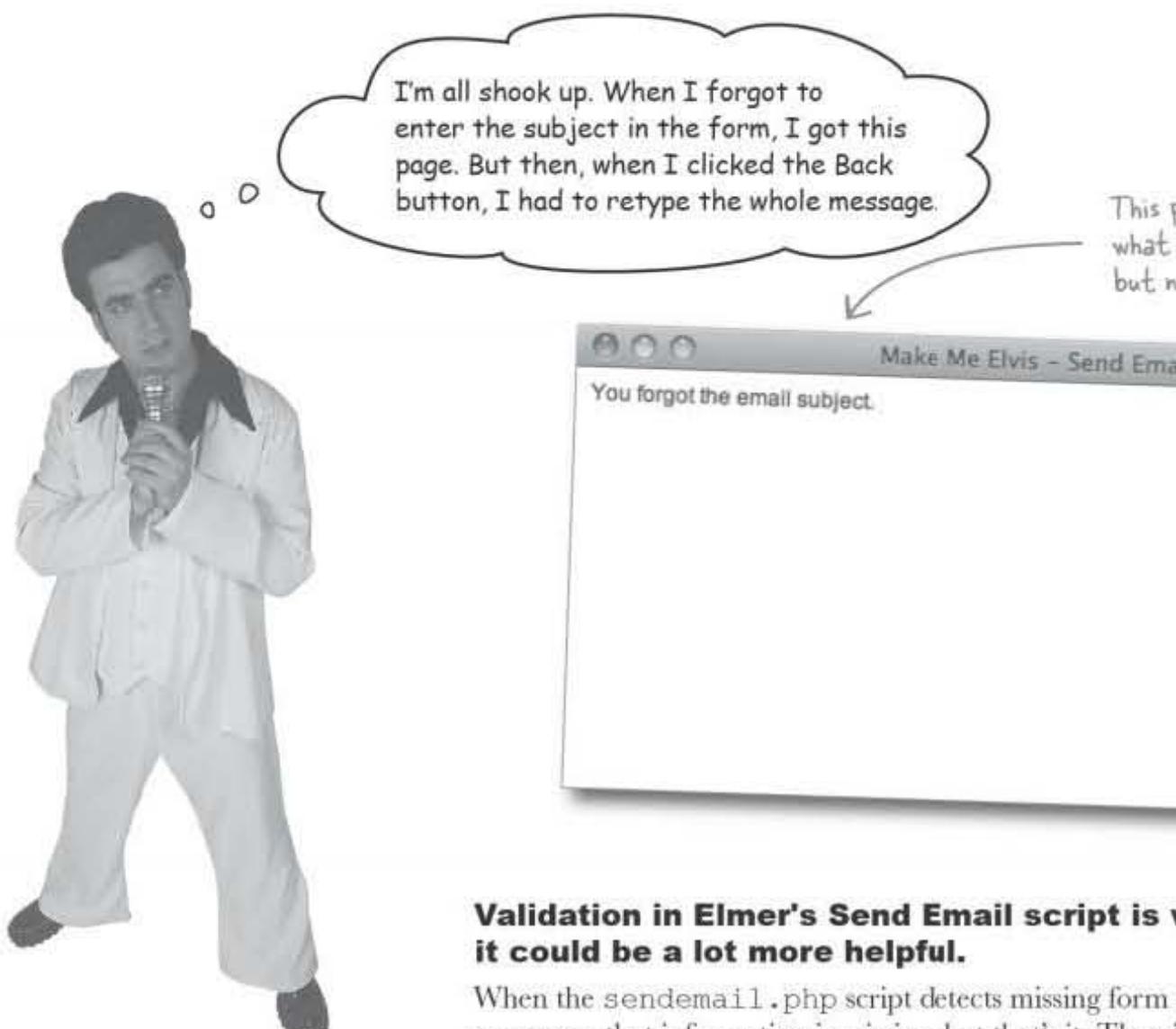
→ if ((!empty(\$subject)) && empty(\$text)) {
 echo 'You forgot the email body text.
';
}

If we didn't use the non-empty subject we could end up getting a message. Same thing if \$subject and empty \$text

And here, we're testing to see if neither \$subject nor \$text is empty.

→ if ((!empty(\$subject)) && (!empty(\$text))) {
 //...Everything is fine. send the email...
}

The NOT operator (!) is used because \$subject and \$text being



Validation in Elmer's Send Email script is v it could be a lot more helpful.

When the sendemail.php script detects missing form data, it displays a message that information is missing, but that's it. There's no link back to the original form, for example. And even worse, when he clicks the Back button, he has to go back to the original form and re-enter his message. He has to retype both the subject and body of his email message.



What would you do to improve the error handling of the Send Email script to make it more helpful?

regenerating the HTML form code



This indentation isn't strictly necessary, but it helps to see the structure of the original HTML code.

It would be cool to show the form along with the error message. Couldn't we just echo the form if the email subject and body text are empty?

Displaying the form would definitely be helpful, save Elmer having to navigate back in his browser.

So in addition to echoing an error message when one of the form fields is empty, we also need to regenerate the HTML form code from PHP by echoing it directly into the browser. This code shows that PHP is capable of generating some complex HTML code:

This PHP code generates the entire HTML form, starting with the `<form>` tag.

```

echo '<form method="post" action="sendemail.php">';
echo '  <label for="subject">Subject of email:<br />';
echo '  <input id="subject" name="subject" type="text" size="30" /><br />';
echo '  <label for="elvismail">Body of email:<br />';
echo '  <textarea id="elvismail" name="elvismail" cols="40"></textarea><br />';
echo '  <input type="submit" name="submit" value="Send Email" />';
echo '</form>';

```

Since HTML code is enclosed in double quotes, it's better to use single quotes to enclose strings of HTML code.

If you're thinking this code looks a bit chaotic, that's because it is. The fact that you **can** do something in PHP doesn't mean you should. In this case, the complexity of echoing all that HTML code is a problem. This is a good example of code that generating it via PHP with `echo` is really not a good idea.

Ease in and out of PHP as needed

It's sometimes easy to forget that a PHP script is really just an HTML web page that is capable of holding PHP code. Any code in a PHP script that isn't enclosed by the <?php and ?> tags is assumed to be HTML. This means you can close a block of PHP code and revert to HTML as needed, and then pick back up with a new block of PHP code. This is an extremely handy technique for outputting a chunk of HTML code that is unwieldy to generate through PHP echo statements... like our Send Email form code.

```

<?php
    $from = 'elmer@makemeelvis.com';
    $subject = $_POST['subject'];
    $text = $_POST['elvismail'];

This ?> tag
closes the PHP
block, returning
us to HTML.
    →?>

        if (empty($subject) && empty($text)) {
            // We know both $subject AND $text are blank
            echo 'You forgot the email subject and body text.<br />';

<form method="post" action="sendemail.php">
    <label for="subject">Subject of email:</label><br />
    <input id="subject" name="subject" type="text" size="30">
    <label for="elvismail">Body of email:</label><br />
    <textarea id="elvismail" name="elvismail" rows="8" cols="30">
    <input type="submit" name="submit" value="Submit" />
</form>

<?php
    } ← The <?php tag starts a new PHP block. Since
        we're still inside the if action, we have to
        close the if statement before continuing.
    if (empty($subject) && (!empty($text))) {
        echo 'You forgot the email subject.<br />';
    }

    if ((!empty($subject)) && empty($text)) {
        echo 'You forgot the email body text.<br />';
    }

    if ((!empty($subject)) && (!empty($text))) {
        // Code to send the email
        ...
    }
?>

```

Write down anything you think might be wrong with this code. How would you fix it?

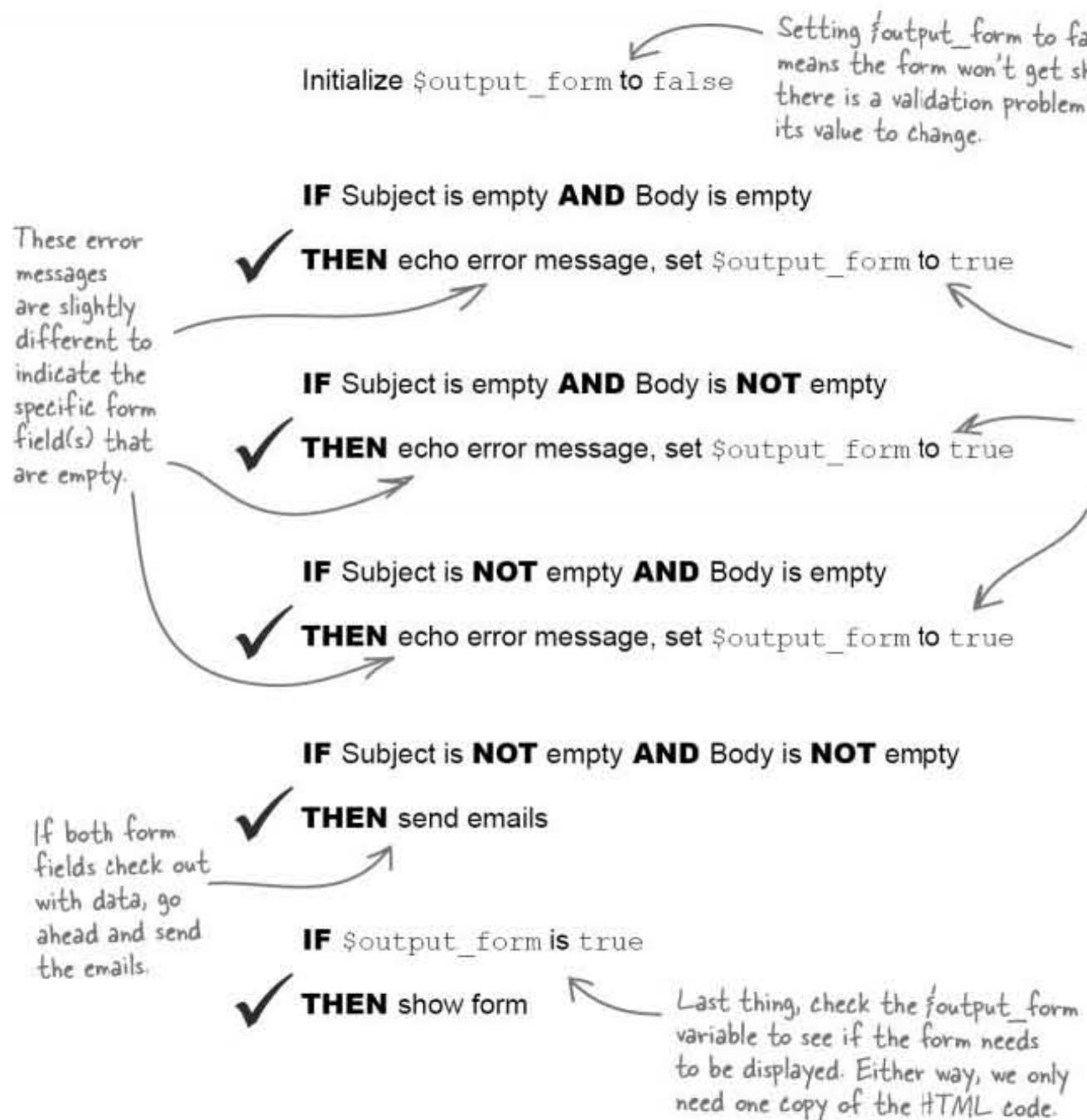
.....
.....
.....

avoiding duplicate code with a flag

Use a flag to avoid ~~duplicate~~ duplicate code

The problem with the previous code is that it will have to drop out of PHP and duplicate the form code in three different places (once for each validation error). We can use a true/false variable known as a **flag** to keep track of whether or not we need to output the form. Let's call it \$output_form. Then we can check the variable **later in the code** and display the form if the variable is true.

So we need to start out the script with \$output_form set to false, and then only change it to true if a form field is empty and we need to show the form:



Code the HTML form only once

Turning the new validation logic into PHP code involves creating and initializing the new `$output_form` variable, and then making sure to set it throughout the validation code. Most important is the new `if` statement at the end of the code that only displays the form if `$output_form` is set to true.

```
<?php
    $from = 'elmer@makemeelvis.com';
    $subject = $_POST['subject'];
    $text = $_POST['elvismail'];
    $output_form = false; We create our new variable here and set it to false initially.

    if (empty($subject) && empty($text)) {
        // We know both $subject AND $text are blank
        echo 'You forgot the email subject and body text.<br />';
        $output_form = true; Set the variable to true if $subject and $text are empty so that the form is shown.
    }

    if (empty($subject) && (!empty($text))) {
        echo 'You forgot the email subject.<br />';
        $output_form = true; Also set the variable to true if $subject is empty.
    }

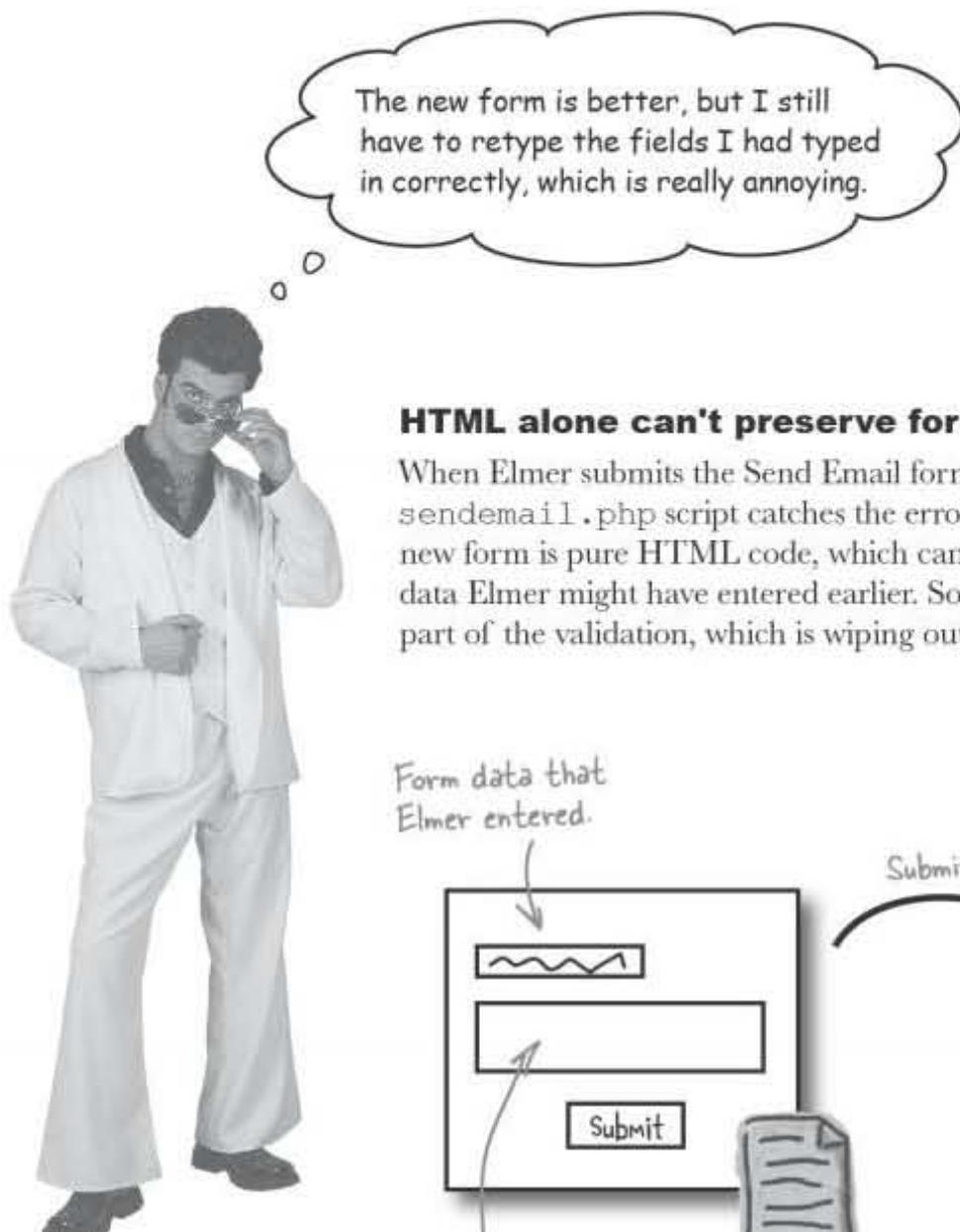
    if ((!empty($subject)) && empty($text)) {
        echo 'You forgot the email body text.<br />';
        $output_form = true; And set the variable to true if $text is empty.
    }

    if ((!empty($subject)) && (!empty($text))) {
        // Code to send the email
        ...
    }
}

if ($output_form) { This if statement checks the $output_form variable and displays the form if it is true.
    <form method="post" action="sendemail.php">
        <label for="subject">Subject of email:</label><br />
        <input id="subject" name="subject" type="text" size="30" /><br />
        <label for="elvismail">Body of email:</label><br />
        <textarea id="elvismail" name="elvismail" rows="8" cols="40"></textarea>
        <input type="submit" name="submit" value="Submit" />
    </form>
}

<?php
    ?> Don't forget to jump back into PHP code and close the if statement. The HTML code on we've crunched all this into a single variable.
```

the form data still disappears



HTML alone can't preserve form data.

When Elmer submits the Send Email form with an empty field, the sendemail.php script catches the error and generates a new form. The new form is pure HTML code, which can't possibly know anything about the data Elmer might have entered earlier. So we're generating a clean part of the validation, which is wiping out any data Elmer might have entered.

Elmer accidentally
left this field blank.

sendemail.html

All the fields
are now empty
because this is a
shiny new form.

Ack. We can't get around the fact that a new form will have to be generated by the PHP script. But we need a way to remember any data Elmer might have already entered, and plug it back into the new form so that Elmer can focus solely on filling out the form field that he accidentally left empty...



Sharpen your pencil

Draw what Elmer's form should look like after he submits his first form field filled out. Then write down how you think the files (HTML and PHP) should be altered to carry out this request.

A rectangular form containing two horizontal input fields stacked vertically, followed by a single "Submit" button at the bottom.



sendemail.php



sendemail.htm

.....
.....
.....
.....
.....
.....
.....

.....
.....
.....
.....
.....
.....
.....

make the form "sticky"



Sharpen your pencil

Draw what Elmer's form should look like after he submits his first form field filled out. Then write down how you think the files (HTML and PHP) should be altered to carry out this request.

The error message
is still displayed...

This field is still blank
because Elmer never
entered anything into it.

... but the
remember
Elmer ent
it back in



The PHP script takes over the job of displaying....
the form, both before and after submission. And....
since the script has access to any form data that...
has been entered, it can plug the data back into....
the form when it is generated. This solves Elmer's....
problem of having to re-enter form data he's....
already filled out.

If we display the form entirely....
we can do away with the HTML....
the PHP script both show the....
In doing so, the PHP script can....
data entered into the form, wh....
pure HTML code.

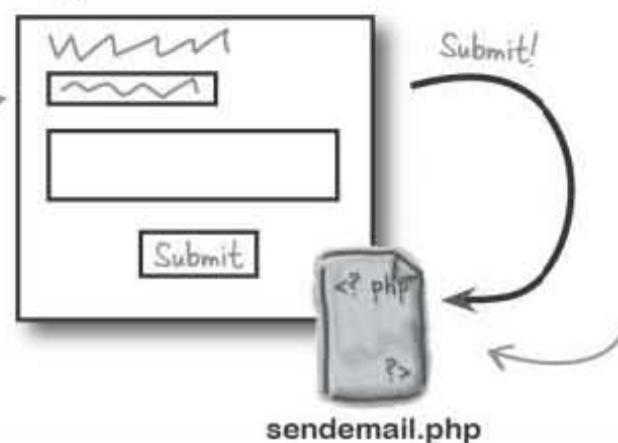
A form that references itself

How can it be possible to remove `sendemail.html` from the Send Email form equation? The answer is that we're not actually eliminating any HTML code, we're just moving it to the PHP script. This is made possible by the fact that a PHP script can contain HTML code just like a normal web page. So we can structure our script so that it not only processes the form on submission but also displays the form initially, which is all `sendemail.html` was doing.

The key to the `sendemail.php` script being able to fill the role left by `sendemail.html` is the form action. Since the script itself now contains the HTML form, the form action leads back to the script... a **self-referencing form**.

We no longer need `sendemail.html`—users navigate directly to the PHP script to use the form.

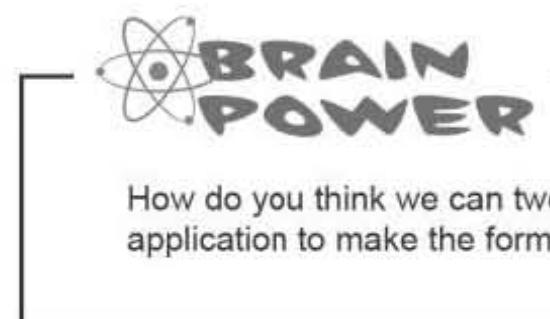
The form data's submitted to the same script, which processes it and displays the form again, but this time it remembers data already entered.



The script initially displays the form and then processes it when submitted. Processing involves either displaying the data or an error message.

To understand what's going on here, think about the first time Elmer visits the page (script). An empty form is generated as HTML code and displayed. Elmer fills out a field of the form and clicks Submit. The script processes its own form, and displays an error message if any data's missing. More importantly, the script displays the form again, but this time it includes any data Elmer has already entered. When a form's smart enough to remember data entered into it in prior submissions, it's known as a **sticky form**... the data sticks to it!

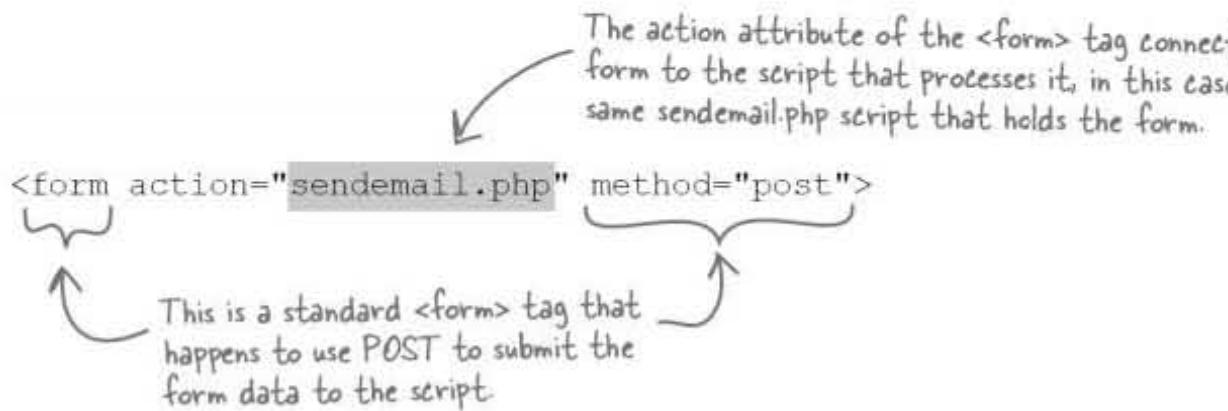
**Sticky forms
remember the data
the user has already
correctly entered.**



make the form action self-referencing

Point the form action at the script

As we've seen several times, the `action` attribute of the `<form>` tag is what connects a form to a PHP script that processes it. Setting the `action` of Elmer's form to `sendemail.php` works just fine in allowing it to process itself, which is the first step toward form stickiness. In fact, the form already has its `action` attribute set to the script:



This code works, assuming you don't ever rename the script and forget to update the code. But there's a better way that works no matter what because it doesn't rely on a specific script filename. It's the built-in PHP superglobal variable `$_SERVER['PHP_SELF']`, which stores the name of the current script. You can replace the script URL in the form action to `$_SERVER['PHP_SELF']`, and not ever have to worry about updating anything if you ever need to rename the script.

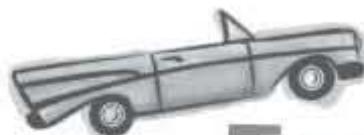
The only catch is that `$_SERVER['PHP_SELF']` is PHP code, which means you have to echo its value so that it is output as part of the HTML code, like this:

```
<form action="php echo $_SERVER['PHP_SELF']; ?" method="po
```

Instead of hardcoding the name of the script, we can tell it to reference itself by echoing `$_SERVER['PHP_SELF']` superglobal

Granted, using `$_SERVER['PHP_SELF']` instead of the script name isn't an earth shattering improvement but it's one of the many little things you can do to make your scripts easier to maintain over time.

`$_SERVER['PHP_SELF']`
stores away the name of the current script



Test Drive

Try out the new self-referencing script with improved form validation logic.

Modify the code in `sendemail.php` to use the `$output_form` variable to selectively display the form as shown a few pages back. Also change the `action` attribute of the `<form>` tag so that the form is self-referencing.

You no longer need the `sendemail.html` page on your web server, so feel free to delete it. Then upload the new version of the `sendemail.php` script to your web server and open the script in a web browser. How does it look?

For some reason the script's showing an error message even though the form hasn't even been submitted... not good.

Not only that but it still isn't sticky. We still have some work to do!

Make Me Elvis - Send Email

MAKEMEELVIS.COM

Private: For Elmer's use ONLY
Write and send an email to mailing list members.

You forgot the email subject and body text.
Subject of email:

Body of email:

Submit

First things first – the sticky stuff in a

Write down why you think the script is showing an error message the first time the form is shown.

.....
.....
.....

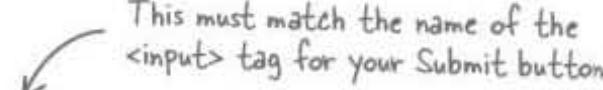
`check for $_POST['submit']`

Check to see if the form has been submitted

The problem is that the script can't distinguish between the form being displayed for the first time and it being submitted with incomplete data. So the script reports missing data the very first time the form is displayed, which is confusing. The question is, how can we check to see if the form is being submitted? If we know that, we can make sure we only validate data on a submission.

Remember how, when a form is submitted using the POST method, its data is stored away in the `$_POST` array? If the form hasn't been submitted, then the `$_POST` array isn't filled with any data. Or to put it another way, the `$_POST` array **hasn't been set**. Any guess what function we could call to see if the `$_POST` array's been set?

The `isset()` function checks to see if a variable has been set



```

if (isset($_POST['submit'])) {
    ...
}

```

Any code in here will only get executed if the form's been submitted.

Since every form has a Submit button, an easy way to check to see if a form has been submitted is to see if there's `$_POST` data for the Submit button. The data's just the label on the button, which isn't important. What's important is simply the existence of `$_POST['submit']`, which tells us that the form has been submitted. Just make sure that 'submit' matches up with the `id` attribute of the Submit button in the form code.

there are no
Dumb Questions

Q: How does knowing if the form was submitted stop us from accidentally displaying validation error messages?

A: The reason the error messages are being shown incorrectly is because the script doesn't distinguish between the form being submitted vs. being displayed for the first time. So we need a way to tell if this is the first time the form is being shown, in which case empty form fields are perfectly fine—it's not an error. We should *only* validate the form fields if the form's been submitted, so being able to detect a form submission is very important.

Q: So why don't we check to see if instead of the Submit button?

A: It would work perfectly fine to check `$_POST['subject']` or `$_POST['name']` but only for this particular form. Since every button that can be consistently named `$_POST['submit']` gives you a reliable submission in all of our scripts.



The Send Email Script Up Close

```

<?php
    if (isset($_POST['submit'])) (
        $from = 'elmer@makemeelvis.com';
        $subject = $_POST['subject'];
        $text = $_POST['elvismail'];
        $output_form = false;

        if (empty($subject) && empty($text)) {
            // We know both $subject AND $text are blank
            echo 'You forgot the email subject and body text.<br />';
            $output_form = true;
        }

        if (empty($subject) && (!empty($text))) {
            echo 'You forgot the email subject.<br />';
            $output_form = true;
        }

        if ((!empty($subject)) && empty($text)) {
            echo 'You forgot the email body text.<br />';
            $output_form = true;
        }

        if ((!empty($subject)) && (!empty($text))) {
            // Code to send the email
            ...
        }
    } else {
        $output_form = true;
    }

    if ($output_form) {
        ?>

        <form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
            <label for="subject">Subject of email:</label><br />
            <input id="subject" name="subject" type="text" size="30" /><br />
            <label for="elvismail">Body of email:</label><br />
            <textarea id="elvismail" name="elvismail" rows="8" cols="40"></textarea>
            <input type="submit" name="submit" value="Submit" />
        </form>

        <?php
    }
?>

```

We check the value of `$_POST`. If the form has never been submitted, this will be unset.

This parenthesis closes the first if, which tells us if the form was submitted.

If the form's never been submitted, we definitely need to show it!

make the form fields "sticky"



Cool. So we can now detect the form submission and show error messages correctly. But we still haven't made the form fields sticky, right?

That's right. Detecting the form submission is we still need to plug the sticky form data back

Knowing if the form's been submitted is an important part of making a form work, but it isn't the only part. The part we're missing is taking any form data that was submitted and plugging it back into the form as the form is being output. We do this by setting the value of a form field using the `value` attribute of the HTML `<input>` element. This attribute presets the value of an input field using the `value` attribute.

This value is hardcoded – it's always the same every time the form is shown.

```
<input name="subject" type="text" value="Fantasyland!>
```

But we don't want to hardcode a specific value. We want to insert the value from a PHP variable. How is that possible? Remember that we can use PHP to dynamically generate HTML code from PHP in other situations. So we can use `echo` to generate a value for the `value` attribute from a PHP variable.

Since we're switching to PHP to echo the variable, we have to use a `<?php` tag.

```
<input name="subject" type="text" value="<?php echo $subject; ?>">
```

And to return back to regular HTML, we close up the PHP code with the `</?php` tag.

For a text area input field, we echo the sticky data in between the `<textarea>` and `</textarea>` tags instead of using the `value` attribute.

Elmer's form can then be modified similarly to take advantage of this.

```
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
    <label for="subject">Subject of email:</label>
    <input id="subject" name="subject" type="text" value="<?php echo $subject; ?>"/><br />
    <label for="elvismail">Body of email:</label>
    <textarea id="elvismail" name="elvismail" rows="10" cols="50"><?php echo $text; ?></textarea><br />
    <input type="submit" name="submit" value="Submit" />
</form>
```



Test Drive

Check to see how sticky Elmer's data really is.

Change the code in `sendemail.php` to check `$_POST` for the form submission as well as adding `echo` code to the form so that its fields are sticky. Upload the new version of the script to your web server and open the script in a web browser. Experiment with different form field values, including leaving one or both fields empty, and submit it a few times.

Boy, that was dumb leaving the body of the email blank. Thankfully, I'll never do that again now that the form is on top of things. And I don't have to keep re-entering the same data to fix it either.

[make Me Elvis - Send Email](#)

MakEMEELViS.COM

Private: For Elmer's use ONLY
Write and send an email to mailing list members.

You forgot the email body text.
Subject of email:

Fall Clearance!
Body of email:

Submit

The Send Email script now shows an error message when Elmer leaves a form field blank, but it remembers any data he did enter.

*when good **DELETEs** go bad*

Some users are still disgruntled

Form validation has gone a long way toward dealing with Elmer's frustrated customers, particularly those who were receiving blank emails. But not everyone is happy. It seems a few people are receiving duplicate emails... remember this guy from earlier in the chapter?



This customer is frustrated because he keeps receiving multiple copies of Elmer's emails.

Elmer knows he didn't send a message more than once, leading him to suspect that maybe some users have accidentally subscribed to his email list more than once. Not a problem, just use the Remove Email page/script from the last chapter to remove the user, right?

Unfortunately, it's not that simple. Removing Elbert using his email address will completely delete him from the `email_list` table, causing him to no longer receive any email messages from Elmer. We need a way to only delete Elbert's **extra** rows from the table, making sure to leave one.

Using the Remove Email page from the previous chapter would remove the customer entirely from Elmer's database, which is not what we want.



How can Elmer delete all but one of the multiple rows in his table that have identical email addresses?



Hmm. The problem is that there are multiple rows in the table but no way to distinguish them from each other. Without a way to isolate them individually, any DELETE we try to do will delete all of them.

Joe: Maybe our Add Email form should check for duplicate email addresses before adding new users. That would fix it, right?

Frank: Excellent idea.

Jill: Yes, that would solve the problem moving forward, but it doesn't help us remove duplicate email addresses that are already in the database.

Frank: Right. What if we tried to use a different column in the table to identify extra rows, like `last_name`?

Jill: I wondered about that, but using a last name is potentially problematic. Two people could have the same last name. What if we wanted to delete someone named John Smith from our mailing list, and we ran the following SQL code:

```
DELETE FROM email_list WHERE last_name = 'Smith'
```

Joe: We wouldn't just delete John Smith from our table; we'd be deleting Will Smith, Maggie Smith, and every other John Smith in the database.

Frank: Wow, that wouldn't be good. Last names are more likely to be common across rows than first names. First names would be even worse than that. We could lose dozens and dozens of rows with one simple mistake.

Jill: Exactly. We can't risk using a WHERE clause that will delete rows we need to keep. We need to find a way to identify individual rows that won't affect other rows.

Joe: So what the heck do we do? We can't use `email`, `last_name`, or `first_name` in our WHERE clause.

Frank: We're out of columns in our table to use. Looks like we're out of luck.

Jill: Not necessarily. What we really need is something to make each row of the table unique—something that we can add to the table without changing the existing data. And just because we don't currently have a column that has a unique value doesn't mean we can't add one.

Joe: A new column? But we've already decided on our table structure.

Frank: Yeah, but what we've got isn't meeting our needs. You're right that it would be better if we had a unique column from the start, but we can still add one now. We can add a column beforehand, so we could have designed our table accordingly, but it's not too late to fix what we have.

Joe: OK, but what would we call our new column? What data would we put into it?

Jill: Well, since its purpose would be to uniquely identify each row in the table, we could call it `id`. It's a common convention in databases, and it's short, so maybe just `id` for short.

Frank: Nice, and we can fill the `id` column with a different ID number for each row, so when we want to remove a specific row, we'll be removing rows based on a unique number, instead of an email address or surname.

Joe: Exactly. It's really a great idea, isn't it? I'm so glad I thought of it.

adding a primary key column to a table

Table rows should be uniquely identifiable

Part of the whole idea of sticking something in a database is that later on you'd like to look it up and do something with it. Knowing this, it's incredibly important for each row in a table to be **uniquely identifiable**, meaning that you can specifically access one row (and only that row!). Elmer's `email_list` table makes a dangerous assumption that email addresses are unique. That assumption works as long as no one accidentally subscribes to the mailing list twice, but when they do (and they will!), their email address gets stored in the table twice... no more uniqueness!

What Elmer's table contains now:

first_name	last_name	email
Denny	Bubbleton	denny@mightygumball.net
Irma	Werlitz	iwer@aliensabductedme.com
Elbert	Kreslee	elbert@kresleespockets.biz
Irma	Kreslee	elbert@kresleespockets.biz

More than one person can have the same first name, so this isn't a good choice for a unique column.

Same here, we can't count on unique last names.

Nothing is
of this to
uniqueness

And
time
can
alw

When you don't have a column of truly unique values in a table, you should create one. MySQL gives you a way to add a unique integer column, also called a **primary key**, for each row in your table.

What Elmer's table should contain:

We need a new column that contains a value that is unique for every row in the table.

id	first_name	last_name	email
1	Denny	Bubbleton	denny@mightygumball.net
2	Irma	Werlitz	iwer@aliensabductedme.com
3	Elbert	Kreslee	elbert@kresleespockets.biz
4	Irma	Kreslee	elbert@kresleespockets.biz

Now that this column contains a unique value, we can be sure that every row in our table is truly unique.

Duplic
columns
the uni
because
takes ea



Hey genius, you know if we want to make a change to the table structure, we have to do a `DROP TABLE` and then recreate it from scratch. Elmer's email data will be toast!

It's true that `DROP TABLE` would destroy Elmer's data, but SQL has another command that lets you make changes to an existing table without losing any data.

It's called **`ALTER TABLE`**, and we can use it to create a new column without having to drop the table and destroy its data. Here's what the general syntax for an `ALTER TABLE` statement looks like for adding a new column to a table:

`ALTER TABLE table_name ADD column_name`

The name of the table to be altered.

The column name.

The data type of the new column.

We can use the `ALTER TABLE` command to add a new column to the `email_list` table, which we'll name `id`. We'll give the `id` column the data type of `INT`, since integers work great for establishing uniqueness. Some other parameters are also required, as this code reveals:

`ALTER TABLE email_list ADD id INT NOT NULL AUTO_INCREMENT`

The name of the table that we want to alter.

We want to ADD a new column, which we call `id`.

This tells the MySQL database to add the value stored in `id` for each new row that is inserted.

`ADD PRIMARY KEY (id)`

The data type of the column makes it an `INTEGER`.

FIRST tells MySQL to make `id` the primary key first in the table. This is good form to put your primary key at the top of the table.

This little chunk of code tells MySQL that the new `id` column is the primary key for the table. More on that in just a sec!

This `ALTER TABLE` statement has a lot going on because primary keys are created with very specific features. For example, `NOT NULL` tells MySQL that there must be a value in the `id` column—you can never leave it empty. `AUTO_INCREMENT` further describes the traits of the `id` column by telling MySQL to automatically get set to a unique numeric value when a new row is inserted. As the name suggests, `AUTO_INCREMENT` automatically adds one to the last value in a row and places this value into the `id` column when you `INSERT` a new row into your table. Finally, `PRIMARY KEY` tells MySQL that each value in the `id` column must be unique, but there's more to it than just uniqueness...

all about primary keys

Primary keys enforce uniqueness

A **primary key** is a column in a table that distinguishes each row in that table as unique. Unlike normal columns, which could also be designed to be unique, only one common can be made the primary key. This provides a clear choice for what column to use in any queries that need to pinpoint specific rows.

In order to ensure this uniqueness for primary keys, MySQL imposes a bunch of restrictions on the column that has been declared as PRIMARY KEY. You can think of these restrictions as rules to be followed as you work with primary keys:

The five rules of primary keys:



The data in a primary key can't be repeated.

Two rows should never have the same data in their primary keys. No exception! Every primary key should always have unique values within a given table.



A primary key must have a value.

If a primary key was left empty (NULL), then it might not be unique because another row could potentially also be NULL. Always set your primary keys to unique values.



The primary key must be set when a new row is inserted.

If you could insert a row without a primary key, you would run the risk of inserting a NULL primary key and duplicate rows in your table, which would defeat the purpose of having a primary key.



A primary key must be as efficient as possible.

A primary key should contain only the information it needs to be unique. It should not contain unnecessary data, such as names or addresses, for example. That's why integers make good primary keys—they allow for unique values without requiring much storage.



The value of a primary key can't be changed.

If you could change the value of your key, you'd risk accidentally setting a primary key to a value that's already used. Remember, it has to remain unique at all costs.

The id column in Elmer's table doesn't have repeat data, has a value for every row, is automatically set when a new row is inserted, is compact, and doesn't change. Perfect!

id	first_name	last_name	
1	Denny	Bubbleton	denny@
2	Irma	Werlitz	iwer@ali
			...



Test Drive

Alter Elmer's table and try out inserting a new row of data with a primary key.

Using a MySQL tool such as the MySQL terminal or the SQL tab of phpMyAdmin, enter the `ALTER TABLE` statement to add a primary key column named `id`:

```
ALTER TABLE email_list ADD id INT NOT NULL AUTO_INCREMENT
ADD PRIMARY KEY (id)
```

Now insert a new customer to the database to see if the `id` column is automatically set for the new row. Here's an example of an `INSERT` statement to use (notice the primary key isn't mentioned):

```
INSERT INTO email_list (first_name, last_name, email)
VALUES ('Don', 'Draper', 'draper@sterling-cooper.com')
```

Finally, issue a `SELECT` statement to view the contents of the table and see the new primary key in all its glory! Just in case you've forgotten, here's the `SELECT` statement:

```
SELECT * FROM email_list
```

The new `id` column
is auto-incremented
so that it remains
unique for the new
row of data.

```
File Edit Window Help Email
mysql> SELECT * FROM email_list;
+----+----+----+----+
| id | first_name | last_name | email
+----+----+----+----+
| 1  | Denny      | Bubbleton | denny@mighty
| 2  | Irma       | Werlitz   | iwer@aliensal
| 3  | Elbert     | Kreslee   | elbert@kresle
| 4  | Irma       | Kreslee   | elbert@kresle
| 5  | Don        | Draper    | draper@sterli
+----+----+----+----+
5 rows in set (0.0005 sec)
```

cubicle conversation



Okay, so now every row in the table has a unique primary key. How does that help? Elmer still deletes based on email addresses.

Joe: The problem is that the user needs to pinpoint rows of data by primary key instead of the email address.

Frank: That's right! So we just need to change the form so that it takes the ID of a customer instead of their email address. No problem!

Jill: Actually, big problemo. The user has no way of knowing without somehow finding them in the database. In fact, the user has no way of knowing anything about the database structure. Maybe what we need is something like a search function so that it lists out all the names and email addresses in a list with a link to each one. Here, I'll sketch it for you.

A hand-drawn sketch of a form. On the left, there is a vertical list of five items, each preceded by an empty square checkbox. The first item is "John Doe johndoe@someemail.com". To the right of the list, there are four wavy lines, each starting from a line above a checkbox and ending at a line below it. At the bottom right is a rectangular button labeled "SUBMIT". A curved arrow points from the word "the" on the far left to the first checkbox.

Frank: Nice sketch, but how does that help Elmer isolate a cut using their ID?

Joe: Hmm. What if we stored the customer ID in the value of a variable? It's not actually visible, but the script can get to it.

Jill: That's a great idea. So we could generate the form automatically by doing a SELECT to get all the data, and then creating each checkbox from a row of query data.

Joe: Cool. But what happens when the Submit button is pressed? What does POST have in it?

Frank: Hang on, Joe, we'll get there in a minute. Let's just start part of the script, the part that displays all the data from the table, those checkboxes.



PHP & MySQL Magnets

Use the magnets below to finish the missing code for the Remove Email script, which lists checkboxes for the customers in Elmer's database. Note that this code just creates the form; you'll have to worry about the code that performs the DELETE just yet.

```






Please select the email addresses to delete from the email list and click Remove.



<?php
$dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking', 'elvis_stor');
or die('Error connecting to MySQL server.');

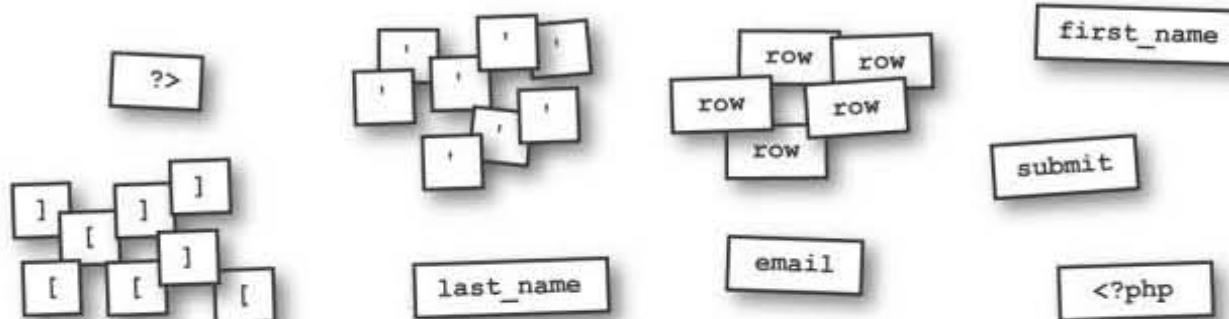
// Display the customer rows with checkboxes for deleting
$query = "SELECT * FROM email_list";
$result = mysqli_query($dbc, $query);

while ( $row = mysqli_fetch_array($result) ) {
    echo '<input type="checkbox" value="';
    echo $row['email'];
    echo '" checked="checked" />';
    echo '<br />';
}

mysqli_close($dbc);
?>

<input type="submit" name="submit" value="Remove" />
</form>

```



php & mysql magnets solution



PHP & MySQL Magnets Solution

Use the magnets below to finish the missing code for the Remove Email script, which lists all the checkboxes for the customers in Elmer's database. Note that this code just creates the form - you'll have to worry about the code that performs the DELETE just yet.

```

<?php
$dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking', 'elvis_story');
or die('Error connecting to MySQL server.');

// Display the customer rows with checkboxes for deleting
$query = "SELECT * FROM email_list";
$result = mysqli_query($dbc, $query);

while ($row = mysqli_fetch_array($result)) {
    echo '<input type="checkbox" value="' . $row['id'] . '"';
    echo ' ' . $row['first_name'] . ' ';
    echo ' ' . $row['last_name'] . ' ';
    echo ' ' . $row['email'] . ' ';
    echo '<br />';
}

mysqli_close($dbc);
?>

<input type="submit" name="submit" value="Remove" />

```

This form is self-referencing!

?>

This is where the
gets used in the
we can use this
any checked

Each checkbox
field is constru
a row of custo

The script doesn't actually
do any deleting yet. For
now it just presents a list
of checkboxes.

You can name your Submit button anything
you want - just be sure to remember the
name later if you decide to check \$_POST
to see if the form was submitted.

From checkboxes to customer IDs

The checkbox code generated by the Remove Email script is simple HTML with our primary key (`id`) stuffed into the `value` attribute of the `<input>` tag. There's one small, but very important change from ordinary checkbox HTML code, though. You might have noticed square brackets (`[]`) at the end of the checkbox name—they serve a vital purpose.

```
echo '<input type="checkbox" value="' . $row['id'] . '" name=
```

The square brackets result in the creation of an array within `$_POST` that stores the contents of the `value` attribute of every checked checkbox in the form. Since each checkbox's `value` attribute contains a primary key, **each value in the `todelete` array is the ID of the row in our table that needs to be deleted**. This makes it possible for us to loop through the `todelete` array and issue an SQL query to delete each customer that is checked in the form.

Each checkbox form field has the ID of the customer stored away, which is accessible through the `$_POST` superglobal.

I get it. We just use a while loop to cycle through the `todelete` array and delete each of the customers using their IDs.

We could use a while loop but there's a more solution using a different kind of loop.

Write down how you think a `foreach` loop might look for an array of Elmer's customer IDs:

.....

Make Me Elvis - Remove

MAKEELVIS.COM

Please select the email addresses to delete from the list and click Remove.

- Denny Bubbleton denny@mightygumball.net
- Irma Wuritz iwrz@aliensabductedme.com
- Elbert Kreslee elbert@kresleesprockets.biz
- Irma Kreslee elbert@kresleesprockets.biz
- Don Draper draper@sterling-cooper.com

Remove

anatomy of a foreach loop

Loop through an array with foreach

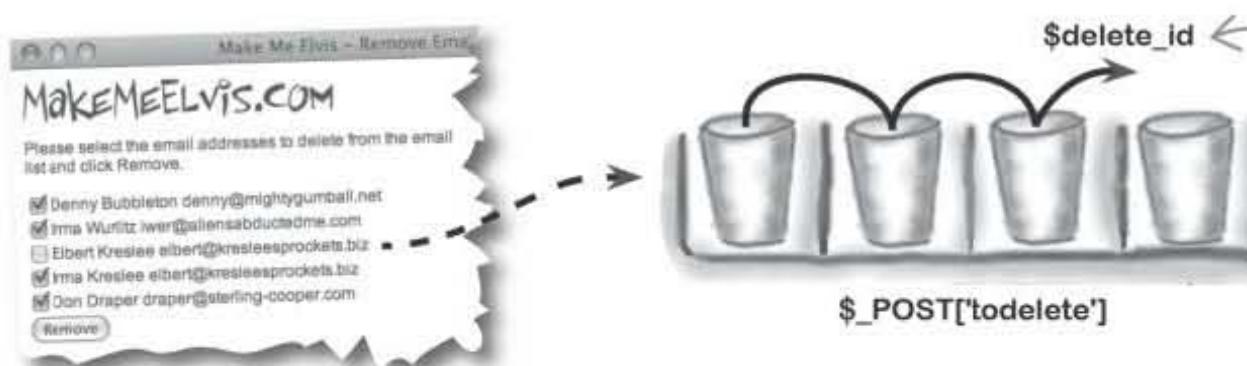
The `foreach` loop takes an array and loops through each element in the array without the need for a test condition or loop counter. As it steps through each element in the array, it temporarily stores the value of that element in a variable. Assuming an array is stored in a variable named `$customers`, this code steps through each one:

```
The array you want to
loop through appears first
↓
foreach ($customers as $customer) {
    echo $customer;
}
As the loop goes through each individual
element in the array, it will temporarily
store them in a variable with this name.
↓
Inside of the loop, you can access each element
using the variable name you just provided.
```

So if we want to loop through the customer IDs stored in the `$_POST` array in the Remove Email script, we can use the following `foreach` code:

```
Here the array is stored inside
of the $_POST superglobal,
and identified by "todelete".
↓
foreach ($_POST['todelete'] as $delete_id) {
    // Delete a row from the table
}
Each element of the
will be accessible thro
variable $delete_id.
↓
We can use $delete_id to delete each
of the customers from the database.
```

The `$delete_id` variable holds the value of each array element as the loop progresses through them one at a time.



With the `foreach` loop now stepping through each of the **checked** checkboxes in the Remove Email form, we just need to add code inside of the loop to issue a `DELETE` query and actually delete each row from the `email_list` table.



Finish the code for Elmer's new and improved `removeemail.php` script that removes customers that have been checked in the form when the form is submitted.

```
...
$dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking', 'elv');
or die('Error connecting to MySQL server.');

// Delete the customer rows (only if the form has been submitted)
if (.....) {
    foreach ($_POST['todelete'] as $delete_id) {
        .....
    }
    echo 'Customer(s) removed.<br />';
}

// Display the customer rows with checkboxes for deleting
$query = "SELECT * FROM email_list";
$result = mysqli_query($dbc, $query);
while ($row = mysqli_fetch_array($result)) {
    echo '<input type="checkbox" value="' . $row['id'] . '" name="todele';
    echo $row['first_name'];
    echo ' ' . $row['last_name'];
    echo ' ' . $row['email'];
    echo '<br />';
}

mysqli_close($dbc);
?>

<input type="submit" name="submit" value="Remove" />
</form>
```

the revised removeemail.php script

Finish the code for Elmer's new and improved `removeemail.php` script that removes customers that have been checked in the form when the form is submitted.

*Only delete
customers
if the form
has been
submitted!*

```

$dbc = mysqli_connect('data.makemeelvis.com', 'elmer', 'theking', 'elmer');
or die('Error connecting to MySQL server.');

// Delete the customer rows (only if the form has been submitted)
if (!isset($_POST['submit'])) {
    foreach ($_POST['todelete'] as $delete_id) {
        $query = "DELETE FROM email_list WHERE id = $delete_id";
        mysqli_query($dbc, $query);
        or die('Error querying database.');
    }

    echo 'Customer(s) removed.<br />';
}

// Display the customer rows with checkboxes for deleting
$query = "SELECT * FROM email_list";
$result = mysqli_query($dbc, $query);
while ($row = mysqli_fetch_array($result)) {
    echo '<input type="checkbox" value="' . $row['id'] . '" name="todelete">';
    echo $row['first_name'];
    echo ' ' . $row['last_name'];
    echo ' ' . $row['email'];
    echo '<br />';
}

mysqli_close($dbc);

?>

<input type="submit" name="submit" value="Remove" />
</form>

```

*Use \$_POST
choose customer*

The code to generate the customer checkboxes is the same as you created it before.



Test Drive

Take Elmer's newly revamped Remove Email script for a spin

Modify the code in the `removeemail.php` script so that it generates customer checkboxes instead of using the old email text field. Then add the code to delete customers whenever the form's submitted. Also change the `action` attribute of the `<form>` tag so that the form's self-referencing.

Now that `removeemail.php` uses a self-referencing form, you no longer need the `removeemail.html` page on your web server, so feel free to delete it. Then upload the new version of `removeemail.php` to your web server and open the script in a web browser. Check off a few customers and click Submit. The form immediately changes to reflect the customer removal.



When you check off a customer and click Submit, the customer's removed from the database.



The script confirms the customer removal and also updates the checklist – the deleted customer is now gone.

viva makemeelvis.com!



Totally digging my new Remove Email form. Time for a vacation. Viva Las Vegas, baby!



Elmer's got a fully functioning application. He can send out spectacular sale emails to just the customers who receive them, and delete customers who have moved on the side, or just want to be removed from his list.

[Send Email](#)

MakeMEELVIS.COM

PRIVATE: For Elmer's use ONLY. Write and send an email to mailing list members.

Subject of email:

Body of email:

Big Sale! Big sales this week at MakeMEELVIS.com! Genuine home run sideburns 20% off! And don't forget the "buy one, get one free" leisure suits — only three days left!

Customer list:

- Debra
- Jim
- Elmer
- Don
- Alice

Customer list:

MakeMEELVIS.COM

Enter your first name, last name, and email to be added to the MakeMEELVIS mailing list.

First name: Julian

Last name:

Email: julian@makeelvis.com

Submit



Your PHP & MySQL Toolbox

You bagged quite a few new PHP and MySQL skills while taking Elmer's web application to a whole new level...

!

The negation operator, or NOT operator, reverses a true/false value. So true becomes false and false becomes true.

ALTER TABLE

This SQL statement changes the structure of a table, such as adding a new column of data. This allows you to alter a table structurally without having to drop it and start over.

foreach

A PHP looping construct that lets you loop through an array one element at a time without using a test condition. Inside the loop, you can access each element of the array.

&&, OR

These are logical operators used to build conditions involving true/false values. Combining two (&&) results in true if both values are true. Combining two (OR) results in true if either of the values is true.

isset(), empty()

The built-in PHP `isset()` function tests to see if a variable exists, which means that it has been assigned a value. The `empty()` function takes things one step further and determines whether a variable contains an empty value (0, an empty string, false, or `NULL`).

5 working with data stored in files

When a database just isn't enough



Don't believe the hype...about databases, that is. Sure, they work wonders for storing all kinds of data involving text, but **what about binary data?** You know, stuff like **JPEG images** and **PDF documents**. Does it really make sense to store all those pictures of your rare guitar pick collection in a database table? Usually not. That kind of data is typically stored in files, and we'll leave it in files. But it's entirely possible to have your virtual cake and eat it too—this chapter reveals that you can **use files and databases together to build PHP applications** that are awash in binary data.

guitar wars needs screenshots

Virtual guitarists like to compete

Apparently creating art for art's sake isn't always enough because players of the hot new game Guitar Wars are quite enamored with competitive virtual guitar playing. So much so that they regularly post their high scores at the Guitar Wars web site, which you are now in charge of maintaining. Problem is, there isn't currently a good way to verify the scores.

The Gu...
allows
scores

The screenshot shows a web browser window titled "Guitar Wars - High Scores". The page content is as follows:

Guitar Wars - High Scores

Welcome, Guitar Warrior, do you have what it takes to crack the high score list? If so, just add your own score.

Score	Name	Date
127650	Paco Jastorius	2008-04-22 14:37:34
98430	Nevil Johansson	2008-04-22 21:27:54
345900	Eddie Vanilli	2008-04-23 09:06:35
282470	Belita Chevy	2008-04-23 09:12:53
368420	Ashton Simpson	2008-04-23 09:13:34

A thought bubble from Belita Chevy contains the text: "This is so bogus. There's no way all those scores are real. I'd like to see proof!" An arrow points from the text "With no way to verify, we can't know whose score is valid and whose isn't." to the thought bubble.

With no way to verify, we can't know whose score is valid and whose isn't.

Belita, skeptical
Guitar Wars rocker.

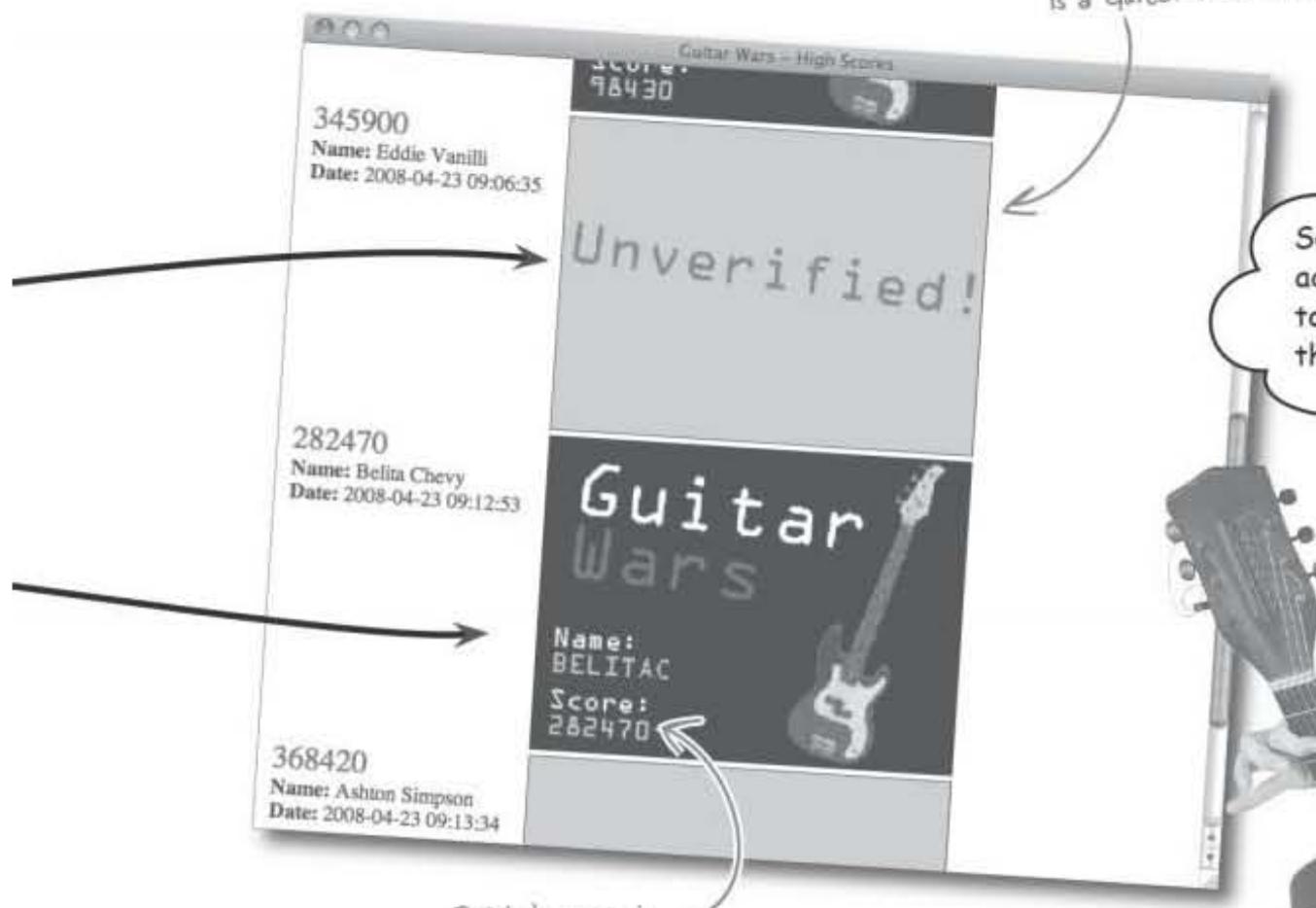
Text can't be trusted

Right now players just post up their high scores purely as text, and there have been lots of disputes over whose scores are valid and whose aren't. There's only one way to put an end to all this bickering and crown a legitimate Guitar Wars champion...

The proof is in the ~~rockin'~~ picture

Visual verification of a high score is what we need to determine who's for real and who isn't. So the Guitar Wars application needs to allow users to submit a screen shot of their high score when posting their score. This means the high score list will not only be a list of scores, names, and dates, but also a list of images (screen shots).

With photo verification
we find out that Eddie
is a Guitar Wars fraud.



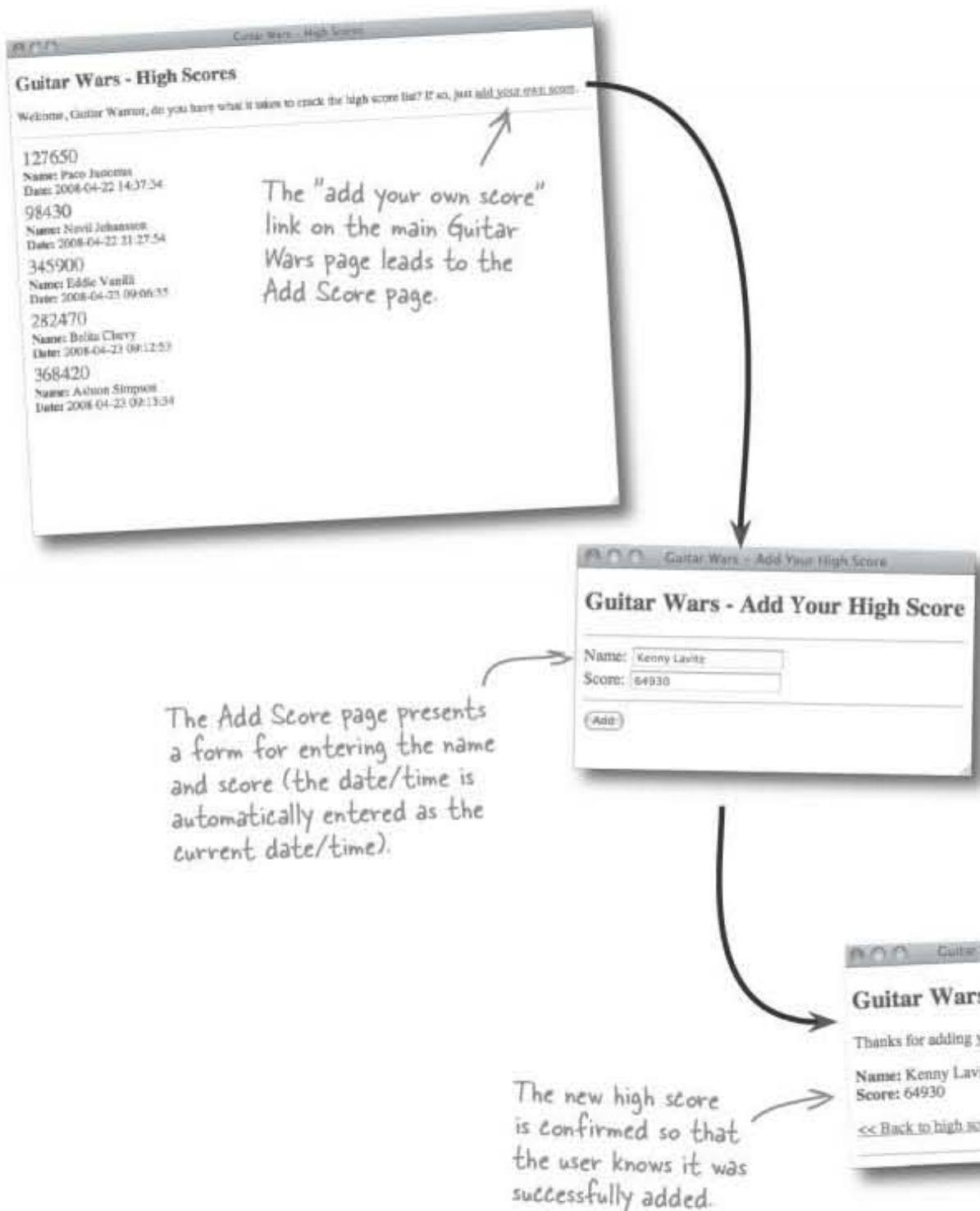
Belita's score is
legit, thanks to her
submitted screen shot.

Eddie, rocker
wannabe and Guitar
Wars score faker.

the guitar wars application design

The application needs to store images

Currently, the Guitar Wars high score application keeps track of three pieces of information: the date and time of a new score, the name of the person submitting the score, and the score itself. This information is entered through a form as part of the application's user interface, after which it gets stored in a MySQL database table called *guitarwars*.



This ID is the primary key for the database and is automatically generated for each row.

This is the exact date (and time) that a score was submitted to the Guitar Wars application.

id	date	name	score
1	2008-04-22 14:37:34	Paco Jastorius	127650
2	2008-04-22 21:27:54	Nevil Johansson	98430
3	2008-04-23 09:06:35	Eddie Vanilli	345900
4	2008-04-23 09:12:53	Belita Chevy	282470
5	2008-04-23 09:13:34	Ashton Simpson	368420
6	2008-04-23 14:09:50	Kenny Lavitz	64930

After entering a name and score and clicking Add, the new score is confirmed and added to the guitarwars table in the database.

The guitarwars table also stores the name and score for each high score entry in the database.

The newly added score immediately appears on the main Guitar Wars page.

Guitar Wars - High Scores		
Welcome, Guitar Warrior, do you have what it takes?		
127650	Name:	Paco Jastorius
98430	Date:	2008-04-22 14:37:34
345900	Name:	Nevil Johansson
282470	Date:	2008-04-22 21:27:54
368420	Name:	Eddie Vanilli
64930	Date:	2008-04-23 09:06:35
2008-04-23 09:12:53	Name:	Belita Chevy
2008-04-23 09:13:34	Name:	Ashton Simpson
2008-04-23 14:09:50	Name:	Kenny Lavitz

annotate the code

The Guitar Wars high score application will have to change to accommodate files for high score screen shots. Circle and annotate the parts of the application change to support user-submitted images.

```

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <title>Guitar Wars - High Scores</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
    <h2>Guitar Wars - High Scores</h2>
    <p>Welcome, Guitar Warrior, do you have what it takes to crack the
    high score list? If so, just <a href="addscore.php">add your own
    score</a>. </p>
    <hr />

    <?php
        // Connect to the database
        $dbc = mysqli_connect('www.guitarwars.net', 'admin', 'rockit', 'gwdb');

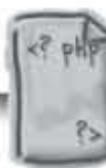
        // Retrieve the score data from MySQL
        $query = "SELECT * FROM guitarwars";
        $data = mysqli_query($dbc, $query);

        // Loop through the array of score data, formatting it as HTML
        echo '<table>';
        while ($row = mysqli_fetch_array($data)) {
            // Display the score data
            echo '<tr><td class="scoreinfo">';
            echo '<span class="score">' . $row['score'] . '</span><br />';
            echo '<strong>Name:</strong> ' . $row['name'] . '<br />';
            echo '<strong>Date:</strong> ' . $row['date'] . '</td></tr>';
        }
        echo '</table>';

        mysqli_close($dbc);
    ?>

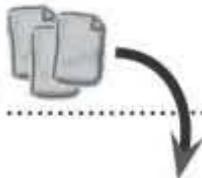
</body>
</html>

```



index.php

Download It!



The complete source code for the Guitar Wars application is available for download from the Head First Labs web site:

www.headfirstlabs.com/books/hfphp

This
need
You do
worry

guita

id	date
1	2008-04-22 14:37:34
2	2008-04-22 21:27:54
3	2008-04-23 09:06:35
4	2008-04-23 09:12:53
5	2008-04-23 09:13:34
6	2008-04-23 14:09:50

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <title>Guitar Wars - Add Your High Score</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
    <h2>Guitar Wars - Add Your High Score</h2>

<?php
if (isset($_POST['submit'])) {
    // Grab the score data from the POST
    $name = $_POST['name'];
    $score = $_POST['score'];

    if (!empty($name) && !empty($score)) {
        // Connect to the database
        $dbc = mysqli_connect('www.guitarwars.net', 'admin', 'rockin123');

        // Write the data to the database
        $query = "INSERT INTO guitarwars VALUES (0, NOW(), '$name', '$score')";
        mysqli_query($dbc, $query);

        // Confirm success with the user
        echo '<p>Thanks for adding your new high score!</p>';
        echo '<p><strong>Name:</strong> ' . $name . '<br />';
        echo '<strong>Score:</strong> ' . $score . '</p>';
        echo '<p><a href="index.php">&lt;&lt; Back to high scores</a></p>';

        // Clear the score data to clear the form
        $name = "";
        $score = "";

        mysqli_close($dbc);
    }
    else {
        echo '<p class="error">Please enter all of the information to add your high score.</p>';
    }
}

?>

<hr />
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
    <label for="name">Name:</label><input type="text" id="name" name="name" value="<?php if (!empty($name)) echo $name; ?>" /><br />
    <label for="score">Score:</label><input type="text" id="score" name="score" value="<?php if (!empty($score)) echo $score; ?>" />
    <hr />
    <input type="submit" value="Add" name="submit" />
</form>
</body>
</html>
```

annotated guitar wars code



The Guitar Wars high score application will have to change to accommodate files for high score screen shots. Circle and annotate the parts of the application to support user-submitted images.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <title>Guitar Wars - High Scores</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
    <h2>Guitar Wars - High Scores</h2>
    <p>Welcome, Guitar Warrior, do you have what it takes to crack the high score list? If so, just <a href="addscore.php">add your own score</a>.</p>
    <hr />

    <?php
        // Connect to the database
        $dbc = mysqli_connect('www.guitarwars.net', 'admin', 'rockit');

        // Retrieve the score data from MySQL
        $query = "SELECT * FROM guitarwars";
        $data = mysqli_query($dbc, $query);

        // Loop through the array of score data, formatting it as HTML
        echo '<table>';
        while ($row = mysqli_fetch_array($data)) {
            // Display the score data
            echo '<tr><td class="scoreinfo">';
            echo '<span class="score">' . $row['score'] . '</span><br />';
            echo '<strong>Name:</strong> ' . $row['name'] . '<br />';
            echo '<strong>Date:</strong> ' . $row['date'] . '</td>';
            echo '</tr>';
        }
        echo '</table>';

        mysqli_close($dbc);
    ?>
</body>
</html>
```

The screen shot image needs to be displayed on the main page.



index.php

The image should be displayed to the user to confirm success.

guitarwars			
id	date	name	score
1	2008-04-22 14:37:34	Paco Jastorius	127650
2	2008-04-22 21:27:54	Nevil Johansson	98430
3	2008-04-23 09:06:35	Eddie Vanilli	345900
4	2008-04-23 09:12:53	Belita Chevy	282470
5	2008-04-23 09:13:34	Ashton Simpson	368420
6	2008-04-23 14:09:50	Kenny Lavitz	64930

The table needs a new column to store the screen shot image filename for each score.

The form needs an tag for the image file selection.

The screen shot image file must be obtained from form POST data.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <title>Guitar Wars - Add Your High Score</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
    <h2>Guitar Wars - Add Your High Score</h2>

    <?php
        if (isset($_POST['submit'])) {
            // Grab the score data from the POST
            $name = $_POST['name'];
            $score = $_POST['score'];

            if (!empty($name) && !empty($score)) {
                // Connect to the database
                $dbc = mysqli_connect('www.guitarwars.net', 'admin', 'rockit');

                // Write the data to the database
                $query = "INSERT INTO guitarwars VALUES (0, '$name', '$score')";
                mysqli_query($dbc, $query);

                // Confirm success with the user
                echo '<p>Thanks for adding your new high score!</p>';
                echo '<p><strong>Name:</strong> ' . $name . '<br />';
                echo '<strong>Score:</strong> ' . $score . '<br />';
                echo '<p><a href="index.php"><input type="button" value="Back to Main Page"></a></p>';

                // Clear the score data to clear the form
                $name = "";
                $score = "";

                mysqli_close($dbc);
            } else {
                echo '<p class="error">Please enter all of the required fields to add your high score.</p>';
            }
        }
    ?>

    <hr />
    <form method="post" action="<?php echo $_SERVER('PHP_SELF') ?>">
        <label for="name">Name:</label><input type="text" name="name" value="<?php if (!empty($name)) echo $name; ?>" />
        <label for="score">Score:</label><input type="text" name="score" value="<?php if (!empty($score)) echo $score; ?>" />
        <input type="submit" value="Add" name="submit" />
    </form>
</body>
</html>
```

This query takes a guess by not specifying a column.

Upon success, make sure to update the image form field with the new score.

Planning for image file uploads in Guitar Wars

Although it may not seem like a big deal to add support for uploadable screen shot images to Guitar Wars, the application must change in a variety of ways. For this reason, it's a good idea to have a plan of attack before diving into any code. Let's nail down the steps required to revamp the Guitar Wars high scores for screen shots.

1

Use ALTER to add a screenshot column to the table.

First off is the database, which needs a new column for storing the name of each screen shot image file. Since we plan on putting all image files in the same folder, all we need to store in the database is the filename itself (no path).



2

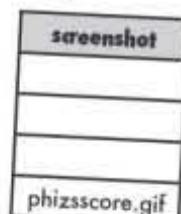
Change the Add Score uses a file input field to file uploads.

The Add Score page already has scores, so we need to modify that file input field. This input field will work with the web browser to present an interface for selecting a file to up-

3

Write a query to INSERT the screen shot image filename into the screenshot column of the table.

The Add Score script that processes the form for adding scores must also take into consideration the new file input form field, and handle inserting a screen shot image filename into the screenshot column when inserting a new high score row into the guitarwars table.



4

Change the main Guitar Wars page to show screen shot images for the high scores.

Last on the laundry list of changes involves the main index.php Guitar Wars page, which must be changed to actually show the screen shot image for each high score that is displayed.

Screen shot:

ALTER your table

The high score database must be ALTERed

In addition to a variety of PHP scripting tweaks, the image-powered Guitar Wars application needs a new column in the `guitarwars` table to store screen shot image filenames. Enter SQL, which offers a statement called `ALTER` that is capable of modifying tables in all kinds of interesting ways, including adding new columns of data. You used the `ALTER` statement in the previous chapter to tweak Elmer's `email_list` table, but let's recap how the command works.

```
ALTER TABLE guitarwars DROP COLUMN score
```

The `DROP COLUMN` ↑
statement drops an entire
column from a table.

OK, maybe that's a dangerous example since it reveals how to drop an entire column from a table, data and all. Yet there certainly may be a situation where you need to remove a column of data from a table. It's more likely that you need to add a column of data, as is the case with Guitar Wars. This is made possible by `ADD COLUMN`, which is one of several table alterations you can carry out with `ALTER`.

ADD COLUMN

Adds a new column to a table—just specify the name of the column and its type following `ADD COLUMN`.

```
ALTER TABLE guitarwars  
ADD COLUMN age TINYINT
```

DROP COLUMN

Drops a column (and any data stored in it) from a table—just specify the name of the column following `DROP COLUMN`.

```
ALTER TABLE guitarwars  
DROP COLUMN age
```

CHANGE COLUMN

Changes the name and data type of a column—just specify the name of the old column, new column, and new data type following `CHANGE COLUMN`.

```
ALTER TABLE guitarwars  
CHANGE COLUMN score high_score INT
```

MODIFY COLUMN

Changes the data type or position of a column within a table—just specify the name of the column and its new data type following `MODIFY COLUMN`. To change the position of a column, specify the name of the column and its new position (`FIRST` is the only option for changing the position relative to other columns; `AFTER` another column is the only option for specifying the position by name).

```
ALTER TABLE guitarwars  
MODIFY COLUMN date DATETIME
```

The **ALTER**
statement
changes the
of a database

The **ALTER** statement followed by `TABLE` that you're going to use. It's also possible to change the structure of the table with `ALTER TABLE`, that's another story.



Sharpen your pencil

Write an SQL statement that adds a new column named `rating` to the `guitarwars` table. Make sure to give the new column a suitable MySQL data type. Then write another SQL query to select the structure of the table and make sure the column was successfully added.

guitarwars

id	date	name	store	screenshots
1	2008-04-22 14:37:34	Paco Jastorius	127650	
2	2008-04-22 21:27:54	Nevil Johansson	98430	
3	2008-04-23 09:06:35	Eddie Vanilli	345900	
4	2008-04-23 09:12:53	Belita Chevy	282470	
5	2008-04-23 09:13:34	Ashton Simpson	368420	
6	2008-04-23 14:09:50	Kenny Lavitz	64930	

Write the statement
that adds a column here.



.....

Write the other
SQL statement here.



.....

sharpen your pencil solution

Sharpen your pencil Solution

Write an SQL statement that adds a new column named screenshot to the guitarwars table. Make sure to give the new column a suitable MySQL data type. Then write another SQL query to show the structure of the table and make sure the column was successfully added.

id	date	name	score	screenshot
1	2008-04-22 14:37:34	Paco Jastorius	127650	
2	2008-04-22 21:27:54	Nevil Johansson	98430	
3	2008-04-23 09:06:35	Eddie Vanilli	345900	
4	2008-04-23 09:12:53	Belita Chevy	282470	
5	2008-04-23 09:13:34	Ashton Simpson	368420	
6	2008-04-23 14:09:50	Kenny Lavitz	64930	

The ALTER statement doesn't affect any of the other table data.

ALTER TABLE guitarwars

ADD COLUMN screenshot varchar(64)

ADD COLUMN indicates that we want to alter the table by adding a new column of data.

The name of the table to be altered follows ALTER TABLE.

The name and data type of the specified last in the SQL query are enough to accommodate most data types, although you can make the column nullable if you want to be extra safe.

DESCRIBE guitarwars

This statement displays the structure of the table, including the column names and their data types.

The first step is done!

1 Use ALTER to add a column to the table

DONE



Test Drive

Add the screenshot column to the guitarwars table.

Using a MySQL tool, execute the ALTER statement to add the screenshot column to the guitarwars table. Then issue the DESCRIBE statement to take a look at the table structure and make sure the column was added.

Issuing the DESCRIBE statement reveals the new screenshot column.

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	
date	timestamp	NO		CURRENT_TIMESTAMP	auto_increment
name	varchar(32)	NO		NULL	
score	int(11)	NO			
screenshot	varchar(64)	NO			

5 rows in set (0.03 sec)

there are no
Dumb Questions

Q: Do new columns added with ALTER have to be added to the end of a database table?

A: No, they can be added anywhere. But keep in mind that the order of the columns in a table isn't terribly important. In other words, you can structure query results so that data is organized in any order you want. But maybe you like the sense of structural order brought about by a specific ordering of columns, in which case you may want to add a column in an exact location. You can do this by tacking on the keyword FIRST to the ALTER query. Or use AFTER column to place a column relative to another column:

ALTER TABLE guitarwars
ADD COLUMN age TINYINT AFTER name

If you don't specify where a new column is added, it defaults to the end of the table.

Q: What happens to the existing high score database rows of data after adding the new screenshot column?

A: Since the ALTER statement only affects the structure of a database, the new screenshot column is empty for all pre-existing rows of high scores. While it's possible to populate the screenshot column of future rows, pre-existing rows all have an empty screenshot column.

Q: Can I add to

A: Yes, use the UPDATE command. There is no need to upload the file, then use the INSERT command. Remember to submit the file to allow us to see it. By using the UPDATE command, you can easily add new data to the database without having to upload a new file.

add images with the add score form

How do we get an image from the user?

With a new column added to the high score database, we're ready to focus on allowing the user to upload an image file. But how exactly is that possible? FTP? Mental telepathy? This actually leads back to the Add Score form, where we can use a form field to allow the user to select an image file to upload.

Guitar Wars - Add Your High Score

Name: Phiz Lairston
Score: 186580
Screen shot: Choose File

Add

The specifics of this button are controlled by the web browser and the native operating system. Usually it triggers a file browser dialog box where the user can navigate to find a file on their hard drive.

The Add Score form is what allows users to add a new high score to the Guitar Wars high score list.



phizsscore.gif

Upon submitting the form, the binary image file is uploaded to the server.

So an input field helps the user find the file to be uploaded, then what? The file upload form field also takes care of the selected image getting uploaded to a folder on the server, where it can then be displayed as part of the Guitar Wars high score list.

A folder on the server receives the image file and stores it away.

Is this file upload form field some kind of strange extension to HTML? No, not at all. The HTML `<input>` tag supports file form fields and works in conjunction with PHP to allow file uploads. But before we get into the PHP side of things, let's take a closer look at the form field itself...



The Add Score Form Up Close

This form attribute tells the form to use a special type of encoding required for file uploading - it affects how the POST data is bundled and sent when the form is submitted.

Establishes a maximum file size for file uploads, in this case 32 KB (32,768 bytes).

```
<form enctype="multipart/form-data" method="post" action="php echo $_SERVER['PHP_SELF']?"&gt;
    &lt;input type="hidden" name="MAX_FILE_SIZE" value="32768" /&gt;
    &lt;label for="name"&gt;Name:&lt;/label&gt;
    &lt;input type="text" id="name" name="name" value="<?php if (!empty($name)) echo $name; ?" />
    <br />
    <label for="score">Score:</label>
    <input type="text" id="score" name="score" value="php if (!empty($score)) echo $score; ?" />
    <br />
    <label for="screenshot">Screen shot:</label>
    <input type="file" id="screenshot" name="screenshot" />
    <hr />
    <input type="submit" value="Add" name="submit" />
</form>
```

The actual file input which ultimately runs native operating systems for file browsing.

2

DO
Change
form so
file input
image fi

store image filenames in the database

filename

Insert the image into the database

Simply uploading an image file to the web server via a form isn't enough. We have to store the filename in the new `screenshot` column of the database so that the image can be accessed and displayed. As it stands, the Add Score script already inserts new high scores into the `guitarwars` table using the SQL `INSERT` statement, but this statement doesn't factor in the new `screenshot` column:

`INSERT INTO guitarwars VALUES (0, NOW(), '$name', '$score')`

The id column gets set automatically via
`AUTO_INCREMENT` – the 0 is ignored,
 although the query does require a value here.

The MySQL NOW() function is used to insert the current date and time into the database.

Since this SQL statement is inserting values without identifying the column names for each, it must include a value for every column. But we just added a new column, which means the query no longer works—it's missing a value for the new `screenshot` column. So adding a screenshot image filename to the database as part of a new high score row requires us to add a new value to the `INSERT` statement:

`INSERT INTO guitarwars VALUES (0, NOW(), '$name', '$score', 'phizsscore.gif')`

Rows of data inserted prior to the addition of the screenshot column don't have a screenshot filename.

guitarwars

id	date	name	score	screenshot
1	2008-04-22 14:37:34	Paco Jastorius	127650	
2	2008-04-22 21:27:54	Nevil Johansson	98430	
3	2008-04-23 09:06:35	Eddie Vanilli	345900	
4	2008-04-23 09:12:53	Belita Chevy	282470	
5	2008-04-23 09:13:34	Ashton Simpson	368420	
6	2008-04-23 14:09:50	Kenny Lavitz	64930	
7	2008-04-24 08:13:52	Phiz Lairston	186580	phizsscore.gif

The new `INSERT` statement results in the screenshot filename getting inserted into the screenshot column.

Find out the name of the uploaded file

The query looks good, but we still don't know what the actual filename of the image is. It's fair to assume that the file input field in the form somehow provides access to the filename, but how? The answer lies in a built-in PHP superglobal variable named `$_FILES`, which is similar to the `$_POST` superglobal we've used to access form data. Like `$_POST`, `$_FILES` is an array, and within it is not only the name of the uploaded file, but also some other information about the file that might prove useful.

The form passes some useful information about the file to the PHP script via the `$_FILES` superglobal variable.



`phizsscore.gif`

This is the image file being uploaded thanks to the file input field in the form.

The `$_FILES` built-in superglobal variable provides access to information about uploaded files.

`$_FILES['screenshot']`

`phizsscore.`

`$_FILES['screenshot']['ty`

`image/gif`

`$_FILES['scre`

`$_FILES['screenshot']['tmp`

`/tmp/phpE7qJky`

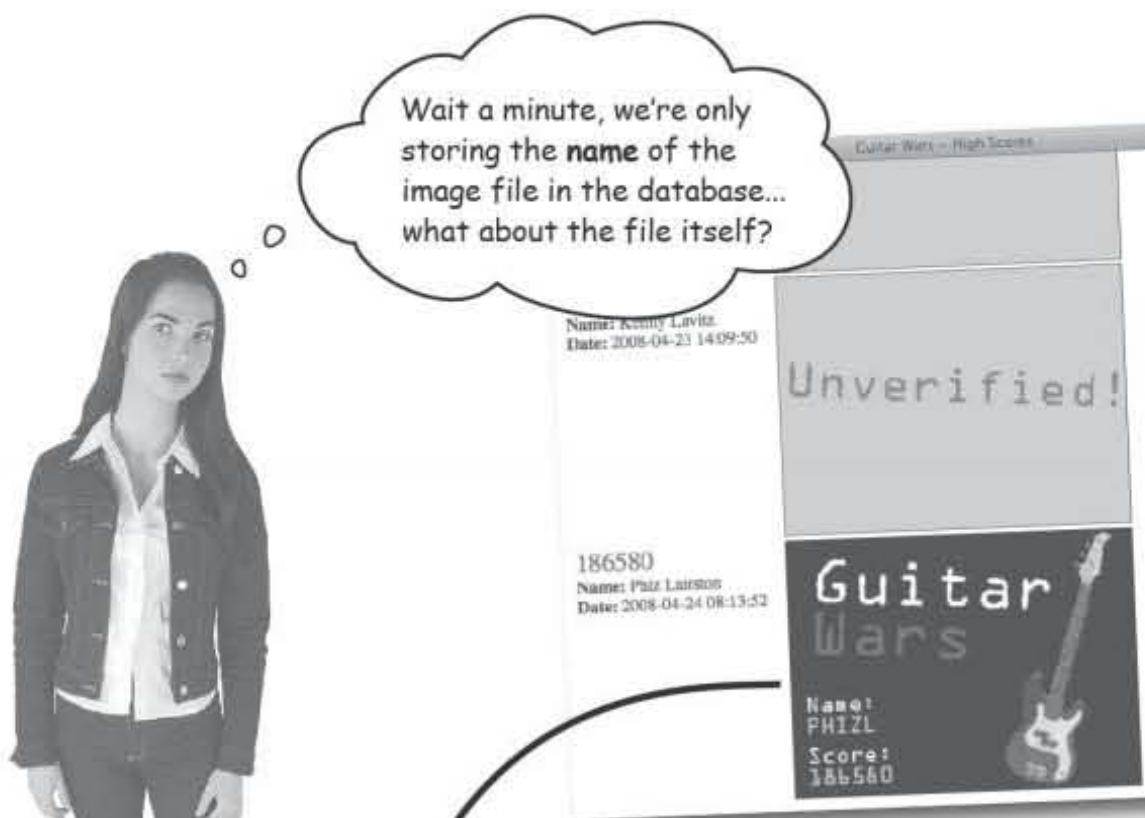
`$_FILES['screenshot']['`

`0`

The other information made available is certainly useful, but right this moment, image, which can be stored away in a location used in the SQL INSERT statement.

`$screenshot = $_FILES['s`

keep external file data in external files



Data stored in external files is typically left even in database applications.

In this case, the data is a collection of pixels that make up an image, which is stored in an external file—a GIF, JPEG, or PNG image file. Databases store references to image data, not raw binary data such as images, so it's better to just store a reference to an image in the database. This reference is the name of the image file.

Another reason images in web applications aren't stored in databases is that it would be much harder to display them using HTML code. References to images in web code references images from external files using filenames. So, for example, the `` tag in HTML involves using an image filename, not raw image data.

```

```

The image
filename

The HTML `` tag uses
the filename of an image
to reference the image file
on the web server.

**Placing an image
on a web page only
requires a reference
to the image file.**



Sharpen your pencil

The main Guitar Wars page (`index.php`) still isn't displaying images for the high scores. Finish the code so that it shows

```
<?php

// Connect to the database
$dbc = mysqli_connect('www.guitarwars.net', 'admin', 'rockit', 'gwdb');

// Retrieve the score data from MySQL
$query = ..... ;
$data = mysqli_query($dbc, $query);

// Loop through the array of score data, formatting it as HTML
echo '<table>';
while ($row = mysqli_fetch_array($data)) {
    // Display the score data
    echo '<tr><td class="scoreinfo">';
    echo '<span class="score">' . $row['score'] . '</span><br />';
    echo '<strong>Name:</strong> ' . $row['name'] . '<br />';
    echo '<strong>Date:</strong> ' . $row['date'] . '</td>';
    if (is_file( ..... ) && filesize( ..... ) > 0) {
        echo '<td></td></tr>';
    }
    else {
        echo '<td></td></tr>';
    }
}
echo '</table>';

mysqli_close($dbc);
?>
```

sharpen your pencil solution

Sharpen your pencil Solution

The main Guitar Wars page (`index.php`) still isn't displaying images for the high scores. Finish the code so that it shows them.

```
<?php
    // Connect to the database
    $dbc = mysqli_connect('www.guitarwars.net', 'admin', 'rockit', 'gwdbs');

    // Retrieve the score data from MySQL
    $query = "SELECT * FROM guitarwars";
    $data = mysqli_query($dbc, $query);

    // Loop through the array of score data, formatting it as HTML
    echo '<table>';
    while ($row = mysqli_fetch_array($data)) {
        // Display the score data
        echo '<tr><td class="scoreinfo">';
        echo '<span class="score">' . $row['score'] . '</span><br />';
        echo '<strong>Name:</strong>' . $row['name'] . '<br />';
        echo '<strong>Date:</strong>' . $row['date'] . '</td>';
        if (is_file( $row['screenshot'] ) && filesize( $row["screenshot"] ) > 0) {
            echo '<td></td>';
            } This function checks to see if a screenshot image file actually exists.
            } The screenshot column of the database contains the screen shot image for a given score.
        echo '<td></td>';
    }
    echo '</table>';

    mysqli_close($dbc);
?>
```

In a move to help simplify the code, we're using `die()` to produce error messages and exit when a `mysqli` function fails. You may want to include this code in your own applications, so you're going to skip it from here on for the sake of readability.

The SQL statement that requests the score data from the database doesn't change at all!

This function checks to see if a screenshot image file actually exists.

The screenshot column of the database contains the screen shot image for a given score.

4 Change the Guitar Wars page to show the high scores.



Test Drive

Add a new high score to Guitar Wars, complete with a screen shot image.

If you haven't already done so, download the Guitar Wars example code from the Head First Labs web site at www.headfirstlabs.com/books/hfphp. It's in the **chapter05** folder. The code consists of the main page (**index.php**), the Add Score script (**addscore.php**), and a style sheet (**style.css**).

First you need to change the **addscore.php** script so that its Add Score form supports file uploads. This includes adding new form fields, adjusting the **<form>** tag, and checking to make sure the **\$screenshot** variable isn't empty. Then incorporate the new high score **INSERT** query into the script.

Now shift to the **index.php** script, and add the new code from the facing page so that it displays the screen shot image for each high score.

Upload all of these files to your web server and open the **addscore.php** page in a web browser. Enter a new high score in the form, and click Submit. Then navigate to the **index.php** page and take a look at the new score.

368420
Name: Ashton Simpson
Date: 2008-04-23 09:13:34

64930
Name: Kenny Lavitz
Date: 2008-04-23 14:09:50

186580
Name: Phiz Lairston
Date: 2008-04-24 08:13:52

Something isn't right! The image doesn't appear with the new score as expected.



Why do you think the screen shot didn't show up for the new score? What about the scores that were already in the database?

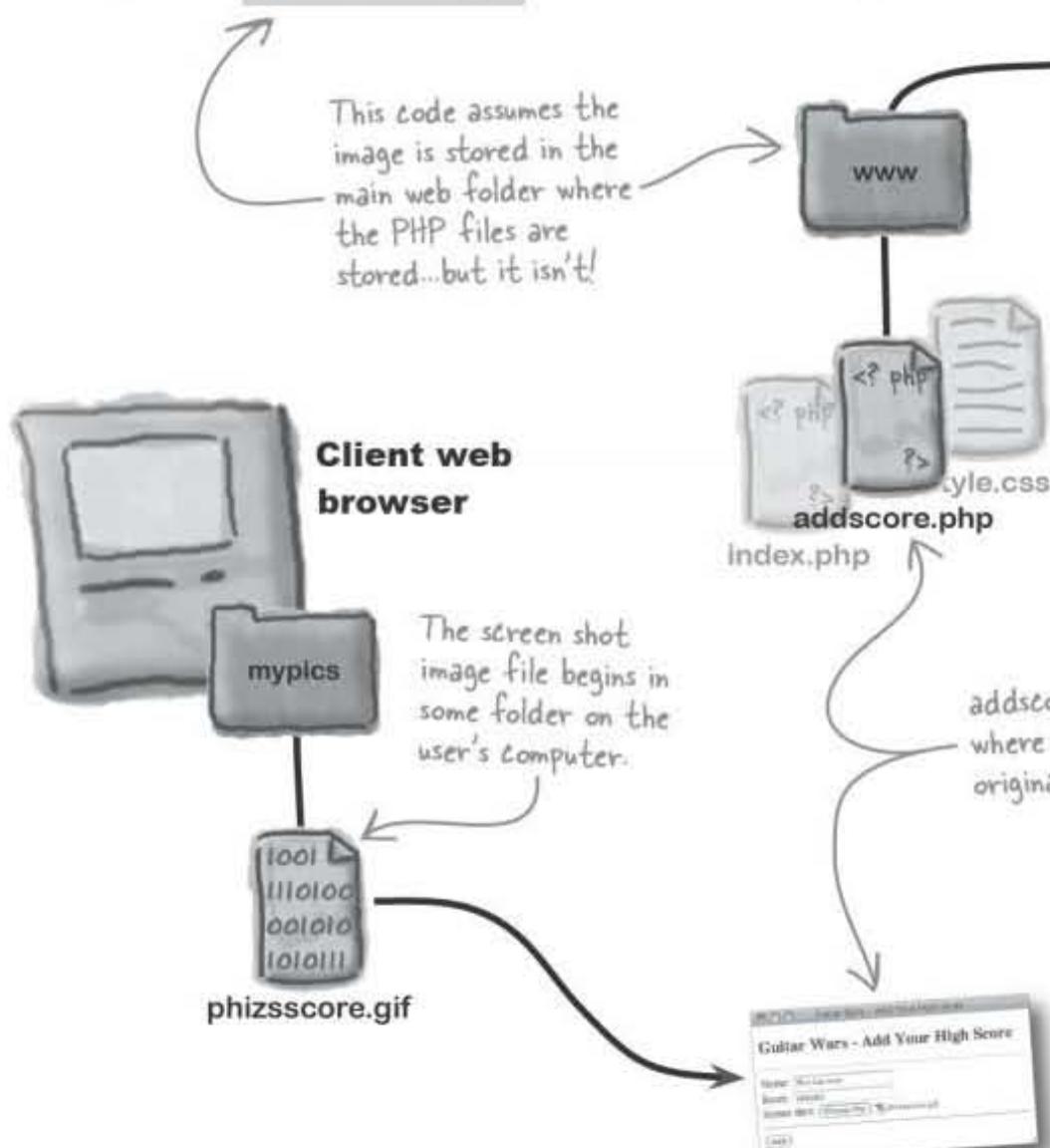
uploaded files are stored in a temporary folder

Where did the uploaded file go?

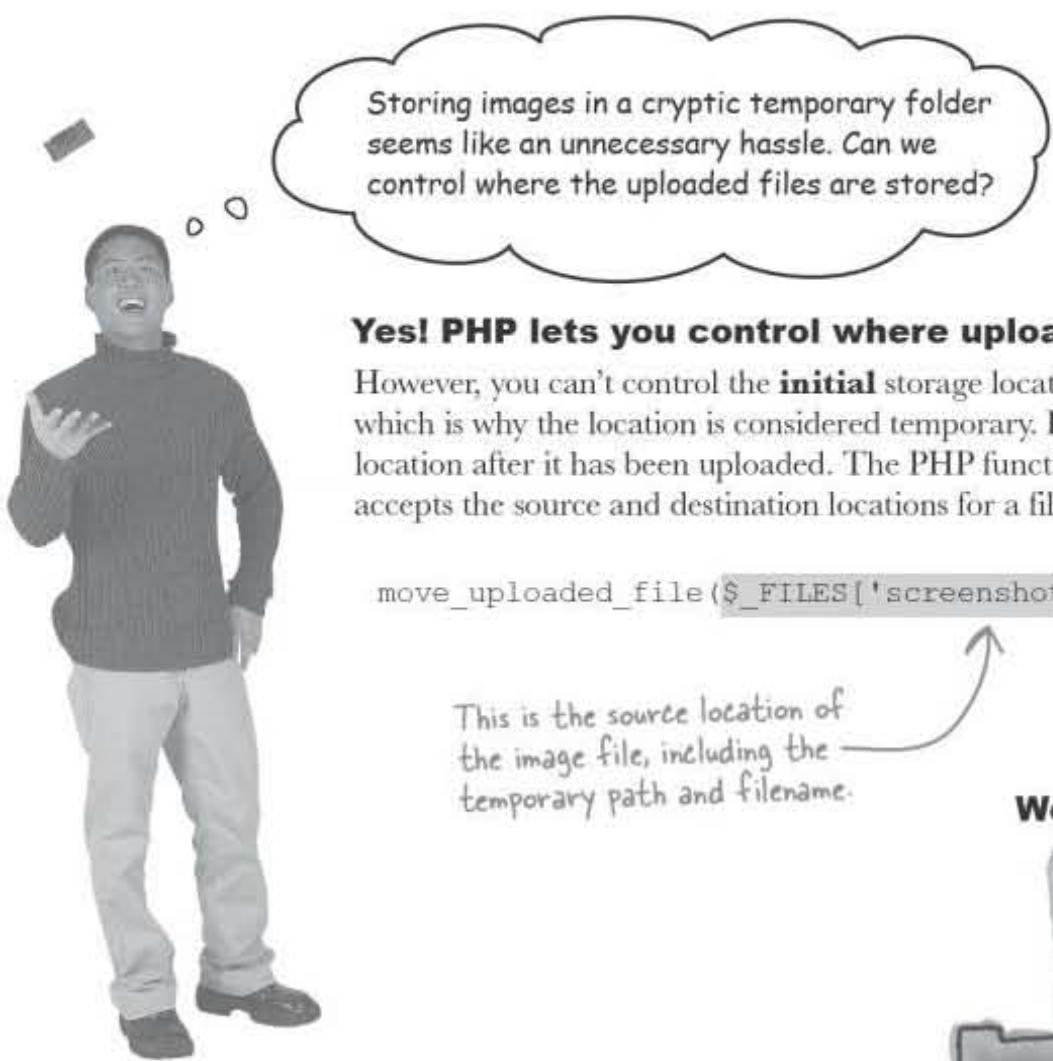
The problem with the uploaded image not appearing is that we made an assumption that the file would be uploaded to the same folder on the web server as our PHP scripts. As it turns out, this assumption is dead wrong. The Add Score form lets the user select a file from their own computer, but the file is actually uploaded to a **temporary folder** on the server. The temporary folder is created automatically on the server and usually has a weird name with a bunch of random letters and numbers.

This presents a problem for our `` code in `index.php` because it assumes the image is located in the main web folder:

``



root
This temporary folder name, location of, varies with PHP instance
Client web browser
mypics
phizsscore.gif
The screen shot image file begins in some folder on the user's computer.
index.php
addscore.php
style.css
<? php
>?>
The Add Score takes care of image file to folder on the server.



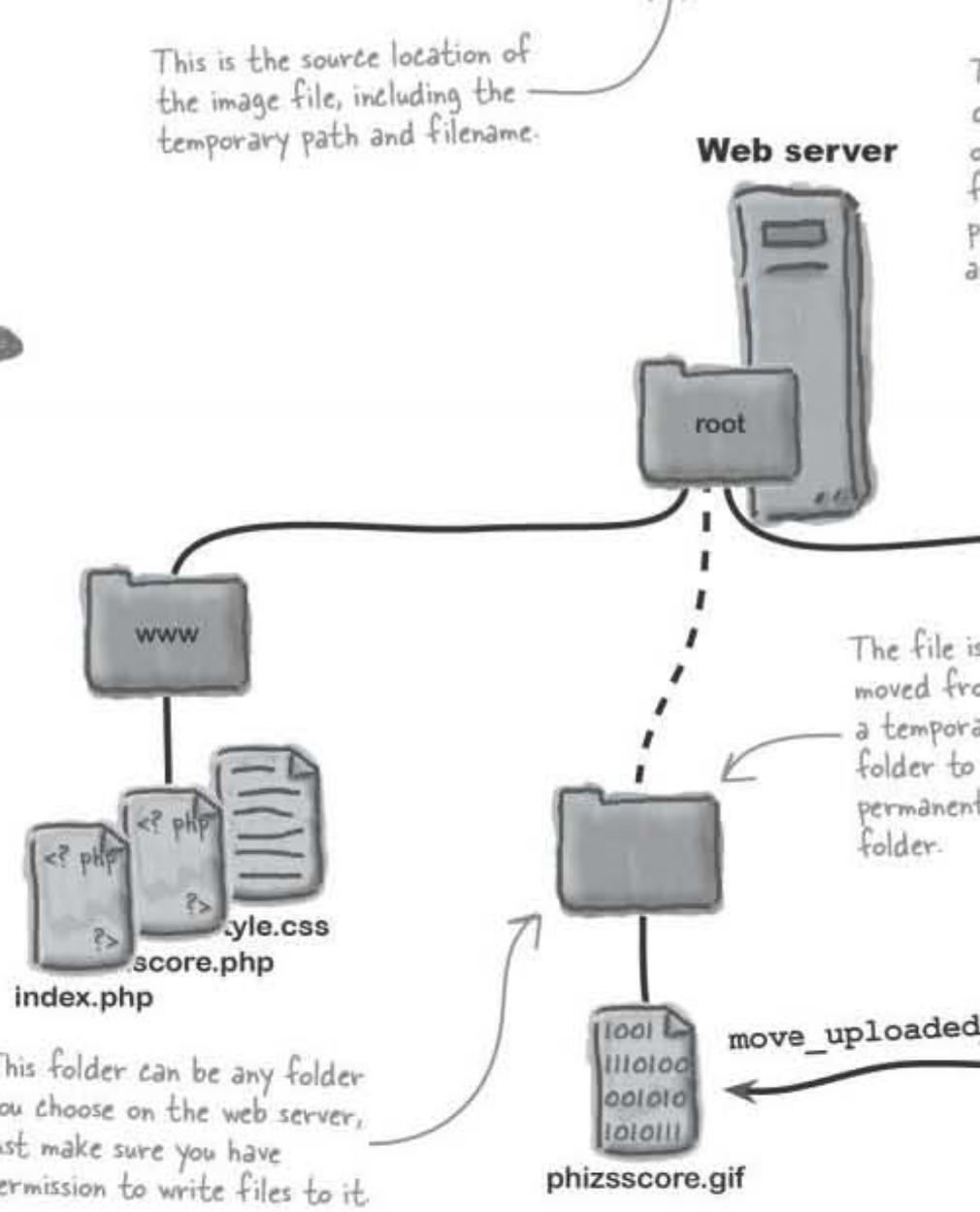
Yes! PHP lets you control where uploaded files are stored.

However, you can't control the **initial** storage location of uploaded files, which is why the location is considered temporary. But you can move a file to a permanent location after it has been uploaded. The PHP function `move_uploaded_file` accepts the source and destination locations for a file, and then takes care of the rest.

```
move_uploaded_file($_FILES['screenshot']['tmp_name'],
```

This is the source location of the image file, including the temporary path and filename:

Web server



no dumb questions

there are no **Dumb Questions**

Q: Can't I change the initial storage location of uploaded files by modifying the `php.ini` file?

A: Yes. The PHP initialization file (`php.ini`) can be used to change the initial storage location of uploaded files through the `upload_tmp_dir` option. But if your application is hosted on a virtual server, you may not have access to this file, which means you'll have to move the file to your own folder via PHP script code.

Q: Why is the initial upload folder called a "temporary" folder? Does it go away after a file is moved?

A: No. The folder is "temporary" in a sense that it isn't intended to serve as the final storage location for uploaded files. You can think of it as a holding area where uploaded files are stored until they are moved to their final storage location.

Q: Why can't I just leave a file in the temporary folder?

A: You can, in which case you'd need to add `$_FILES['Screenshot']['tmp_name']` to the path of the image to make sure it is found in the temporary folder. But keep in mind that you don't typically control the name or location of the folder. Even more important is the fact that temporary folders can be automatically emptied periodically on some systems. Another potential issue is that the temporary upload folder may not be publicly accessible, so you won't be able to reference uploaded files from HTML code, which is the whole point in *Guitar Wars* and most other PHP applications. By moving uploaded files out of the temporary upload folder, you can carefully control exactly where they are stored and how they are accessed.



OK, so now I know how to move uploaded files around. That's really special. But I still don't have a clue where they are supposed to go.

Every application needs an `images` folder.

OK, maybe "need" is a bit strong, but it's important to organize the pieces and parts of PHP applications as much as possible, and one way to do so is to create folders for different components. Since uploaded images are submitted by users, they aren't something you typically have direct control over, at least in terms of filenames and quantity. So it's a good idea to store them separately from other application files.

All this said, we need an `images` folder, where image files that are uploaded to the Guitar Wars application are stored. This folder can also serve as the storage location for any other images the application may use, should the need arise.



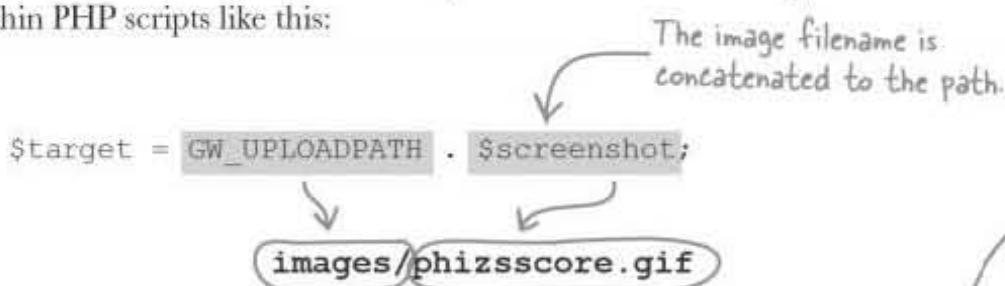
The `images` folder isn't actually any bigger than other folders, but it helps organize image files into one place.

make a folder for uploaded images

Create a home for uploaded image files

The images folder is just like any other folder on the web server except that it must be placed somewhere beneath the main web folder for the application. It's usually fine to just place the folder directly beneath this web folder, but you're free to create a more complex folder hierarchy if you want.

With the images folder created immediately beneath the main web folder on the web server, it becomes possible to reference image files from within PHP scripts like this:



The \$target path is built out of a new constant we're going to add to the script called GW_UPLOADPATH, which holds the path to our images folder. Like a variable, a **constant** stores a piece of data. But the value of a constant can't change once it's set. The image filename as entered into the Add Score form is then concatenated to the images path.



If your PHP application is hosted anywhere other than your local computer, you'll need to use FTP to create the images folder.

Use an FTP program to access the file system of your web site and create the images folder beneath the web folder of the application.

This is the web application where files are stored, including:

The images folder is typically just beneath the web folder.

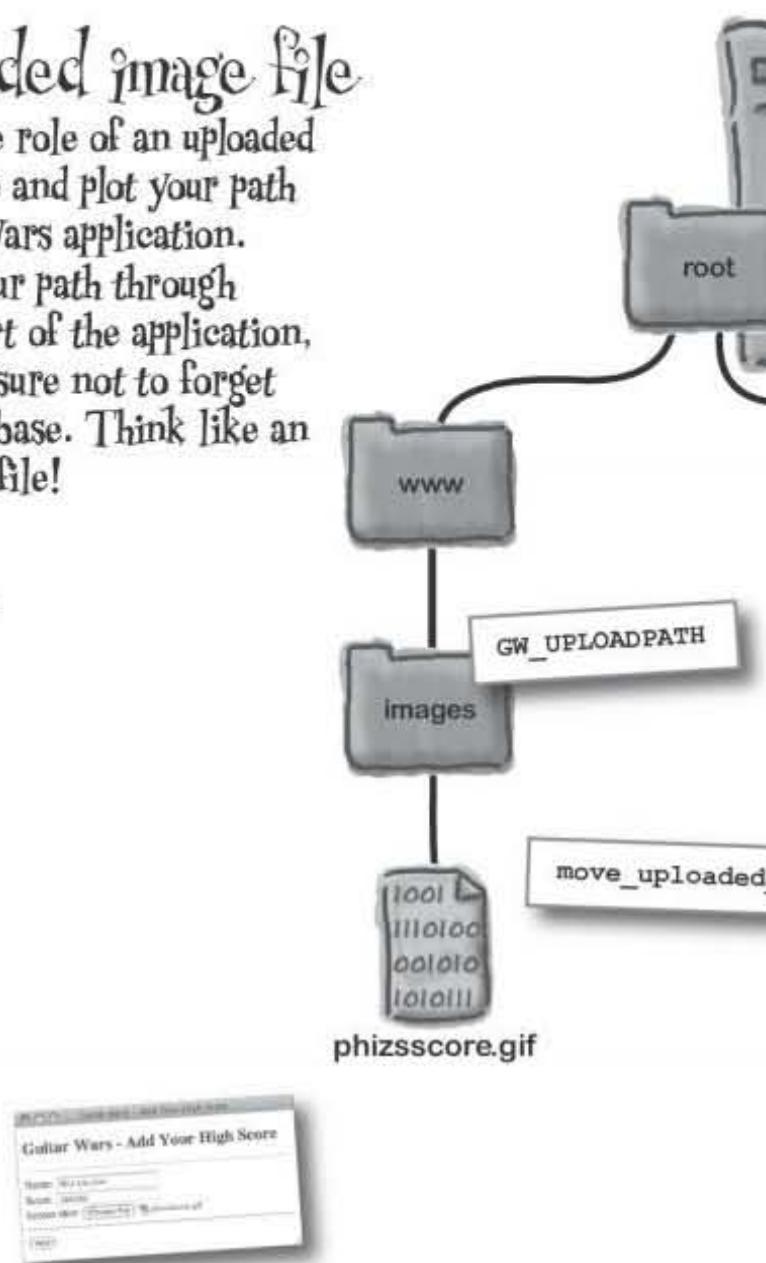
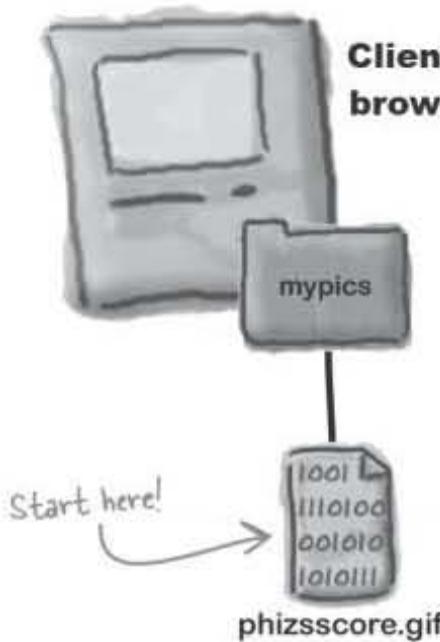
Uploaded image files are moved to the images folder, where they can be displayed via HTML tags.

move_uploaded_file
\$_FILES['screenshot']
\$target);

BE the uploaded image file

Your job is to play the role of an uploaded screen shot image file and plot your path through the Guitar Wars application.

Draw your path through each part of the application, making sure not to forget the database. Think like an uploaded file!



```
$screenshot = $_FILES['screenshot']['name'];
```

```
INSERT INTO guitarwars VALUES (0, NOW(), '$name', '$score', '$screenshot')
```

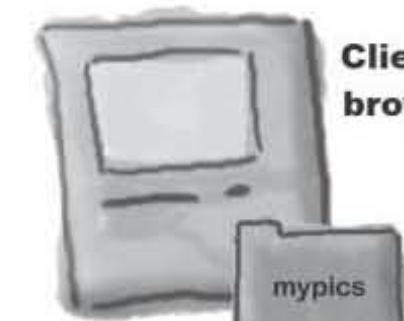
be the uploaded image file solution

BE the uploaded image file solution



Your job is to play the role of an uploaded screen shot image file and plot your path through the Guitar Wars application.

Draw your path through each part of the application, making sure not to forget the database. Think like an uploaded file!



Client web browser

This folder is on the user's computer - you have no control over what it's called or where it's stored, and neither do you care.

You care a lot about the name and location of this folder because it is used throughout Guitar Wars to store and reference uploaded image files.

First, the file is uploaded using a file input form field.

```
$screenshot = $_FILES['screenshot']['name'];
```

```
INSERT INTO guitarwars VALUES (0, NOW(), '$name', '$score', '$screenshot')
```

Whew!

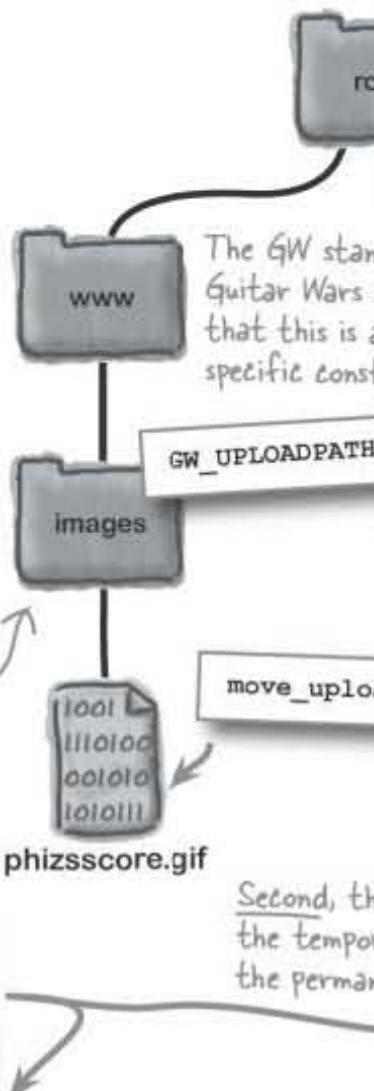
Yep, this is a new step we didn't plan on earlier - your design must be flexible!



addscore.php

5

DONE
Move the uploaded image file from a temporary upload folder to a permanent folder for images.



The GW stands for Guitar Wars, and this is a specific constant.

GW_UPLOADPATH

move_uploaded_file

Second, the temporary file is moved to the permanent images folder.

phizsscore.gif

Second, the temporary file is moved to the permanent images folder.

id	date	name
1	2008-04-22 14:37:34	Paco Jastorius
2	2008-04-22 21:27:54	Nevil Johansson
3	2008-04-23 09:06:35	Eddie Vanille
4	2008-04-23 09:12:53	Belito Chevy
5	2008-04-23 09:13:34	Ashton Simpson
6	2008-04-23 14:09:50	Kenny Lovitz
7	2008-04-24 08:13:52	Phiz Lairston



Test Drive

Give uploaded screen shot images a permanent home in the image folder.

Modify the `addscore.php` script to use the `GW_UPLOADPATH` constant and move screen shot images in the path it points to. Here's a peek at the code that needs to change:

```
<?php
    // Define the upload path and maximum file size constants
    define('GW_UPLOADPATH', 'images/');

    if (isset($_POST['submit'])) {
        // Grab the score data from the POST
        $name = $_POST['name'];
        $score = $_POST['score'];
        $Screenshot = $_FILES['Screenshot']['name'];

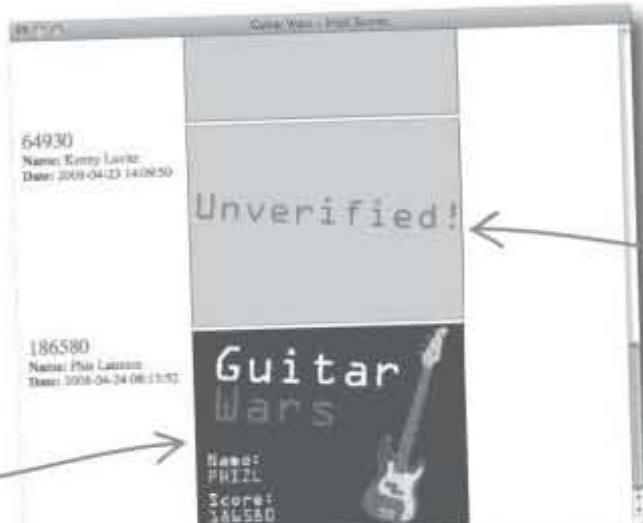
        if (!empty($name) && !empty($score) && !empty($Screenshot)) {
            // Move the file to the target upload folder
            $target = GW_UPLOADPATH . $Screenshot;
            if (move_uploaded_file($_FILES['Screenshot']['tmp_name'], $target)) {
                // Connect to the database
                $dbc = mysqli_connect('www.guitarwars.net', 'admin', 'rockit', 'gwdb');

                // Write the data to the database
                $query = "INSERT INTO guitarwars VALUES (0, NOW(), '$name', '$score', '$Screenshot')";
                mysqli_query($dbc, $query);

                // Confirm success with the user
                echo '<p>Thanks for adding your new high score!</p>';
                echo '<p><strong>Name:</strong> ' . $name . '<br />';
                echo '<strong>Score:</strong> ' . $score . '<br />';
                echo '</p>';
                echo '<p><a href="index.php">xit</a> Back to high scores</a></p>';
            }
        }
    }
}
```

The `index.php` script is also affected by the `GW_UPLOADPATH` constant. Don't change it as well. After making these changes, upload the scripts to your server and add a high score again.

The uploaded screen shot image is now visible on the main page.



more no dumb questions

there are no Dumb Questions

Q: If the `php.ini` file can be used to control the storage location of uploaded files, why is it necessary to move the file?

A: Because it isn't always possible to change `pnp.ini`. For example, if you're building a PHP application on a virtual web server, you very likely won't be able to change the settings in `php.ini`. And even if you are able to change `php.ini`, you run the risk of breaking your application if you ever need to move it to another server. In other words, the application will be dependent on a path controlled by `php.ini`, as opposed to a path controlled by your own PHP code.

Q: Why isn't the date something that users can enter in Guitar Wars?

A: The date is an important part of a high score in that it establishes when a score was officially posted to the site. Like any record, the first person to achieve a certain score gets all the glory. Rather than trust a user to tell us when they achieved their high score, we can just use the post date/time as the official recording of the score. This eliminates bogus dates and lends more credibility to the high score list. Users of such a competitive application will always be looking for an angle, so eliminate as many of them as you can!

It is worth pointing out that the `NOW()` function uses the time on the web server, which may not be the same as the user's local time. This shouldn't be a problem, however, since all users are held to that same server time.

Databases are great for storing text data, but it's usually better for them to reference binary data in external files.

Q: Isn't it possible for people to overwrite each other's screen shot images by uploading image files with the same names?

A: Yes. The problem has to do with the fact that the screen shot image stored on the web server uses the exact same filename provided by the user in the file upload form field. So if two users upload image files with the same filenames, the first user's image will get overwritten by the second user's image. Not good. One solution is to add a degree of uniqueness to the image filename on the server. A simple way to do this is to add the current server time, in seconds, to the front of the filename, like this:

```
Starget = GW_UPLOADPATH . time()
$Screenshot;
```

The result of this code is a filename of `1221634560phizsscore.gif` instead of `phizsscore.gif`, where `1221634560` is the current time on the server expressed in seconds.

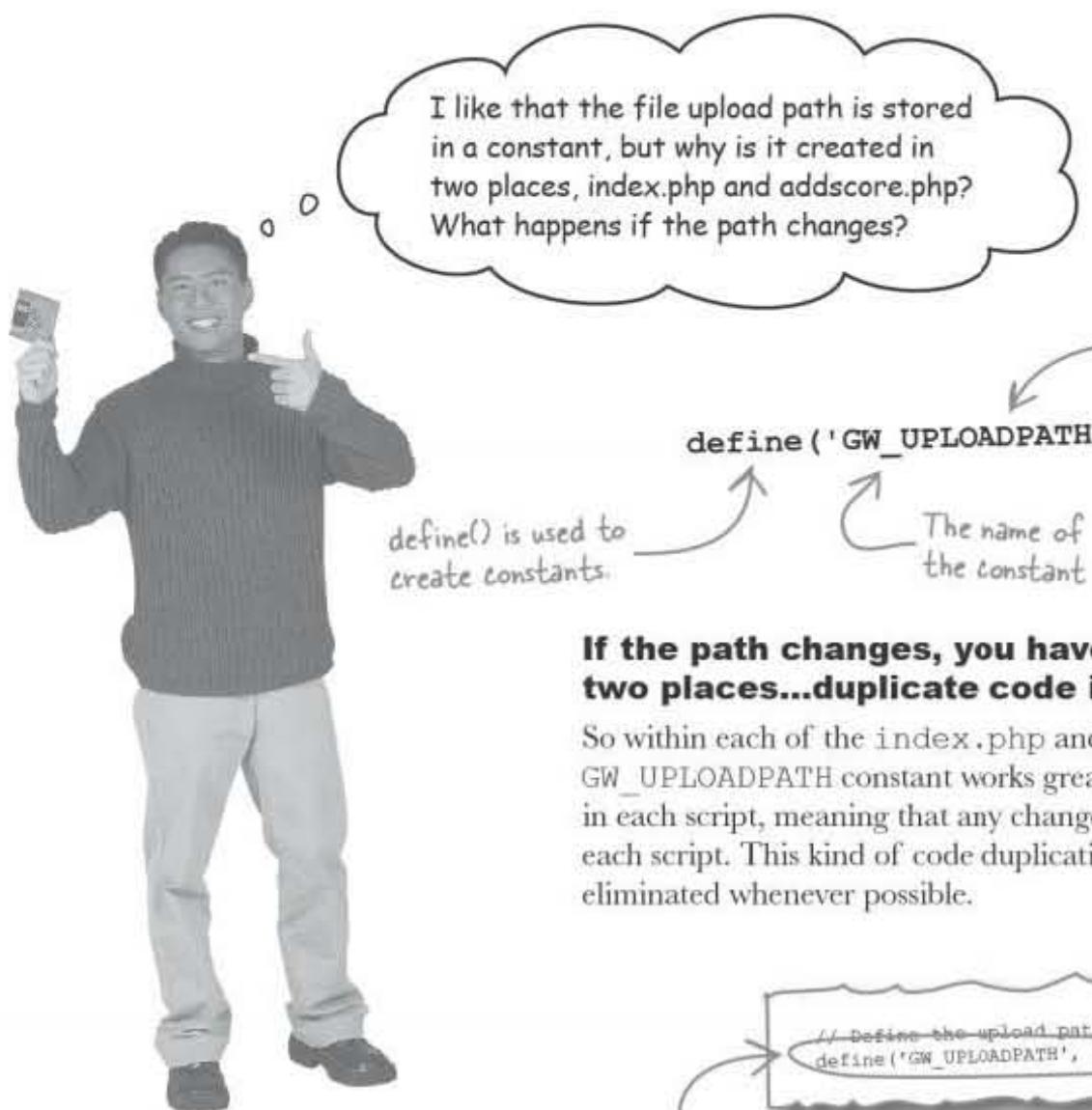
Q: Could we image data for screen shot in t

A: Yes. Datab allow you to stor However, the big Guitar Wars uses HTML code so th on the main index tag is d image file stored a chunk of binar a database. So e guitarwar data, you'd be fa trying to get the can be displayed

There's nothin about the tim time() functio that it sourc number it ret

BULLET POINTS

- The `ALTER` statement is used to change the structure of a MySQL database by adding a new column of data.
- With a little help from PHP and the `<input>` tag can be used to store data.
- The superglobal variable `$HTTP_POST_VARS` PHP stores information about the variables sent to the server.
- The standard PHP function `file()` allows you to move files between the web server and is critical for reading and writing files.
- Most web applications benefit from having a separate images folder for storing images used by the application, especially those that involve file uploads.



If the path changes, you have to change two places...duplicate code is a bad idea!

So within each of the index.php and addscore.php files, the GW_UPLOADPATH constant works great. But the code is repeated in each script, meaning that any change in the path would need to be made in each script. This kind of code duplication is bad design. It's better to eliminate whenever possible.

The constant is stored twice, meaning it has to be maintained in two different places.



To solve the duplicate code problem, we need to store the GW_UPLOADPATH constant in a single place. Would you store it in index.php or addscore.php? Why?

```
// Define the upload path constant
define('GW_UPLOADPATH', 'images/');
```

```
// Define the upload path and maximum file size
define('GW_UPLOADPATH', 'images/');
define('GW_MAXFILESIZE', 32768);
```

sharing script data with include files

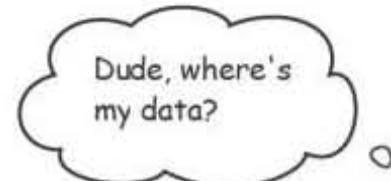
Shared data has to be shared

When it comes to data that is shared across multiple scripts in an application, you need a way to store the data in one place and then pull it into the different scripts. But that still doesn't answer the question of where exactly the data should go...?

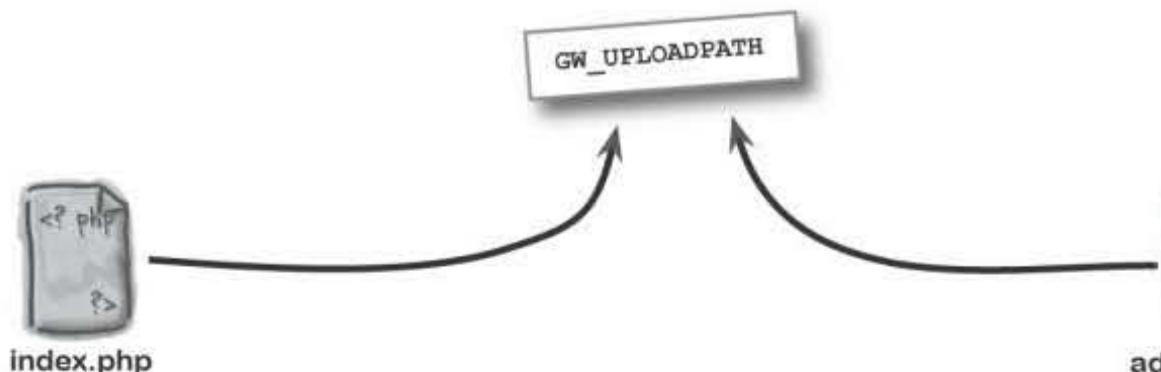
You could store the data only in index.php....



...but then other scripts wouldn't have it



So storing shared script data in an existing script file doesn't really work because the data isn't really shared any more. The answer lies in somehow making the data accessible to multiple scripts but without directly storing it in any of them.

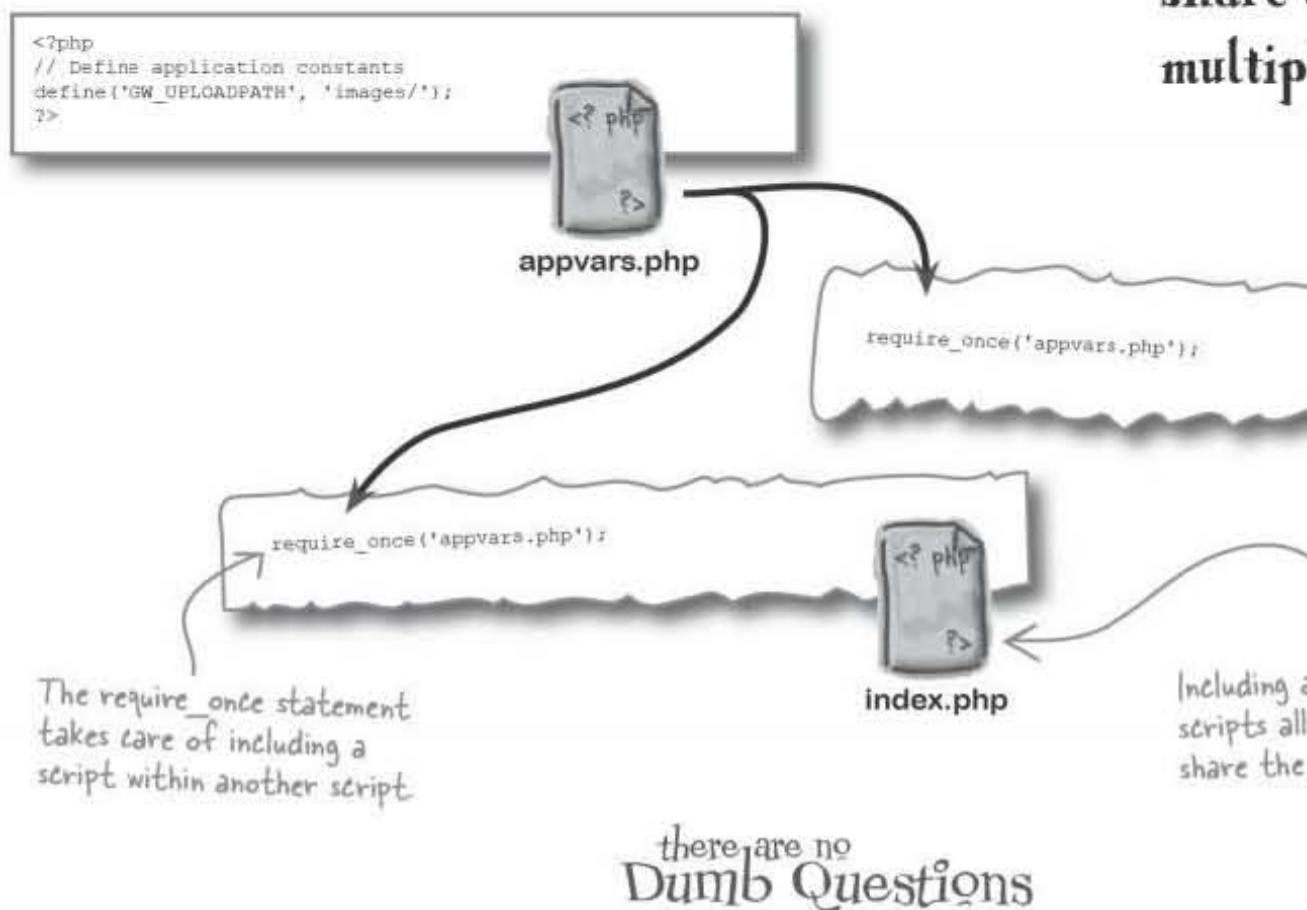


How can you make the data accessible to both scripts without storing it in either of them?

The solution to shared script data lies in **include files**, source code files that are inserted into other PHP files.

Shared script data is required

Include files are very powerful because you create them once but then reuse them as needed in other script files, effectively sharing the code within. The GW_UPLOADPATH constant can be placed within an include file to establish a collection of "application variables."



Q: Hey, aren't these application "variables" really constants?

A: Sometimes, yes. But that's OK. The point is not to split hairs over variables vs. constants. Instead, we're just trying to establish a common place to store shared script data within a given application. And that place is a script file called `appvars.php`.

Q: Is code in shared script files limited to data?

A: No, not at all. Any PHP code can be placed in its own script file and shared using the `require_once` statement. In fact, it's very common for applications to share lots of functional code across multiple script files. Not only is it common to use shared script files, but it's often a great idea in terms of code organization.

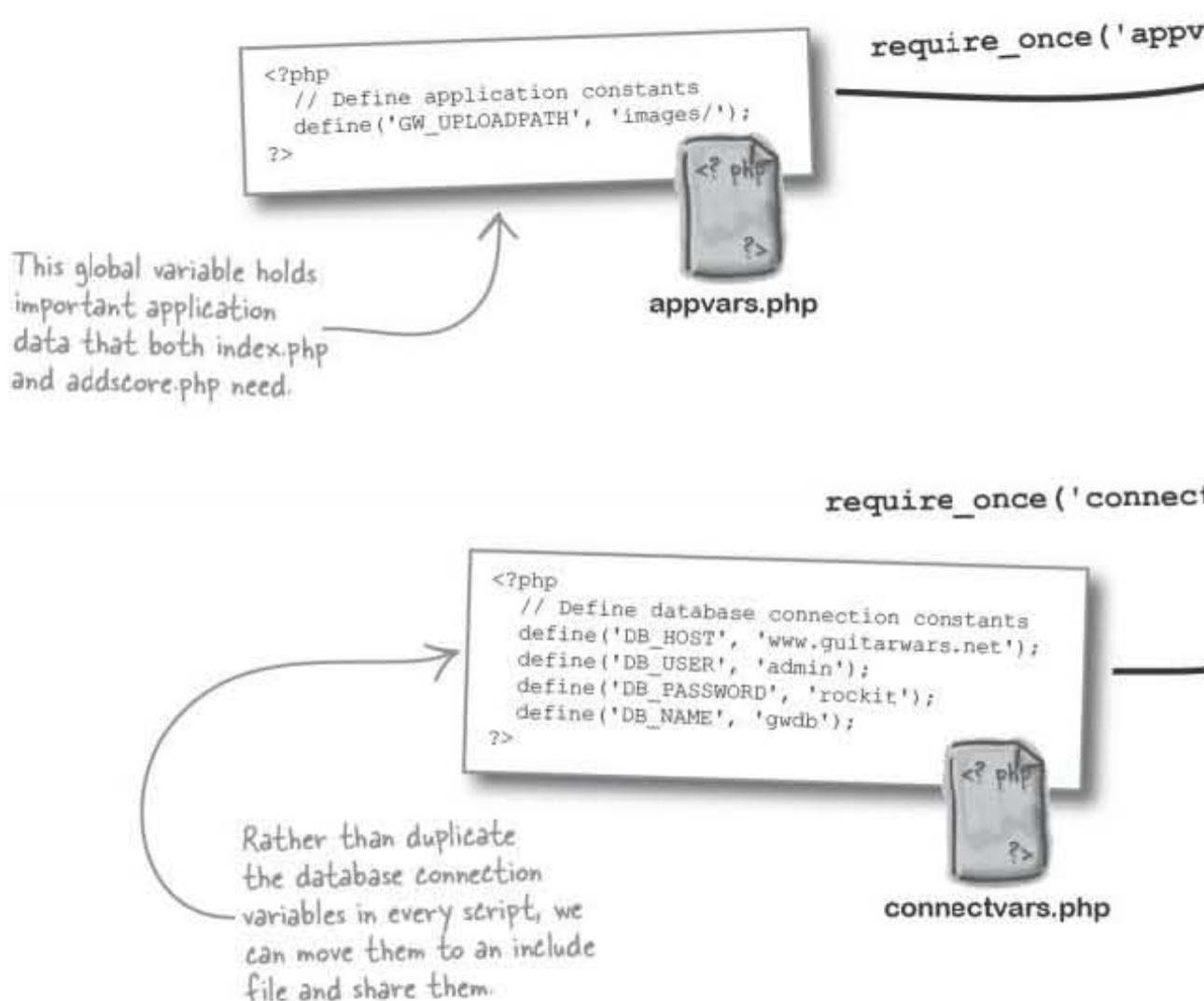
Q: Why is the PHP statement to include files called `require_once`?

A: The name "include file" comes from the original `include` statement that is very similar to `require`. The main difference is that `require_once` requires that the included file be included only once. If an include file cannot be found—`include` will just ignore it—but if an include file is missing. Also, the "once" part of the name means that it keeps a file from being accessed more than once. You sometimes see `include` and `require` used interchangeably to include code that is mostly just as pure HTML code that doesn't perform any logic. PHP also has `include_once` and `require_once` variations on `require_once` and `include`.

the require_once statement

Think of `require_once` as "insert"

Includes aren't limited to one shared PHP file, and they can appear anywhere you want within a script. You can think of the `require_once` statement as an "insert" statement that gets replaced with the contents of the script file it references. In the case of Guitar Wars, the database connection variables could also benefit from being moved into an include file. So the contents of two shared script files are inserted directly into other script files at the points where they are required.



The REQUIRE_ONCE statement inserts shared script code into other scripts.

```

<?php

// Define application constants
define('GW_UPLOADPATH', 'images/');
define('DB_HOST', 'www.guitarwars.net');
define('DB_USER', 'admin');
define('DB_PASSWORD', 'chiefrocker');
define('DB_NAME', 'guitarwarsdb');

// Connect to the database
$dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

// Retrieve the score data from MySQL
$query = "SELECT * FROM guitarwars";
$data = mysqli_query($dbc, $query);

// Loop through the array of score data, formatting it as HTML
echo '<table>';
while ($row = mysqli_fetch_array($data)) {
    // Display the score data
    echo '<tr><td class="scoreinfo">';
    echo '<span class="score">' . $row['score'] . '</span><br />';
    echo '<strong>Name:</strong> ' . $row['name'] . '<br />';
    echo '<strong>Date:</strong> ' . $row['date'] . '</td>';
    if (is_file(GW_UPLOADPATH . $row['Screenshot']) &&
        filesize(GW_UPLOADPATH . $row['Screenshot']) > 0) {
        echo '<td></td></tr>';
    }
    else {
        echo '<td></td></tr>';
    }
}
echo '</table>';

mysqli_close($dbc);
?>
```



index.php



Test Drive

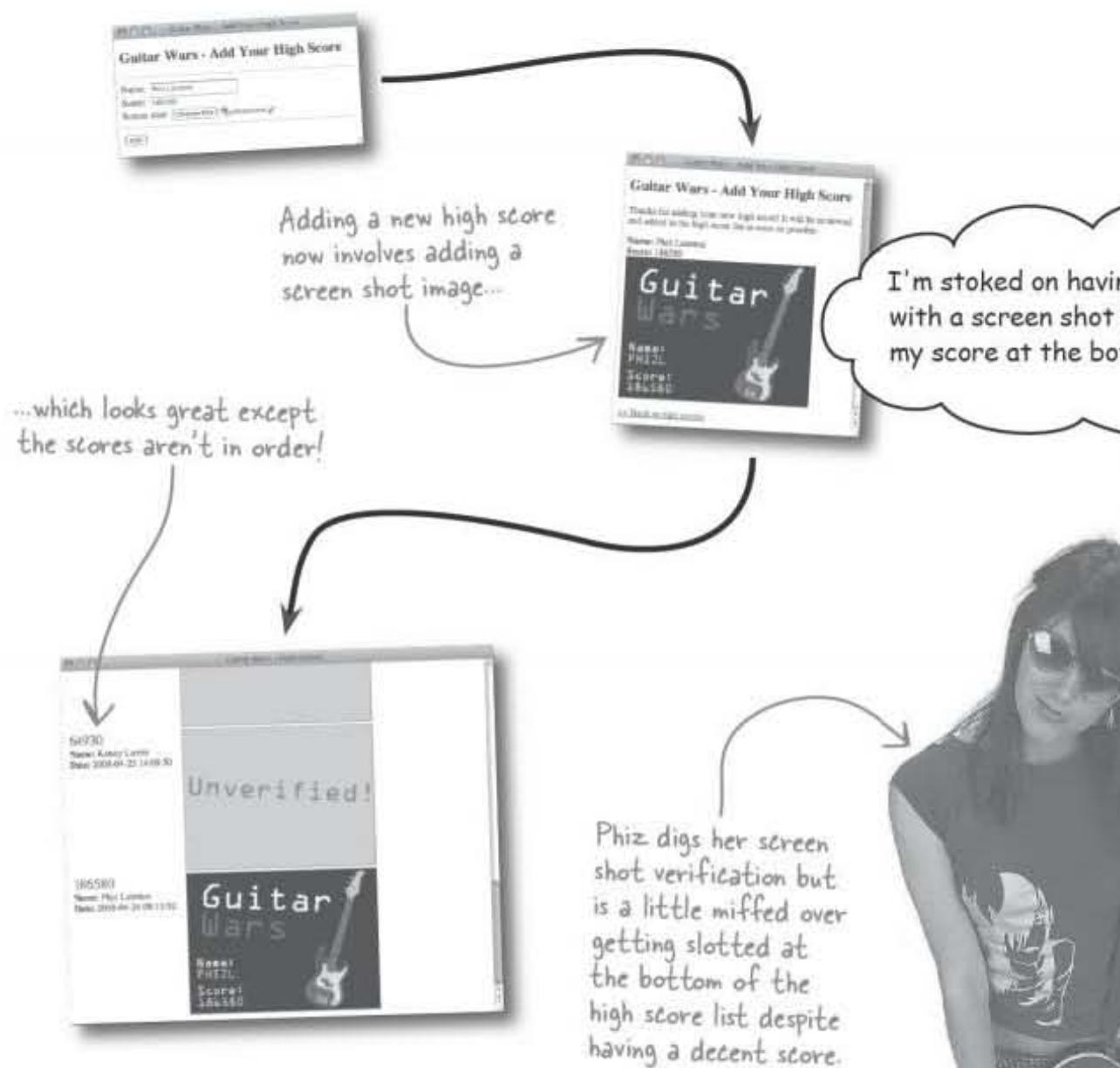
Create two include files for Guitar Wars, and then share the other scripts.

Create two new text files, appvars.php and connectvars.php, and upload them shown on the facing page. Then add require_once statements to addscore.php so that both shared script files are included. Upload all files to your web server and try out the Add Score form and main page to make sure the new and improved include file organizational structure works.

the ORDER BY statement

Order timing is everything with high scores

Guitar Wars is finally image-powered, allowing users to upload screen shot images to help verify their high scores. While this is a major improvement to the application, it hasn't solved a problem that users have actually been grumbling about for quite a while—the order of the scores on the main page.



It's true, the scores aren't in order. They are being displayed in whatever order they're stored in the database, which is entirely arbitrary. You should never rely on the order that data is stored in a database unless order truly doesn't matter. In this case it does, so we need to impose some order on the query results. The ORDER BY SQL statement makes such ordering possible.





PHP & MySQL Magnets

See if you can figure out how **ORDER BY** works by using the magnets below to create **SELECT** statements that result in the output below. Also circle which query you think is the best fix for Guitar Wars. Hint: **ASC** stands for ASCending and **DESC** stands for DESCending.

```
File Edit Window Help YYZ
mysql>

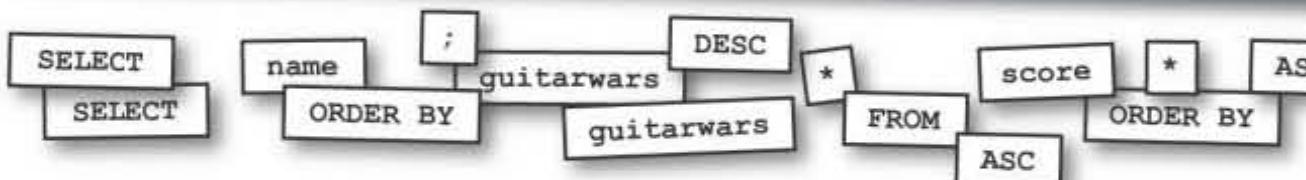
+----+-----+-----+-----+-----+
| id | date | name | score | screenshot |
+----+-----+-----+-----+-----+
| 5  | 2008-04-23 09:13:34 | Ashton Simpson | 368420 |
| 4  | 2008-04-23 09:12:53 | Belita Chevy | 282470 |
| 3  | 2008-04-23 09:06:35 | Eddie Vanilli | 345900 |
| 6  | 2008-04-23 14:09:50 | Kenny Lavitz | 64930 |
| 2  | 2008-04-22 21:27:54 | Nevil Johansson | 98430 |
| 1  | 2008-04-22 14:37:34 | Paco Jastorius | 127650 |
| 7  | 2008-04-24 08:13:52 | Phiz Lairston | 186580 |
+----+-----+-----+-----+-----+
7 rows in set (0.0005 sec)
```

The query results are returned in descending numerical order by score, and then in ascending order by date.

The query results are returned in ascending alphabetical order by name.

```
File Edit Window Help YYZ
mysql>

+----+-----+-----+-----+-----+
| id | date | name | score | screenshot |
+----+-----+-----+-----+-----+
| 5  | 2008-04-23 09:13:34 | Ashton Simpson | 368420 |
| 3  | 2008-04-23 09:06:35 | Eddie Vanilli | 345900 |
| 4  | 2008-04-23 09:12:53 | Belita Chevy | 282470 |
| 7  | 2008-04-24 08:13:52 | Phiz Lairston | 186580 |
| 1  | 2008-04-22 14:37:34 | Paco Jastorius | 127650 |
| 2  | 2008-04-22 21:27:54 | Nevil Johansson | 98430 |
| 6  | 2008-04-23 14:09:50 | Kenny Lavitz | 64930 |
+----+-----+-----+-----+-----+
7 rows in set (0.0005 sec)
```



php & mysql magnets solution



PHP & MySQL Magnets Solution

See if you can figure out how **ORDER BY** works by using the magnets below to create **SELECT** statements that result in the output below. Also circle which query you think is the best fix for Guitar Wars. Hint: **ASC** stands for **ASC**ending and **DESC** stands for **D**escending.

```
File Edit Window Help YYZ
mysql> SELECT * FROM guitarwars ORDER BY name ASC ;
+----+-----+-----+-----+-----+
| id | date | name | score | screenshot |
+----+-----+-----+-----+-----+
| 5  | 2008-04-23 09:13:34 | Ashton Simpson | 368420 |
| 4  | 2008-04-23 09:12:53 | Belita Chevy | 282470 |
| 3  | 2008-04-23 09:06:35 | Eddie Vanilli | 345900 |
| 6  | 2008-04-23 14:09:50 | Kenny Lavitz | 64930  |
| 2  | 2008-04-22 21:27:54 | Nevil Johansson | 98430 |
| 1  | 2008-04-22 14:37:34 | Paco Jastorius | 127650 |
| 7  | 2008-04-24 08:13:52 | Phiz Lairston | 186580 |
+----+-----+-----+-----+-----+
7 rows in set (0.0005 sec)
```

The query results are returned in descending numerical order by score, and then in ascending order by date.

The query results are returned in ascending alphabetical order by name.

```
File Edit Window - help + F2
mysql> SELECT * FROM guitarwars ORDER BY score DESC , name ASC ;
+----+-----+-----+-----+-----+
| id | date | name | score | screenshot |
+----+-----+-----+-----+-----+
| 5  | 2008-04-23 09:13:34 | Ashton Simpson | 368420 |
| 3  | 2008-04-23 09:06:35 | Eddie Vanilli | 345900 |
| 4  | 2008-04-23 09:12:53 | Belita Chevy | 282470 |
| 7  | 2008-04-24 08:13:52 | Phiz Lairston | 186580 |
| 1  | 2008-04-22 14:37:34 | Paco Jastorius | 127650 |
| 2  | 2008-04-22 21:27:54 | Nevil Johansson | 98430 |
| 6  | 2008-04-23 14:09:50 | Kenny Lavitz | 64930  |
+----+-----+-----+-----+-----+
7 rows in set (0.0005 sec)
```

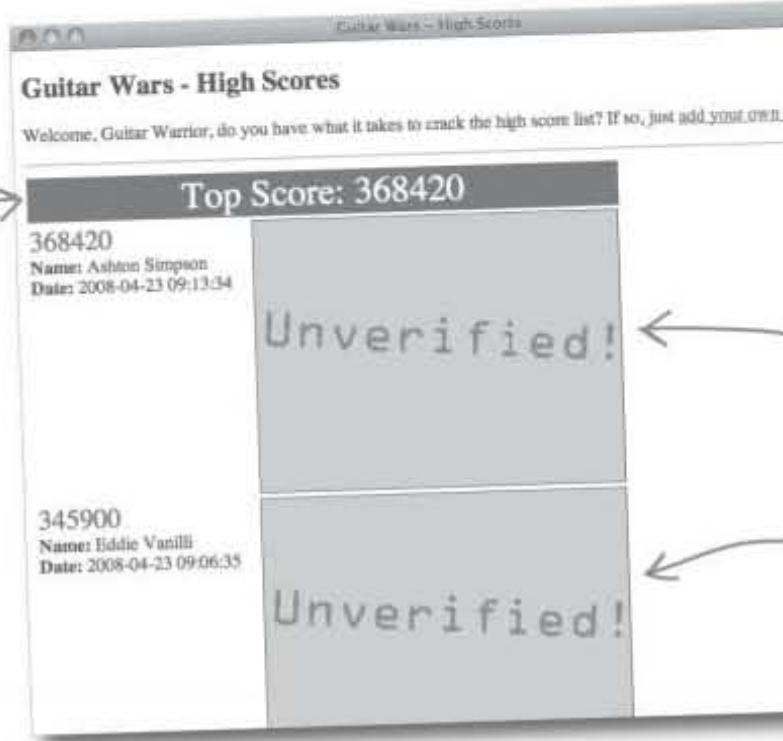
The ordering by date is secondary and only applies when there are two identical scores, which doesn't happen here but is likely in a large enough data set.

The ordering by date is secondary and only applies when there are two identical scores, which doesn't happen here but is likely in a large enough data set.

Honoring the top Guitar Warrior

With the order of the scores fixed, it's now possible to make an unexpected improvement to the high score list by calling out the highest scorer at the top of the list. The top scoring Guitar Warrior deserves a top score header that clearly displays the highest score, so there is no doubt who the top Guitar Warrior is... and what score to gun for.

A top score header clearly highlights the top score, providing a target for competing Guitar Warriors.



*there are no
Dumb Questions*

Q: Many of the scores are still unverified? Isn't that a problem?

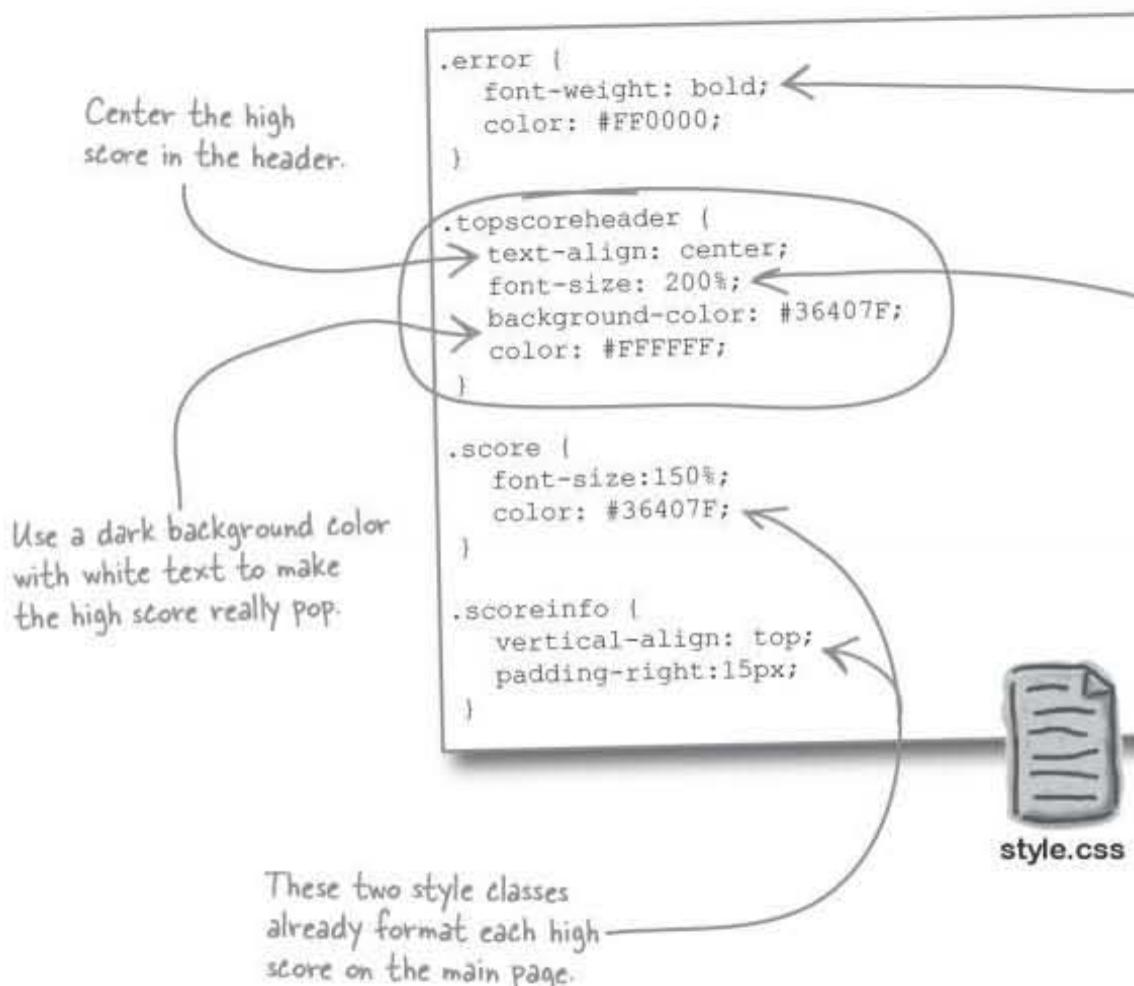
A: Yes it is. But it doesn't stop us from going ahead and calling attention to the top score. It just means that we'll need to eventually clean up the high score list by removing unverified scores. In fact, we'll tackle the unverified high scores just as soon as we finish highlighting the top score.

adding a little CSS

Format the top score with HTML and CSS

The most important thing about the new high score header is that it be clearly seen above all other scores in the high score list. This requires the help of both HTML and CSS to add some visual flair. The header will be generated as a row in the HTML table with a special CSS style applied to it. This style, `topscoreheader`, must be added to the `style.css` stylesheet for Guitar Wars.

This style class used to highlight errors in the A



The `index.php` script already generates an HTML table containing the high score list. Generating a header just for the top score involves isolating the first score, which is guaranteed to be the top score since the list is now in order. A while loop takes care of looping through the scores, so we need to somehow count the scores, and only generate the header for the first one...



Finish the code for the `index.php` Guitar Wars script so that it adds a font for the top score that uses the `topscoreheader` CSS style. Hint: Don't forget that the top score header is part of the high score HTML table, which has two columns.

```
...
// Loop through the array of score data, formatting it as HTML
echo '<table>';
$i = 0;
while ($row = mysqli_fetch_array($data)) {
    // Display the score data
    if (.....) {
        .....
    }
    echo '<tr><td class="scoreinfo">';
    echo '<span class="score">' . $row['score'] . '</span><br />';
    echo '<strong>Name:</strong> ' . $row['name'] . '<br />';
    echo '<strong>Date:</strong> ' . $row['date'] . '</td>';
    if (is_file(GW_UPLOADPATH . $row['screenshot']) &&
        filesize(GW_UPLOADPATH . $row['screenshot']) > 0) {
        echo '<td></td></tr>';
    }
    else {
        echo '<td></td></tr>';
    }
}
echo '</table>';
...
```

exercise solution

Finish the code for the index.php Guitar Wars script so that it adds a footer for the top score that uses the topscoreheader CSS style. Hint: Don't forget that the top score header is part of the high score HTML table, which has two columns.

```

...
// Loop through the array of score data, formatting it as HTML
echo '<table>';
$fi = 0; // fi is the variable that counts through the high scores - we can use it to isolate the first score
while ($row = mysqli_fetch_array($data)) {
    // Display the score data
    if ($fi == 0) { // If fi is 0, it's the first score, so render the top score header
        echo '<tr><td colspan="2" class="topscoreheader">Top Score:</td></tr>';
    }
    echo '<tr><td class="scoreinfo">';
    echo '<span class="score">' . $row['score'] . '</span><br />';
    echo '<strong>Name:</strong> ' . $row['name'] . '<br />';
    echo '<strong>Date:</strong> ' . $row['date'] . '</td>';
    if (is_file(GW_UPLOADPATH . $row['screenshot']) &&
        filesize(GW_UPLOADPATH . $row['screenshot']) > 0) {
        echo '<td></td></tr>';
    }
    else {
        echo '<td></td></tr>';
    }
    $fi++; // Increment the counter at the end of the score loop - this code is the same as $fi = $fi + 1;
}
echo '</table>';
...

```

If \$fi is 0, it's the first score, so render the top score header

The topscoreheader style class is in style.css



Test Drive

Order the high scores and showcase the highest score of all!

Modify the `index.php` script to use the new ordered `SELECT` query, and then change the code that generates the top score header. Upload the new script to your web server and open it in your browser to see the top score prominently displayed.

The highest score is now shown loud and clear at the top of the high score list.

Guitar Wars - High Scores

Welcome, Guitar Warrior, do you have what it takes to crack the high score list? If so, just add your name and score to the list below.

Top Score: 368420

368420
Name: Ashton Simpson
Date: 2008-04-23 09:13:34

345900
Name: Eddie Vanilli
Date: 2008-04-23 09:06:35

Unverified!

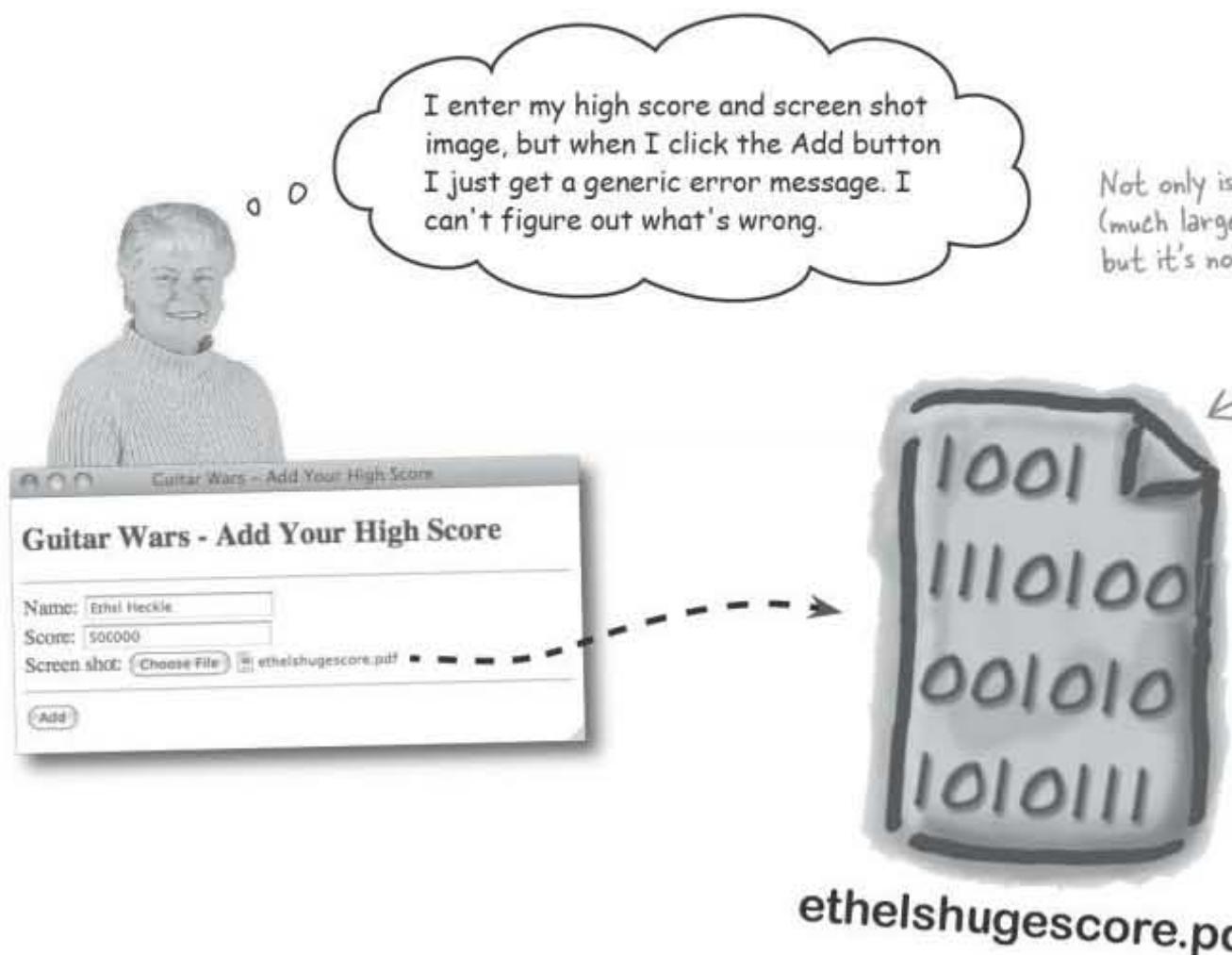
Unverified!

It's cool that the order is fixed...but you know any of those unverified scores could be bogus.



It's true, the unverified scores need to go.

But one thing at a time. It seems another problem has been preventing people from uploading their high score submissions.

adding a size restriction for images**Not only is the file huge, but it's not an image!**

We have a problem in that our form is rejecting some files but not telling users why. It's actually good that the form is rejecting files, in this case because they're too big—remember we capped the file size at 32 KB in the form code. But we need to be clear about telling the user why. Not only that, but we don't want users uploading files that aren't images. Adding validation to the Add Score form will allow us to better control how files are uploaded.

So validation on the image file upload form (`addscore.php`) serves two vital purposes. First, it can beef up the prevention of large file uploads, providing users with notification that a file can't be larger than 32 KB. And secondly, it can stop people from uploading files that aren't images. The file upload form needs validation for both file size and type.

This error message doesn't tell the user much about what went wrong with the high score submission.

Guitar Wars - Add Your High Score

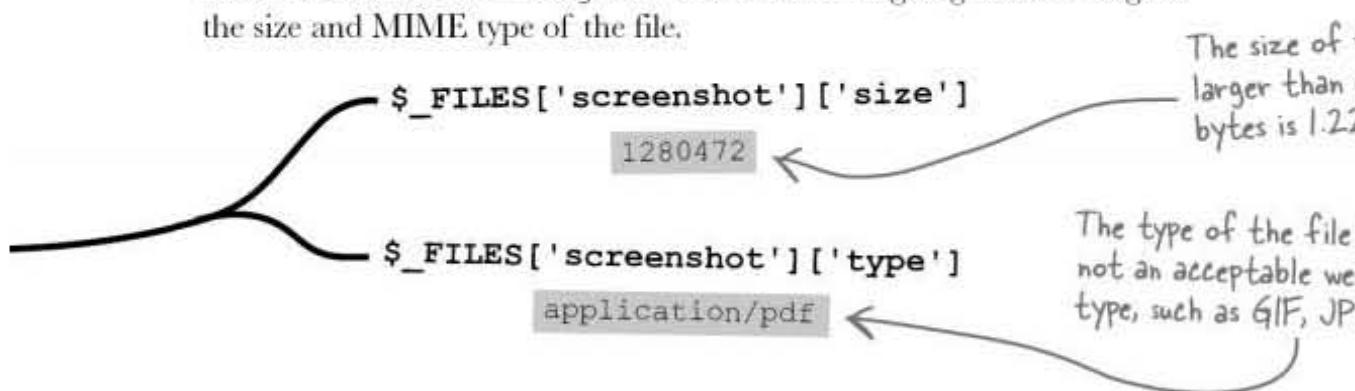
Sorry, there was a problem uploading your file.

Name: Ethel Heckle
Score: 500000
Screen shot: Choose File

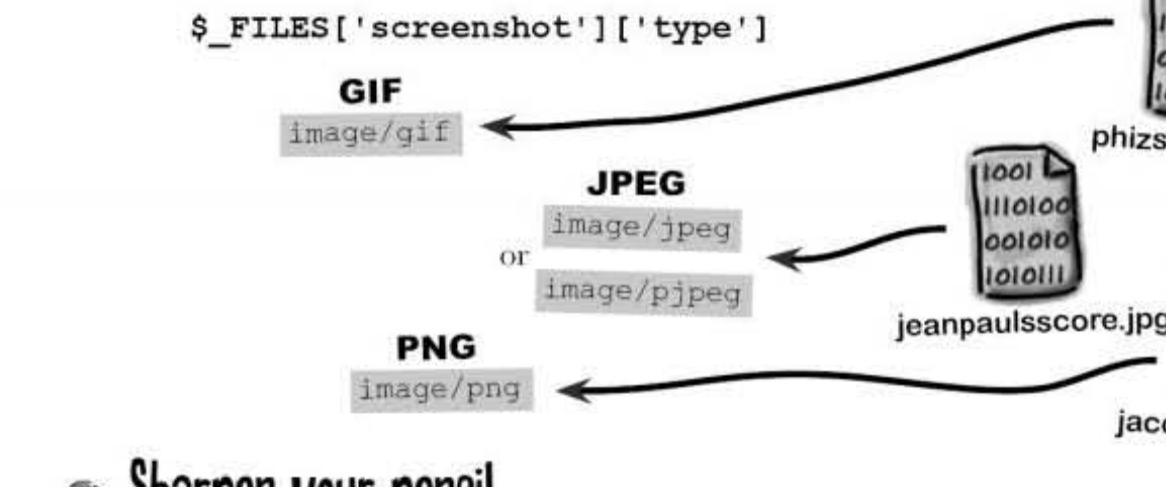
Add

small Only images allowed

So how exactly do we check the Add Score form and make sure uploaded images adhere to a certain size and type? The answer lies in the built-in `$_FILES` superglobal variable, which if you recall, is where we earlier obtained the temporary storage location of the uploaded file so that it could be moved to the `images` folder. Now we're going to use it to grab the size and MIME type of the file.



We don't just want image files to be smaller than our 32 KB size limit, but we also need them to be a file type that can be displayed as a web image. The following MIME types are commonly used to represent web images:



Sharpen your pencil

Write an `if` statement that checks to make sure the file is an image, as well as checking to make sure the file is less than 32 KB in size and less than the constant `$MAX_FILE_SIZE`. Assume the file size and type have already been stored in variables named `$screenshot_size` and `$screenshot_type`.

```
if (
```

.....
.....
.....

incorporating file validation in the app

Sharpen your pencil Solution

Some browsers

use this MIME type to recognize JPEG images.

```
if (((${screenshot_type} == 'image/gif') || (${screenshot_type} == 'image/png')) && (${screenshot_size} > 0) && (${screenshot_size} <= ${GW_MAXFILESIZE}))
```

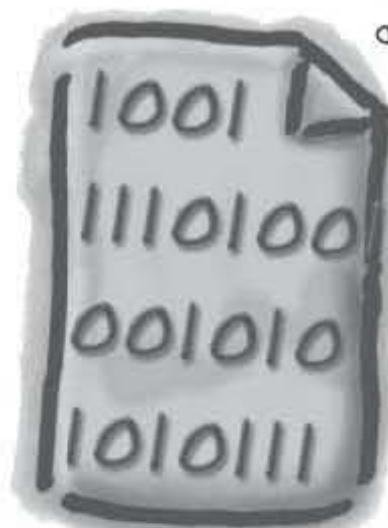
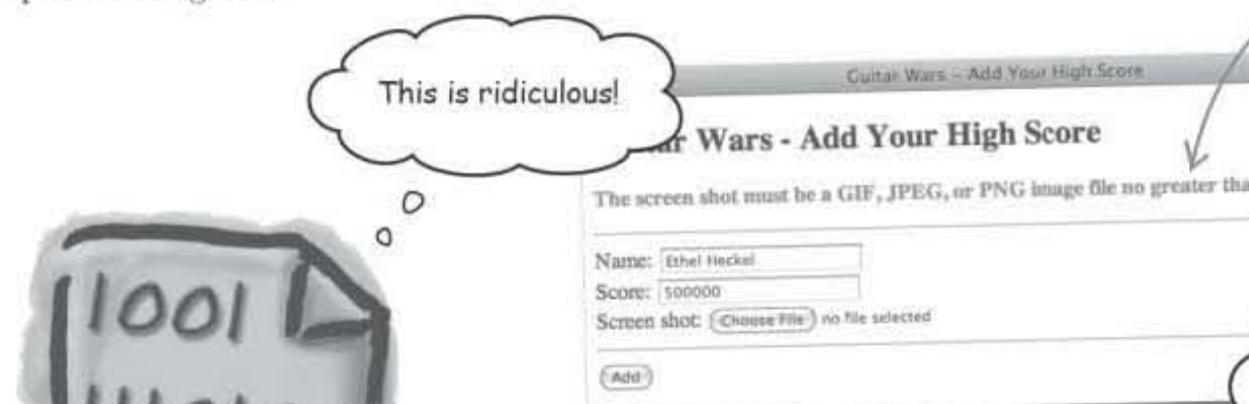
```
<?php
// Define application constants
define('GW_UPLOADPATH', 'images/');
define('GW_MAXFILESIZE', 32768); // 32 KB
?>
```

appvars.php

Since the maximum appears in more than one place in the Add Score script, it makes sense to store it as a constant.

File validation makes the app more robust

A little validation goes a long way toward making any PHP application more intuitive and easier to use, not to mention safer from abuse. Now a helpful error message lets the user know the exact constraints imposed on uploaded image files.



ethelshugescore.pdf



phizsscore.gif

jeanp

Since we're making the script more robust, it's also a good idea to check the `$_FILES` superglobal to make sure there wasn't an upload error.

```

if (!empty($name) && !empty($score) && !empty($screenshot)) {
    if (($screenshot_type == 'image/gif') || ($screenshot_type == 'image/jpeg')
        || ($screenshot_type == 'image/pjpeg') || ($screenshot_type == 'image/png')) {
        if ($screenshot_size > 0) && ($screenshot_size <= GW_MAXFILESIZE) {
            if ($_FILES['file']['error'] == 0) {
                // Move the file to the target upload folder
                $target = GW_UPLOADPATH . $screenshot;
                if (move_uploaded_file($_FILES['screenshot']['tmp_name'], $target))
                    // Connect to the database
                    $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

                // Write the data to the database
                $query = "INSERT INTO guitarwars VALUES (0, NOW(), '$name', '$score')";
                mysqli_query($dbc, $query);

                // Confirm success with the user
                echo '<p>Thanks for adding your new high score!</p>';
                echo '<p><strong>Name:</strong> ' . $name . '<br />';
                echo '<strong>Score:</strong> ' . $score . '<br />';
                echo '<a href="index.php">xit;</a> Back to high scores</a></p>';

                // Clear the score data to clear the form
                $name = "";
                $score = "";
                $screenshot = "";

                mysqli_close($dbc);
            }
            else {
                echo '<p class="error">Sorry, there was a problem uploading your score.</p>';
            }
        }
        else {
            echo '<p class="error">The screen shot must be a GIF, JPEG, or PNG file and less than ' . (GW_MAXFILESIZE / 1024) . ' KB in size.</p>';
        }
    }
    // Try to delete the temporary screen shot image file
    unlink($_FILES['screenshot']['tmp_name']);
}
else {
    echo '<p class="error">Please enter all of the information to add your score.</p>';
}

```

The `unlink()` function deletes a file from the web server. We suppress its error reporting with `@` in case the file upload didn't actually succeed.

The new and improved Add Score script has image fi

test drive addscore.php

Test Drive

Add screen shot image file validation to the Add Score script

Modify the `addscore.php` script to use the new image file validation code. Upload the modified script to your web server and try out the Add Score form with both valid images and a variety of invalid files (huge images and non-images).

*there are no
Dumb Questions*

Q: Why are there two different MIME types for JPEG images?

A: This is a question better asked of browser vendors, who, for some reason, decided to use different MIME types for JPEG images. To make sure the JPEG file validation works across as many browsers as possible, it's necessary to check for both MIME types.

Q: Why is it necessary to check for image files larger than 0 bytes? Aren't all images larger than 0 bytes?

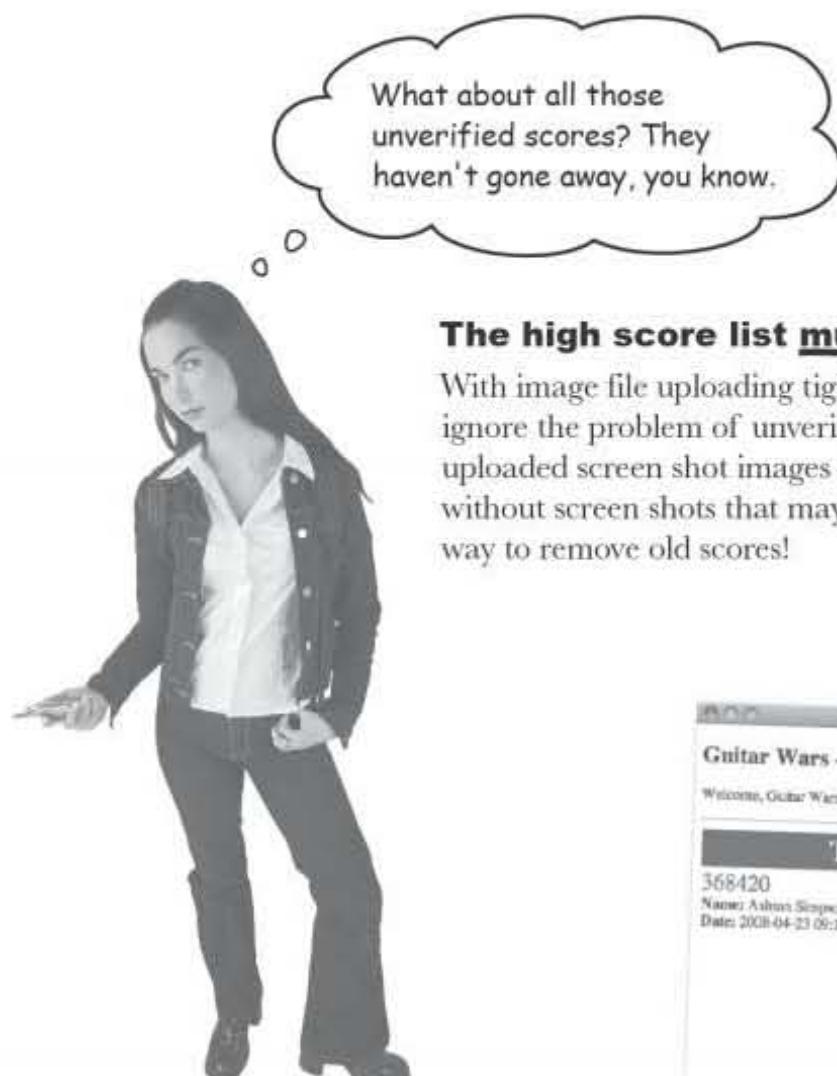
A: In theory, yes. But it is technically possible for a 0 byte file to get created on the server if the user specifies a file that doesn't actually exist on their own computer. Just in case this happens, `addscore.php` plays it safe and checks for an empty file.

Q: Why is `GW_MAXFILESIZE` defined in `appvars.php` even though it is used in `addscore.php`?

A: While it's true that `appvars.php` is used for storing script data that is shared across multiple scripts, it is also a good place to store **any** constant values. In this case, placing `GW_MAXFILESIZE` in `appvars.php` makes it easier to find if you ever want to change the limit larger.

Q: How does that line of code work?

A: The built-in PHP `unlink()` function removes a file from the web server, in our case the temporary image file that was uploaded. Since it's possible that the file may still exist after the `unlink()` call, we catch any errors generated by `unlink()` by prefixing the function with the error suppression symbol (@). You can stick @ in front of any PHP function to suppress its error reporting.



The high score list must be cleaned up.

With image file uploading tightened up thanks to validation, we can ignore the problem of unverified scores any longer. New uploaded screen shot images shouldn't play second fiddle to old ones without screen shots that may or may not be valid. Guitar Wars has a simple way to remove old scores!

The current
verified,
much co

guitarwars

id	date	name	score	screenshot
1	2008-04-22 14:37:34	Paco Jastorius	127650	
2	2008-04-22 21:27:54	Nevil Johansson	98430	
3	2008-04-23 09:06:35	Eddie Vanilli	345900	
4	2008-04-23 09:12:53	Belita Chevy	282470	
5	2008-04-23 09:13:34	Ashton Simpson	368420	
6	2008-04-23 14:09:50	Kenny Lavitz	64930	
7	2008-04-24 08:13:52	Phiz Lairston	186580	phizsscore.gif

Write down how you would go about cleaning up unverified scores in the high score list:

.....

.....

add an admin page

Plan for an Admin page

Since we just need to remove some unverified scores from the database, it's perfectly reasonable to just fire up an SQL tool and manually remove rows from the database with a few DELETE queries. But this may not be the last time you'll need to remove a score, and it's no fun having to resort to manual SQL queries to maintain a web application. The idea here is to build an application that can be maintained with as little hassle as possible.

What we need is a page that only the web site administrator has access to and can use to remove scores... an Admin page! But we need to be very careful in making a clear distinction between what parts of Guitar Wars are for the administrator and what parts are for users.

These pages are for users:



The Add Score page and main page of Guitar Wars designed for end users to add and view their high scores

This page is only for the administrator

The Admin page is designed only for use by the site administrator – you wouldn't want end users removing high scores

Guitar Wars - High Scores Admin	
Below is a list of all Guitar Wars high scores. Click the link next to each score to edit or delete it.	
Name	Date
Adam Simpson	2008-04-21 09:13:44 388420
Edgar Vazquez	2008-04-21 09:06:33 347900
Selina Cleary	2008-04-24 08:02:11 282470
Paul Laius	2008-04-24 08:13:27 186580
Peter Jantzen	2008-04-24 08:02:11 176450
Neill Johnstone	2008-04-24 08:02:11 168410
Benny Laius	2008-04-23 14:09:59 94930

Click the link next to each part of the table

Web applications often include pages for user access, and admin pages are only maintained



Write down what the Admin and Remove Score scripts need to do in order to implement a score removal feature for Guitar Wars. Then draw how a score removal affects the guitarwars table and the screen shot image file associated with it.



admin.php

.....

.....

.....

.....

.....

Web server



.....

.....

.....

.....

.....

.....



removesco

guitarwars

id	date	name	score
1	2008-04-22 14:37:34	Paco Jastorius	127650
2	2008-04-22 21:27:54	Nevil Johansson	98430
3	2008-04-23 09:06:35	Eddie Vanilli	345900
4	2008-04-23 09:12:53	Belita Chevy	282470
5	2008-04-23 09:13:34	Ashton Simpson	368420
6	2008-04-23 14:09:50	Kenny Lavitz	64930
7	2008-04-24 08:13:52	Phiz Lairston	186580

exercise solution

Write down what the Admin and Remove Score scripts need to do in order to implement a score removal feature for Guitar Wars. Then draw how a score removal would affect the guitarwars table and the screen shot image file associated with it.

Guitar Wars - High Scores Administration

Below is a list of all Guitar Wars high scores. Use this page to remove scores as needed.

Ashton Simpson	2008-04-23 09:13:34	368420	Remove
Eddie Vanilli	2008-04-23 09:06:35	345900	Remove
Belita Chevy	2008-04-24 08:02:11	282470	Remove
Phiz Lairston	2008-04-24 08:13:52	186580	Remove
Paco Jastorius	2008-04-24 08:02:11	127650	Remove
Nevil Johansson	2008-04-24 08:02:11	98430	Remove
Kenny Lavitz	2008-04-23 14:09:50	64930	Remove



The admin.php script lists all the score rows, each with a Remove link that passes information to the Remove Score script.

Web server

The removescore.php script takes care of the actual removal of the score from the database, the deletion of the image file from the server, and the display of a confirmation message.

Guitar Wars - Remove

Are you sure you want to delete the following score?

Name: Ashton Simpson
Date: 2008-04-23 09:13:34
Score: 368420

[Yes](#) [No](#) [Cancel](#)

[<< Back to admin page](#)

guitarwars

id	date	name	score
1	2008-04-22 14:37:34	Paco Jastorius	127650
2	2008-04-22 21:27:54	Nevil Johansson	98430
3	2008-04-23 09:06:35	Eddie Vanilli	345900
4	2008-04-23 09:12:53	Belita Chevy	282470
5	2008-04-23 09:13:34	Ashton Simpson	368420
6	2008-04-23 14:09:50	Kenny Lavitz	64930
7	2008-04-24 08:13:52	Phiz Lairston	186580

Although this particular example row is missing a screen shot file, the Remove Score script will need to delete the image file from the server for scores that do have an image.

Generate score removal links on the Admin page

Although the Remove Score script is responsible for the actual score removal, we need an Admin script that allows us to select a score to remove. The `admin.php` script generates a list of high scores with Remove links for each one. These links pass along data about a given score to the `removescore.php` script.

```
<?php

require_once('appvars.php');
require_once('connectvars.php');

// Connect to the database

$dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

// Retrieve the score data from MySQL

$query = "SELECT * FROM guitarwars ORDER BY score DESC, date ASC";
$data = mysqli_query($dbc, $query);

// Loop through the array of score data, formatting it as HTML

echo '<table>';

while ($row = mysqli_fetch_array($data)) {
    // Display the score data

    echo '<tr class="scorerow"><td><strong>' . $row['name'] . '</strong></td>';
    echo '<td>' . $row['date'] . '</td>';
    echo '<td>' . $row['score'] . '</td>';
    echo '<td><a href="removescore.php?id=' . $row['id'] . '&date=' . $row['date'] . '&name=' . $row['name'] . '&score=' . $row['score'] . '&Screenshot=' . $row['Screenshot'] . '">Remove</a></td></tr>';

}

echo '</table>';

mysqli_close($dbc);
```

This code generates an HTML link to the removescore.php script, passing a parameter containing information about the score to be removed.

[remove](removescore.php?id=5&date=2008-3-34&name=Ashton%20Simpson&score=368420)

introducing the GET request

Scripts can communicate with each other

In order for the Remove Score script to remove a high score, it must know what score to remove. But that's decided in the Admin script. This begs the question, how does the Admin script tell the Remove Score script what score to remove? This communication between scripts is accomplished by packaging up the data as part of a "Remove" URL for each high score shown on the Admin page. If you closely analyze the URL for a particular score, you'll notice that all the high score data is in there.

```

<a href="removescore.php?
id=5&
date=2008-04-23%2009:13:34&
name=Ashton%20Simpson&
score=368420&screenshot=>Remove</a>

```

Each piece of data has a name and a value, and is separated from other name/value pairs by an ampersand (&).

The "Remove" URL links to the removescore.php script but also includes data for the row to be deleted.

Guitar Wars - High Scores Admin

Below is a list of all Guitar Wars high scores. Use this page to manage them.

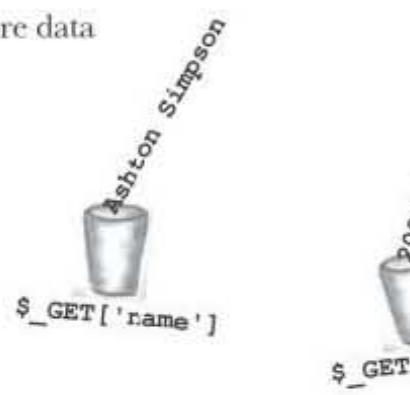
User	Date	Score	Action
Ashton Simpson	2008-04-23 09:13:34	368420	Remove
Eddie Vanilli	2008-04-23 09:06:35	345900	Remove
Belita Chevy	2008-04-24 08:02:11	282470	Remove
Phiz Lairston	2008-04-24 08:13:52	186580	Remove
Paco Jastorius	2008-04-24 08:02:11	127650	Remove
Nevil Johansson	2008-04-24 08:02:11	98430	Remove
Kenny Lavitz	2008-04-23 14:09:50	64930	Remove

Clicking this link not only opens the Remove Score script, but it also passes along data to the script as a GET request.

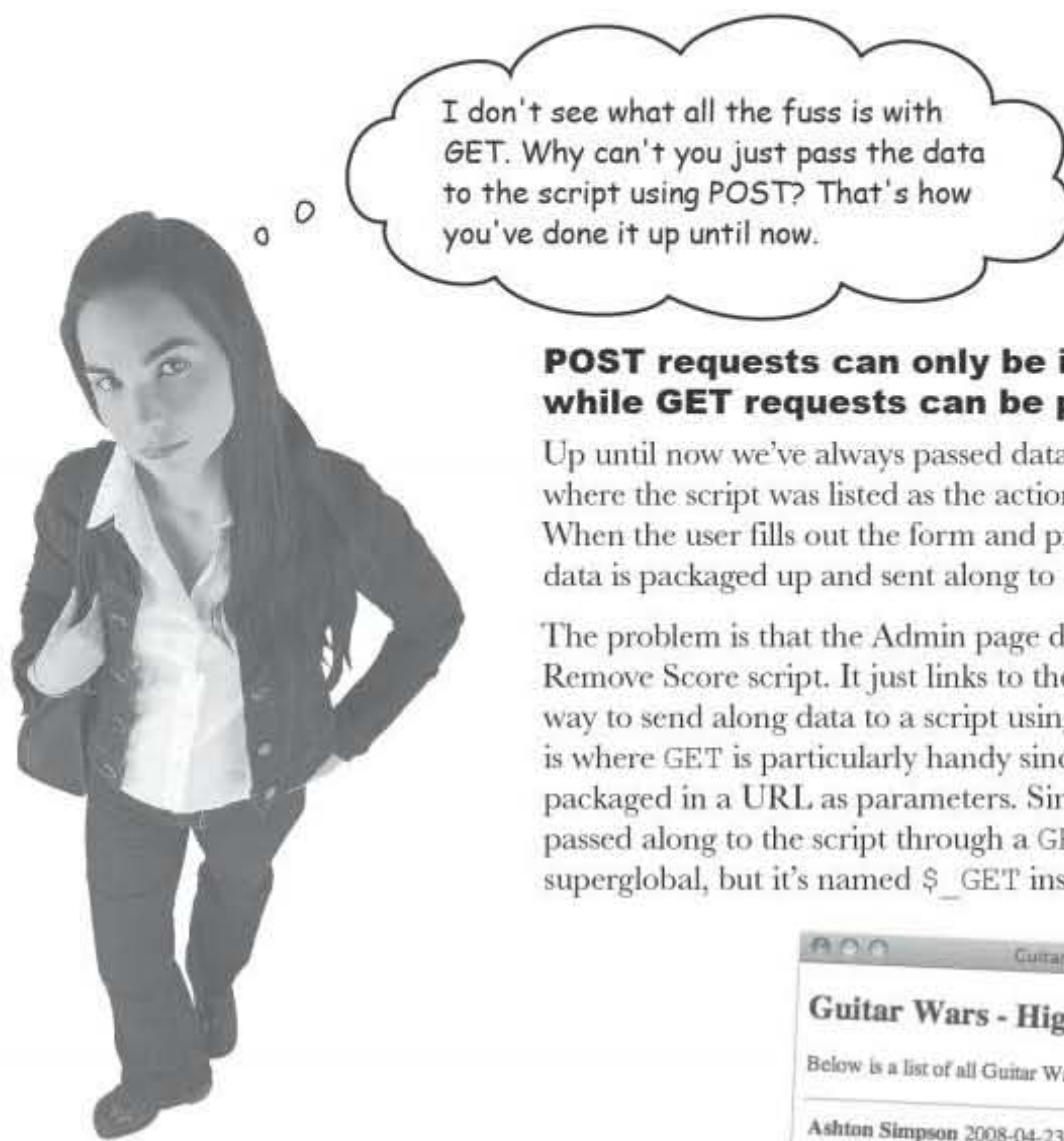
OK, so data gets passed along through a URL, but how exactly does the Remove Score script get its hands on that data? Data passed to a script through a URL is available in the `$_GET` superglobal, which is an array very similar to `$_POST`. Packaging data into a linked URL is the same as using a GET request in a web form. In a traditional HTML GET request, form data is **automatically** sent along to the form processing script as part of the script's URL. We're doing the same thing by **manually** building our own GET request as a custom URL.

Similar to `$_POST`, using the `$_GET` array to access the high score data requires the name of each piece of data.

The URL for a script serves as a handy way to pass important data, such as the ID of a database row.



`$_GET`

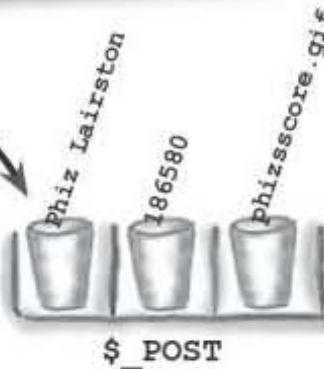


POST requests can only be initiated through forms while GET requests can be packaged as URLs

Up until now we've always passed data to a script through forms where the script was listed as the action for the form's Submit button. When the user fills out the form and presses the Submit button, the data is packaged up and sent along to the form as a POST request.

The problem is that the Admin page doesn't use a form. It uses a Remove Score script. It just links to the script via a URL. This is a better way to send along data to a script using nothing more than a URL. This is where GET is particularly handy since it provides access to data that's been packaged in a URL as parameters. Similar to POST, the data passed along to the script through a GET request is available via a superglobal, but it's named `$_GET` instead of `$_POST`.

Web forms often use POST requests to submit data, which is stored in the `$_POST` array.



Guitar Wars - High Scores Admin		
Below is a list of all Guitar Wars high scores. Use this page to remove old scores.		
Ashton Simpson	2008-04-23 09:13:34	368420
Eddie Vanilli	2008-04-23 09:36:35	345900
Belita Chevy	2008-04-24 08:02:11	282470
Phiz Lairston	2008-04-24 08:13:52	186580
Paco Jastorius	2008-04-24 08:02:11	127650
Nevil Johansson	2008-04-24 08:02:11	98430
Kenny Lavitz	2008-04-23 14:09:50	64930



GET vs POST

Of GETs and POSTs

The difference between GET and POST isn't just form vs. URL since GET requests can (and often are) used to submit form data as well. The real distinction between GET and POST has to do with the **intent** of a request. GET is used primarily to retrieve data from the server **without affecting anything** on the server. POST, on the other hand, typically involves sending data to the server, after which **the state of the server usually changes** somehow in response to the data that was sent.

The two web requests and POSTs how you send data between servers

POST

Used to send data to the server that somehow causes a change in the state of the server, such as inserting data in a database. Data can still be returned in a response. Unlike GET, POST requests can only be made through the action of a web form. Also unlike GET, the data sent in a POST request is hidden from view.

GET

Typically used for data retrieval that doesn't change anything on the server. It's good for small amounts of data, GET is also useful for directly sending data to the server in a URL. Unlike POST, GET is primarily suited to sending small amounts of data.

*there are no
Dumb Questions*

Q: I've seen web forms that use GET. How does that work?

A: Both GET and POST have their place when it comes to web forms. When creating a web form, the `method` attribute of the `<form>` tag controls how the data is sent, while the `action` attribute identifies the script to receive the data and process it:

```
<form method="post" action="addscore.php">
```

When the submit button is clicked to submit this form, the `addscore.php` script is executed, and the form data is passed along to it through the `$_POST` array. But you could've just as easily written the `<form>` tag like this, in which case the data would get passed along through the `$_GET` array:

```
<form method="get" action="addscore.php">
```

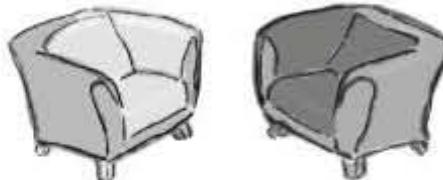
Q: Ah, so it doesn't matter which request method I use, GET or POST?

A: Wrong. It matters quite a lot. GET is used to retrieve data from the server, not changing anything. It's perfect for forms that make information available without altering the state of the server, such as querying a database. POST, on the other hand, is intended to affect the server's state, such as issuing a query that changes the database. Another difference between GET and POST is that data passed through a GET request is visible in the URL, while POST data is hidden, and, therefore, cannot be easily viewed.

Q: How does this distinction between GET and POST affect the passing of data to a script through a URL?

A: Well, first of all, you can only pass data to a URL using a GET request, so POST is out of the question. Furthermore, since GET is intended purely for retrieving data from the server, this means you can't use it to perform INSERTs, DELETEs, or UPDATEs, or any other operation that alters the state of the database. If you try to use a GET request to perform such an operation, the server will return an error message.

Fireside Chats



Tonight's talk: **GET and POST**

GET:

So, word on the street is you've been saying all I'm good for is asking questions but not really doing anything with the answers. Is that true?

OK, so it's true that I'm not really intended to be causing changes on the server such as deleting files or adding database rows, but that doesn't mean I'm not important.

True, but you're permanently connected to your good buddy, Form, whereas Form and I are merely casual acquaintances. I leave room for other friends, such as URL.

Well, then I have a question for you. How exactly do you take action when your little sidekick, Form, isn't around? You know sometimes Page doesn't find it necessary to go to the trouble of involving Form.

Calm down. I'm just pointing out that while I'm geared toward retrieving data from the server, I'm fairly flexible in how I can be used to do it.

Glad to hear it. It's been good talking to you...

POST:

Sure is. Let's face it, you don't just the ability to ask the serv

If you say so. All I know is no done without people like me on the server. If the server was same state, it would be pretty

So you think your "circle of overcomes your inability to t

Listen, Form is my friend, and commitment not to do any r So judge my loyalty if you m my friend!

I'll give you that. You're alrig

how removescore.php will work

GET, POST, and high score removal

We've established that the removal of scores in Guitar Wars starts with a "Remove" link on the Admin page that links to the Remove Score script. We also know that score data can be passed through the link URL to the Remove Score script. But we have a problem in that a GET request really shouldn't be changing anything on the server, such as deleting a score. A possible solution is to not change anything on the server... yet. What if the Remove Score script initially displayed a confirmation page before actually removing a score from the database?



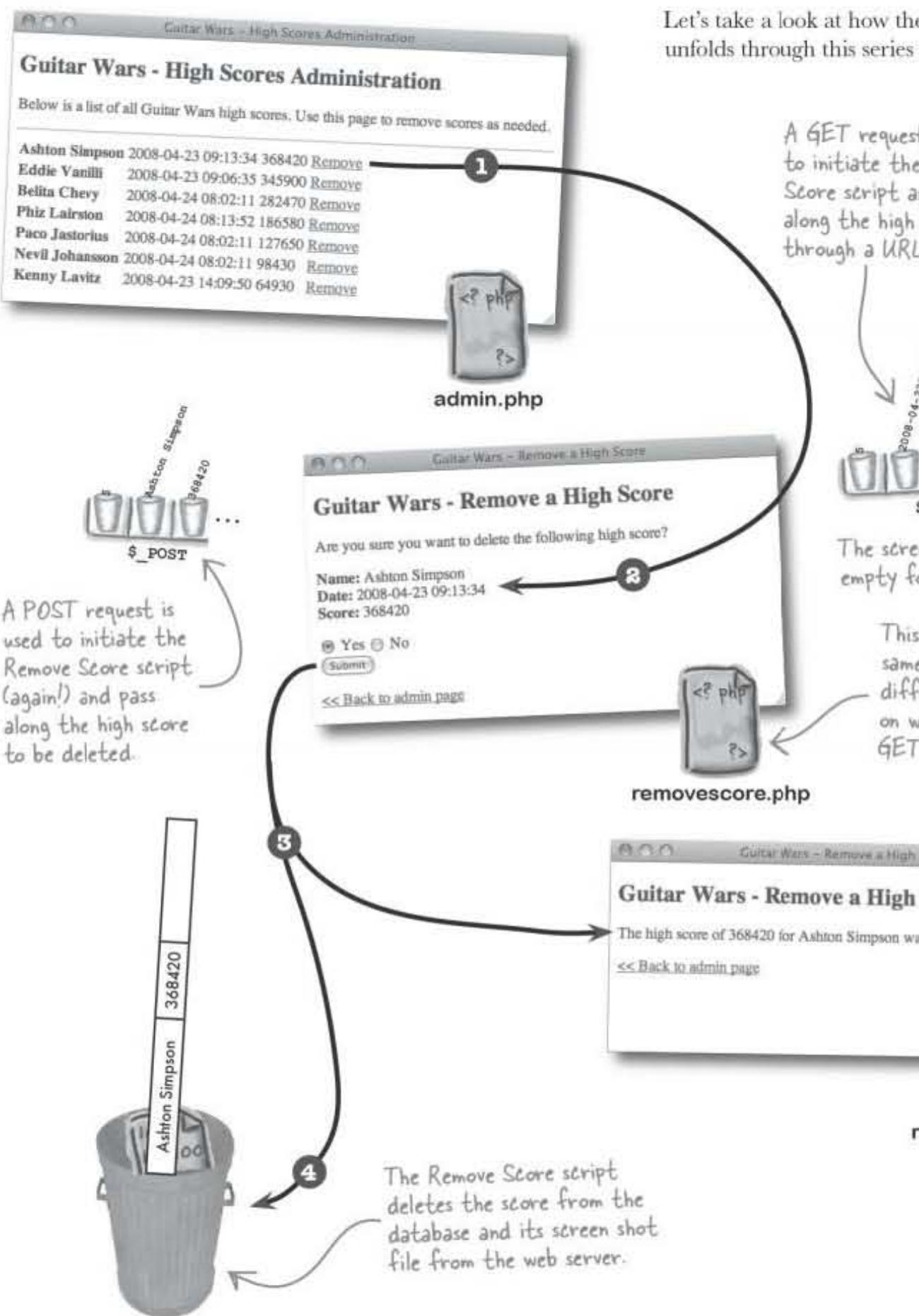
A confirmation page gives the user a chance to confirm high score removal instead of just removing it instantaneously.

The confirmation page shows the score that is up for removal with a simple Yes/No form. Selecting Yes and clicking the Submit button results in the score being removed, while choosing No cancels the score removal.

Thinking in terms of GETs and POSTs, the Remove Score script can display the confirmation page as a response to the GET request from the Admin script. And since the confirmation itself is a form, it can issue its own POST request when submitted. If the form is a self-referencing form, the same script (`removescore.php`) can process the POST and carry out the score removal. Here are the steps involved in this process:

It's entire
even help
cases, for
script to 1
both GET
requests.

- 1 The Remove Score script is initiated through a GET request by the user clicking the "Remove" link on the Admin page.**
- 2 The Remove Score script uses the high score data stored in the `$_GET` array to generate a removal confirmation form.**
- 3 The Remove Score script is initiated again, this time, through a POST request by the user submitting the confirmation form.**
- 4 The Remove Score script deletes the score from the database and also deletes the screen shot image file from the web server.**



more on GET and POST

there are no Dumb Questions

Q: How can the same script process both GET and POST requests?

A: It all has to do with how a script is invoked. In the case of the Remove Score script, it is invoked in two different ways. The first way is when the user clicks a "Remove" link on the Admin page, in which case a URL leads them to the script. Since data is packaged into the URL, this is considered a GET request. This GET request causes the script to generate a web form whose action refers back to the same Remove Score script. So when the user submits the form, the script is invoked a second time. But unlike the first time, there is no fancy URL with data packaged into it and, therefore, no GET request. Instead, the high score data is passed along through a POST request and is, therefore, available in the `$_POST` array.

Q: So the manner in which the script is invoked actually determines what it does?

A: Yes! When the script sees that data has been sent through a URL as a GET request, it knows to display a confirmation form, as opposed to deleting anything from the database. So the data sent along in the `$_GET` array is used only within the confirmation page and has no lasting effect on the server.

When the script sees that data is being delivered through a POST request, the script knows that it can delete the data from the database. So it uses the `$_POST` array to access the data and assemble a `DELETE FROM` query that deletes the score. And since most high scores also have a screen shot image file stored on the web server, the script also deletes that file.

Isolate the high score for deletion

With the score removal process laid out, we can now focus our attention on the database side of things. The Remove Score script is responsible for removing a high score, which means deleting a row from the database of scores. If you recall, the SQL `DELETE FROM` statement allows us to delete rows. But in order to delete a row, we must first find it. This is accomplished by tacking a `WHERE` clause onto a query that uses `DELETE FROM`. For example, this SQL query deletes the row with the name column set to 'Ashton Simpson':

```
DELETE FROM guitarwars WHERE name = 'Ashton Simpson'
```

guitarwars

id	date	name	score	screenshot
1	2008-04-22 14:37:34	Paco Jastorius	127650	
2	2008-04-22 21:27:54	Nevil Johansson	98430	
3	2008-04-23 09:06:35	Eddie Vanilli	345900	
4	2008-04-23 09:12:53	Belita Chevy	282470	
5	2008-04-23 09:13:34	Ashton Simpson	368420	
6	2008-04-23 14:09:50	Kenny Lavitz	64930	
7	2008-04-24 08:13:52	Phiz Lairston	186580	phizsscore.gif

There's a problem with this query, however. In a world full of millions of Guitar Warriors, odds are there will be more than one Ashton Simpson. This query doesn't just delete a single row, it deletes **all** rows matching the name 'Ashton Simpson'. The query needs more information in order to delete the right row:

```
DELETE FROM guitarwars WHERE name = 'Ashton Simpson'
```

This query deletes all rows with a name column value matching 'Ashton Simpson'.

The table name is specified by `DELETE`, so we know which table we're deleting data from.

The `name` column is the condition for the deletion.

By matching the additional condition of `score`, we can ensure the deletion is isolated to the correct row.

`AND score`

guitarwars

id	date	name	score	screenshot
1	2008-04-22 14:37:34	Paco Jastorius	127650	
2	2008-04-22 21:27:54	Nevil Johansson	98430	
3	2008-04-23 09:06:35	Eddie Vanilli	345900	
4	2008-04-23 09:12:53	Belita Chevy	282470	
5	2008-04-23 09:13:34	Ashton Simpson	368420	
6	2008-04-23 14:09:50	Kenny Lavitz	64930	
7	2008-04-24 08:13:52	Phiz Lairston	186580	phizsscore.gif

The `AND` condition in the query isolates the row with the correct name and score.

Now that both the name and score have been specified, the accidentally selected rows with the wrong name and score are deleted.

*putting a **LIMIT** on your **DELETE***

Control how much you delete with **LIMIT**

Using both the `name` and `score` columns as the basis for deleting a row is good... but not good enough. Application development is about minimizing risks at all cost, and there's still a slight risk of deleting multiple rows that match both the same name and score. The solution is to force the query to only delete **one row** no matter what. The `LIMIT` clause makes this happen:

For maximum limit on the n
that can be d

```
DELETE FROM guitarwars WHERE name = 'Ashton Simpson' AND score =
```

The number following `LIMIT` lets MySQL know the maximum number of rows to delete—in this case, one. So we're guaranteed to never delete more than one row with this query. But what if there were two Ashton Simpsons with the same score? Sure, this is an unlikely scenario, but it's sometimes worth considering extreme scenarios when working out the best design for an application.

guitarwars

id	date	name	score	screenshot
1	2008-04-22 14:37:34	Paco Jastorius	127650	
2	2008-04-22 21:27:54	Nevil Johansson	98430	
3	2008-04-23 09:06:35	Eddie Vanilli	345900	
4	2008-04-23 09:12:53	Belita Chevy	282470	
5	2008-04-23 09:13:34	Ashton Simpson	368420	
6	2008-04-23 14:09:50	Kenny Lavitz	64930	
7	2008-04-24 08:13:52	Phiz Lairston	186580	phizsscore.gif
...				
523	2008-11-04 10:03:21	Ashton Simpson	368420	ashtonsscore.jpg

Write down what happens to this table when the statement above is executed. How could you ensure that the right Ashton Simpson score is deleted?

.....

.....

.....

Would it be any better to use the ID of the score in the WHERE clause of the DELETE FROM query? It might help make sure we delete the right score, no?

Yes, it would! The ID of a high score is the perfect way to isolate the score for deletion.

Uniqueness is one of the main advantages of creating primary keys for your tables. The `id` column in the `guitarwars` table is the primary key and is, therefore, unique for each and every high score. By using this column in the WHERE clause of the `DELETE FROM` query, we eliminate all doubt surrounding which score we're deleting. Here's a new query that uses the `id` column to help ensure uniqueness:

```
DELETE FROM guitarwars WHERE id = 5
```

Trusting that the `id` column is indeed a primary key results in this code safely deleting only one row. But what if **you** didn't create the database, and maybe uniqueness wasn't properly enforced? Then a `LIMIT` clause might still make some sense. The rationale is that if you intend for a query to only affect one row, then say it in the query.

```
DELETE FROM guitarwars WHERE id = 5 LIMIT 1
```

It's never a bad idea to be very explicit with what you expect to be done in a query, and in this case `LIMIT` adds an extra degree of safety to the `DELETE` query.

Deleting data
Primary key
accuracy in
right row f

The `LIMIT` c
states that t
delete more

finish the removescore.php script



PHP & MySQL Magnets

The removescore.php script is almost finished, but it is missing a few important pieces of code. Use the magnets to plug in the missing code and give Guitar Wars the ability to eradicate unwanted scores.

```

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Guitar Wars - Remove a High Score</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
    <h2>Guitar Wars - Remove a High Score</h2>

<?php
    (*appvars.php');
    ..... (*connectvars.php');
    ..... if (isset($_GET['id'])) && isset($_GET['date']) && isset($_GET['name']) &&
    isset($_GET['score']) && isset($_GET[.....]);
    ..... // Grab the score data from the GET
    $id = $_GET['id'];
    $date = $_GET['date'];
    $name = $_GET['name'];
    $score = $_GET['score'];
    ..... = $_GET[.....];
    ..... }
    else if (isset($_POST['id'])) && isset($_POST['name']) && isset($_POST['scor
    ..... // Grab the score data from the POST
    ..... = $_POST[.....];
    ..... $name = $_POST['name'];
    $score = $_POST['score'];
    }
    else {
        echo '<p class="error">Sorry, no high score was specified for removal.</p>';
    }
    ..... if (isset($_POST['submit'])) {
        ..... if ($_POST['confirm'] == ..... ) {
            ..... // Delete the screen shot image file from the server
            @unlink(GW_UPLOADPATH . $Screenshot);
            ..... // Connect to the database
            $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);
        }
    }

```

```

// Delete the score data from the database
$query = "DELETE FROM guitarwars WHERE
          ..... LIMIT .....";
mysqli_query($dbc, $query);
mysqli_close($dbc);

// Confirm success with the user
echo '<p>The high score of ' . $score . ' for ' . $name . ' was successful
';
}
else {
    echo '<p class="error">The high score was not removed.</p>';
}
}

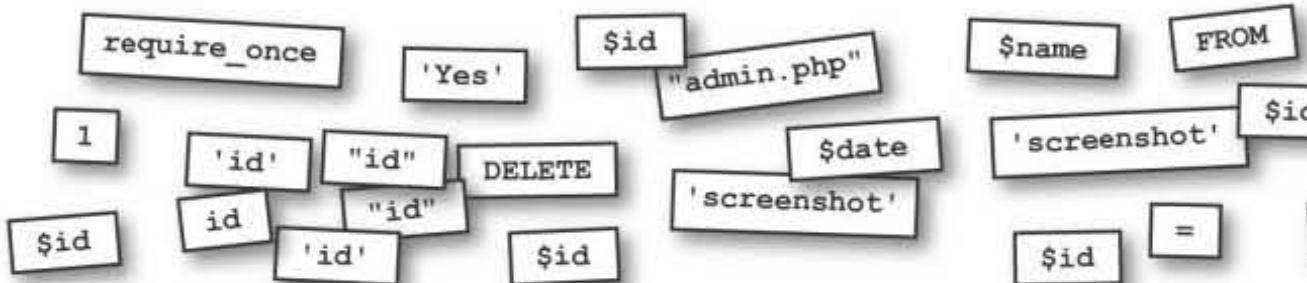
else if (isset(.....) && isset(.....) && isset(.....) &&
isset($score) && isset($Screenshot)) {
    echo '<p>Are you sure you want to delete the following high score?</p>';
    echo '<p><strong>Name: </strong>' . $name . '<br /><strong>Date: </strong>' .
        '<br /><strong>Score: </strong>' . $score . '</p>';
    echo '<form method="post" action="removescore.php">';
    echo '<input type="radio" name="confirm" value="Yes" /> Yes ';
    echo '<input type="radio" name="confirm" value="No" checked="checked" /> No';
    echo '<input type="submit" value="Submit" name="submit" />';
    echo '<input type="hidden" name="name" value="' . $name . '" />';
    echo '<input type="hidden" name="score" value="' . $score . '" />';
    echo '</form>';
}

echo '<p><a href="admin.php" >&lt;&lt; Back to admin page</a></p>';
?

```

</body>

</html>



the finished removescore.php script



PHP & MySQL Magnets Solution

The removescore.php script is almost finished, but it is missing a few important pieces of code. Use the magnets to plug in the missing code and give Guitar Wars the ability to eradicate unwanted scores.

```

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Guitar Wars - Remove a High Score</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
    <h2>Guitar Wars - Remove a High Score</h2>
    <?php
        require_once ('appvars.php');
        require_once ('connectvars.php');

        if (isset($_GET['id']) && isset($_GET['date']) && isset($_GET['name']) &&
            isset($_GET['score']) && isset($_GET[ 'screenshot' ])) {
            // Grab the score data from the GET
            $id = $_GET['id'];
            $date = $_GET['date'];
            $name = $_GET['name'];
            $score = $_GET['score'];

            $screenshot = $_GET[ 'screenshot' ];
        }
        else if (isset($_POST['id']) && isset($_POST['name']) && isset($_POST['score'])) {
            // Grab the score data from the POST
            $id = $_POST[ 'id' ];
            $name = $_POST['name'];
            $score = $_POST['score'];
        }
        else {
            echo '<p class="error">Sorry, no high score was specified for removal.</p>';
        }

        if (isset($_POST['submit'])) {
            if ($_POST['confirm'] == 'Yes') {
                Delete the screen shot image file from the server
                unlink(GW_UPLOADPATH . $screenshot);
            }
        }
        // Connect to the database
        $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);
    
```

Include the shared script files but use required_once since they are critical to the score removal.

The script reacts differently depending on whether the request is a GET or a POST.

The @ PHP error suppression directive prevents errors from being displayed. This is used for unlink() since we may be attempting to delete a file that doesn't exist and we don't want the user to see an error message.

This script uses the unlink() function to remove the uploaded file from the server so the uploaded file must be part of the file path.

```

// Delete the score data from the database
$query = "DELETE FROM guitarwars WHERE id = $id";
mysqli_query($dbc, $query);
mysqli_close($dbc);

// Confirm success with the user
echo '<p>The high score of ' . $score . ' for ' . $name . ' was successfully removed.';
}

else {
    echo '<p class="error">The high score was not removed.</p>';
}

}

else if (isset( $id ) && isset( $name ) && isset( $date ) && isset($score) && isset($screenshot)) {
    echo '<p>Are you sure you want to delete the following high score?</p>';
    echo '<p><strong>Name: </strong>' . $name . '<br /><strong>Date: </strong>' . $date . '<br /><strong>Score: </strong>' . $score . '</p>';
    echo '<form method="post" action="removescore.php">';
    echo '<input type="radio" name="confirm" value="Yes" /> Yes ';
    echo '<input type="radio" name="confirm" value="No" checked="checked" /> No';
    echo '<input type="submit" value="Submit" name="submit" />';
    echo '<input type="hidden" name="id" value="' . $id . '" />';
    echo '<input type="hidden" name="name" value="' . $name . '" />';
    echo '<input type="hidden" name="score" value="' . $score . '" />';
    echo '</form>';
}

echo '<p><a href= "admin.php" >&lt;&lt; Back to admin page</a></p>';

?>

</body>
</html>

```

Provide a link back to the Admin page to improve navigation.

\$id
"id"
'id'

A few hidden form fields are used to store the score data so that it gets sent along as part of the POST request.

The id column is matched by the DELETE query, along with using a LIMIT of one row.

↑

↑

↑

↑

↑

↑

↑

We don't use `$_SERVER['PHP_SELF']` here because it would include any data that had been passed through the URL query string as a GET. We want to make sure no GET data is passed along with this form - only POST data.

test drive the final guitar wars app



Test Drive

Add Remove Score and Admin scripts to Guitar Wars so they can be removed.

Create two new text files, `removescore.php` and `admin.php`, and add the code you've just worked through. Upload the new scripts to your web server, and open the Admin script in your web browser. Click the "Remove" link for a score you'd like to delete, and then confirm its removal on the Remove Score page. Return to the Admin page to make sure the score is gone, and then go to the main Guitar Wars page (`index.php`) to make sure the change there.

The new Admin page provides links to remove unverified high scores.

There's only room in this town for one top rocker, and it's me!

Little Jacob, Guitar Warrior rock prodigy

The legit Guitar Warriors are now happy to see only verified high scores.

Unverified high scores, those without screen shots, have now been removed from the system.

Guitar Wars - High Scores Administration
Below is a list of Guitar Wars high scores. Use this page to remove scores as needed.

User	Date	Score	Notes
Ashley Bishop	2008-04-19 00:13:14	308127	Verified
Eddy Yanki	2008-04-21 00:06:29	349900	Unverified
Bella Cherry	2008-04-21 04:02:11	225200	Verified
FBI Latrine	2008-10-22 00:11:02	386500	Verified
Pete Justice	2008-04-24 00:02:11	127000	Unverified
Nerd Johnson	2008-04-24 00:02:11	189000	Unverified
Savvy Lurk	2008-04-22 14:00:53	689793	Unverified

Guitar Wars - Remove a High Score
Are you sure you want to delete the following high score?

Name: Ashley Bishop
Date: 2008-04-19 00:13:14
Score: 308127

Yes No

Check location

Guitar Wars - Remove a High Score
The high score of 308127 for Ashley Bishop was successfully deleted.

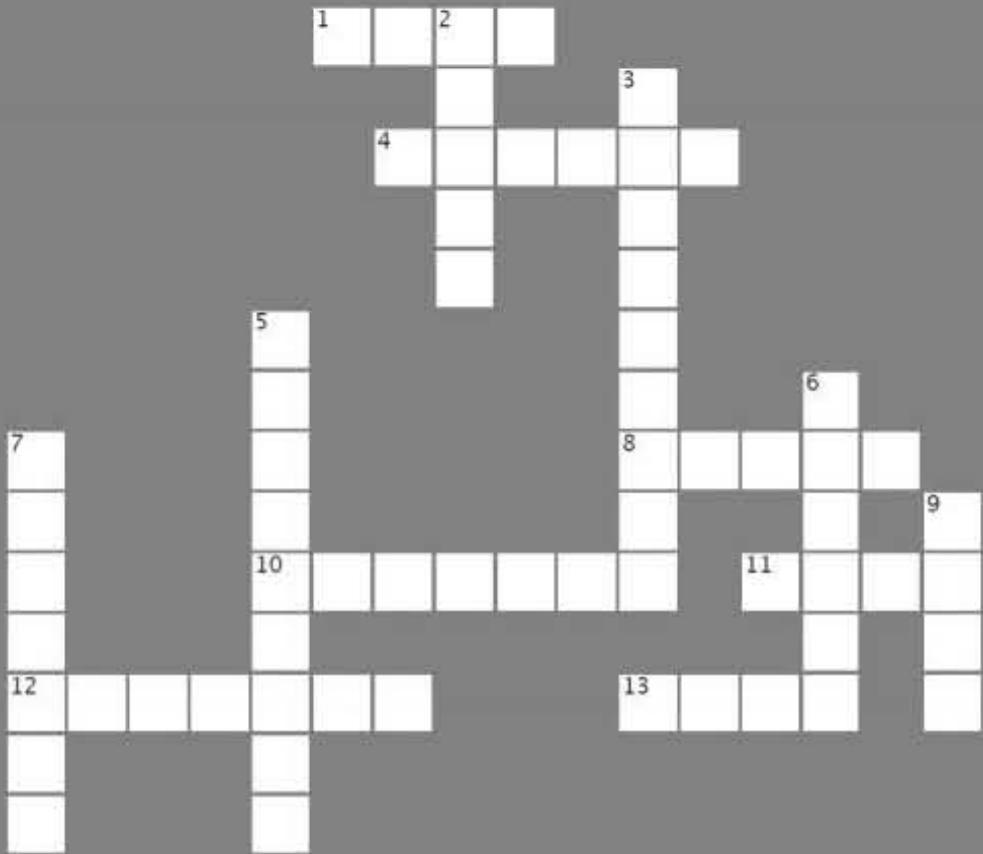
Guitar Wars - High Scores Administration
Below is a list of Guitar Wars high scores. Use this page to remove scores as needed.

User	Date	Score	Notes
FBI Latrine	2008-10-22 00:11:02	386500	Verified
Bella Cherry	2008-04-21 04:02:11	225200	Verified
Eddy Yanki	2008-04-21 00:06:29	349900	Unverified
Ashley Bishop	2008-04-19 00:13:14	308127	Verified



PHP&MySQLcross

Tired of uploading image files? How about uploading some knowledge into a bunch of squares laid out in a puzzle?



Across

- The type attribute of the <input> tag must be set to this for a file upload form field.
- It's usually a good idea to store uploaded application images in an folder.
- This SQL statement is used to change the structure of a table.
- This SQL statement is used to put the results of a query in a certain order.
- Information about uploaded files is stored in the \$..... superglobal variable.
- This PHP statement is used to insert code from another script.
- It's a good idea to do this to newly uploaded files.

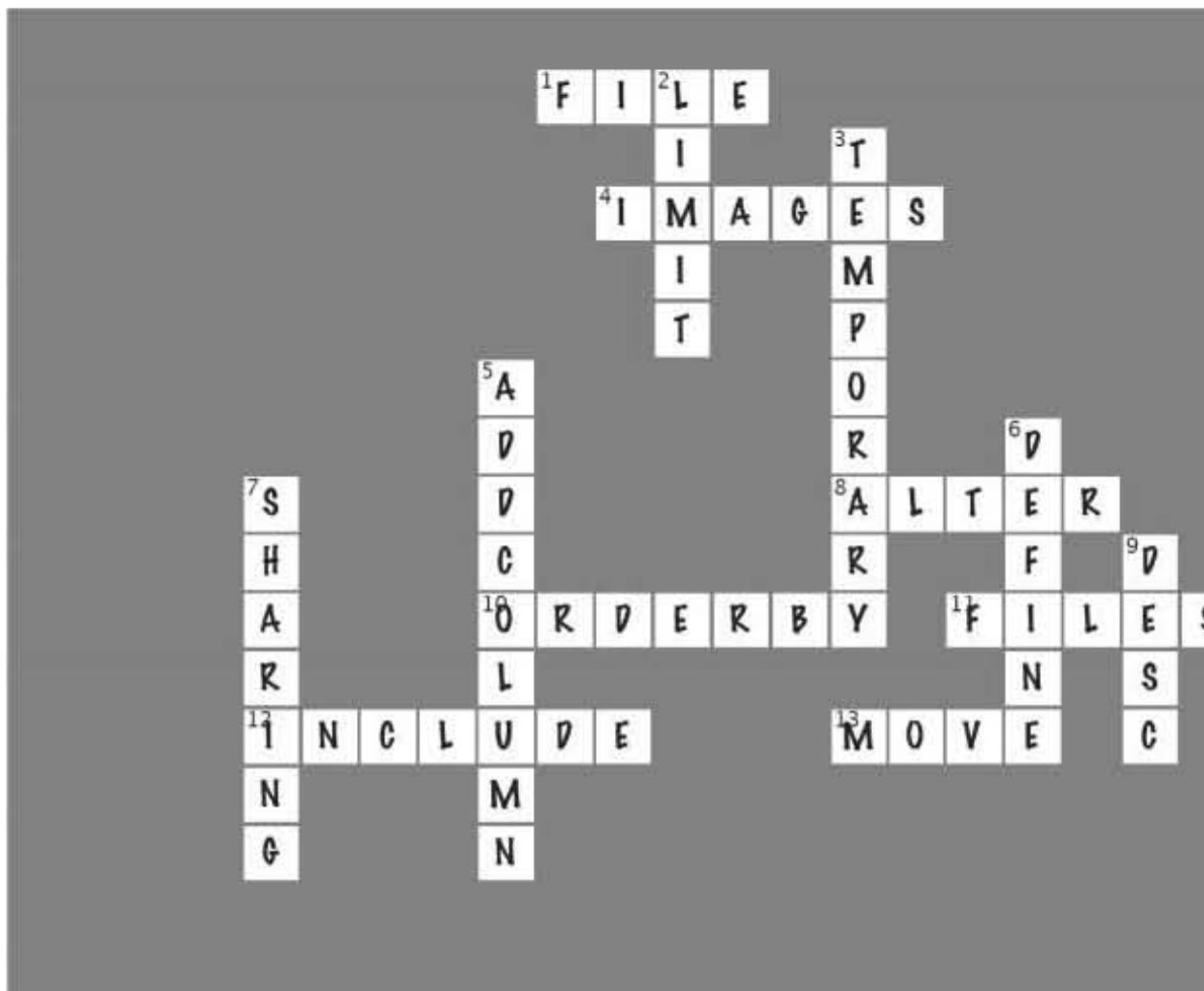
Down

- To prevent a DELETE FROM statement from deleting more than one row, use this SQL statement.
- When a file is uploaded through a form, it is stored in this folder on the web server.
- When altering a table, this SQL command is used to add a new column.
- This PHP statement is used to create a temporary file.
- Include files are very handy for dynamic script files.
- This SQL statement is used as part of a query to reverse the order of query results in descending order.

php&mysqlcross solution



PHP&MySQL cross Solution





Your PHP & MySQL Toolbox

Feel free to take a virtual bow. Not only are you loved by virtual guitarists worldwide, but you've also added quite a few new skills to your PHP and MySQL skillset: altering the structure of tables, handling file uploads, controlling the order of data, and removing data.



`ALTER TABLE table
ADD COLUMN column type`

Use this SQL statement to add a new column of data to an existing database table. The column is added to the end of the table and is initially empty for rows that are already in the database.

`include`, `include_once`,
`require`, `require_once`

These PHP statements allow you to share script code across multiple script files in an application, eliminating duplicate code and making the code easier to maintain.

`$_FILES`

This built-in PHP superglobal variable stores information about files that have been uploaded through a file input form. You can use it to determine the filename, the temporary storage location of the file, the file size, and the file type, among other things.

`ORDER BY column`

This SQL statement orders the results of a query based on a certain column of data. Use ASC or DESC after the statement to sort the data in ascending or descending order. ASC is the default ordering for ORDER BY, and is, therefore, optional.

`DELETE FROM table
WHERE column = value
LIMIT num`

Use this SQL statement to remove a row from a table. More than one row can be removed (and often should be) to improve the accuracy of the deletion, not to increase the deletion time.

6 securing your application

* Assume they're all out to get you

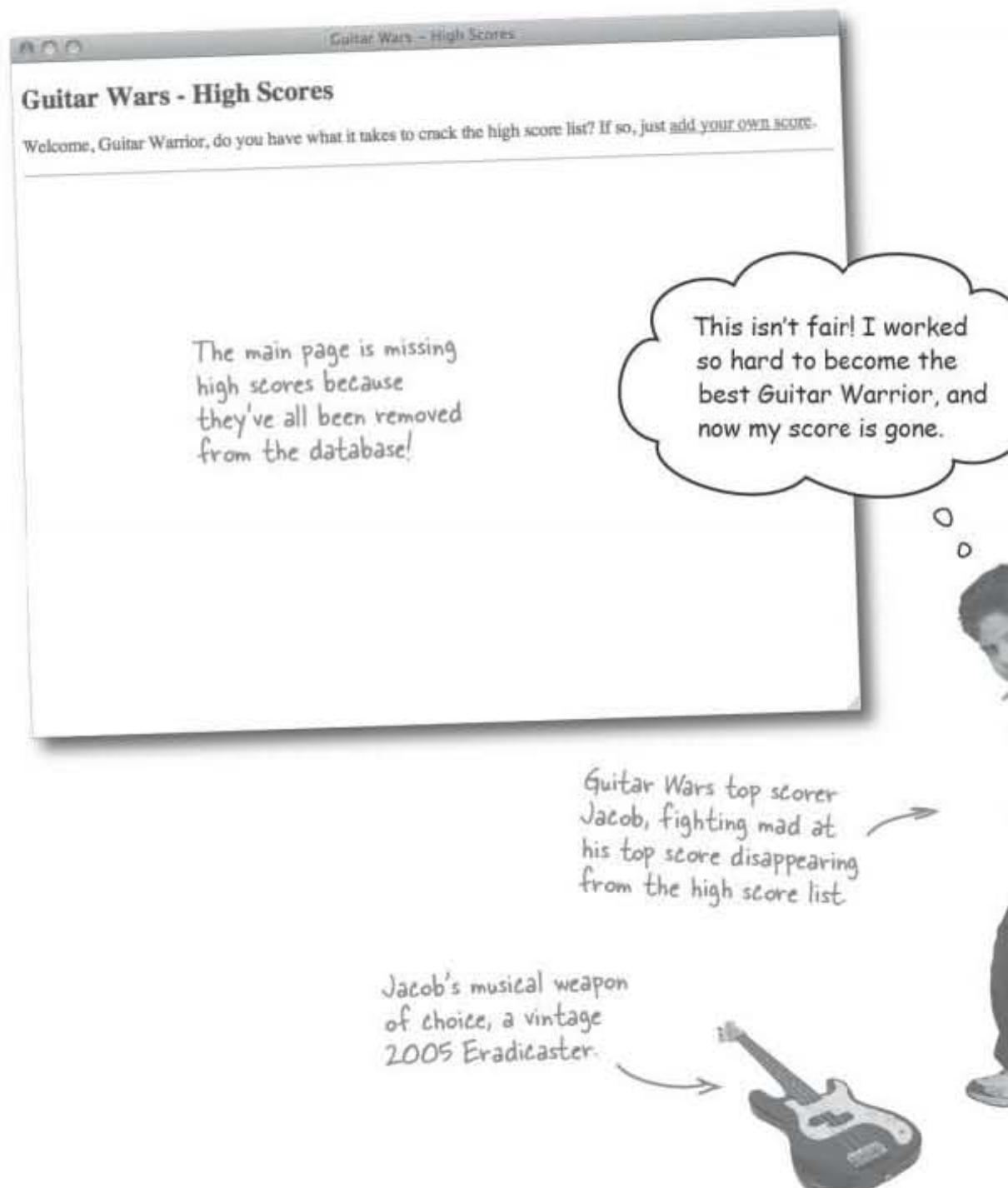


Your parents were right: don't talk to strangers. Or at least don't trust them. If nothing else, *don't give them the keys to your application data*, assuming they'll do the right thing. It's a cruel world out there, and you can't count on everyone to be trustworthy. In fact, as a web application developer, you have to be part cynic, part conspiracy theorist. Yes, people are generally bad, and they're definitely out to get you! OK, maybe that's a little extreme, but it's very important to **take security seriously and design your applications so that they're protected** against anyone who might choose to do harm.

guitar wars has been hacked

The day the music died

Uh oh, our young virtual rock prodigy's moment in the limelight has been short-lived, as Jacob's top Guitar Wars score is somehow missing, along with all the other scores. It seems a diabolical force is at work to foil the high score application and prevent Guitar Warriors from competing online. Unhappy virtual guitarists are unhappy users, and that can only lead to unhappy application developers... you!



Where did the high scores go?

We know that the main Guitar Wars page is empty, but does that mean the database is empty too? A SELECT query can answer that question:

```
File Edit Window Help iFWash
mysql> SELECT * FROM guitarwars;
+----+-----+-----+-----+
| id | date | name | score | screenshot |
+----+-----+-----+-----+
0 rows in set (0.0005 sec)
```

Somehow all of the high score rows of data have been deleted from the Guitar Wars database. Could it be that maybe someone out there is using our Remove Score script to do evil? We need to protect the scores!



Sharpen your pencil

Circle which of the following techniques you could use to protect the Guitar Wars high scores from bitter virtual guitar haters, and then implement them.



Password protect the Admin page so that only people who know the password (you!) can remove scores.

.....
.....
.....
.....
.....



Create a user registration and then only give some administrative privilege.

.....
.....
.....
.....
.....



Check the IP address of the computer trying to access the Admin page, and only allow certain ones (yours!).

.....
.....
.....
.....
.....



Eliminate the score removal altogether.

.....
.....
.....
.....
.....

A SELECT
that the
is comple
the score

protecting guitar wars' high scores

Sharpen your pencil Solution

Circle which of the following techniques you could use to get high scores from bitter virtual guitar haters, and then explain why.



Password protect the Admin page so that only people who know the password (you!) can remove scores.

Password protecting the Admin page is a good quick and dirty solution because it's not too complicated and it secures the site quickly.



Check the IP address of the computer trying to access the Admin page, and only allow certain ones (yours!).

Checking the IP address works but it makes the site dependent upon your computer's IP address, which very well may change.



Create a user registration
and then only give some
administrative privilege

A user registration system with privileges is a great solution because it requires a large amount of planning and coding. It also needs security now!



Eliminate the score removal altogether.

Removing the feature certainly specific problem, but if you recall feature was originally added in chapter to make the site easier

Securing the teeming hordes

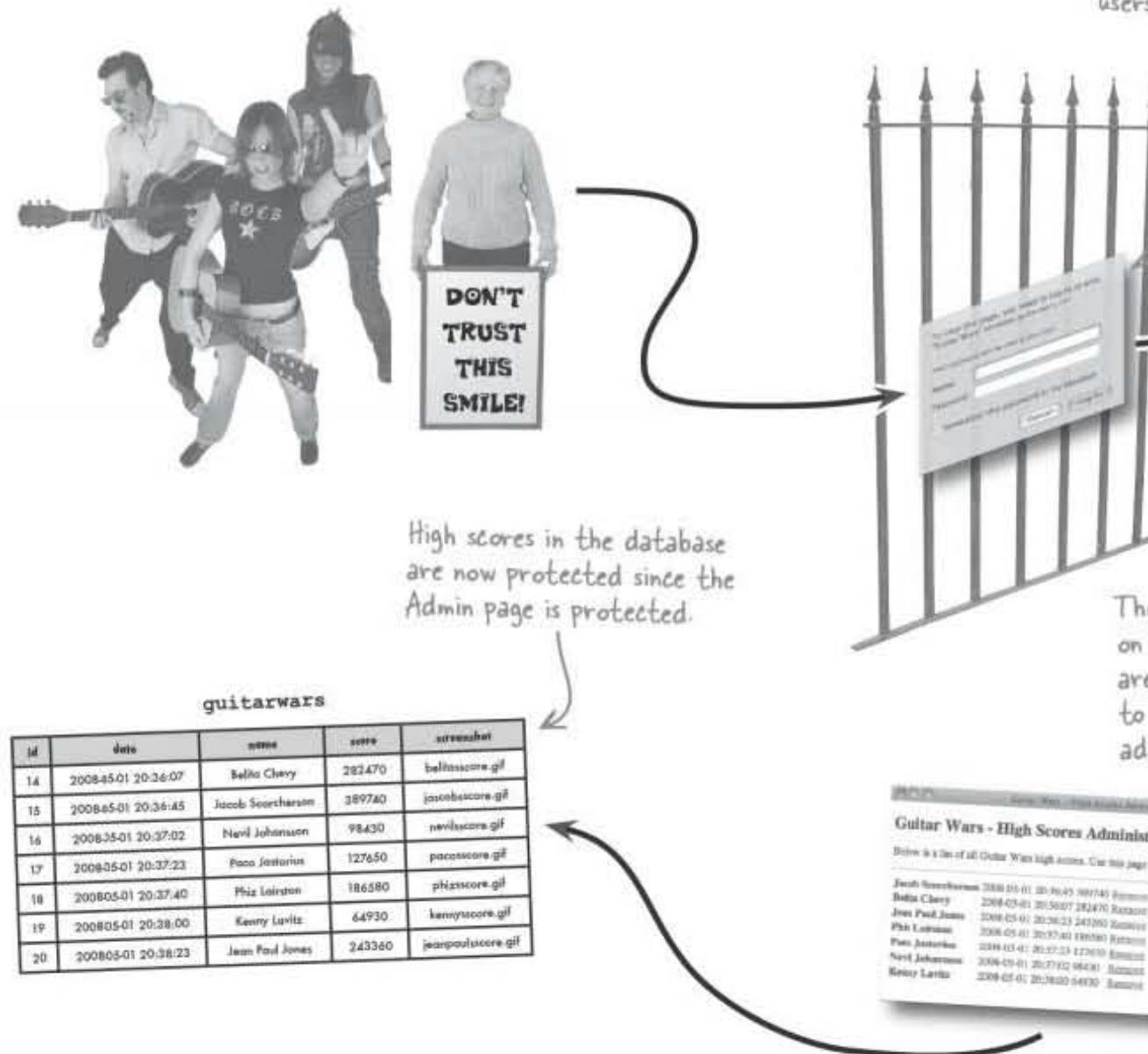
A simple and straightforward way to quickly secure the Guitar Wars high scores is to use HTTP authentication to password protect the Admin page. This technique actually involves both a user name and a password, but the idea is to require a piece of secret information from an administrator before they have access to restricted application features, such as the score removal links.

When a page is secured using HTTP authentication, a window pops up requesting the user name and password before access is allowed to the protected page. In the case of Guitar Wars, you can limit access to the Admin page to as few people as you want, potentially just you!

**HTTP auth
provides a
way to se**

using PH

The
wind
users



Guitar Wars - High Scores Admin				
Below is a list of all Guitar Wars high scores. Use the input fields to search for specific names or scores.				
Josh Greenbottom	2008-05-01 20:36:45	980745	greenbottomscore.gif	Remove
Bella Chevy	2008-05-01 20:36:45	282470	bellascore.gif	Remove
Jacob Paul Jones	2008-05-01 20:36:45	389740	jacobsscore.gif	Remove
Phiz Lairton	2008-05-01 20:37:40	186580	phizscore.gif	Remove
Paco Jesterius	2008-05-01 20:37:23	127650	pacoscore.gif	Remove
Newt Johnson	2008-05-01 20:37:43	98430	newtscore.gif	Remove
Kenny Lovitz	2008-05-01 20:38:00	64930	kennyscore.gif	Remove
Jean Paul Jones	2008-05-01 20:38:23	243360	jeanpaulscore.gif	Remove

using http authentication

Protecting the Guitar Wars Admin page

HTTP authentication works like this: when a user tries to access a page protected by authentication, such as our Admin page, they are presented with a window that asks them for a user name and password.

The web browser uses a window like this to request a user name and password before allowing access to a protected page.

To keep things simple, the password isn't encrypted.

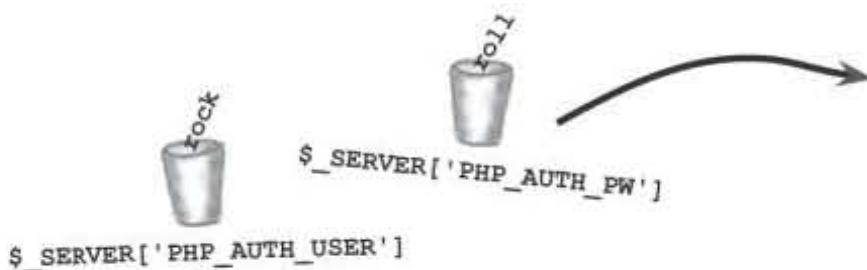


This PHP superglobal stores the user name into the authentication variable.

This variable contains the password entered by the user during authentication.

The Admin page is only accessible if the correct user name and password are entered.

PHP enters the picture through its access to the user name and password entered by the user. They are stored in the `$_SERVER` superglobal, which is similar to other superglobals you've used (`$_POST`, `$_FILES`, etc.). A PHP script can analyze the user name and password entered by the user and decide if they should be allowed access to the protected page. Let's say we only allow access to the Admin page if the user name is "rock" and the password is "roll." Here's how the Admin page is unlocked:



Guitar Wars - High Scores Administration		
Below is a list of all Guitar Wars high scores. Use this page to remove scores.		
Joseph Smirnoff	2008-05-01 20:56:47	187140
Bella Cherry	2008-03-01 20:36:07	128243
John Paul Jones	2008-03-01 20:28:23	124360
Pete Loeffler	2008-03-01 20:27:40	116590
Peter Jameson	2008-03-01 20:27:13	127109
Neil Johannsen	2008-03-01 20:27:12	294337
Henry Lewis	2008-03-01 20:28:00	84930

there are no
Dumb Questions

Q: Is HTTP authentication really secure?

A: Yes. And no. It all depends on what you're trying to accomplish with security. Nothing is ever truly 100% secure, so we're always talking about **degrees of security**. For the purposes of protecting high scores in Guitar Wars, HTTP authentication provides a reasonable level of security. You could add encryption to the password to ramp that up a bit further. However, it's probably not sufficient for an application involving data that is more sensitive, such as financial data.

Q: What happens if the user name and password are entered incorrectly?

A: The browser emits a small electrical shock through the mouse. No, it's nothing that harsh. Usually a message is displayed letting users know that they're attempting to access a secure page that is apparently none of their business. It's ultimately up to you how grim you want this message to read.

Q: Does HTTP authentication require both a user name and password?
What if I only want to use a password?

A: You aren't required to use both a user name and password. If you just want a password, focus solely on checking the `$_SERVER['PHP_AUTH_PW']` global variable. More on how this variable is checked in just a moment...

Q: How exactly do you protect a page with HTTP authentication? Do you call a PHP function?

A: Yes, you do. HTTP authentication involves establishing a line of communication between the browser and the server through HTTP headers. You can think of a header as a short little conversation between the browser and the server. Browsers and servers use headers quite often to communicate outside of the context of PHP, but PHP does allow you to send a header, which is how HTTP authentication works. We're about to dig a lot deeper into headers and their role in HTTP authentication with PHP.



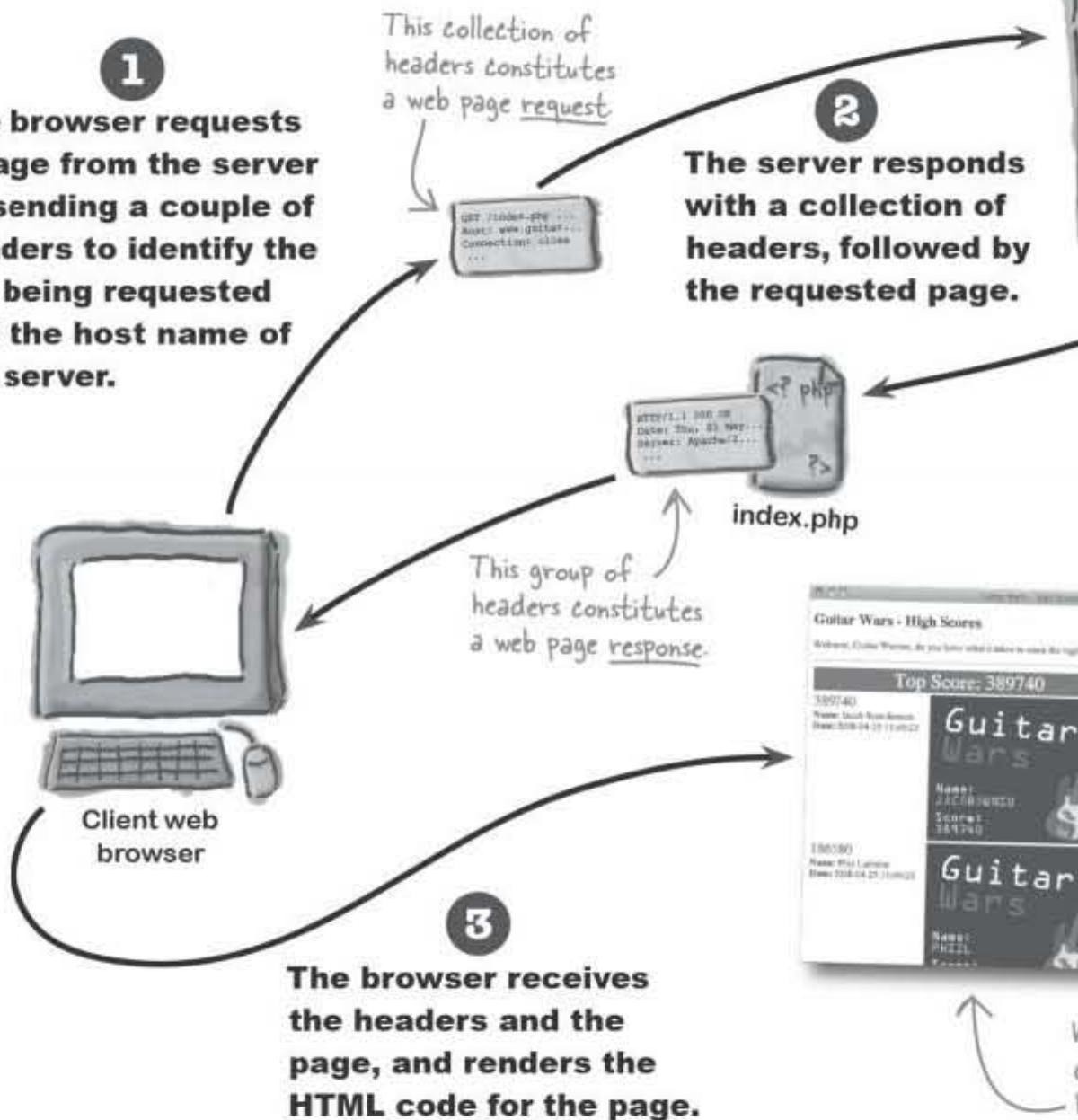
When should
authentication
page actually

authentication and headers

HTTP authentication requires headers

The idea behind HTTP authentication is that the server withholds a protected web page, and then asks the browser to prompt the user for a user name and password. If the user enters these correctly, the browser goes ahead and sends along the page. This dialog between browser and server takes place through **headers**, which are little text messages with specific instructions on what is being requested or delivered.

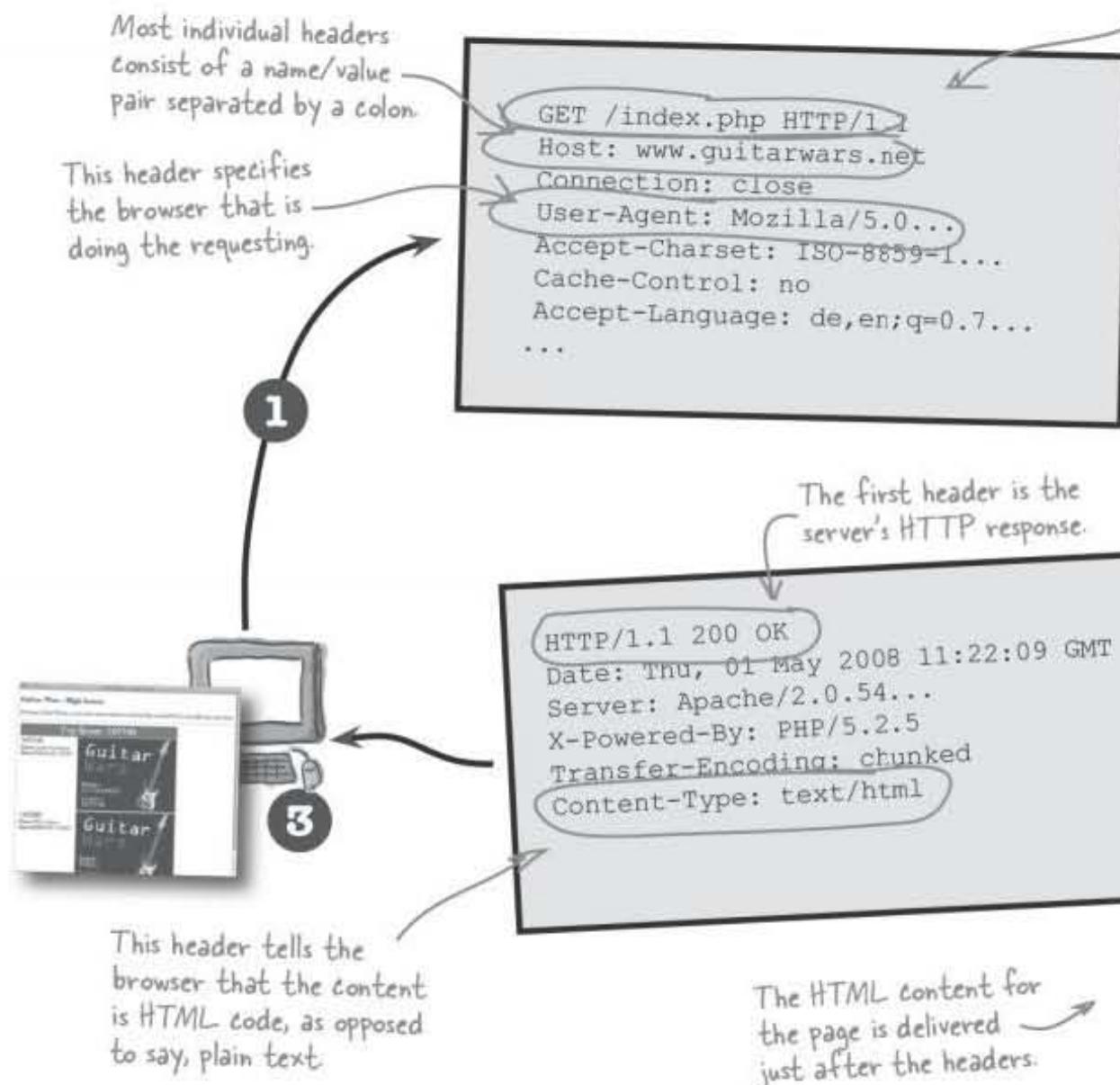
Headers are actually used **every time** you visit a web page, not just when authentication is required. Here's how a normal, unprotected web page is delivered from the server to the browser with the help of headers:





Anatomy of a header

Headers control precisely how and what kind of information is passed back and forth between a web browser and web server. An individual header often consists of a name/value pair. A group of headers is sent to the server as part of a web page request, and then another group is returned to the browser as part of the response. Let's take a closer look at these headers to find out exactly what is sent as the client and server communicate with each other.



Headers matter to us in regard to Guitar Wars because they provide the mechanism for disrupting the delivery of a page from the server and requiring the user to enter a login and password before it can be delivered. In other words, you have to tweak the headers returned by the server to protect a page with HTTP authentication.

interview with a header

Header Exposed

This week's interview:
What's all the fuss about?

Head First: You seem to be grabbing a lot of attention when it comes to authenticating web pages. Is it really justified, or are you just looking for your fifteen minutes of virtual fame?

Header: Oh, I'm justified alright. You more than likely take for granted that I play a role in delivering every single web page in existence. So I guess you could say the web wouldn't even work without me in the picture. I'll be around a lot longer than fifteen minutes, even if I do go largely underappreciated.

Head First: So what exactly is this role you play?

Header: You have to understand that web browsers and web servers aren't people, so they can't just call each other up on the phone or send a text message.

Head First: OMG!

Header: Yeah, I know, it's a little shocking but machines just don't communicate the same way people do. But browsers and servers still have to communicate, and they do so using me.

Head First: So how does that work?

Header: When someone types in a URL or clicks a link on a web page, the browser assembles a GET request that it sends to the server. This request is packaged into a series of headers, each of which contains information about the request. The headers hold information like the name and host of the page being requested, the type of browser doing the requesting, etc.

Head First: I still don't see why that's important.

Header: Well, do you think it's important when you tell the person at the coffee shop that you want a giganto vanilla espressiato with skim milk?

Head First: Of course, they need to know what I want.

Header: That's the same idea here. The browser tells the server what it wants by packaging the request up and sending it along in headers.

Head First: Interesting. But I hear you headers as well. I thought servers just...

Header: Ah, good question. I am on the other side of the communication, so I has to do more than just dump a bunch of data to the browser. The browser wouldn't have any idea what to do with it without knowing a bit more about me.

Head First: Such as what?

Header: The type of the content, probably the most important thing. I also sends along other stuff like the size, date and time of the delivery, and...

Head First: When does the web server send that?

Header: Right after the server sends the headers, it follows up with the actual content, such as PDF data, or image data such as a...

Head First: OK, I'm starting to get it in regard to normal web pages. But what about authentication stuff?

Header: I play the same role for authentication as I do for a normal web page. I just take care of letting the browser know that the user is authenticated. That way the browser can ask me for authentication information.

Head First: You mean a user name and password?

Header: Exactly. And then it's up to the server to decide if the user name and password are valid, in which case, the server can go ahead and serve the rest of the page.

Head First: Fascinating. Thanks for the interview.

Header: No problem. That's just my job.

Take control of headers with PHP

Using PHP, you can carefully control the headers sent by the server to the browser, opening up the possibilities for performing header-driven tasks such as HTTP authentication. The built-in `header()` function is how a header is sent from the server to the browser from within a PHP script.

```
header('Content-Type: text/html');
```

The `header()` function immediately sends a header from the server to the browser and must be called before any actual content is sent to the browser. This is a very strict requirement—if even a single character or space is sent ahead of a header, the browser will reject it with an error. For this reason, calls to the `header()` function should precede any HTML code in a PHP script:

Even an errant space
before the `<?php` tag
would cause an error
in this example script.

```
<?php
```

Spaces inside of the
`<?php ?>` tags aren't
a problem because
they aren't passed
along to the browser.

```
header('Content-Type: text/html');
```

...

?>

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="
```

...

```
</html>
```

The server sends
the browser for
attempting to se
HTML content in

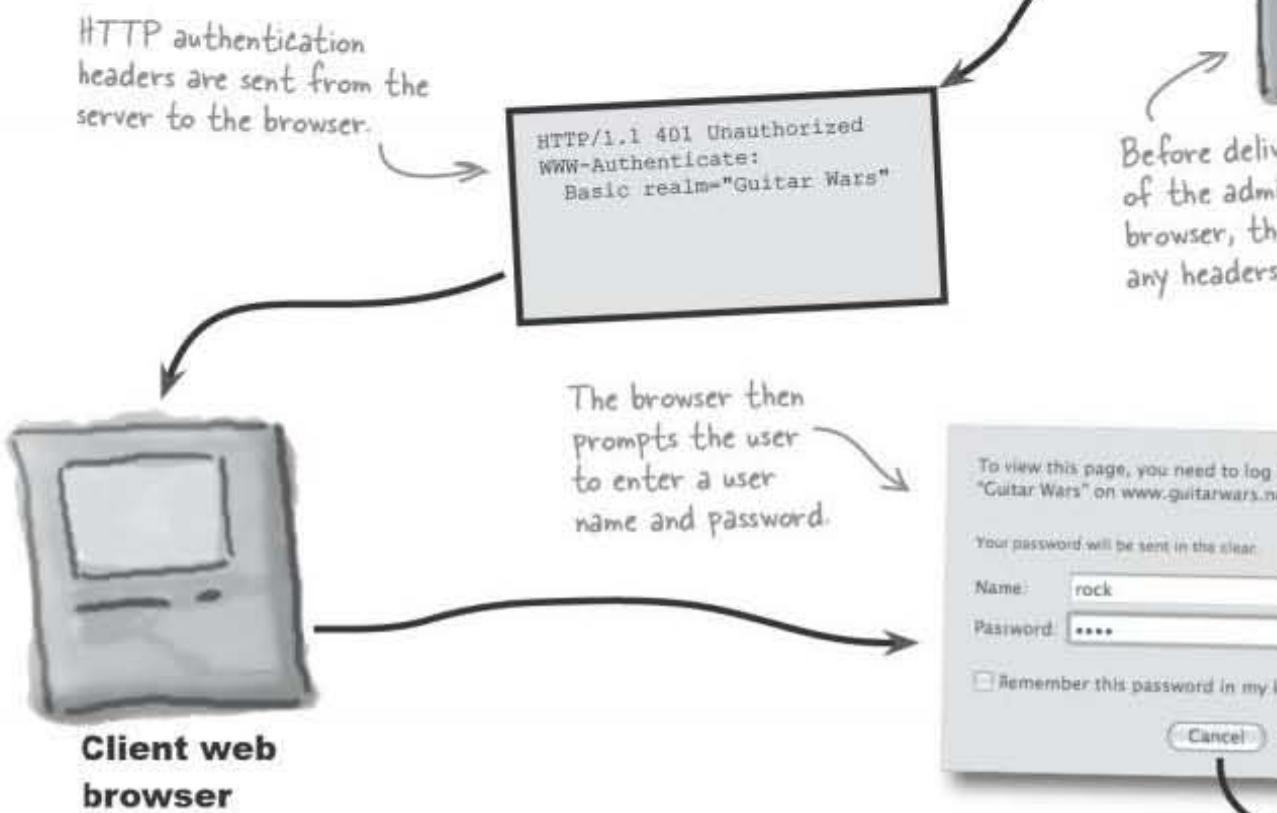
All this header stuff is fa
but how do we actually use
protect pages with authen



how header authentication works

Authenticating with headers

Authenticating the Guitar Wars Admin page using headers involves crafting a very specific set of headers, two in fact, that let the browser know to prompt the user for a user name and password before delivering the page. These two headers are generated by PHP code in the Admin script, and control the delivery of the page to the browser.



Two specific headers are required to request the authentication of a web page.

The two headers required to initiate authentication do two specific things:

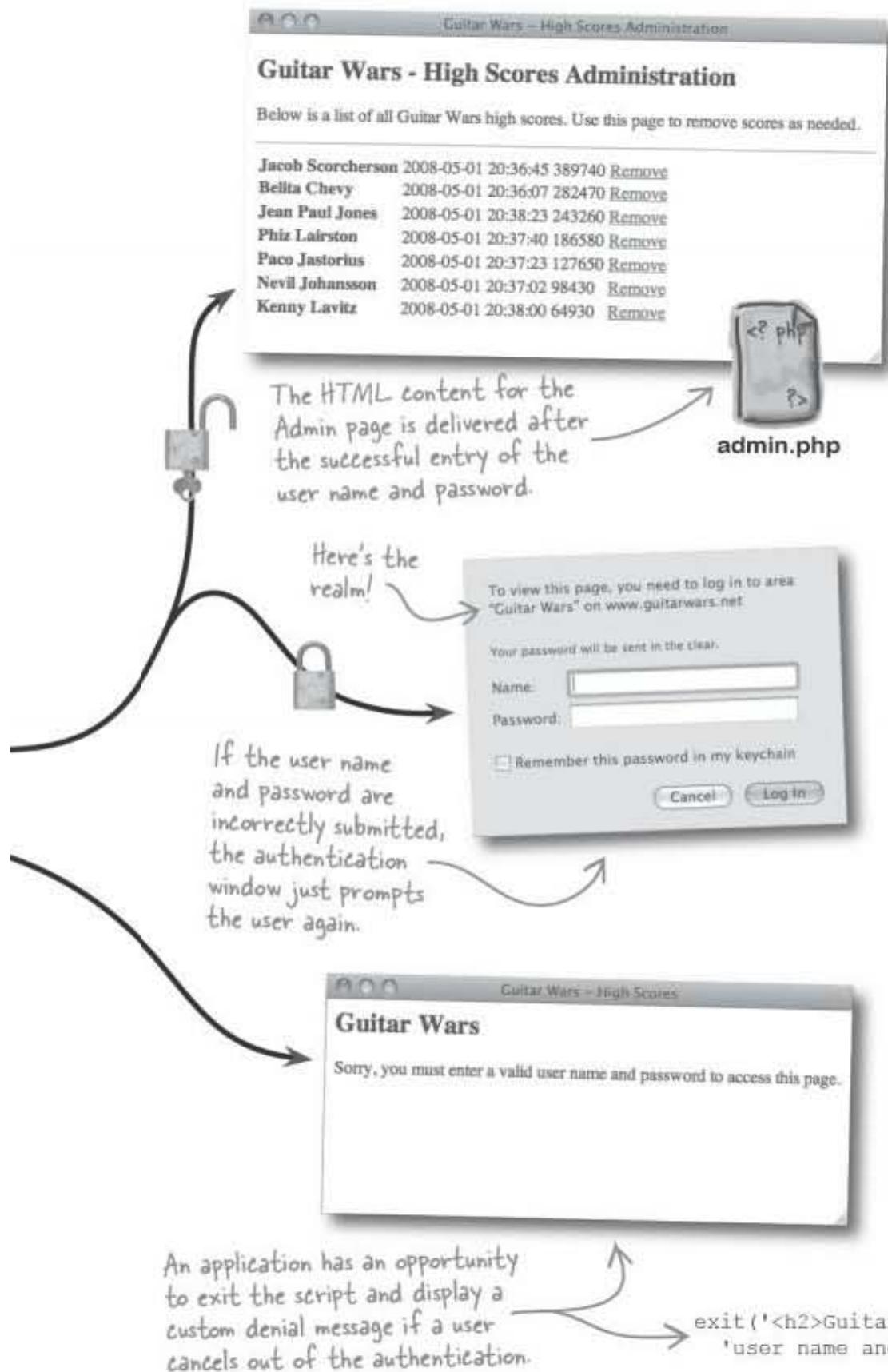
This header asks the browser to attempt to authenticate the user by prompting for a user name and password.

HTTP/1.1 401 Unauthorized

WWW-Authenticate: Basic realm="Guitar Wars"

The "basic realm" is just a placeholder used to uniquely identify this particular authentication—it's displayed in the authentication window.

After processing the authentication headers, the browser waits for the user to take action via the authentication window. The browser takes a dramatically different action in response to what the user does...



If the user enters a user name and password, and the HTML content is delivered to the browser, the page, and the user sees like the previous slide.

If the user enters a user name and password, and the browser to prompt continues this, it keeps entering combinations until it knows the user is out. The way out is to

If the user clicks the cancel button, the authentication window with a denial message. The admin.php page is controlled by a PHP script that is close to the end of the file. This code calls the exit() function to display a message.

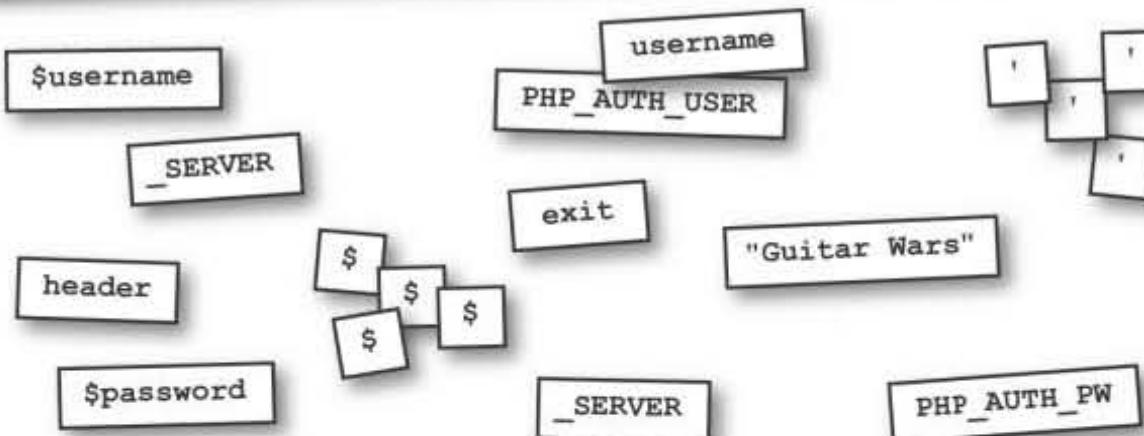
finish the authentication code



PHP Magnets

The Guitar Wars Admin script is missing several important pieces of PHP code that provide HTTP authentication. Use the magnets to fill in the missing code and use headers to make the Admin page secure. Hint: Some magnets may be used more than once.

```
<?php
// User name and password for authentication
    ..... = 'rock';
    ..... = 'roll';
.....
if (!isset(.....)) {
    !isset(.....) || ($_SERVER['PHP_AUTH_USER'] != .....)) || ($_SERVER['PHP_AUTH_PW'] == .....);
    // The user name/password are incorrect so send the authentication headers
        .....('HTTP/1.1 401 Unauthorized');
        .....('WWW-Authenticate: Basic realm=.....');
        .....('<h2>Guitar Wars</h2>Sorry, you must enter a valid user name and
access this page.');
}
?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    ...
</html>
```





I wonder if it's possible
to send other kinds of
headers using PHP?

The browser is
redirected to
the About page
upon receiving
this header.

Indeed it is... headers aren't just for...

Although authentication presents the immediate need for headers, they are quite flexible and can do lots of other interesting things. For example, you can use the header() function with the appropriate name/value pair:

```
<?php
header('Location: http://www.guitarwars.net/about.php');
?>
```

The header is called a **location header** and redirects the browser to a page called about.php on the same Guitar Wars site. You can also use a similar header to redirect to the about.php page on another site:

The browser is
redirected to the
About page after
5 seconds.

```
<?php
header('Refresh: 5; url=http://www.guitarwars.net/about.php');
echo 'In 5 seconds you\'ll be taken to the About page';
?>
```

This header is called a **refresh header** since it refreshes the current page after a period of time has elapsed. You often see the URL in the header so that it references the current page so that it refreshes itself.

One last header is called a **content type header**. It specifies the type of content being delivered by the server. A common use is to force a page to be plain text, as opposed to HTML. You can set this header when calling the header() function:



Watch it!

**Headers must
be the very
first thing sent
to the browser
in a PHP file.**

Because headers must be sent before any content, it is extremely important to not allow even a single space to appear outside of PHP code before calling the header() function in a PHP script.

The content type header
tells the browser what kind of content is coming.

```
<?php
header('Content-Type: text/plain');
echo 'This <strong>text</strong> won\'t actually be rendered';
?>
```

In this example, the text echoed to the browser is plain text and will be shown with no special formatting. In other words, the browser **not** to render the echoed content as HTML. Instead, the words are displayed literally as text.

the completed authentication code



PHP Magnets Solution

The Guitar Wars Admin script is missing several important pieces of PHP code that provide HTTP authentication. Use the magnets to fill in the missing code and use headers to make the Admin page secure. Hint: Some magnets may be used more than once.

```
<?php
// User name and password for authentication
...     $username = 'rock';           ← The user name and password
...     $password = 'roll';           ← are stored in variables at
...                               the start of the script.
if (!isset($_SERVER['PHP_AUTH_USER'])) ||
!isset($_SERVER['PHP_AUTH_PW']) ||
($_SERVER['PHP_AUTH_USER'] != $username) || ($_SERVER['PHP_AUTH_PW'])

// The user name/password are incorrect so send the authentication headers
header('HTTP/1.1 401 Unauthorized');
header('WWW-Authenticate: Basic realm= "Guitar Wars"');
exit('<h2>Guitar Wars</h2>Sorry, you must enter a valid user name
     and password to access this page.');
?>

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<...>
</html>
```

No HTML code is delivered to the browser until after the headers are sent and processed.

HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Guitar Wars"



Test Drive

Add HTTP authorization to the Admin script.

Modify the admin.php script to use HTTP authentication so that only the correct user can access it. Upload the script to your web server and then open it in your web browser. Enter a wrong user name and password first to see how access is restricted.



*there are no
Dumb Questions*

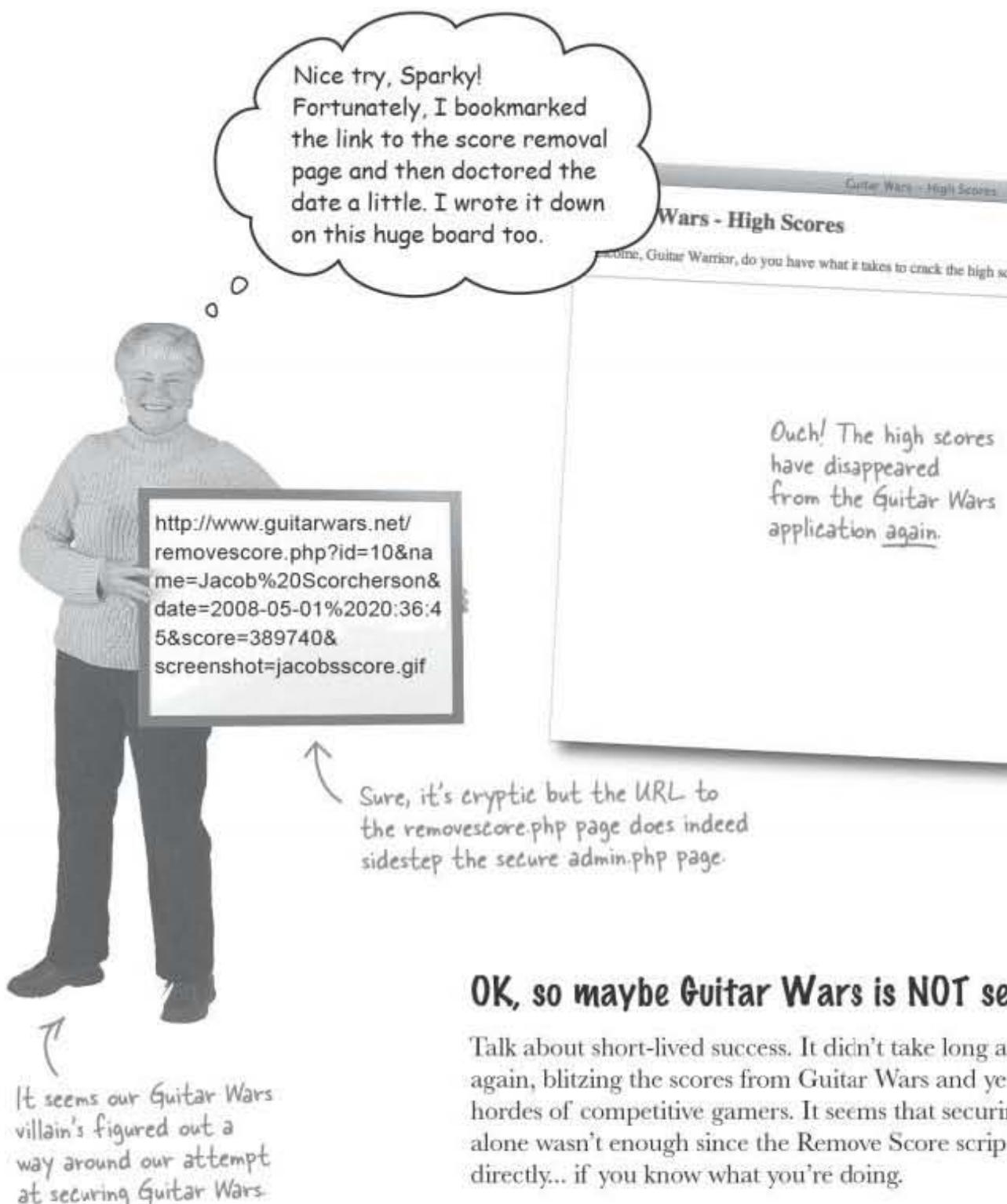
Q: When exactly does the `exit()` function get called in the Guitar Wars Admin script?

A: Even though the `exit()` function appears in the PHP code just below the two calls to the `header()` function, it's only called if the user cancels out of the authentication window by clicking the Cancel button. If the authentication fails, the server doesn't continue executing past the two `header()` calls. Instead, it resends the headers and tries again. Only if the user clicks Cancel does the server make it to the `exit()` function, in which case it sends along the content within the function call and nothing else. If the authentication succeeds, `exit()` isn't called because the script never makes it inside the `if` statement—the code inside the `if` statement is only executed if the user name and password aren't set or have been entered incorrectly.

Q: Does the "basic realm" have any real purpose?

A: Yes. It defines a security realm for the user name and password. If multiple realms have been successfully entered, the user can remember it and not continually enter it again. Subsequent authentication requests for other realms allow a browser to skip the realm selection requirements for a given realm for the authentication process.

another security problem...

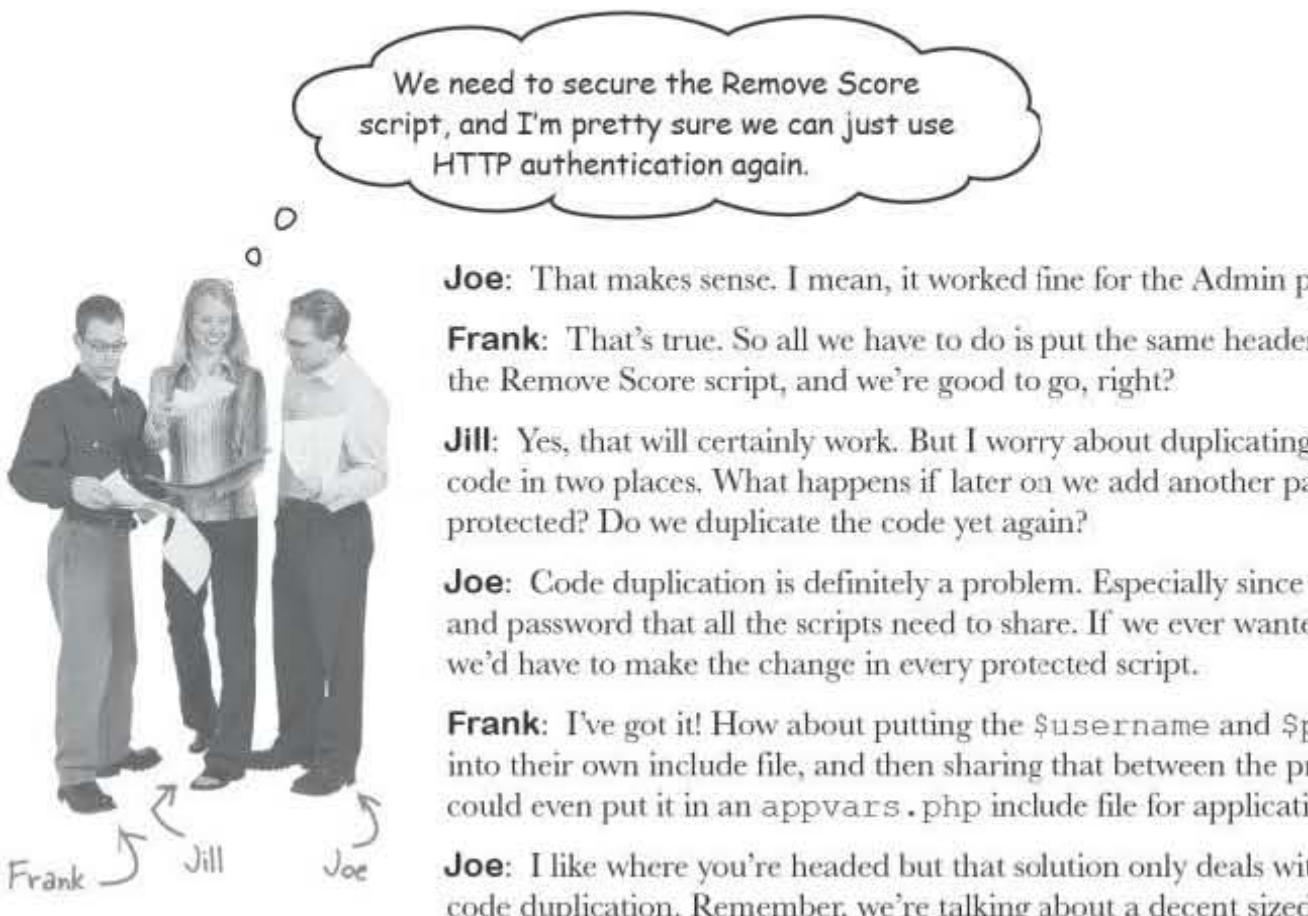


OK, so maybe Guitar Wars is NOT secure.

Talk about short-lived success. It didn't take long at all for the villain to figure out a way around our attempt at securing the application again, blitzing the scores from Guitar Wars and yet again, blinding us to the fact that we still had a problem. It seems that securing the application alone wasn't enough since the Remove Score script can be run directly... if you know what you're doing.

Write down how you think we can solve this problem to prevent high scores from being deleted:

.....
.....
.....



```
<?php
// User name and password for authentication
$username = 'rock';
$password = 'roll';

if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW']) ||
    ($_SERVER['PHP_AUTH_USER'] != $username) || ($_SERVER['PHP_AUTH_PW'] != $password))
// The user name/password are incorrect so send the authentication headers
header('HTTP/1.1 401 Unauthorized');
header('WWW-Authenticate: Basic realm="Guitar Wars"');
exit('<h2>Guitar Wars</h2>Sorry, you must enter a valid user name and password to access the system');
?>

<html>
```

Jill: You're both right, and that's why I think we need a new include file that stores all the authorization code, not just the \$username and \$password variables.

Frank: Ah, and we can just include that script in any page we want to protect with.

Joe: That's right! We just have to make sure we always include it first thing since it handles all the HTTP authorization stuff.

creating authorize.php

Create an Authorize script

We already have all the code we need for a new Authorize script; it's just a matter of moving the code from `admin.php` to a new script file (`authorize.php`), and replacing the original code with a `require_once` statement.

We're
from
we ca
script



```
<?php
// User name and password for authentication
$username = 'rock';
$password = 'roll';

if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW']) ||
    ($_SERVER['PHP_AUTH_USER'] != $username) || ($_SERVER['PHP_AUTH_PW'] != $password)) {
    // The user name/password are incorrect so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Guitar Wars"');
    exit('<h2>Guitar Wars</h2>Sorry, you must enter a valid user name and password to access this page.');
}

<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Guitar Wars-High Scores Administration</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
    <h2>Guitar Wars-High Scores Administration</h2>
    <p>Below is a list of all Guitar Wars high scores. Use this page to remove scores as needed.<br />
    <?php
        require_once('appvars.php');
        require_once('connectvars.php');

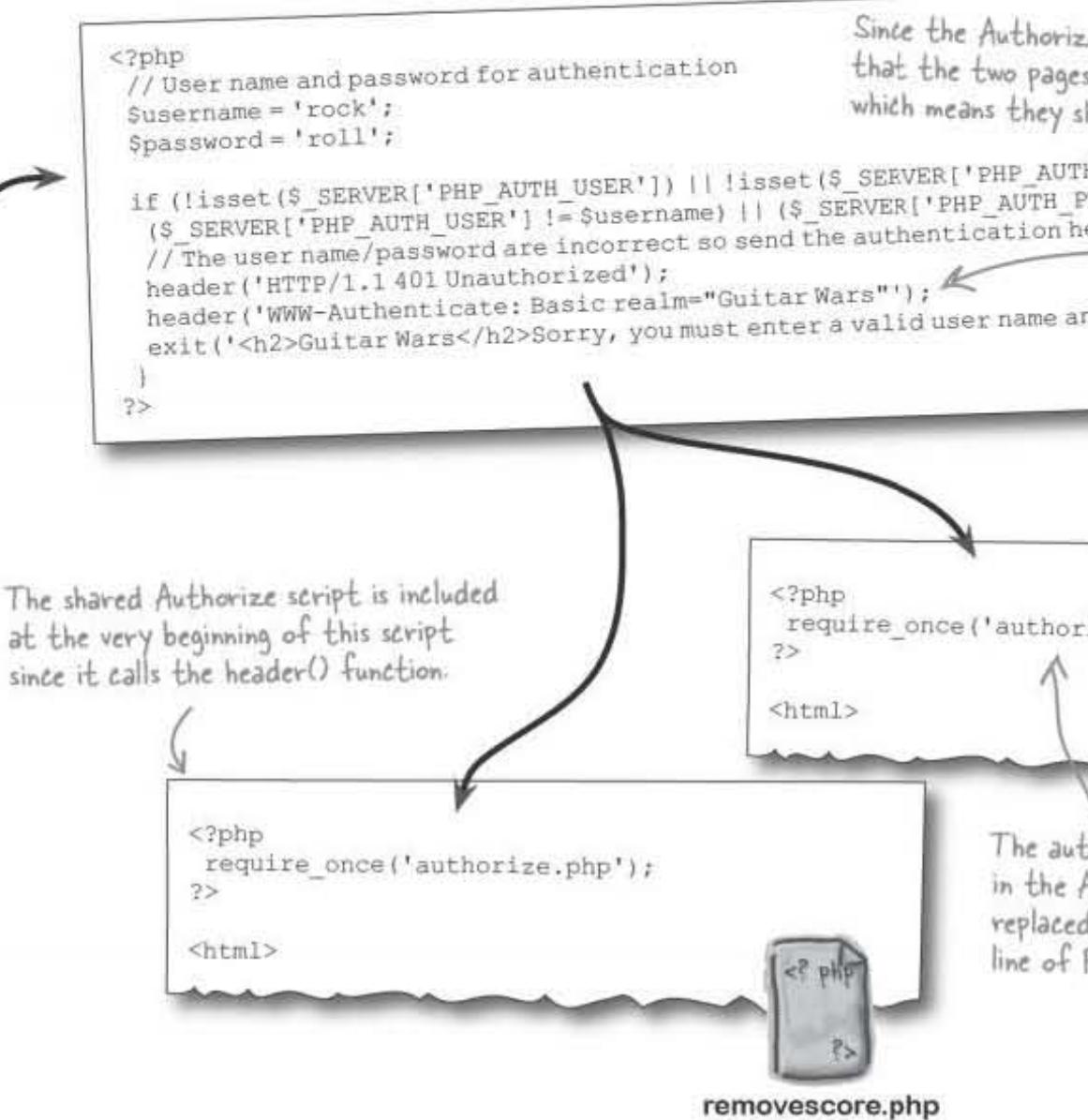
        // Connect to the database
        $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

        // Retrieve the score data from MySQL
        $query = "SELECT * FROM guitarwars ORDER BY score DESC, date ASC";
        $data = mysqli_query($dbc, $query);

        // Loop through the array of score data, formatting it as HTML
        echo '<table>';
        while ($row = mysqli_fetch_array($data)) {
            // Display the score data
            echo '<tr class="scorerow"><td><strong>' . $row['name'] . '</strong></td>';
            echo '<td>' . $row['date'] . '</td>';
            echo '<td>' . $row['score'] . '</td>';
            echo '<td><a href="removescore.php?id=' . $row['id'] . '&date=' . $row['date'] .
                '&name=' . $row['name'] . '&score=' . $row['score'] .
                '&screenshot=' . $row['screenshot'] . '">Remove</a></td></tr>';
        }
        echo '</table>';

        mysqli_close($dbc);
    >

```



BULLET POINTS

- PHP scripts can use headers to control how the server delivers web content to the browser.
- The built-in PHP `header()` function is used to send headers to the browser, which can be used to redirect a page, control the content type of a page, or request the authentication of a page.
- When headers are sent to the browser using the `header()` function, calls to the `header()` function must come before any other content is sent.
- When a page is protected by basic authentication, the user name and password are stored in the `$_SERVER` superglobal variable.
- The "basic realm" of a page is the security zone that gets the user name and password, which are then secured together.
- The built-in PHP `exit()` function prevents any code from being executed after it, preventing any code from being sent to the browser.

security no dumb questions

there are no Dumb Questions

Q: I still don't fully understand how Ethel got around the security in Guitar Wars. What did she do?

A: She capitalized on the weakness inherent in only protecting one page (Admin) when the remove score feature really relies on two pages (Admin and Remove Score). The Admin page presents a series of Remove links that link to the Remove Score page. The specifics about which score to remove are passed in the URL, allowing the Remove Score script to access them through the `$_GET` superglobal. If you were able to put together a legit URL for the Remove Score page, you could remove scores without even going through the Admin page. That's what Ethel did.

Q: But how did she know how to structure the URL to the Remove Score page?

A: She's pretty crafty, but this task didn't require a genius. Remember she mentioned bookmarking the Remove Score page back when the whole site was unprotected. Well, a bookmark is just a URL, and she was able to use it to construct a URL that directly accessed the Remove Score page without having to go through the Admin page.

Q: OK, but the high scores had been re-entered since the previous attack. Doesn't that mean the old URLs wouldn't work since the dates are different?

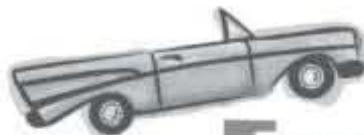
A: Yes, that's a very good point. But remember, Ethel is pretty clever. She could easily look at the main Guitar Wars page and see the new dates, which she then plugged into the old URL to remove the new scores without any trouble. It's important to never underestimate the ability of determined people to reverse-engineer your PHP scripts and exploit weaknesses.

Q: Alright, so protecting both the Admin and Remove Score pages stops Ethel, but don't they now make it a total hassle to remove scores legitimately?

A: No, not at all. Without the help of realms, it would definitely be a hassle removing scores legitimately because you'd have to enter the user name and password separately for the Admin and Remove Score pages. But remember that a realm was established that is the same in both pages, meaning that the pages fall under the same security zone. And once you go through the authentication window for a page in a given realm, the user name and password are remembered throughout the realm. The end result is that successfully entering the user name and password once is sufficient to unlock both pages.



Never under
ability of de
people to re
your PHP s
exploit weak



Test Drive

Create the Authorize script and include it in the Admin and Score scripts to secure them.

Create a new text file called `authorize.php`, and enter the code for the Authorize script into it. Then modify the `admin.php` script so that it includes the `Authorize.php` file at the beginning of the `removescore.php` script so that it is also protected by HTTP authentication.

Upload all of the scripts to your web server and then try to open the Remove Score page directly in your web browser. You may have to clear any previous HTTP authentication sessions in your browser for it to prompt you again—most browsers remember an authentication realm so that you don't have to keep re-entering the user name and password.

The screenshot shows a browser window titled "Guitar Wars - High Scores". The address bar contains the URL `http://www.guitarwars.net/removescore.php?id=10&name=Jacob%20Scorcherson&date=2008-05-01%2020:36:45&score=389740&screenshot=jacobsscore.gif`. A callout bubble points to this URL with the text: "This URL bypasses the Admin page and accesses the Remove Score page directly."

A separate callout bubble points to the right side of the browser window with the text: "The Remove Score page is protected regardless of how the user gets to it."

On the right side of the browser window, there is a sidebar with fields for "Your password", "Name", "Password", and a "Remember me" checkbox. The text "To view this 'Guitar Wars - Remove Score' page, you must log in." is displayed above the sidebar.



Can you think of any other ways the Guitar Wars high score application is at risk?

The screenshot shows a browser window titled "Guitar Wars - Remove Score". The page displays the message: "The high score of 314340 for Bill was removed." Below this message is a link "[<< Back to admin page](#)".

a fake score debacle

High Score

Guitar Wars Episode II : Attack of the Clones

Sadly, happiness in the Guitar Wars universe didn't last for long, as bogus scores are showing up in the application in place of legitimate scores... and still inciting rage throughout the Guitar Wars universe. Apparently it's entirely possible to disrupt the Guitar Wars high score list without removing scores. But how?

Guitar Wars - High Scores

Welcome, Guitar Warrior, do you have what it takes to crack the high score list? If so, just add your own score.

Top Score: 500000																																													
500000	Name: Ethel Heckel Date: 2008-05-02 14:02:54																																												
389740	Name: Jacob Scorcherson Date: 2008-05-01 20:36:45																																												
 <p>Guitar Wars</p> <p>Name: EthelHeck Score: 500000</p>																																													
 <p>Guitar Wars</p> <p>Name: JACOBOWNSU Score: 389740 20:36:07</p>																																													
<table border="1"> <thead> <tr> <th>Rank</th> <th>Date</th> <th>Name</th> <th>Score</th> </tr> </thead> <tbody> <tr><td>22</td><td>2008-05-01 20:36:45</td><td>Belita Chevy</td><td>282470</td></tr> <tr><td>23</td><td>2008-05-01 20:37:02</td><td>Jacob Scorcherson</td><td>389740</td></tr> <tr><td>24</td><td>2008-05-01 20:37:23</td><td>Nevil Johansson</td><td>98430</td></tr> <tr><td>25</td><td>2008-05-01 20:37:40</td><td>Paco Jastorius</td><td>127650</td></tr> <tr><td>26</td><td>2008-05-01 20:38:00</td><td>Phiz Lairston</td><td>186580</td></tr> <tr><td>27</td><td>2008-05-01 20:38:23</td><td>Kenny Lavitz</td><td>64930</td></tr> <tr><td>28</td><td>2008-05-01 21:14:56</td><td>Jean Paul Jones</td><td>243260</td></tr> <tr><td>29</td><td>2008-05-01 21:15:17</td><td>Leddy Gee</td><td>308710</td></tr> <tr><td>30</td><td>2008-05-02 14:02:54</td><td>T-Bone Taylor</td><td>354190</td></tr> <tr style="outline: 2px solid #ccc;"><td>30</td><td>2008-05-02 14:02:54</td><td>Ethel Heckel</td><td>500000</td></tr> </tbody> </table>		Rank	Date	Name	Score	22	2008-05-01 20:36:45	Belita Chevy	282470	23	2008-05-01 20:37:02	Jacob Scorcherson	389740	24	2008-05-01 20:37:23	Nevil Johansson	98430	25	2008-05-01 20:37:40	Paco Jastorius	127650	26	2008-05-01 20:38:00	Phiz Lairston	186580	27	2008-05-01 20:38:23	Kenny Lavitz	64930	28	2008-05-01 21:14:56	Jean Paul Jones	243260	29	2008-05-01 21:15:17	Leddy Gee	308710	30	2008-05-02 14:02:54	T-Bone Taylor	354190	30	2008-05-02 14:02:54	Ethel Heckel	500000
Rank	Date	Name	Score																																										
22	2008-05-01 20:36:45	Belita Chevy	282470																																										
23	2008-05-01 20:37:02	Jacob Scorcherson	389740																																										
24	2008-05-01 20:37:23	Nevil Johansson	98430																																										
25	2008-05-01 20:37:40	Paco Jastorius	127650																																										
26	2008-05-01 20:38:00	Phiz Lairston	186580																																										
27	2008-05-01 20:38:23	Kenny Lavitz	64930																																										
28	2008-05-01 21:14:56	Jean Paul Jones	243260																																										
29	2008-05-01 21:15:17	Leddy Gee	308710																																										
30	2008-05-02 14:02:54	T-Bone Taylor	354190																																										
30	2008-05-02 14:02:54	Ethel Heckel	500000																																										

Ethel's top score suspect due to doctored screen the fact that to score exact

Subtraction by addition

Until now we've operated under the assumption that any high score submitted with a screen shot image is considered verified. It's now reasonably safe to say this is not the case! And it's pretty clear who the culprit is...



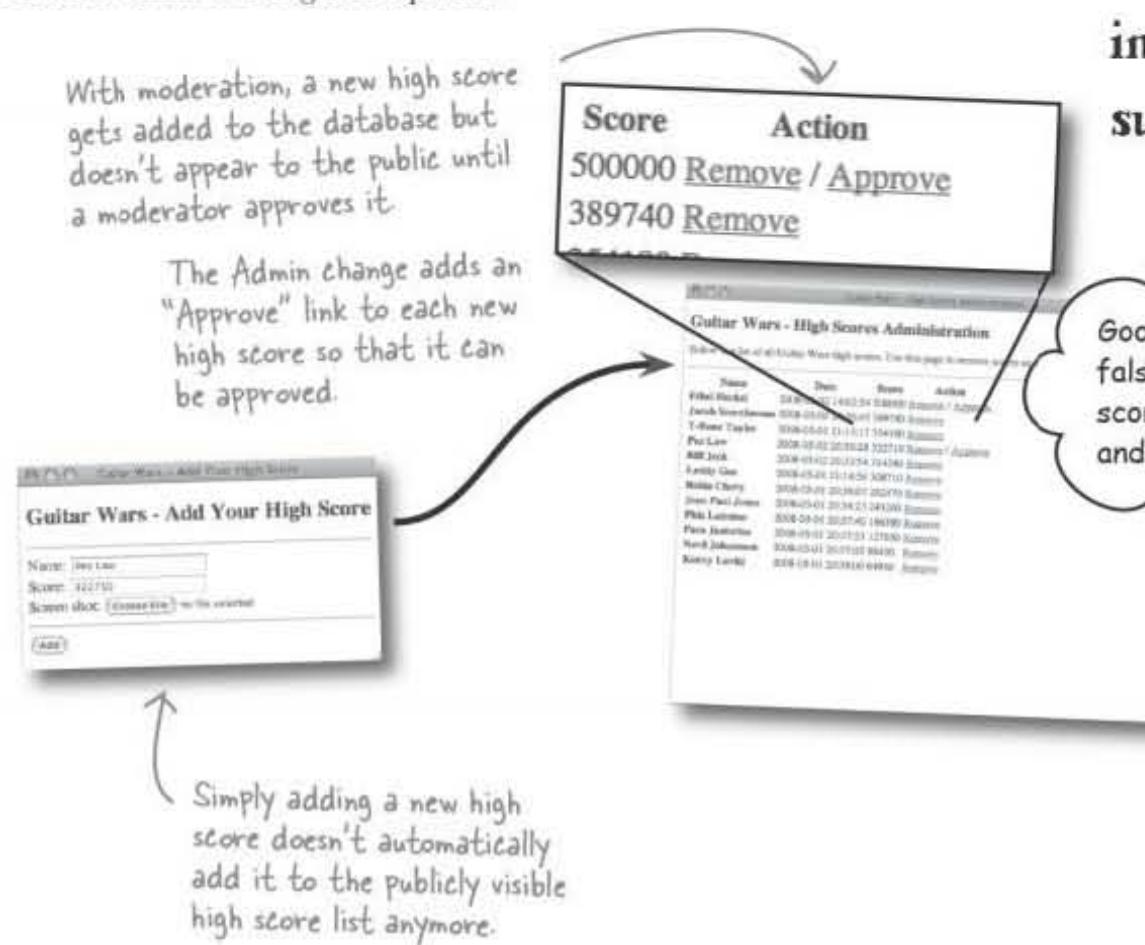
Write down how you would solve the problem of people being able to post bogus high scores to the Guitar Wars website.

.....
.....
.....

guitar wars needs human moderation

Security requires humans

Even in this modern world we live in, sometimes you can't beat a real live thinking, breathing human being. In this case, it's hard to beat a real person when it comes to analyzing a piece of information and assessing whether or not it is valid. We're talking about moderation, where a human is put in charge of approving content posted to a web application before it is made visible to the general public.



Guitar Wars could really use some human moderation. Sure, it's still possible that someone could carefully doctor a screen shot and maybe still sneak a score by a human moderator. But it wouldn't be easy, and it doesn't change the fact that moderation is a great deterrent. Keep in mind that securing a PHP application is largely about prevention.

Our fearless Guitar Wars moderator... never met a high score he really and truly trusted.

Plan for moderation in Guitar Wars

Adding a human moderation feature to Guitar Wars is significant because it affects several parts of the application. The database must change, a new script must be created to carry out an approval, the Admin page must add an “Approve” link to each score, and finally, the main page must change to only show approved scores. With this many changes involved, it’s important to map out a plan and carry out each change one step at a time.

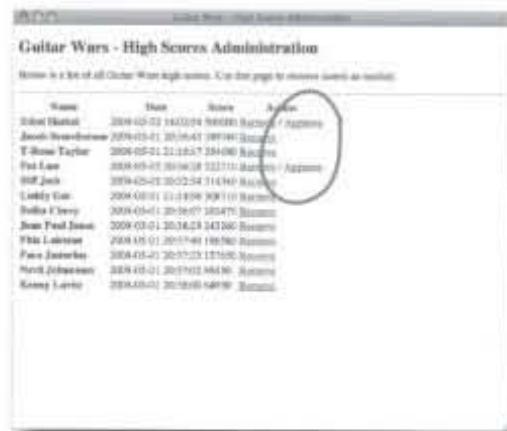
1 Use ALTER to add an approved column to the table.

Let’s start with the database, which needs a new column for keeping up with whether or not a score has been approved.

ID	Date	Name	Score	Approved	Approved
28	2008-05-01 21:12:56	Lucky Guy	308710	luckyguy.gif	0
29	2008-05-01 21:15:17	T-Ross Taylor	534190	trosscore.gif	0
30	2008-05-02 14:02:34	Ehaf Hectal	500000	ehafscore.gif	0
31	2008-05-02 20:32:34	Geff Jack	314340	geffscore.gif	0
32	2008-05-02 20:36:38	Pete Low	323710	petescore.gif	0

3 Modify the Admin page to include an “Approve” link for scores that have yet to be approved.

The Approve Score script is a back-end script that shouldn’t normally be accessed directly. Instead, it is accessed through “Approve” links generated and displayed on the Admin page—only unapproved scores have the “Approve” link next to them.



2 Create an Approve Score script to handle approving a score (sets the approved column to 1).

With the database ready to accept score approvals, you need a script responsible for looking up a score in the database and changing the approved column to 1.



4 Change the query on the main page to only show approved scores.

The last step is to make sure the query gets factored into the main page of the application. If the high scores that have been approved change, all the other approved scores will be pointless.



add an approved column to the guitarwars table

Make room for approvals with ALTER

Adding the new approved column to the guitarwars table involves a one-time usage of the `ALTER TABLE` statement, which is an SQL statement we've used before.

```
ALTER TABLE guitarwars
ADD COLUMN approved TINYINT
```

The MySQL data type `BOOL` is an alias for `TINYINT`, so you can use either one.

The new approved column is a `TINYINT` that uses 0 to indicate an unapproved score, or 1 to indicate an approved score. So all new scores should start out with a value of 0 to indicate that they are **initially unapproved**.

1 Use `ALTER` to add a new column to the table.



Wait a minute. I don't think you can just go adding a column to the database without changing the Add Score script—shouldn't it INSERT data into the new column?

It's true, a new column means a new value in the database, so you'll need to change the code in the Add Score script.

It's important to not lose sight of the fact that a PHP application is an integration of many different parts. It's the orchestration of several pieces and parts: a database consisting of tables and columns, PHP code, HTML code, and usually CSS code. It's not always apparent that changing one part requires changing another. Adding a new approved column in the guitarwars table for the sake of the Add Score script also requires modifying the `INSERT` query in the Add Score script.

All newly inserted high scores will now have approved set to 0.

```
INSERT INTO guitarwars
VALUES (0, NOW(), '$name', '$score', '$category')
```

<code>id</code>	<code>date</code>	<code>name</code>	<code>store</code>	<code>screenshot</code>	<code>approved</code>
30	2008-05-02 14:02:54	Ethel Heckel	500000	ethelsscore.gif	0
31	2008-05-02 20:32:54	Biff Jeck	314340	biffsscore.gif	0
32	2008-05-02 20:36:38	Pez Law	322710	pezsscore.gif	0



Sharpen your pencil

The Approve Score script is similar in structure to the Reject Score script, except that its job is to approve a score. Finish the missing code in the Approve Score script, making sure to secure the page and set the appropriate score based on score data passed through a POST variable.

```

<?php
    .....
?>
...
<?php
    require_once('appvars.php');
    require_once('connectvars.php');
    ...
    if (isset($_POST['submit'])) {
        if (.....) {
            // Connect to the database
            $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

            // Approve the score by setting the approved column in the database
            $query = "UPDATE guitarwars SET .....";
            mysqli_query($dbc, $query);
            mysqli_close($dbc);

            // Confirm success with the user
            echo .....;
        }
        else {
            echo .....;
        }
    }
    ...
    echo '<p><a href=".....">&lt;&lt; Back to admin page</a></p>';
?>
...

```

the completed approve score script

Sharpen your pencil Solution

```
<?php
    require_once('authorize.php');
?>
...
<?php
    require_once('appvars.php');
    require_once('connectvars.php');
    ...
    if (isset($_POST['submit'])) {
        if ($_POST['confirm'] == 'Yes') {
            // Connect to the database
            $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

            // Approve the score by setting the approved column in the database
            $query = "UPDATE guitarwars SET approved = 1 WHERE id = '$id'";
            mysqli_query($dbc, $query);
            mysqli_close($dbc);
        }
        // Confirm success with the user
        echo '<p>The high score of '. $score . ' for '. $name . ' was successfully approved.' . '</p>';
    } else {
        echo '<p class="error">Sorry, there was a problem approving the high score.</p>';
    }
}
...
echo '<p><a href="admin.php" >&lt;&lt; Back to admin page</a></p>?>
...

```

The Approve Score script is similar in structure to the Remove Score script, except that its job is to approve a score. Finish the missing code in the Approve Score script, making sure to secure the page and set the appropriate score based on score data passed through a POST variable.

Including the Authorize script is all that's needed to secure the Approve Score page with a user ID and password, but it must be done first to run this script since it relies on headers.

2 Create an Approve Score script handles approving a new high score (sets the approved column to 1).

Setting the approved column to 1 approves the score.

Confirm the user approved.

It's important to reveal when a score can't be approved, similar to how other Guitar Wars scripts report errors.

Provide a link back to the Admin page for easier navigation.

there are no
Dumb Questions

Q: Why isn't it necessary to pass along the screen shot filename when approving a score?

A: Because the process of approving a high score only requires enough information to look up a score row and then approve it. This means you really only need enough data to hone in on a particular row. The date, name, and score are enough to find a particular row and set its approved column to 1.

Q: It seems kind of cryptic to use 0 for the column. Are there other ways to represent this?

A: Yes. The MySQL ENUM data type, "enumerated," allows you to create a column with a limited number of possible values. So instead of adding the value 0 to the TINYINT that is intended to be 0 or 1, you could add an ENUM that can only have values of 'yes' or 'no'.

```
ALTER TABLE guitarwars
ADD COLUMN approved ENUM('yes', 'no')
```



Sharpen your pencil

The score data used to approve a score in the Approve Scores section of the Admin interface is generated by looping through "Approve" links that are generated in the Admin script. You will need to add missing code in the Admin script so that it generates these links.

```
...
// Loop through the array of score data, formatting it as HTML
echo '<table>';
echo '<tr><th>Name</th><th>Date</th><th>Score</th><th>Action</th></tr>';
while ($row = mysqli_fetch_array($data)) {
    // Display the score data
    echo '<tr class="scorerow"><td><strong>' . $row['name'] . '</strong></td>';
    echo '<td>' . $row['date'] . '</td>';
    echo '<td>' . $row['score'] . '</td>';
    echo '<td><a href="removescore.php?id=' . $row['id'] . '&date=' . $row['date'] . '&name=' . $row['name'] . '&score=' . $row['score'] . '&Screenshot=' . $row['Screenshot'] . '">Remove</a>;';
    if (.....) {
        echo .....;
        .....
    }
    echo '</td></tr>';
}
echo '</table>';
...

```

Hint: only
should have

generating approve links


Sharpen your pencil Solution

The score data used to approve a score in the Approve Score through "Approve" links that are generated in the Admin screen missing code in the Admin script so that it generates these links.

```
...
// Loop through the array of score data, formatting it as HTML
echo '<table>';
echo '<tr><th>Name</th><th>Date</th><th>Score</th><th>Action</th></tr>';
while ($row = mysqli_fetch_array($data)) {
    // Display the score data
    echo '<tr class="scorerow"><td><strong>' . $row['name'] . '</strong>';
    echo '<td>' . $row['date'] . '</td>';
    echo '<td>' . $row['score'] . '</td>';
    echo '<td><a href="removescore.php?id=' . $row['id'] . '&date='
        . $row['date'] . '&name=' . $row['name'] . '&score=' . $row['score']
        . '&screenshot=' . $row['screenshot'] . '">Remove</a>';
    if (.....$row['approved'] == '0'.....) ( ← Check
        echo ..... / <a href="approvescore.php?id=' . $row['id'] . '&date=' . $row['date']
            . '&name=' . $row['name'] . '&score=' . $row['score'] . '&screenshot='
            . $row['screenshot'] . '>Approve</a>; → unapp
    )
    echo '</td></tr>';
}
echo '</table>';
...

```

Check
unapp
the "

Generate
so that t
score, and
name are

The "Approve" link ties
the Admin page to the
Approve Score page.

Guitar Wars - High Scores Administration

Here is a list of all Guitar Wars high scores. Use this page to remove scores as needed.

Name	Date	Score	Action
Evan Hecht	2008-04-01 19:03:24	800000	Remove
Jacob Sonnenburg	2008-05-01 22:26:07	880790	Remove
T-Bone Taylor	2008-03-01 21:18:17	334090	Remove
Pete Law	2008-03-01 20:16:18	322310	Remove
Hoff Zeki	2008-03-01 20:32:54	314340	Remove
Lucky Gun	2008-03-01 21:16:06	308770	Remove
Bella Chevy	2008-03-01 20:39:07	285070	Remove
John Paul Jones	2008-03-01 19:38:22	245320	Remove
Phin Laine	2008-03-01 20:27:06	180580	Remove
Pete Jester	2008-02-01 22:57:21	173090	Remove
Nevil Johnson	2008-03-01 22:27:02	168430	Remove
Kenny Lavelle	2008-03-01 22:38:00	169330	Remove

Guitar Wars - Approve a High Score

Are you sure you want to approve the following high score?

Name: Pete Law
Date: 2008-03-01 20:16:18
Score: 322310

I'm sure!

Guitar Wars - Approve a High Score

The high score of 322310 for Pete Law was successfully approved.
on March 1, 2008 at 20:16:18

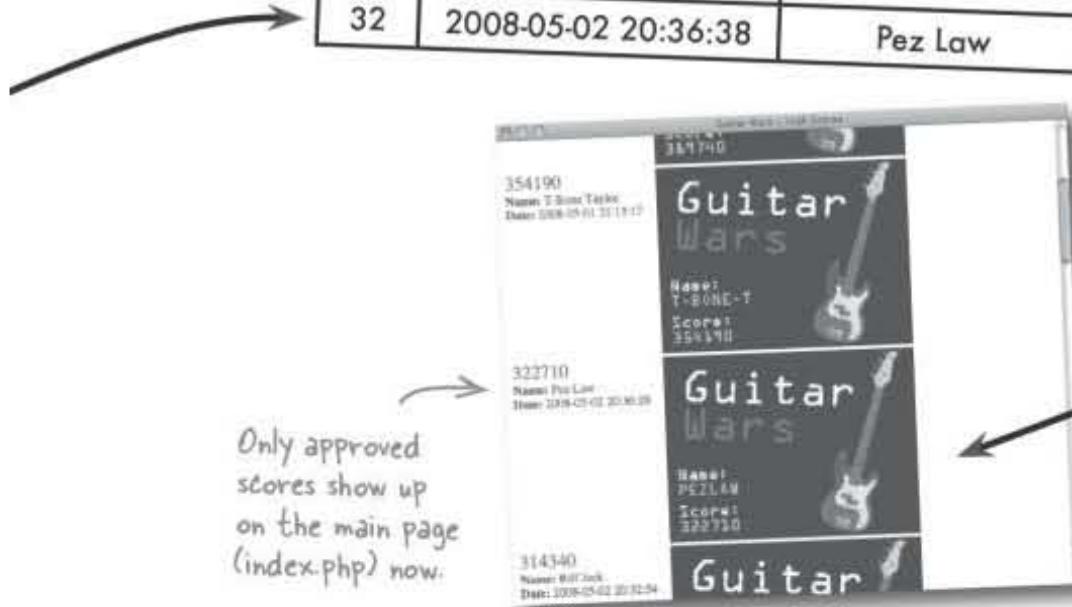
Unapproved scores aren't worthy

All the infrastructure is now in place for the moderation feature in the Guitar Wars high score application. All that's missing is the final step, which is altering the main page to only show approved scores. This involves tweaking the SQL SELECT query so that it only plucks out scores whose approved column is set to 1 (approved). This is accomplished with a WHERE statement.

```
SELECT * FROM guitarwars
WHERE approved = 1 ←
ORDER BY score DESC, date ASC
```

The addition of the WHERE statement to this query eliminates any scores that haven't been approved, which includes all new scores. This gives the moderator a chance to look them over and decide whether they should be removed or made visible to the public (approved).

id	date	name	store	score
28	2008-05-01 21:14:56	Leddy Gee	308710	leddy
29	2008-05-01 21:15:17	T-Bone Taylor	354190	tbone
30	2008-05-02 14:02:54	Ethel Heckel	500000	ethel
31	2008-05-02 20:32:54	Biff Jeck	314340	biffs
32	2008-05-02 20:36:38	Pez Law	322710	pezs



Us
sel
on
cer

If
to
the

test drive approvescore.php



Test Drive

Create the Approve script and rework the rest of the Guitar Wars application to use it.

Using a MySQL tool, issue the ALTER query to add the new approved column to the guitarwars table. Then change the INSERT query in the addscore.php script to add a 0 in the approved column for new rows of data.

Now create a new text file called approvescore.php, and enter the code for the Score script into it. Then modify the admin.php script to include an “Approve” link next to each score in the list. Finally, change the SELECT query in the admin.php script so it only shows approved scores.

Upload all of the scripts to your web server, and open the main Guitar Wars page in a browser. Take note of the scores that are visible, and then open the Admin page. Click on one of the “Approve” links and continue along to approve the score. Then go back to the main page and see if the score appears.

Guitar Wars - High Scores Administration

Below is a list of all Guitar Wars high scores. Use this page to remove scores as needed.

Name	Date	Score	Action
Ethel Heckel	2008-05-02 14:02:54	500000	Remove / Approve
Jacob Scorcherson	2008-05-01 20:36:45	389740	Remove
T-Bone Taylor	2008-05-01 21:15:17	354190	Remove
Pez Law	2008-05-02 20:36:28	322710	Remove / Approve
Bill Jeck	2008-05-02 20:32:54	314340	Remove
Leddy Gee	2008-05-01 21:14:56	308710	Remove
Belta Chevy	2008-05-01 20:36:07	282470	Remove
Jean Paul Jones	2008-05-01 20:38:23	243260	Remove
Phiz Lairston	2008-05-01 20:37:40	186580	Remove
Paco Jastorius	2008-05-01 20:37:23	127650	Remove
Nevil Johansson	2008-05-01 20:37:02	98430	Remove
Kenny Lavitz	2008-05-01 20:38:00	64930	Remove

The new “Approve” links on the Admin page provide access to the Approve Score page, where individual scores can be approved.

Guitar Wars - Approve a High Score

Are you sure you want to approve the following high score?

Name: Pez Law
Date: 2008-05-02 20:36:28
Score: 322710



Yes No

[submit](#)

[<< Back to admin page](#)

Guitar Wars - Approve a High Score

The high score of 322710 for Pez Law was successfully approved.

[<< Back to admin page](#)

A simple form requires a confirmation before actually approving the score.

Score:	Guitar Wars
389740	
354190	Name: T-Bone Taylor Date: 2008-05-01 21:15:17
322710	Name: Pez Law Date: 2008-05-02 20:36:28
314340	Name: Biff Jeck Date: 2008-05-02 20:32:54

The newly approved score now appears on the main Guitar Wars page.

Upon finalizing the approval with the confirmation dialog, a confirmation message is displayed.

ethel strikes again

The million-point hack

The moderated version of Guitar Wars represents a significant security improvement, but it's far from bulletproof. It seems our wily infiltrator has managed to find another weakness in the high score system and somehow sneak her high scores past the moderator. Ethel must be stopped, **permanently**, in order to restore trust throughout the Guitar Wars universe.

Guitar Wars - High Scores

Welcome, Guitar Warrior, do you have what it takes to crack the high score list? If so, just add your own score.

Top Score: 1000000		
1000000	Name: Ethel Heckel	Date: 2008-05-05 14:58:59
389740	Name: Jacob Scorcherson	Date: 2008-05-01 20:36:45

This is precisely the kind of high score the moderator would've stopped dead in its tracks... yet there it is!

I've gotta be honest, I'm not sure if I should be proud of this or embarrassed. I mean, I did break the system, but it was just a simple exploit. Still, I feel like I accomplished something. I wonder if the moderator will ever find out about this...

Ethel can't help but gloat over her success now that she beat the system yet again.

Everything in moderation... ?

Even though the moderator knows without a doubt that he never approved Ethel's high score submission, it nevertheless is there in plain view with the approved column set to 1. We know the Add Score script sets the approved column to 0 for new high scores because we just modified the INSERT query in that script. Something just doesn't add up!

The Guitar V
moderator c
out what ha



id	date	name	score	screenshot
21	2008-05-01 20:36:07	Belita Chevy	282470	belitasscore.gif
22	2008-05-01 20:36:45	Jacob Scorcherson	389740	jacobsscore.gif
23	2008-05-01 20:37:02	Nevil Johansson	98430	nevillsscore.gif
24	2008-05-01 20:37:23	Paco Jastorius	127650	pacosscore.gif
25	2008-05-01 20:37:40	Phiz Lairston	186580	phizsscore.gif
26	2008-05-01 20:38:00	Kenny Lavitz	64930	kennysscore.gif
27	2008-05-01 20:38:23	Jean Paul Jones	243260	jeanpaulsscore.gif
28	2008-05-01 21:14:56	Leddy Gee	308710	leddysscore.gif
29	2008-05-01 21:15:17	T-Bone Taylor	354190	tbonesscore.gif
31	2008-05-02 20:32:54	Biff Jeck	314340	biffsscore.gif
32	2008-05-02 20:36:38	Pez Law	322710	pezsscore.gif
33	2008-05-05 14:58:59	Ethel Heckel	1000000	ethelsscore2.gif



How do you think Ethel's bogus post got past the moderator?

try out ethel's hack



BRAIN BARBELL

As it turns out, Ethel's million-point hack had nothing to do with the Approve Score form. Her mischief was completely isolated to the Add Score form. Below is the exact form data that Ethel entered into the Add Score form to carry out her hack. Enter the same form data in your own form and add the score. What do you think is going on?

Don't
after

Ethel Heckel

1000000', 'ethelsscore2

Guitar Wars - Add Your High Score

Guitar Wars - Add Your High Score

Name:

Score:

Screen shot:

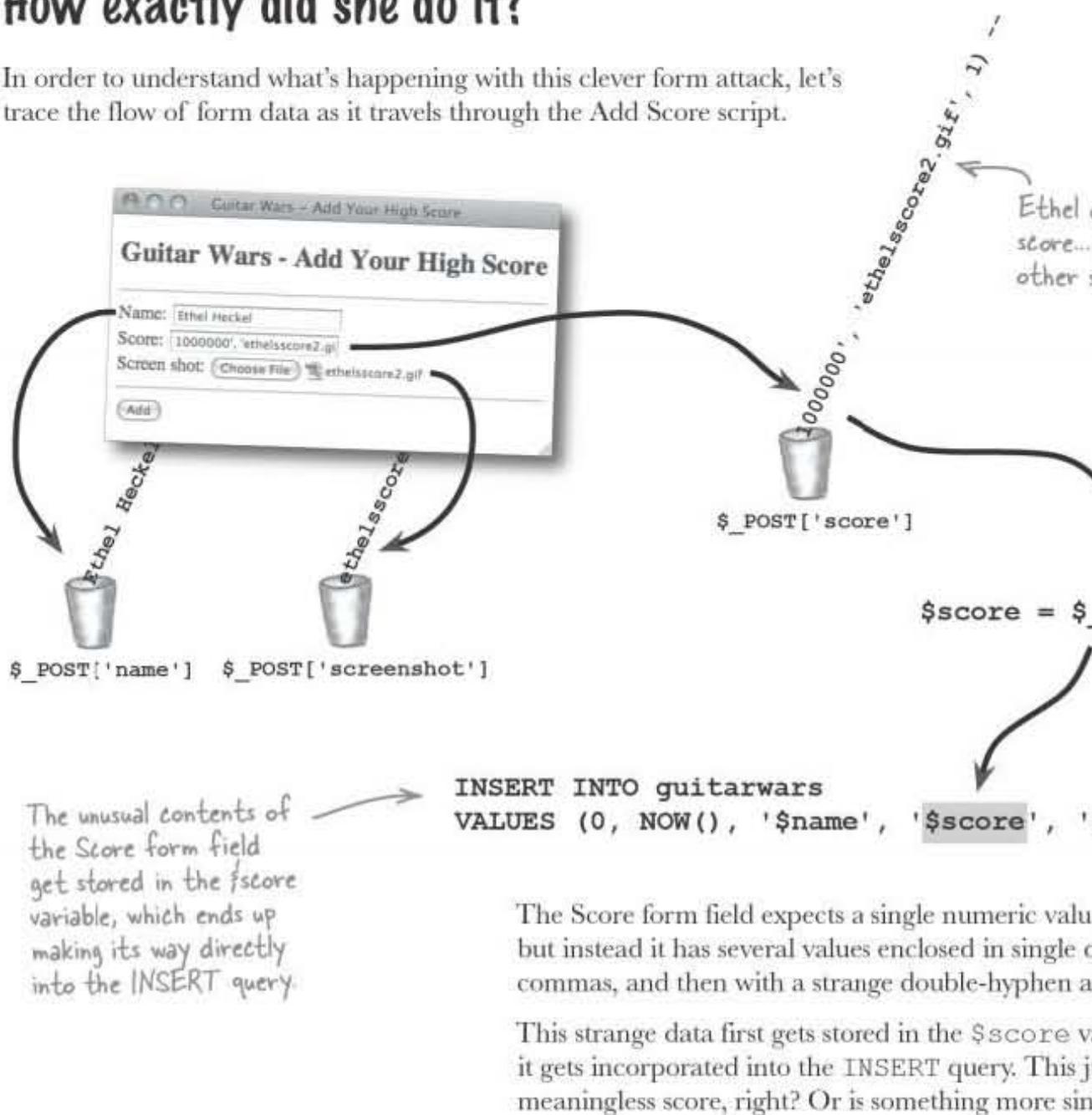
This can be any GIF or JPEG →
image file that is under 32 KB.



ethelsscore2.gif

How exactly did she do it?

In order to understand what's happening with this clever form attack, let's trace the flow of form data as it travels through the Add Score script.



Sharpen your pencil

Using the exact form data shown on the facing page, write out the full Add Score SQL query for the million-point score. Be sure to replace the variables in the query with the actual data. Annotate what you think is happening.

how sql injection works

Sharpen your pencil Solution

Using the exact form data shown on the facing page, write the full Add Score SQL query for the million-point attack. Be sure to replace the variables in the query with the data. Annotate what you think is happening.

INSERT INTO guitarwars

VALUES (0, NOW(), 'Ethel Heckel', '1000000', 'ethelsscore2.gif', 1) -- ', 'ethelsscore2.gif')

Ethel has somehow created her own version of the query that is superseding the original query.

That's a weird looking query. The screenshot filename appears twice, and I don't know what to make of that double-hyphen... does the query work?

Since is last struc to a

O O

Tricking MySQL with comments

The real culprit in Ethel's million-point attack is, strangely enough, SQL comments. A double-hyphen (--) is used in SQL to comment out the remainder of a line of SQL code. You **must follow the double-hyphen with a space** for it to work (--) , but everything after the space is ignored. Now take a look at Ethel's full query with that little nugget of wisdom.

INSERT INTO guitarwars

VALUES (0, NOW(), 'Ethel Heckel', '1000000', 'ethelsscore2.gif', 1) --

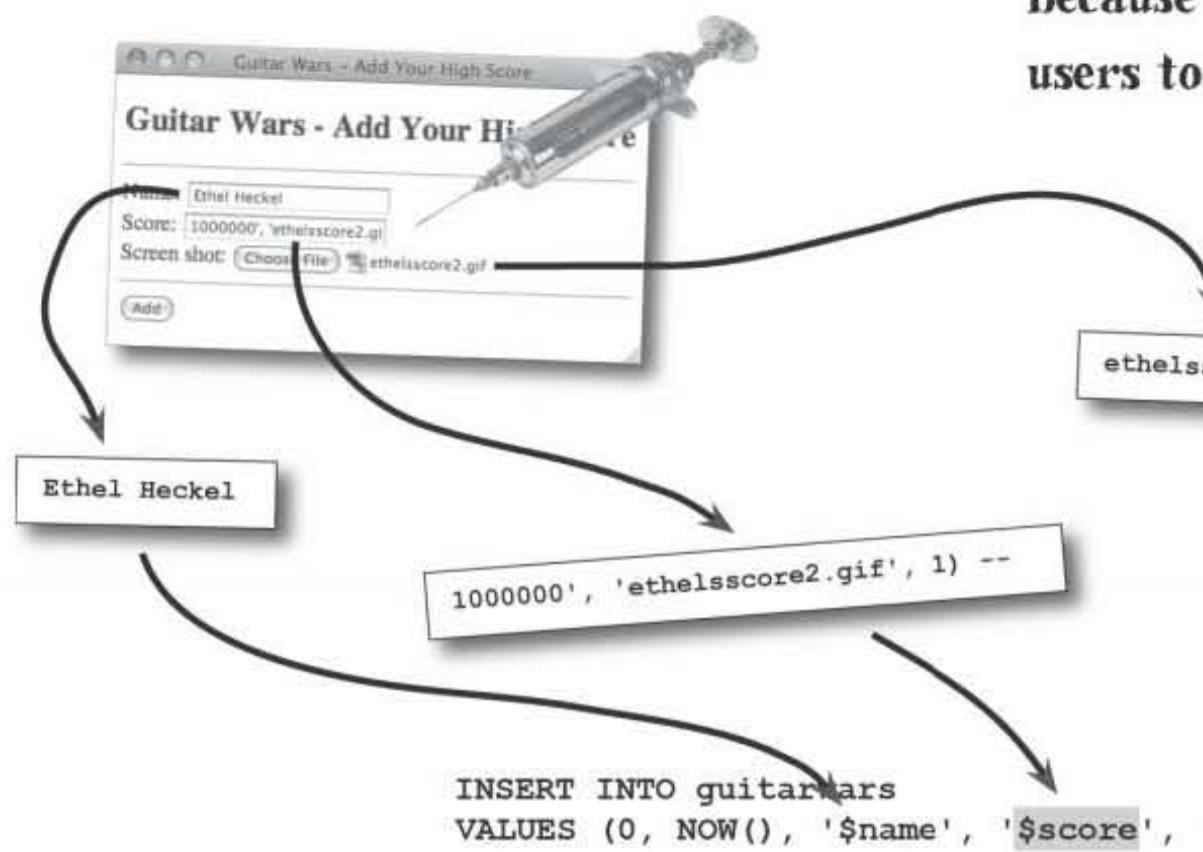
Is it making more sense? The comment effectively erased the remaining SQL code so that it wouldn't generate an error, allowing Ethel's version of the query to slip through without a snag. The end result is an instantly approved new high score that the moderator never got a chance to catch.

id	date	name	score	screenshot	approved
33	2008-05-05 14:58:59	Ethel Heckel	1000000	ethelsscore2.gif	1

The Add Score form was SQL injected

Ethel's attack is known as an **SQL injection**, and involves an extremely sneaky trick where form data is used as a means to change the fundamental operation of a query. So instead of a form field just supplying a piece of information, such as a name or score, it meddles with the underlying SQL query itself. In the case of Guitar Wars, Ethel's SQL injection used the Score field as a means of not only providing the score, but also the screen shot filename, the approval value, and a comment at the end to prevent the original SQL code from generating an error.

Form fields
a security
point for
application
because
users to



there are no
Dumb Questions

Q: Are there any other kinds of comments in SQL besides --?

A: Yes. Another variation on the single-line comment involves the use of # instead of --, but still results in commenting out any SQL code to the end of the line following the comment. SQL also supports multi-line comments that are similar to PHP's multi-line comments in that you enclose commented code between /* and */.

Q: Would Ethel's SQL injection attack have worked if the approved column wasn't at the end of the INSERT statement?

A: No, and that's a really important point. If the SQL INSERT query relies on the default order of columns, then adding a value to the approved is the last column, appearing after the screenshot column.

preventing sql injection

Protect your data from SQL injections

The real weakness that SQL injections capitalize on is form fields that aren't validated for dangerous characters. "Dangerous characters" are any characters that could potentially change the nature of an SQL query, such as commas, quotes, or -- comment characters. Even spaces at the end of a piece of data can prove harmful. Leading or trailing spaces are easy enough to eliminate with the built-in PHP function `trim()`—just run all form data through the `trim()` function before using it in an SQL query.

```
$name = trim($_POST['name']);
$score = trim($_POST['score']);
$screenshot = trim($_FILES['screenshot']['name']);
```

The `trim()` function
rid of leading and
spaces in this form

But leading and trailing spaces aren't the whole problem. You still have the commas, quotes, comment characters, and on, and on. So in addition to trimming form fields of extra spaces, we also need a way to find and render harmless other problematic characters. PHP comes to the rescue with another built-in function, `mysqli_real_escape_string()`, which escapes potentially dangerous characters so that they can't adversely affect how a query executes. These characters can still appear as data in form fields, they just won't interfere with queries.

Putting the `trim()` and `mysqli_real_escape_string()` functions together provide a solid line of defense against SQL injections.

SQL inj
be preve
properly
form dat

The `mysq
function
charact
format
affect s`

```
$name = mysqli_real_escape_string($dbc, trim($_POST['name']));
$score = mysqli_real_escape_string($dbc, trim($_POST['score']));
$screenshot = mysqli_real_escape_string($dbc, trim($_FILES['screenshot']));
```

`mysqli_real_escape_string`
considered a database
is why it requires
database connect
one used when su

Processing the three Guitar Wars form fields with the `trim()` and `mysqli_real_escape_string()` functions greatly reduces the chances of another SQL injection attack. But these two functions aren't enough—maybe there's a way to make the query itself less vulnerable...

A safer INSERT (with parameters)

Aside from exploiting weak form field protection, Ethel's SQL injection also relied on the fact that the approved column followed the screenshot column in the database structure. That's how she was able to get away with just adding 1 onto the end of `INSERT` and have it go into the approved column. The problem is that the `INSERT` query is structured in such a way that it has to insert data into all columns, which adds unnecessary risk.

Ideally, we shouldn't be setting the id and approved columns since they could just have default values.

```
INSERT INTO guitarwars
VALUES (0, NOW(), '$name', '$score', '$screenshot', 0)
```

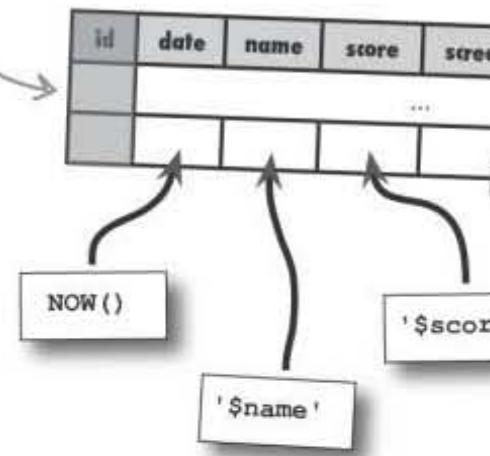
When data is inserted into a table like this, the order of the data must line up with the order of the columns in the table structure. So the fifth piece of data will go into the screenshot column because it's the fifth column in the table. But it really isn't necessary to explicitly insert the `id` or `approved` columns since `id` is auto-incremented and `approved` should always be 0. A better approach is to focus on inserting only the data explicitly required of a new high score. The `id` and `approved` columns can then be allowed to **default** to `AUTO_INCREMENT` and 0, respectively.

We need a restructured `INSERT` query that expects a list of columns prior to the list of data, with each matching one-to-one. This eliminates the risk of the `approved` column being set—it's no longer part of the query. If this kind of query looks familiar, it's because you've used it several times in other examples.

```
INSERT INTO guitarwars (date, name, score, screenshot)
VALUES (NOW(), '$name', '$score', '$screenshot')
```

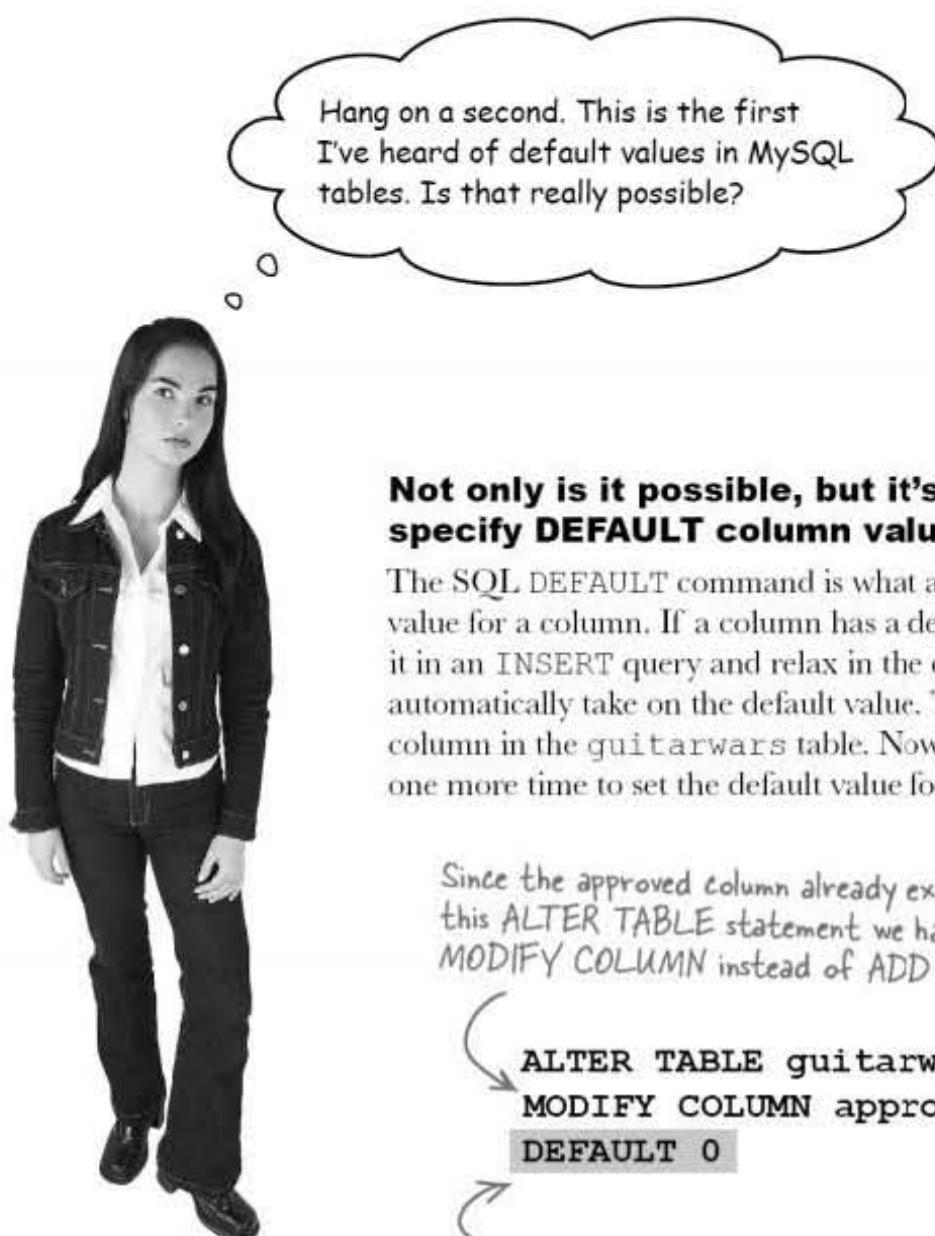
The id column can be left out since it auto-increments anyway.

This version of the `INSERT` query spells out exactly which column each piece of data is to be stored in, allowing you to insert data without having to worry about the underlying table structure. In fact, it's considered better coding style to use this kind of `INSERT` query so that data is inserted exactly where you intend it to go, as opposed to relying on the structural layout of the table.



An `INSERT` can be written in such a way that it matches exactly what you want to go in where.

the **DEFAULT** command



Not only is it possible, but it's a very good idea to specify **DEFAULT column values whenever possible.**

The SQL **DEFAULT** command is what allows you to specify a default value for a column. If a column has a default value, you can omit it in an **INSERT** query and relax in the confidence of knowing that the column will automatically take on the default value. This is perfect for the **approved** column in the **guitarwars** table. Now we just need to make one more time to set the default value for **approved** to 0.

Since the **approved** column already exists, in this **ALTER TABLE** statement we have to use **MODIFY COLUMN** instead of **ADD COLUMN**.

ALTER TABLE **guitarwars**
MODIFY COLUMN **approved** TINYINT
DEFAULT 0

DEFAULT results in the **approved** column being automatically assigned a value of 0 unless an **INSERT** query explicitly sets it otherwise.

You still type of...
sure it's first add...

With the **approved** column now altered to take on a new and improved **INSERT** query in the Add Score function, we can now add high scores without even mentioning the **approved** column. This is good design since there's no need to explicitly insert a value into the **approved** column, which could be defaulted, and it adds a small extra degree of security by preventing someone from tampering with the **approved** column to a potential attack.

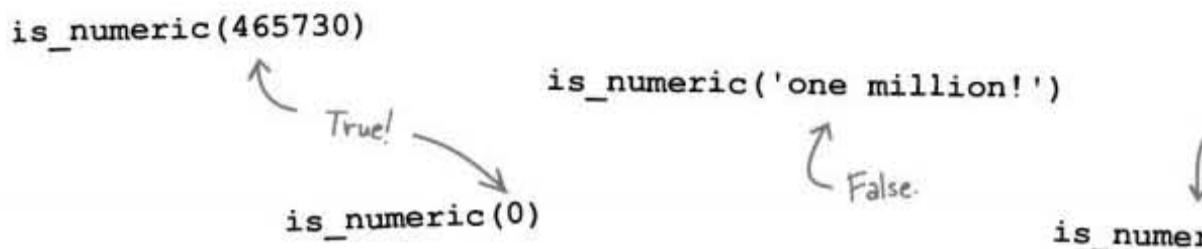
Form validation can never be too smart

One last step in minimizing the risks of SQL injection attacks involves the form validation in the Add Score script. Before checking to see if the screen shot file type or size is within the application-defined limits, the three Add Score form fields are checked to make sure they aren't empty.

```
if (!empty($name) && !empty($score) && !empty($screenshot)) {  
    ...  
}
```

This if statement makes sure all fields are not empty

There is nothing wrong with this code as-is, but securing an application is often about going above and beyond the call of duty. Since the Score field expects a number, it makes sense to not just check for a non-empty value but for a numeric value. The PHP `is_numeric()` function does just that by returning `true` if a value passed to it is a number, or `false` otherwise. It's consistently doing the little things, like checking for a number when you're expecting a number, that will ultimately make your application as secure as possible from data attacks.



Whenever possible, insist on form data being in the format you've requested.



Rewrite the Add Score form validation `if` statement to use the `isnumeric()` function only a numeric score is allowed.

.....
.....
.....

test drive the new addscore.php



Rewrite the Add Score form validation `if` statement to use the `isnumeric()` function so that only a numeric score is allowed.

```
if (!empty($name) && is_numeric($score) && !empty($screenshot)) {
    .....
}
```



— Test DRIVE —

Beef up the handling of form data in the Add Score script.

Tweak the assignment of form data to variables in the `addscore.php` script so that `trim()` and `mysqli_real_escape_string()` functions are used to clean the data. Then change the `INSERT` query so that it specifies both the column names, eliminating the need to provide values for the `id` and `approved` columns. Also add an `if` statement that validates the form fields so that it checks to make sure the score is a number.

Finally, use a MySQL tool to run the `ALTER` query that defaults the `approved` column to `0`.

Upload the new Add Score script to your web server, navigate to it in a web browser, and then try the same SQL injection attack again.

Now the Score form field will only accept numbers and nothing else.

Guitar Wars - Add Your High Score

Please enter all of the information to add your high score.

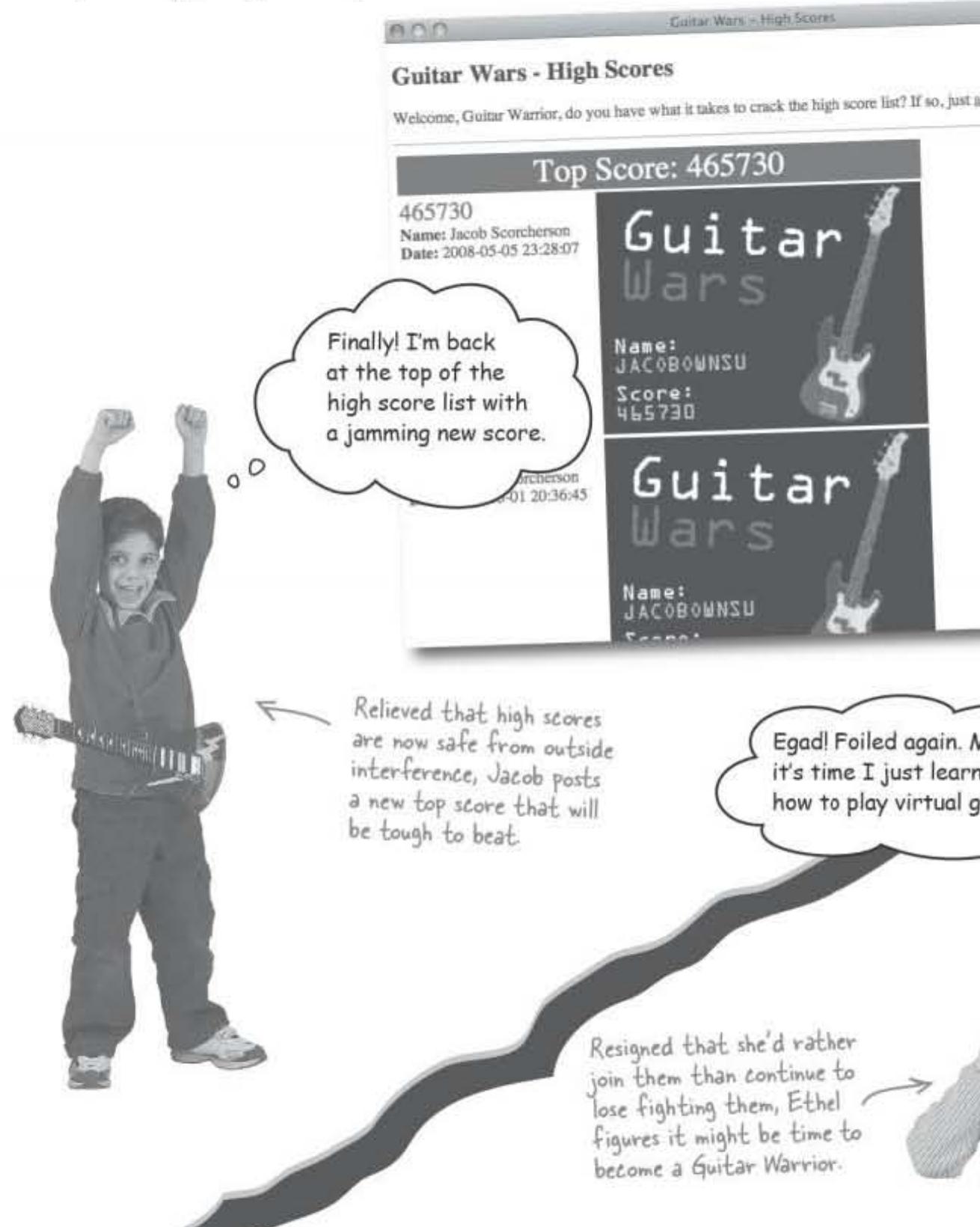
Name:	<input type="text" value="Ethel Heckel"/>
Score:	<input type="text" value="1000000' OR '1=1--"/>
Screen shot:	<input type="file"/> no file selected

Sure, this example could be a bit specific, but the job done without extra logic.

Form validation reaches security validation.

Cease fire!

It seems Ethel's will to interfere with the Guitar Wars high scores has finally been broken thanks to the improvements to the application that render it immune to SQL injections. The reigning Guitar Wars champion has responded by posting a new top score.



CHAPTER 6

php&mysql toolbox



Your PHP & MySQL Toolbox

In addition to taking the Guitar Wars high score application to a new level, you've acquired several new tools and techniques. Let's revisit the most important ones.

`header()`

This built-in PHP function is used to send a header from the server to the browser, allowing you to perform tasks such as redirecting the page, specifying a certain content type, or carrying out HTTP authentication.

`$_SERVER`

Among other things, this built-in PHP superglobal stores the user name and password entered by the user when attempting to access a page requiring HTTP authentication. You can check these against expected values to protect pages that need to be secured.

`is_numeric()`

This built-in PHP function checks to see if a value is a number. It is useful for checking to see if a numeric form field actually holds a numeric value.

`trim()`, `mysqli_real_escape_string()`

These two built-in PHP functions are handy for processing form data and preventing problematic characters from interfering with SQL queries. The first function trims leading and trailing spaces, while the latter escapes special characters.

exit
This b
a PHP
Once i
exit()
PHP
addit
to th

DEFI
This S
the d
table.
the co
on the

Human Moderation

Everything in moderation! In this case, it means that a human being is often the best line of defense in identifying and eliminating unwanted content being posted by others. Automated security techniques are still important, but it's hard to beat a living, breathing person with a brain!

HTTP Authentication

A simple web security technique that limits access to a web page or script using a user name and password. Although not intended for highly sensitive security applications, HTTP authentication can be handy for quickly adding a degree of security to a web application.

Column/Value Query

A type of INSERT query where columns and their respective values are carefully matched to each other, as opposed to relying on the order of the data matching the structural order of the columns in the table.

SQL Injection

A security breach that allows an evil-doer somehow to insert a SQL query to gain unauthorized access to a database. SQL injections involve tricking a user into passing all their data directly to a constructed query. Input validation is often used to prevent this.

Form Validation

The process of checking the data entered into a form to ensure that it matches the expected format. This is important to making forms easier to use and validating data. Form validation can help make web applications more secure by preventing users from entering invalid data.

7 building personalized web apps

Remember me?



No one likes to be forgotten, especially users of web applications. If an application has any sense of “membership,” meaning that users somehow interact with the application in a personal way, then the application needs to remember the users. You’d hate to have to reintroduce yourself to your family every time you walk through the door at home. You don’t have to because they have this wonderful thing called **memory**. But *web applications don’t remember people automatically*—it’s up to a savvy web developer to use the tools at their disposal (PHP and MySQL, maybe?) to **build personalized web apps that can actually remember users**.

a good mismatch is hard to find

They say opposites attract

It's an age-old story: boy meets girl, girl thinks boy is completely nuts, boy thinks girl has issues, but their differences become the attraction, and they end up living happily ever after. This story drives the innovative new dating site, Mis-match.net. Mismatch takes the "opposites attract" theory to heart by mismatching people based on their differences.

Problem is, Mismatch has yet to get off the ground and is in dire need of a web developer to finish building the system. That's where you come in. Millions of lonely hearts are anxiously awaiting your completion of the application... don't let them down!

Sidney loves reality TV, yoga, and sushi, and is hoping for a successful mismatch.

I can't wait to find my perfect mismatch.

Johan Nettles
Male
1981-11-03
Athens, GA

Personal web applications thrive on personal information, which requires users to be able to access an application on a personal level.



Johan loves professional wrestling, weightlifting, and Spam, and is excited about anyone who'll reply to him.

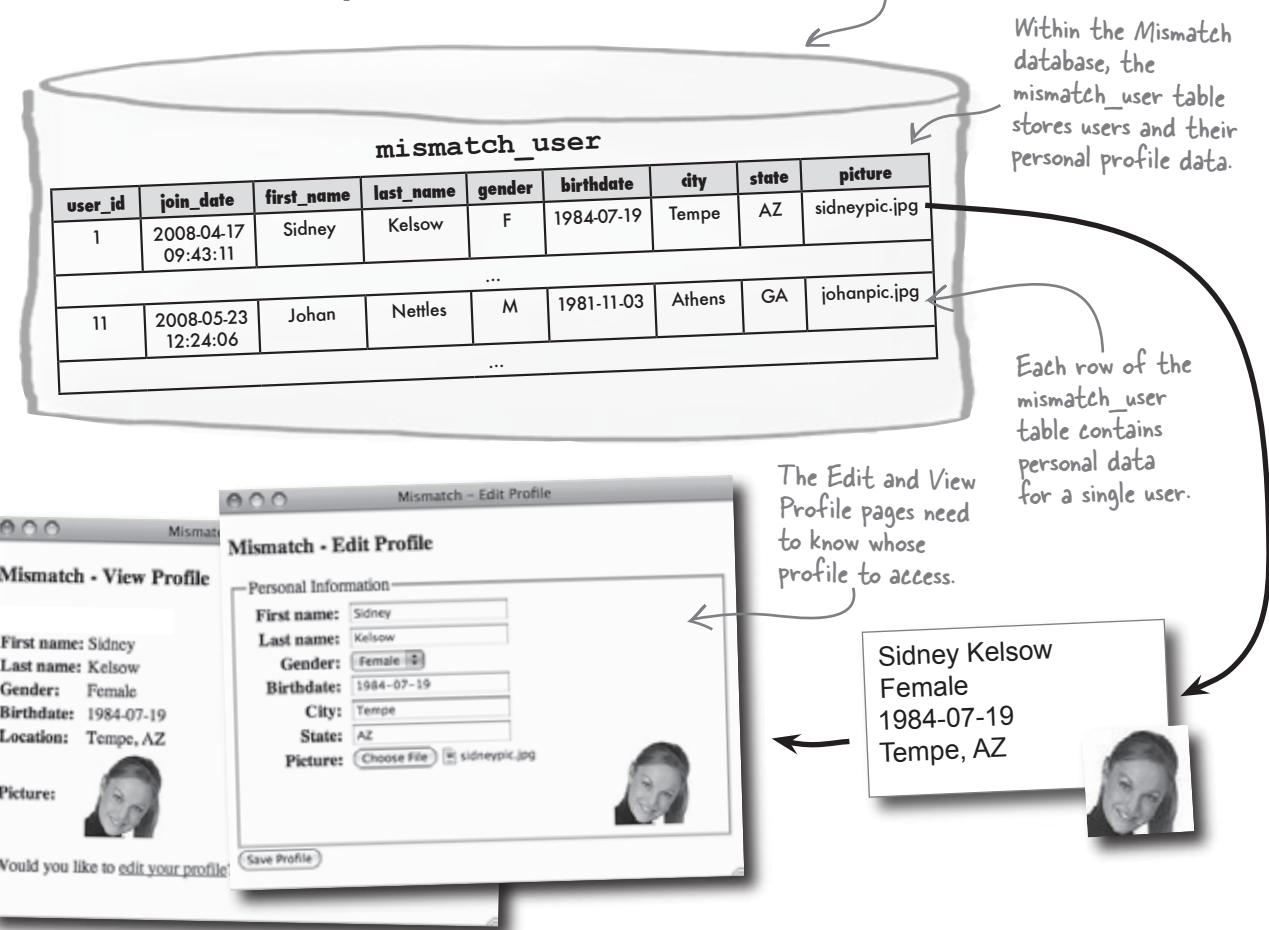


Sidney Kelsow
Female
1984-07-19
Tempe, AZ

Mismatch users need to be able to interact with the site on a personal level. For one thing, this means they need personal profiles where they enter information about themselves that they can share with other Mismatch users, such as their gender, birthdate, and location.

Mismatch is all about personal data

So Mismatch is all about establishing connections through personal data. These connections must take place within a **community of users**, each of whom is able to interact with the site and manage their own personal data. A database called `mismatch_user` is used to keep up with Mismatch users and store their personal information.



In addition to viewing a user profile, Mismatch users can edit their own personal profiles using the Edit Profile page. But there's a problem in that the application needs to know which user's profile to edit. The Edit Profile page somehow needs to keep track of the user who is accessing the page.

BRAIN POWER

How can Mismatch customize the Edit Profile page for each different user?

adding log-ins to mismatch

Mismatch needs user log-ins

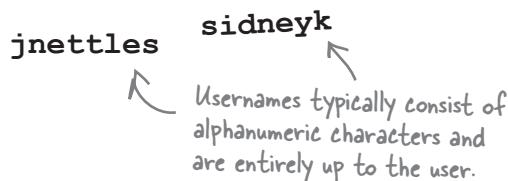
The solution to the Mismatch personal data access problem involves user log-ins, meaning that users need to be able to log into the application. This gives Mismatch the ability to provide access to information that is custom-tailored to each different user. For example, a logged-in user would only have the ability to edit their own profile data, although they might also be able to view other users' profiles. User log-ins provide the key to personalization for the Mismatch application.

A user log-in typically involves two pieces of information, a username and a password.

User log-ins
allow web
applications to
get personal
with users.

Username

The job of the username is to provide each user with a unique name that can be used to identify the user within the system. Users can potentially access and otherwise communicate with each other through their usernames.

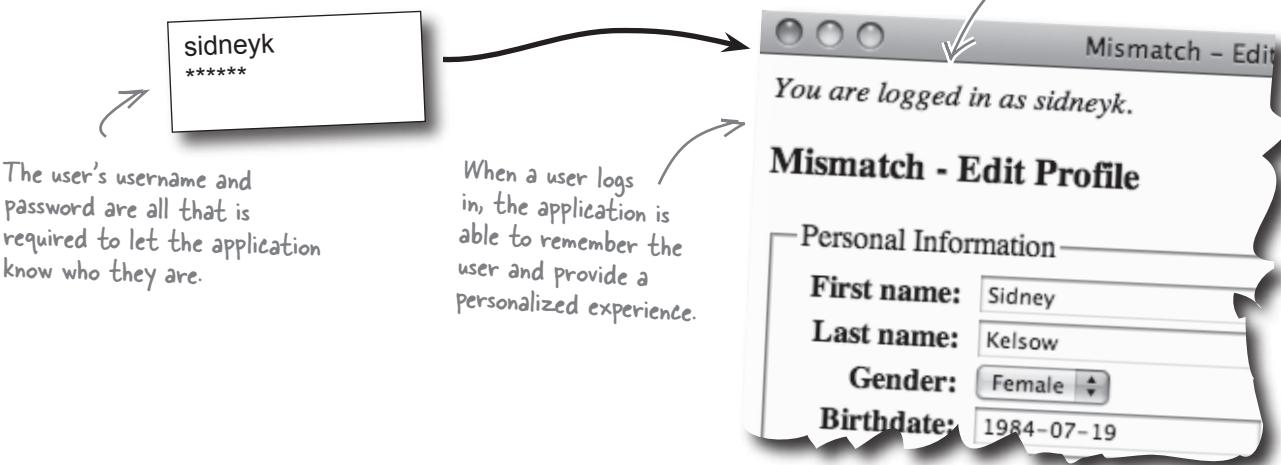


Password

The password is responsible for providing a degree of security when logging in users, which helps to safeguard their personal data. To log in, a user must enter both a username and password.

Passwords are extremely sensitive pieces of data and should never be made visible within an application, even inside the database.

A username and password allows a user to log in to the Mismatch application and access personal data, such as editing their profile.

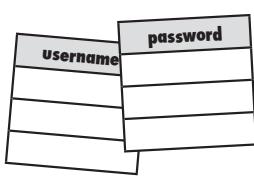


Come up with a user log-in gameplan

Adding user log-in support to Mismatch is no small feat, and it's important to work out exactly what is involved before writing code and running database queries. We know there is an existing table that stores users, so the first thing is to alter it to store log-in data. We'll also need a way for users to enter their log-in data, and this somehow needs to integrate with the rest of the Mismatch application so that pages such as the Edit Profile page are only accessible after a successful log-in. Here are the log-in development steps we've worked out so far:

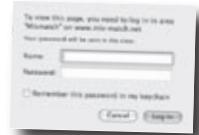
1 Use ALTER to add username and password columns to the table.

The database needs new columns for storing the log-in data for each user. This consists of a username and password.



2 Build a new Log-In script that prompts the user to enter their username and password.

The Log In form is what will ultimately protect personalized pages in that it prompts for a valid username and password. This information must be entered properly before Mismatch can display user-specific data. So the script must limit access to personalized pages so that they can't be viewed without a valid log-in.



3 Connect the Log-In script to the rest of the Mismatch application.

The Edit Profile and View Profile pages of the Mismatch application should only be accessible to logged in users. So we need to make sure users log in via the Log In script before being allowed to access these pages.

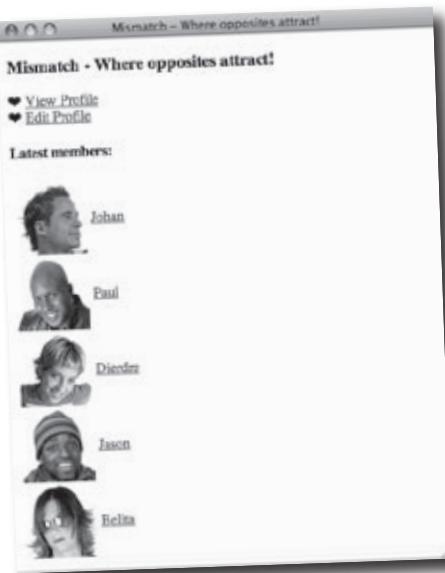


mismatch setup

Before going any further, take a moment to tinker with the Mismatch application and get a feel for how it works.

Download all of the code for the Mismatch application from the Head First Labs web site at www.headfirstlabs.com/books/hfphp. Post all of the code to your web server except for the .sql files, which contain SQL statements that build the necessary Mismatch tables. Make sure to run the statement in each of the .sql files in a MySQL tool so that you have the initial Mismatch tables to get started with.

When all that's done, navigate to the `index.php` page in your web browser, and check out the application. Keep in mind that the View Profile and Edit Profile pages are initially broken since they are entirely dependent upon user log-ins, which we're in the midst of building.



These two links lead into the personalized parts of the application.

The main Mismatch page allows you to see the name and picture of the latest users, but not much else without being logged in.

Download It!

The complete source code for the Mismatch application is available for download from the Head First Labs web site:

www.headfirstlabs.com/books/hfphp

Prepping the database for log-ins

OK, back to the construction. The `mismatch_user` table already does a good job of holding profile information for each user, but it's lacking when it comes to user log-in information. More specifically, the table is missing columns for storing a username and password for each user.

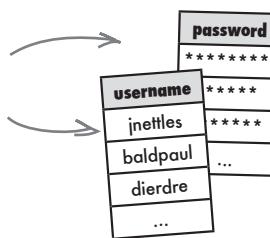
mismatch_user								
user_id	join_date	first_name	last_name	gender	birthdate	city	state	picture

The `mismatch_user` table needs columns for `username` and `password` in order to store user log-in data.

there are no
Dumb Questions

Username and password data both consist of pure text, so it's possible to use the familiar VARCHAR MySQL data type for the new `username` and `password` columns. However, unlike some other user profile data, the `username` and `password` shouldn't ever be allowed to remain empty (NULL).

The `username` and `password` columns contain simple text data but should never be allowed to go empty.



Q: Why can't you just use `user_id` instead of `username` for uniquely identifying a user?

A: You can if you want. In fact, the purpose of `user_id` is to provide an efficient means of uniquely identifying user rows. However, numeric IDs tend to be difficult to remember, and users really like being able to make up their own usernames for accessing personalized web applications. So it's more of a usability decision to allow Johan to be able to log in as "jnettles" instead of "11". No one wants to be relegated to just being a number!



Few people would want to try and remember a password longer than 16 characters!

Finish writing an SQL statement to add the `username` and `password` columns to the table positioned as shown, with `username` able to hold 32 characters, `password` able to hold 16 characters, and neither of them allowing NULL data.

mismatch_user										
user_id	username	password	join_date	first_name	last_name	gender	birthdate	city	state	picture

sharpen your pencil solution

Sharpen your pencil Solution

ALTER TABLE is used to add new columns to an existing table.

`ALTER TABLE mismatch_user ADD username VARCHAR(32) NOT NULL AFTER user_id;`

`ADD password VARCHAR(16) NOT NULL AFTER username.`

The AFTER statement controls where in the table new columns are added.

mismatch_user										
user_id	username	password	join_date	first_name	last_name	gender	birthdate	city	state	picture

The position of columns in a table doesn't necessarily matter, although it can serve an organizational purpose in terms of positioning the most important columns first.

DONE
1 Use ALTER to add username and password columns to the table.



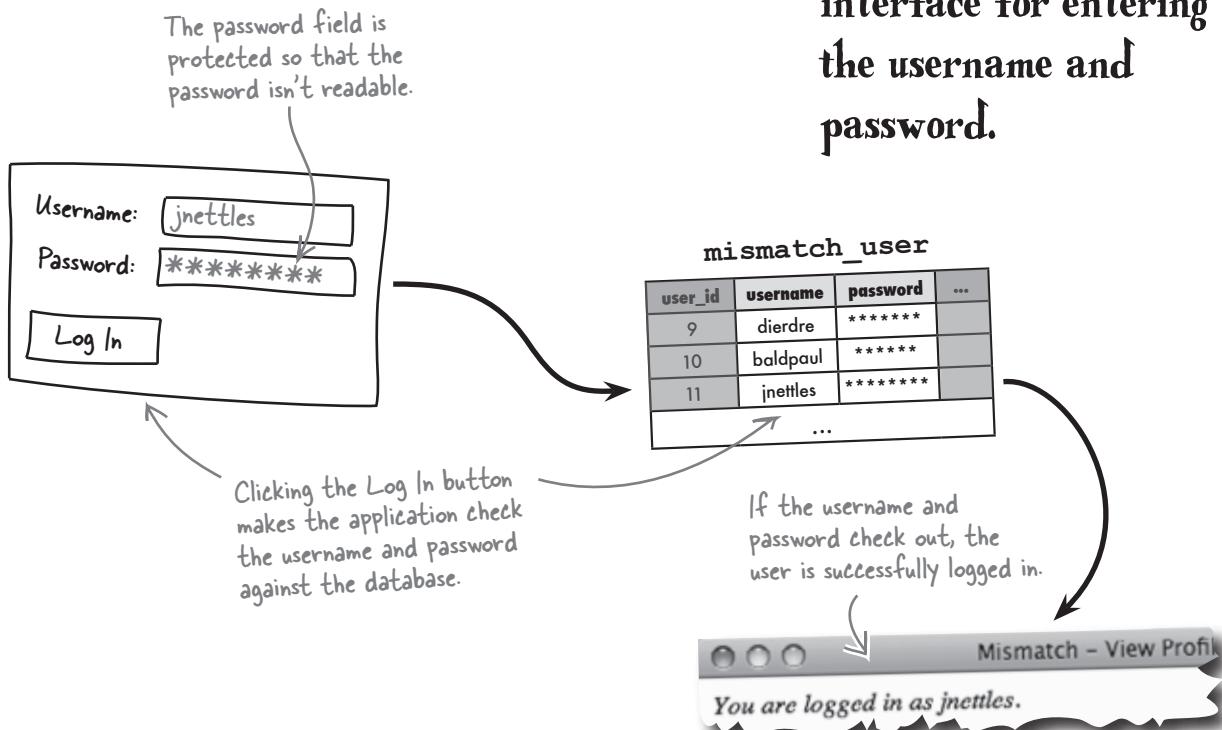
Surely you don't just store a password in the database as-is... don't you also need to encrypt a password before storing it?

Good point... passwords require encryption.

Encryption in Mismatch involves converting a password into an unrecognizable format when stored in the database. Any application with user log-in support must encrypt passwords so that users can feel confident that their passwords are safe and secure. Exposing a user's password even within the database itself is not acceptable. So we need a means of encrypting a password before inserting it into the mismatch_user table. Problem is, encryption won't help us much if we don't have a way for users to actually enter a username and password to log in...

Constructing a log-in user interface

With the database altered to hold user log-in data, we still need a way for users to enter the data and actually log in to the application. This log-in user interface needs to consist of text edit fields for the username and password, as well as a button for carrying out the log-in.



there are no
Dumb Questions

Q: So asterisks aren't actually stored in the database, right?

A: That's correct. The asterisks displayed in a password form field simply provide **visual security**, preventing someone from looking over your shoulder as you enter the password. When the form is submitted, the password itself is submitted, not the asterisks. That's why it's important for the password to be encrypted before inserting it into the database.



If you're worried about how users will be able to log in when we haven't assigned them user names and passwords yet... don't sweat it.

We'll get to creating user names and passwords for users in just a bit. For now it's important to lay the groundwork for log-ins, even if we still have more tasks ahead before it all comes together.

the sha() function

Encrypt passwords with SHA()

The log-in user interface is pretty straightforward, but we didn't address the need to encrypt the log-in password. MySQL offers a function called SHA() that applies an encryption algorithm to a string of text. The result is an encrypted string that is exactly 40 hexadecimal characters long, regardless of the original password length. So the function actually generates a 40-character code that uniquely represents the password.

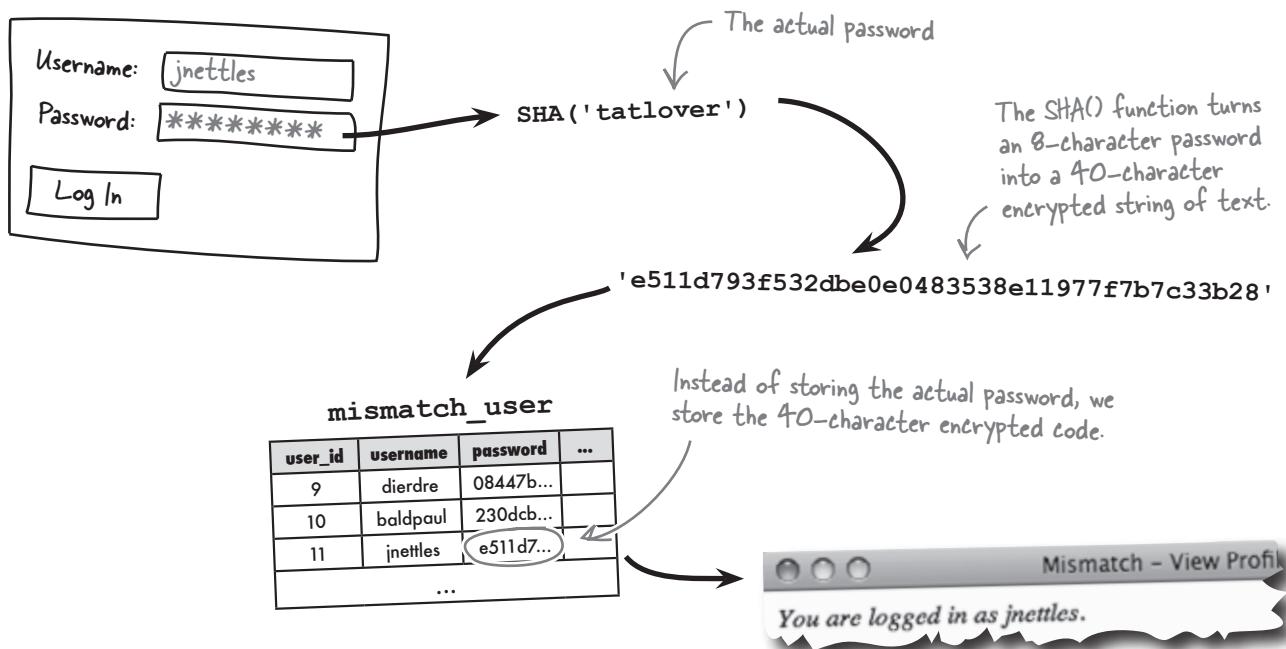
Since SHA() is a MySQL function, not a PHP function, you call it as part of the query that inserts a password into a table. For example, this code inserts a new user into the mismatch_user table, making sure to encrypt the password with SHA() along the way.

```
INSERT INTO mismatch_user
(username, password, join_date) VALUES ('jnettles', SHA('tatlover'), NOW())
```

The SHA() function encrypts the password into a 40-character hexadecimal code that gets stored in the password column of the mismatch_user table.

The MySQL SHA() function encrypts a piece of text into a unique 40-character code.

The same SHA() function works on the other end of the log-in equation by checking to see that the password entered by the user matches up with the encrypted password stored in the database.



Comparing Decrypting passwords

Once you've encrypted a piece of information, the natural instinct is to think in terms of decrypting it at some point. But the SHA() function is a one-way encryption with no way back. This is to preserve the security of the encrypted data—even if someone hacked into your database and stole all the passwords, they wouldn't be able to decrypt them. So how is it possible to log in a user if you can't decrypt their password?

You don't need to know a user's original password to know if they've entered the password correctly at log-in. This is because SHA() generates the same 40-character code as long as you provide it with the same string of text. So you can just encrypt the log-in password entered by the user and compare it to the value in the password column of the mismatch_user table. This can be accomplished with a single SQL query that attempts to select a matching user row based on a password.

```
SELECT * FROM mismatch_user
WHERE password = SHA('tatlover')
```

This is the password entered by the user in order to log in.

The SHA() function is called to encrypt the password so that it can appear in the WHERE clause.

This SELECT query selects all rows in the mismatch_user table whose password column matches the entered password, 'tatlover' in this case. Since we're comparing encrypted versions of the password, it isn't necessary to know the original password. A query to actually log in a user would use SHA(), but it would also need to SELECT on the user ID, as we see in just a moment.

Making room for the encrypted password

The SHA() function presents a problem for Mismatch since encrypted passwords end up being 40 characters long, but our newly created password column is only 16 characters long. An ALTER is in order to expand the password column for storing encrypted passwords.

```
ALTER TABLE mismatch_user
CHANGE password password VARCHAR(40) NOT NULL
```

The size of the password column is changed to 40 so that encrypted passwords will fit.

The SHA()
function provides
one-way
encryption—you
can't decrypt
data that has
been encrypted.

there are no
Dumb Questions

Q: What does SHA() stand for?

A: The SHA() function stands for Secure Hash Algorithm. A "hash" is a programming term that refers to a unique, fixed-length string that uniquely represents a string of text. In the case of SHA(), the hash is the 40-character hexadecimal encrypted string of text, which uniquely represents the original password.

Q: Are there any other ways to encrypt passwords?

A: Yes. MySQL offers another function similar to SHA() called MD5() that carries out a similar type of encryption. But the SHA() algorithm is considered a little more secure than MD5(), so it's better to use SHA() instead. PHP also offers equivalent functions (sha1() and md5()) if you need to do any encryption in PHP code, as opposed to within an SQL query.

modifying mismatch_user

Test Drive

Add the username and password columns to the mismatch_user table, and then try them out.

Using a MySQL tool, execute the ALTER statement to add the username and password columns to the mismatch_user table.

```
ALTER TABLE mismatch_user ADD username VARCHAR(32) NOT NULL AFTER user_id,
ADD password VARCHAR(16) NOT NULL AFTER username
```

But our password column actually needs to be able to hold a 40-character encrypted string, so ALTER the table once more to make room for the larger password data.

```
ALTER TABLE mismatch_user
CHANGE password password VARCHAR(40) NOT NULL
```

Now, to test out the new columns, let's do an INSERT for a new user.

```
INSERT INTO mismatch_user
(username, password, join_date) VALUES ('jimi', SHA('heyjoe'), NOW())
```

Don't forget to encrypt the password by calling the SHA() function.

To double-check that the password was indeed encrypted in the database, take a look at it by running a SELECT on the new user.

```
SELECT password FROM mismatch_user WHERE username = 'jimi'
```

And finally, you can simulate a log-in check by doing a SELECT on the username and using the SHA() function with the password in a WHERE clause.

```
SELECT username FROM mismatch_user WHERE password = SHA('heyjoe')
```

For a successful log-in, this must be the same password used when inserting the row.

username
jimi

Only one user matches the encrypted password.



Yes! HTTP authentication will certainly work as a simple user log-in system.

If you recall from the Guitar Wars high score application in the last chapter, HTTP authentication was used to restrict access to certain parts of an application by prompting the user for a username and password. That's roughly the same functionality required by Mismatch, except that now we have an entire database of possible username/password combinations, as opposed to one application-wide username and password. Mismatch users could use the same HTTP authentication window; however, they'll just be entering their own personal username and password.

A screenshot of a standard HTTP authentication window. The text inside reads:

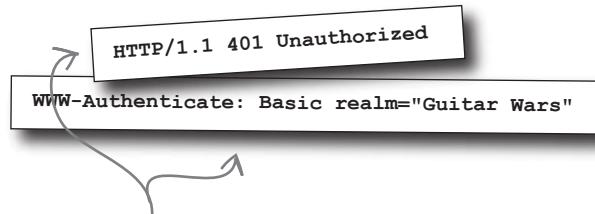
To view this page, you need to log in to area "Mismatch" on www.mis-match.net
Your password will be sent in the clear.

Below this, there are two text input fields labeled "Name:" and "Password:", each preceded by a label and a small input field. Below the password field is a checkbox labeled "Remember this password in my keychain". At the bottom are two buttons: "Cancel" and "Log In".

The standard HTTP authentication window, which is browser-specific, can serve as a simple log-in user interface.

Authorizing users with HTTP

As Guitar Wars illustrated, two headers must be sent in order to restrict access to a page via an HTTP authentication window. These headers result in the user being prompted for a username and password in order to gain access to the Admin page of Guitar Wars.



These two headers must be sent in order to restrict access to a page via HTTP authentication.

Sending the headers for HTTP authentication amounts to two lines of PHP code—a call to the header () function for each header being sent.

```
header('HTTP/1.1 401 Unauthorized');
header('WWW-Authenticate: Basic realm="Mismatch"');
```

← HTTP authentication requires us to send two headers.

This is the realm for the authentication, which applies to the entire application.

Unless a user enters the correct username and password, they cannot see or use this page.



A username and password are required in order to access restricted pages in the Guitar Wars application.

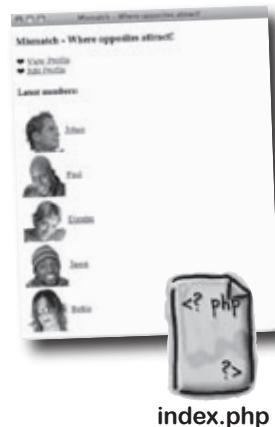


Circle the different parts of the Mismatch application that are impacted by the Log-In script (`login.php`) and its usage of HTTP authentication to control access. Then annotate how those application pieces are impacted.

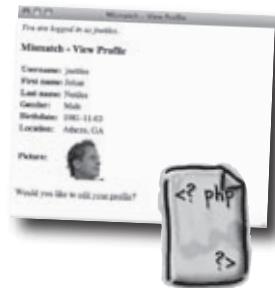
Here's the Log-In script.



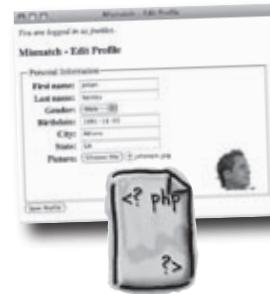
`login.php`



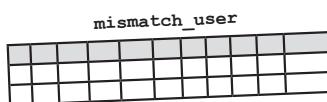
`index.php`



`viewprofile.php`



`editprofile.php`



exercise solution



Circle the different parts of the Mismatch application that are impacted by the Log-In script (`login.php`) and its usage of HTTP authentication to control access. Then annotate how those application pieces are impacted.

The home page plays no direct role in user log-ins because it needs to remain accessible by all.

When a user logs in, their username and password are checked against the database to ensure they are a registered user.

If a row isn't found that matches the username and password, the Log In script displays an error message and prevents further access.

Viewing and editing profiles is restricted, meaning that only logged in users can access these pages.

The Edit Profile page not only relies on the Log In script for restricted access, but it also needs the username in order to determine which profile to edit.

there are no Dumb Questions

Q: Why isn't it necessary to include the home page when requiring user log-ins?

A: Because the home page is the first place a user lands when visiting the site, and it's important to let visitors glimpse the site before requiring a log-in. So the home page serves as both a teaser and a starting point—a teaser for visitors and a starting point for existing users who must log in to go any deeper into the application.

Q: Can logged-in users view anyone's profile?

A: Yes. The idea is that profiles are visible to all users who log in, but remain private to guests. In other words, you have to be a member of Mismatch in order to view another user's profile.

Q: How does password encryption affect HTTP authentication?

A: There are two different issues here: transmitting a password and storing a password. The `SHA()` MySQL function focuses on securely **storing** a password in a database in an encrypted form. The database doesn't care how you transmitted the password initially, so this form of encryption has no impact on HTTP authentication. However, an argument could be made that encryption should also take place during the **transmission** of the password when the HTTP authentication window submits it to the server. This kind of encryption is outside the scope of this chapter and, ultimately, only necessary when dealing with highly sensitive data.

Logging In Users with HTTP Authentication

The Log-In script (`login.php`) is responsible for requesting a username and password from the user using HTTP authentication headers, grabbing the username and password values from the `$_SERVER` superglobal, and then checking them against the `mismatch_user` database before providing access to a restricted page.

```
<?php
require_once('connectvars.php');

if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW'])) {
    // The username/password weren't entered so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Mismatch"');
    exit('<h3>Mismatch</h3>Sorry, you must enter your username and password to log in and access ' .
        'this page.');
}

// Connect to the database
$dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

// Grab the user-entered log-in data
$user_username = mysqli_real_escape_string($dbc, trim($_SERVER['PHP_AUTH_USER']));
$user_password = mysqli_real_escape_string($dbc, trim($_SERVER['PHP_AUTH_PW']));

// Look up the username and password in the database
$query = "SELECT user_id, username FROM mismatch_user WHERE username = '$user_username' AND " .
    "password = SHA('$user_password')";
$data = mysqli_query($dbc, $query);

if (mysqli_num_rows($data) == 1) {
    // The log-in is OK so set the user ID and username variables
    $row = mysqli_fetch_array($data);
    $user_id = $row['user_id'];
    $username = $row['username'];
} else {
    // The username/password are incorrect so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Mismatch"');
    exit('<h2>Mismatch</h2>Sorry, you must enter a valid username and password to log in and ' .
        'access this page.');
}

// Confirm the successful log-in
echo('<p class="login">You are logged in as ' . $username . '</p>');
?>
```

If the username and password haven't been entered, send the authentication headers to prompt the user.

Grab the username and password entered by the user.

Perform a query to see if any rows match the username and encrypted password.

If a row matches, it means the log-in is OK, and we can set the `$user_id` and `$username` variables.

If no database row matches the username and password, send the authentication headers again to re-prompt the user.

All is well at this point, so confirm the successful log-in.

DONE

② Build a new Log-In script that prompts the user to enter their username and password.



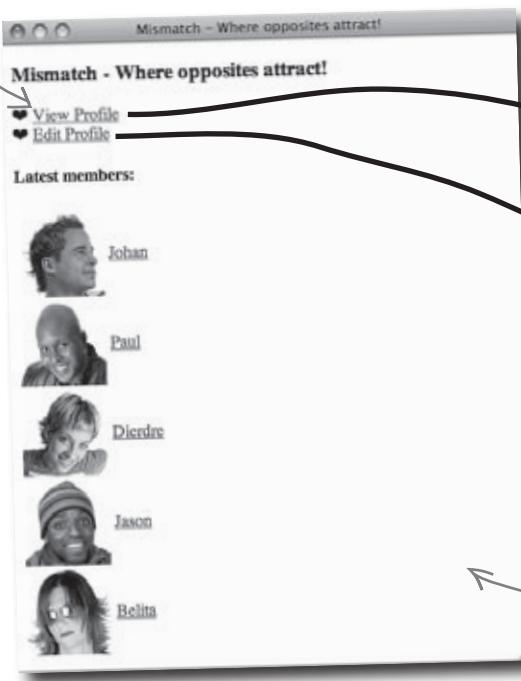
Test Drive

Create the new Log-In script, and include it in the View Profile and Edit Profile scripts.

Create a new text file named `login.php`, and enter the code for the Log-In script in it (or download the script from the Head First Labs site at www.headfirstlabs.com/books/hfphp). Then add PHP code to the top of the `viewprofile.php` and `editprofile.php` scripts to include the new Log-In script.

Upload all of the scripts to your web server, and then open the main Mismatch page in a web browser. Click the View Profile or Edit Profile link to log in and access the personalized pages. Of course, this will only work if you've already added a user with a username and password to the database.

These two links lead to the protected pages, which invoke the Log-In script if a user isn't logged in.

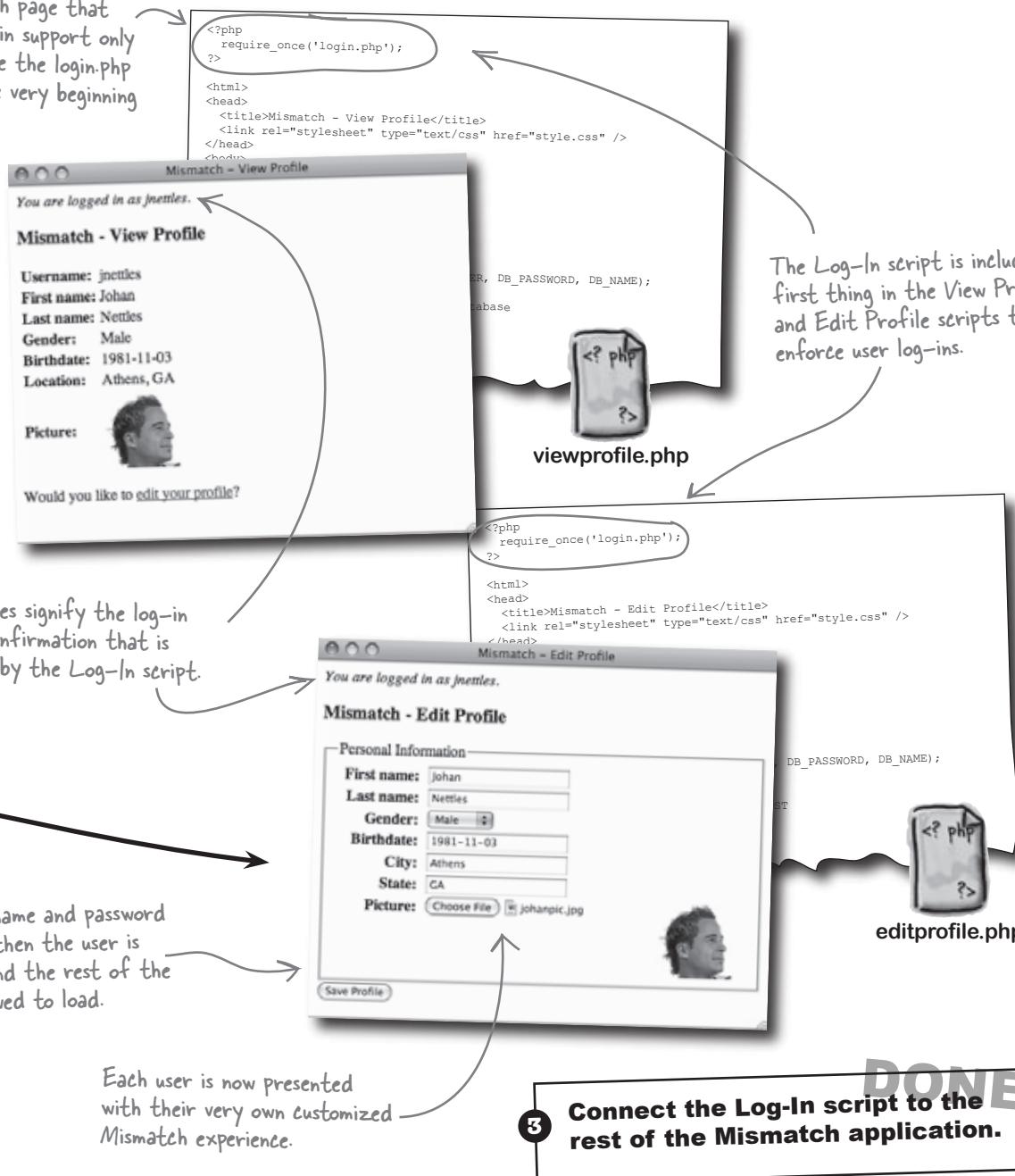


This password is SHA() encrypted and compared with the password in the database to determine if the log-in is allowed.

The Log-In script uses HTTP authentication to prevent unauthorized access to the View Profile and Edit Profile pages.

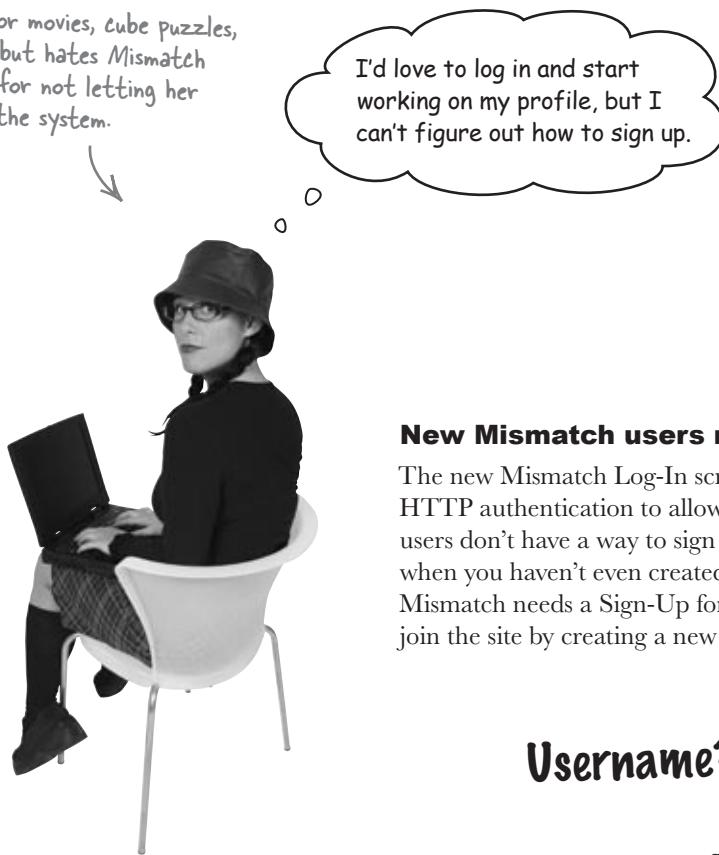
The home page is not protected by the Log-In script, but it does serve as the starting point for navigating deeper into the application.

Any Mismatch page that requires log-in support only has to include the login.php script at the very beginning of its code.



mismatch needs a sign-up form

Ruby loves horror movies, cube puzzles, and spicy food, but hates Mismatch at the moment for not letting her sign up and use the system.



New Mismatch users need a way to sign up.

The new Mismatch Log-In script does a good job of using HTTP authentication to allow users to log in. Problem is, users don't have a way to sign up—logging in is a problem when you haven't even created a username or password yet. Mismatch needs a Sign-Up form that allows new users to join the site by creating a new username and password.

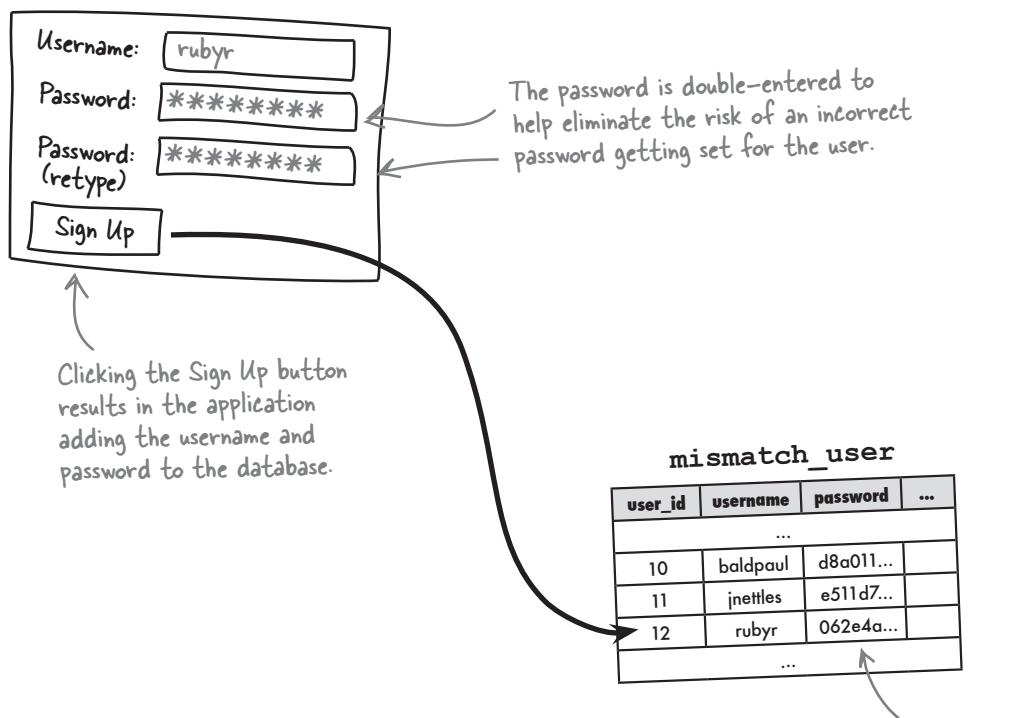
Username?

Password?

A form for signing up new users

What does this new Sign-Up form look like? We know it needs to allow the user to enter their desired username and password... anything else? Since the user is establishing their password with the new Sign-Up form, and passwords in web forms are typically masked with asterisks for security purposes, it's a good idea to have two password form fields. So the user enters the password twice, just to make sure there wasn't a typo.

So the job of the Sign-Up page is to retrieve the username and password from the user, make sure the username isn't already used by someone else, and then add the new user to the `mismatch_user` database.



One potential problem with the Sign-Up script involves the user attempting to sign up for a username that already exists. The script needs to be smart enough to catch this problem and force the user to try a different username. So the job of the Sign-Up page is to retrieve the username and password from the user, make sure the username isn't already used by someone else, and then add the new user to the `mismatch_user` database.

finish signup.php



PHP & MySQL Magnets

The Mismatch Sign-Up script uses a custom form to prompt the user for their desired username and password. Problem is, the script code is incomplete. Use the magnets below to finish up the script so new users can sign up and join the Mismatch community.

Here's the Sign-Up form.

```

<?php
require_once('appvars.php');
require_once('connectvars.php');

// Connect to the database
$dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

if (isset($_POST['submit'])) {
    // Grab the profile data from the POST
    ..... = mysqli_real_escape_string($dbc, trim($_POST['.....']));
    ..... = mysqli_real_escape_string($dbc, trim($_POST['.....']));
    ..... = mysqli_real_escape_string($dbc, trim($_POST['.....']));
    ..... = mysqli_real_escape_string($dbc, trim($_POST['.....']));

    if (!empty($username) && !empty($password1) && !empty($password2) &&
        ..... == .....)) {
        // Make sure someone isn't already registered using this username
        $query = "SELECT * FROM mismatch_user WHERE username = '.....'";
        $data = mysqli_query($dbc, $query);
        if (mysqli_num_rows($data) == 0) {
            // The username is unique, so insert the data into the database
            $query = "INSERT INTO mismatch_user (username, password, join_date) VALUES " .
                " ('.....', SHA('.....'), NOW())";
            mysqli_query($dbc, $query);
            // Confirm success with the user
            echo '<p>Your new account has been successfully created. You\'re now ready to log in and ' .
                '<a href="editprofile.php">edit your profile</a>.</p>';
        }
        mysqli_close($dbc);
        exit();
    }
}

```

Don't forget, you have to escape an apostrophe if it appears inside of single quotes.

```

else {
    // An account already exists for this username, so display an error message
    echo '<p class="error">An account already exists for this username. Please use a different ' .
        'address.</p>';
    .....
    = "";
}

else {
    echo '<p class="error">You must enter all of the sign-up data, including the desired password ' .
        'twice.</p>';
}
}

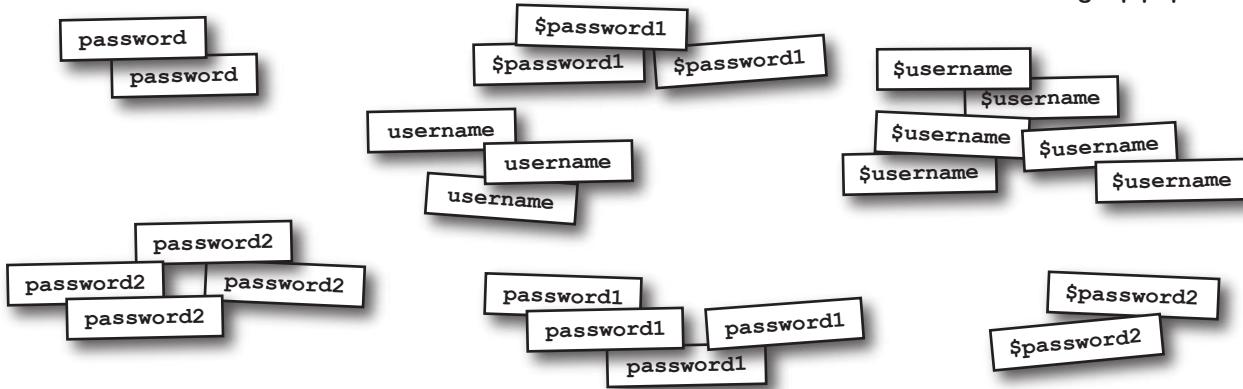
mysqli_close($dbc);
?>

<p>Please enter your username and desired password to sign up to Mismatch.</p>
<form method="post" action=<?php echo $_SERVER['PHP_SELF']; ?>>>
<fieldset>
<legend>Registration Info</legend>
<label for="username">Username:</label>
<input type="text" id="....." name="....." value=<?php if (!empty(.....)) echo .....; ?>><br />
<label for=".....">Password:</label>
<input type="....." id="....." name="....." /><br />
<label for=".....">Password (retype):</label>
<input type="....." id="....." name="....." /><br />
</fieldset>
<input type="submit" value="Sign Up" name="submit" />
</form>

```



signup.php



the completed `signup.php`

PHP & MySQL Magnets Solution

The Mismatch Sign-Up script uses a custom form to prompt the user for their desired username and password. Problem is, the script code is incomplete. Use the magnets below to finish up the script so new users can sign up and join the Mismatch community.

Here's the Sign-Up form.

```

<?php
require_once('appvars.php');
require_once('connectvars.php');

// Connect to the database
$dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

if (isset($_POST['submit'])) {
    // Grab the profile data from the POST
    $username = mysqli_real_escape_string($dbc, trim($_POST['username']));
    $password1 = mysqli_real_escape_string($dbc, trim($_POST['password1']));
    $password2 = mysqli_real_escape_string($dbc, trim($_POST['password2']));
}

if (!empty($username) && !empty($password1) && !empty($password2) &&
    ($password1 == $password2)) {
    // Make sure someone isn't already registered using this username
    $query = "SELECT * FROM mismatch_user WHERE username = '$username'";
    $data = mysqli_query($dbc, $query);
    if (mysqli_num_rows($data) == 0) {
        // The username is unique, so insert the data into the database
        $query = "INSERT INTO mismatch_user (username, password, join_date) VALUES ";
        $query .= "('$username', SHA('$password1'), NOW())";
        mysqli_query($dbc, $query);
    }
}

// Confirm success with the user
echo '<p>Your new account has been successfully created. You\'re now ready to log in and ' .
    '<a href="editprofile.php">edit your profile</a>.</p>';

mysqli_close($dbc);
exit();
}

```

Grab all of the user-entered data, making sure to clean it up first.

Check to make sure that none of the form fields are empty and that both passwords match.

Perform a query to see if any existing rows match the username entered.

If no match is found, the username is unique, so we can carry out the INSERT.

Confirm the successful sign-up with the user, and exit the script.

```

else {
    // An account already exists for this username, so display an error message
    echo '<p class="error">An account already exists for this username. Please use a different ' .
        ' address.</p>';
}

}
else {
    echo '<p class="error">You must enter all of the sign-up data, including the desired password ' .
        ' twice.</p>';
}

}

mysqli_close($dbc);
?>

<p>Please enter your username and desired password to sign up to Mismatch.</p>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
<fieldset>
    <legend>Registration Info</legend>
    <label for="username">Username:</label>

    <input type="text" id="...[username]" name="...[username]" ...
        value="<?php if (!empty(...[username])) echo ...[username]; ?>" /><br />
    <label for="...[password1]">Password:</label>
    <input type="...[password]" id="...[password1]" name="...[password1]" /><br />
    <label for="...[password2]">Password (retype):</label>
    <input type="...[password]" id="...[password2]" name="...[password2]" /><br />
</fieldset>
<input type="submit" value="Sign Up" name="submit" />
</form>

```

The username is not unique, so display an error message.

One or more of the form fields are empty, so display an error message.



signup.php

there are no
Dumb Questions

Q: Why couldn't you just use HTTP authentication for signing up new users?

A: Because the purpose of the Sign-Up script isn't to restrict access to pages. The Sign-Up script's job is to allow the user to enter a unique username and password, and then add them to the user database. Sure, it's possible to use the HTTP authentication window as an input form for the username and password, but the authentication functionality is overkill for just signing up a new user. It's better to create a custom form for sign-ups—then you get the benefit of double-checking the password for data entry errors.

Q: So does the Sign-Up script log in users after they sign up?

A: No. And the reason primarily has to do with the fact that the Log-In script already handles the task of logging in a user, and there's no need to duplicate the code in the Sign-Up script. The Sign-Up script instead presents a link to the Edit Profile page, which is presumably where the user would want to go after signing in. And since they aren't logged in yet, they are presented with the Log-In window as part of attempting to access the Edit Profile page. So the Sign-Up script leads the user to the Log-In window via the Edit Profile page, as opposed to logging them in automatically.

adding a sign-up link

Give users a chance to sign up

We have a Sign-Up script, but how do users get to it? We need to let users know how to sign up. One option is to put a “Sign Up” link on the main Mismatch page. That’s not a bad idea, but we would ideally need to be able to turn it on and off based on whether a user is logged in. Another possibility is to just show a “Sign Up” link as part of the Log-In script.

When a new user clicks the “View Profile” or “Edit Profile” links on the main page, for example, they’ll be prompted for a username and password by the Log-In script. Since they don’t yet have a username or password, they will likely click Cancel to bail out of the log-in. That’s our chance to display a link to the Sign-Up script by tweaking the log-in failure message displayed by the Log-In script so that it provides a link to `signup.php`.

Here’s the original log-in failure code:

```
exit('<h3>Mismatch</h3>Sorry, you must enter your username and password to log in and access ' .
'this page.'');
```

This code just shows a log-in error message with no mention of how to sign up for Mismatch.

This code actually appears in two different places in the Log-In script: when no username or password are entered and when they are entered incorrectly. It’s probably a good idea to go ahead and provide a “Sign Up” link in both places. Here’s what the new code might look like:

```
exit('<h2>Mismatch</h2>Sorry, you must enter a valid username and password to log in and ' .
'access this page. If you aren\'t a registered member, please <a href="signup.php">sign up</a>.'');
```

This code is much more helpful since it generates a link to the Sign-Up script so that the user can sign up.

Nothing fancy here, just a normal HTML link to the `signup.php` script.



Test Drive

Add Sign-Up functionality to Mismatch.

Create a new text file named `signup.php`, and enter the code for the Sign-Up script in it (or download the script from the Head First Labs site at www.headfirstlabs.com/books/hfphp). Then modify the `login.php` script to add links to the Sign-Up script for users who can't log in.

Upload the scripts to your web server, and then open the Sign-Up page in a web browser. Sign up as a new user and then log in. Then edit your profile and view your profile to confirm that the sign-up and log-in worked correctly. The application now has that personalized touch that's been missing.

HTTP authentication is used to log in Ruby based on her sign-up information.

Cool! I can log in to Mismatch and then edit and view my personal profile.

Sign-ups and log-ins turn an impersonal application into a community of interested users.

Ruby's profile is only accessible after logging in.

mismatch also needs to let users log out



Community web sites must allow users to log out so that others can't access their personal data from a shared computer.

Allowing users to log out might sound simple enough, but it presents a pretty big problem with HTTP authentication. The problem is that HTTP authentication is intended to be carried out once for a given page or collection of pages—it's only reset when the browser is shut down. In other words, a user is never “logged out” of an HTTP authenticated web page until the browser is shut down or the user manually clears the HTTP authenticated session. The latter option is easier to carry out in some browsers (Firefox, for example) than others (Safari).

The screenshot shows a web application window titled "Mismatch - Edit Profile". A message at the top says "You are logged in as sidneyk.". Below it, a "Mismatch - Edit Profile" section displays personal information fields: First name: Sidney, Last name: Kelso, Gender: Female, Birthdate: 1984-07-19, City: Tempe, State: AZ. There is a "Picture:" field with a thumbnail image of a woman and a "Choose File" button. To the right, a login dialog box is overlaid with the following text:
To view this page, you need to log in to area "Mismatch" on www.mis-match.net
Your password will be sent in the clear.
Name: sidneyk
Password: *****
 Remember this password in my keychain
Cancel Log In

Once you log in, you stay in until you close the browser.

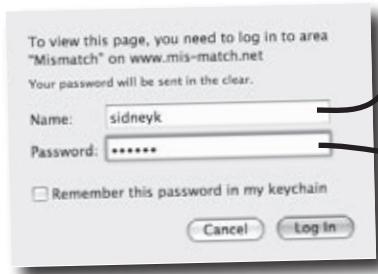
A log-out feature would allow Sidney to carefully control access to her personal profile.

Even though HTTP authentication presents a handy and simple way to support user log-ins in the Mismatch application, it doesn't provide any control over logging a user out. We need to be able to both remember users and also allow them to log out whenever they want.



Sometimes you just need a cookie

The problem originally solved by HTTP authentication is twofold: there is the issue of limiting access to certain pages, and there is the issue of remembering that the user entered information about themselves. The second problem is the tricky one because it involves an application remembering who the user is across multiple pages (scripts). Mismatch accomplishes this feat by checking the username and password stored in the `$_SERVER` superglobal. So we took advantage of the fact that PHP stores away the HTTP authentication username and password in a superglobal that persists across multiple pages.



`$_SERVER['PHP_AUTH_USER']`

`$_SERVER['PHP_AUTH_PW']`

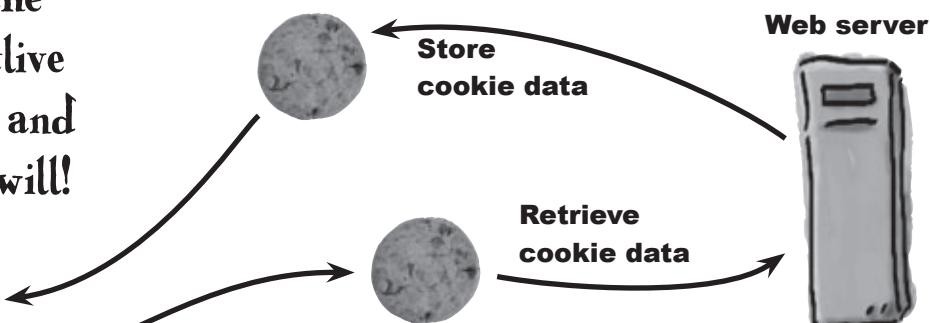
The `$_SERVER` superglobal stores the username and password persistently.

Cookies allow you to persistently store small pieces of data on the client that can outlive any single script... and can be deleted at will!



Client web browser

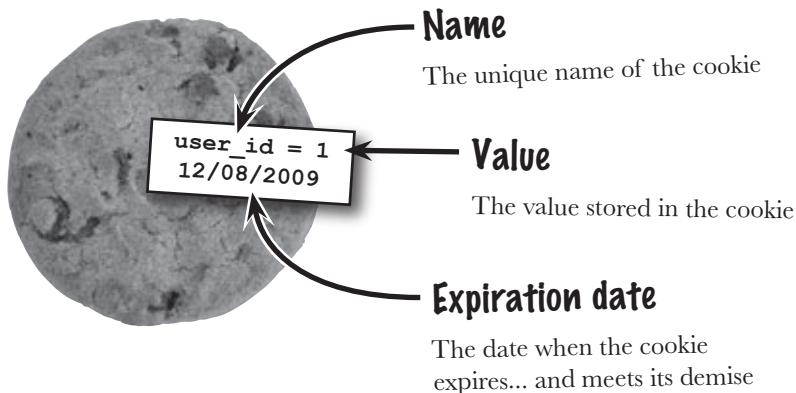
But we don't have the luxury of HTTP authentication anymore because it can't support log-outs. So we need to look elsewhere for user persistence across multiple pages. A possible solution lies in **cookies**, which are pieces of data stored by the browser on the user's computer. Cookies are a lot like PHP variables except that cookies hang around after you close the browser, turn off your computer, etc. More importantly, cookies can be deleted, meaning that you can eliminate them when you're finished storing data, such as when a user indicates they want to log out.



Cookie data is stored on the user's computer by their web browser. You have access to the cookie data from PHP code, and the cookie is capable of persisting across not only multiple pages (scripts), but even multiple browser sessions. So a user closing their browser won't automatically log them out of Mismatch. This isn't a problem for us because we can delete a cookie at any time from script code, making it possible to offer a log-out feature. We can give users total control over when they log out.

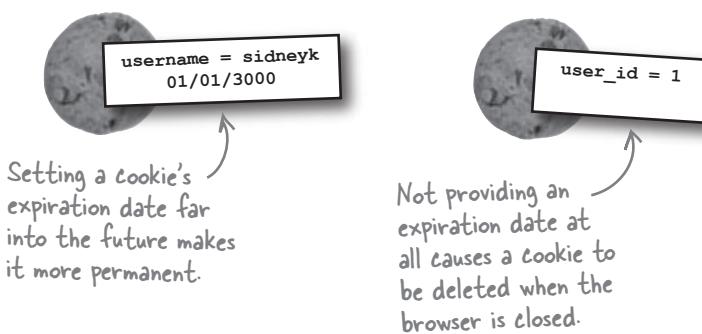
What's in a cookie?

A cookie stores **a single piece of data** under a unique name, much like a variable in PHP. Unlike a variable, a cookie can have an expiration date. When this expiration date arrives, the cookie is destroyed. So cookies aren't exactly immortal—they just live longer than PHP variables. You can create a cookie without an expiration date, in which case it acts just like a PHP variable—it gets destroyed when the browser closes.



Cookies allow you to store a string of text under a certain name, kind of like a PHP text variable. It's the fact that cookies outlive normal script data that makes them so powerful, especially in situations where an application consists of multiple pages that need to remember a few pieces of data, such as log-in information.

*there are no
Dumb Questions*



So Mismatch can mimic the persistence provided by the `$_SERVER` superglobal by setting two cookies—one for the username and one for the password. Although we really don't need to keep the password around, it might be more helpful to store away the user ID instead.

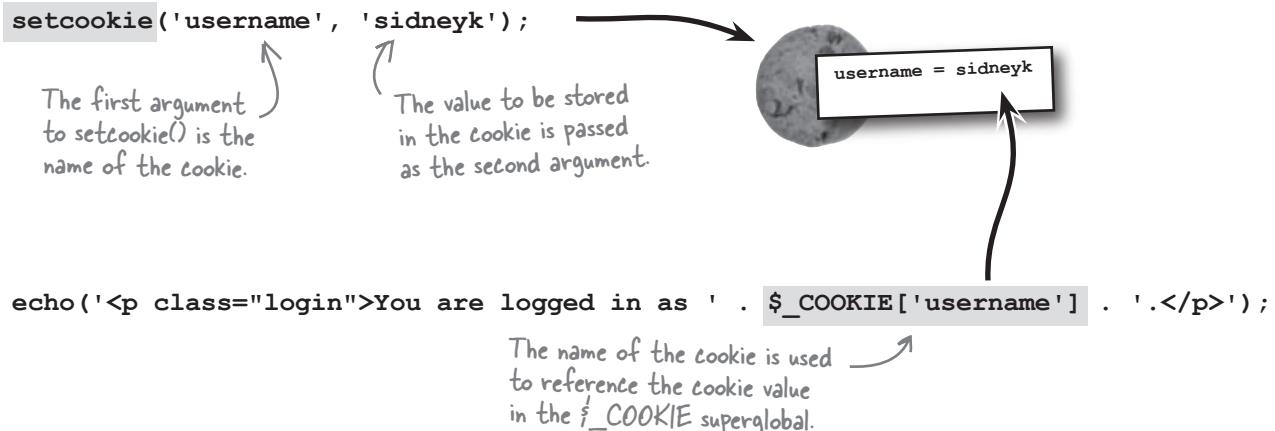
Q: What's the big deal about cookies being persistent? Isn't data stored in a MySQL database persistent too?

A: Yes, database data is most certainly persistent. In fact, it's technically much more persistent than a cookie because there is no expiration date involved—if you stick data in a database, it stays there until you explicitly remove it. The real issue in regard to cookies and persistence is convenience. We don't need to store the current user's ID or username for all eternity just to allow them to access their profile; we just need a quick way to know who they are. What we really need is **temporary persistence**, which might seem like an oxymoron until you consider the fact that we need data to hang around longer than a page (persistent), but not forever.

the setcookie() function

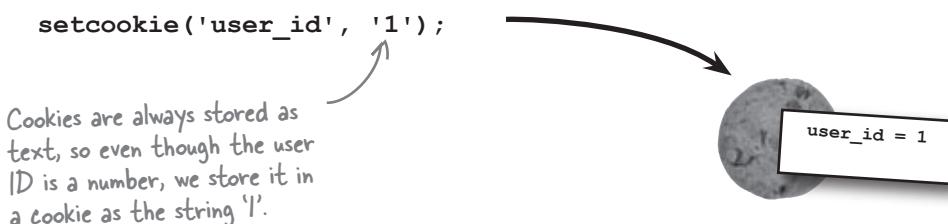
Use Bake cookies with PHP

PHP provides access to cookies through a function called `setcookie()` and a superglobal called `$_COOKIE`. The `setcookie()` function is used to set the value and optional expiration date of a cookie, and the `$_COOKIE` superglobal is used to retrieve the value of a cookie.



The power of setting a cookie is that the cookie data persists across multiple scripts, so we can remember the username without having to prompt the user to log in every time they move from one page to another within the application. But don't forget, we also need to store away the user's ID in a cookie since it serves as a primary key for database queries.

**The PHP
`setcookie()` function
allows you to store
data in cookies.**



The `setcookie()` function also accepts an optional third argument that sets the expiration date of the cookie, which is the date upon which the cookie is automatically deleted. If you don't specify an expiration date, as in the above example, the cookie automatically expires when the browser is closed.



Switching Mismatch to use cookies involves more than just writing a new Log-Out script. We must first revisit the Log-In script and change it to use cookies instead of HTTP authentication. Circle and annotate the parts of the Log-In code that you think need to change to accommodate cookies.

```
<?php
require_once('connectvars.php');

if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW'])) {
    // The username/password weren't entered so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Mismatch"');
    exit('<h3>Mismatch</h3>Sorry, you must enter your username and password to ' .
        'log in and access this page. If you aren\'t a registered member, please ' .
        '<a href="signup.php">sign up</a>.');
}

// Connect to the database
$dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

// Grab the user-entered log-in data
$user_username = mysqli_real_escape_string($dbc, trim($_SERVER['PHP_AUTH_USER']));
$user_password = mysqli_real_escape_string($dbc, trim($_SERVER['PHP_AUTH_PW']));

// Look up the username and password in the database
$query = "SELECT user_id, username FROM mismatch_user WHERE username = " .
    "'$user_username' AND password = SHA('$user_password')";
$data = mysqli_query($dbc, $query);

if (mysqli_num_rows($data) == 1) {
    // The log-in is OK so set the user ID and username variables
    $row = mysqli_fetch_array($data);
    $user_id = $row['user_id'];
    $username = $row['username'];
}
else {
    // The username/password are incorrect so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Mismatch"');
    exit('<h2>Mismatch</h2>Sorry, you must enter a valid username and password ' .
        'to log in and access this page. If you aren\'t a registered member, ' .
        'please <a href="signup.php">sign up</a>');
}

// Confirm the successful log-in
echo('<p class="login">You are logged in as ' . $username . '.</p>');
?>
```



sharpen your pencil solution

Sharpen your pencil Solution

We need to check for the existence of a cookie to see if the user is logged in or not.

Switching Mismatch to use cookies involves more than just writing a new Log-Out script. We must first revisit the Log-In script and change it to use cookies instead of HTTP authentication. Circle and annotate the parts of the Log-In code that you think need to change to accommodate cookies.

Instead of getting the username and password from an authentication window, we need to use a form with POST data.

```
<?php
require_once('connectvars.php');

if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW'])) {
    // The username/password weren't entered so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Mismatch"');
    exit('<h3>Mismatch</h3>Sorry, you must enter your username and password to log in and access this page. If you aren\'t a registered member, please <a href="signup.php">sign up</a>');
}

// Connect to the database
$dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

// Grab the user-entered log-in data
$user_username = mysqli_real_escape_string($dbc, trim($_SERVER['PHP_AUTH_USER']));
$user_password = mysqli_real_escape_string($dbc, trim($_SERVER['PHP_AUTH_PW']));

// Look up the username and password in the database
$query = "SELECT user_id, username FROM mismatch_user WHERE username = '$user_username' AND password = SHA('$user_password')";
$data = mysqli_query($dbc, $query);

if (mysqli_num_rows($data) == 1) {
    // The log-in is OK so set the user ID and username variables
    $row = mysqli_fetch_array($data);
    $user_id = $row['user_id'];
    $username = $row['username'];
} else {
    // The username/password are incorrect so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Mismatch"');
    exit('<h2>Mismatch</h2>Sorry, you must enter a valid username and password to log in and access this page. If you aren\'t a registered member, please <a href="signup.php">sign up</a>');
}

// Confirm the successful log-in
echo('<p class="login">You are logged in as ' . $username . '</p>');
?>
```

We no longer need to send HTTP authentication headers.

The query doesn't have to change at all!

Here we need to set two cookies instead of setting script variables.

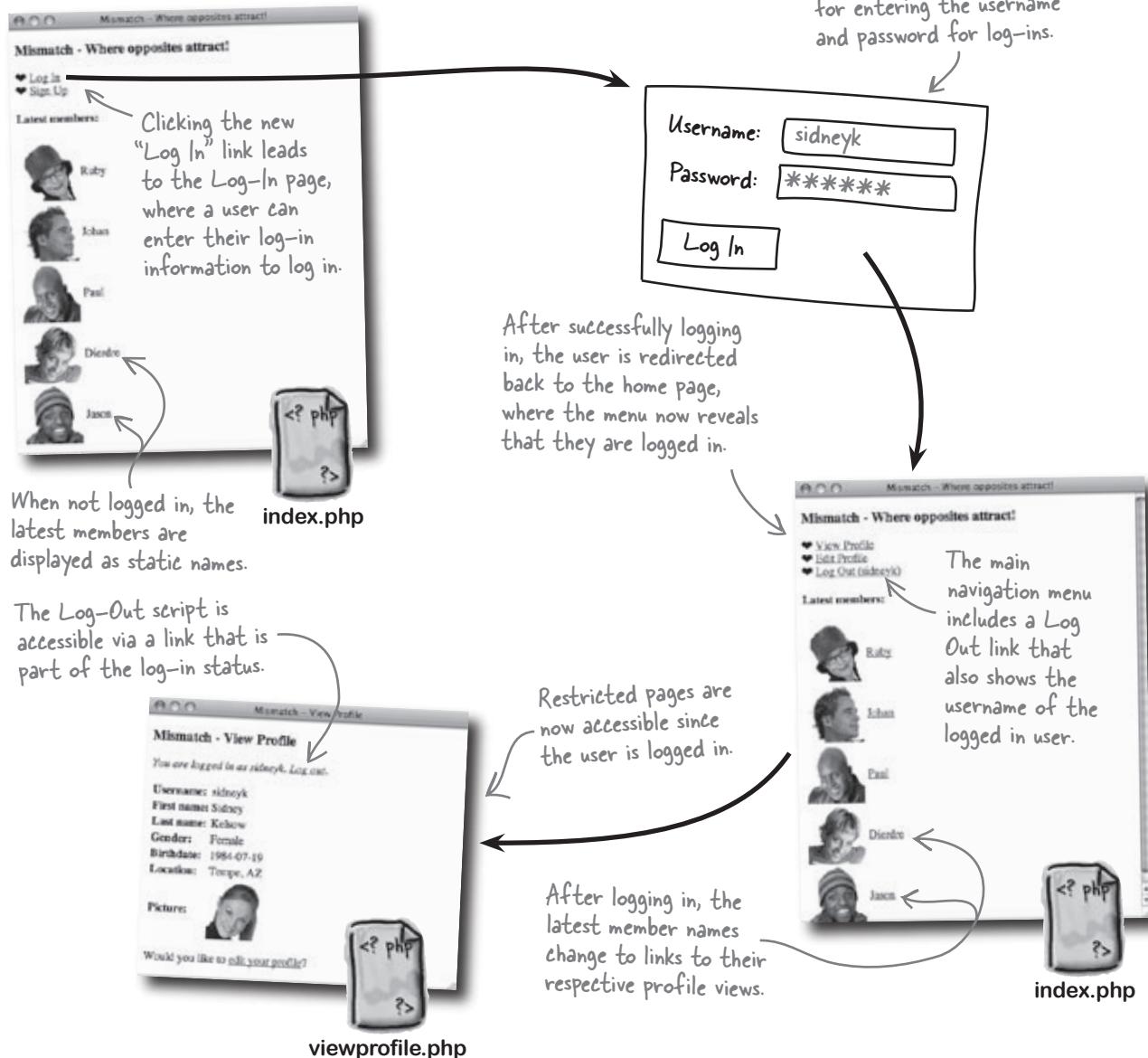
Since we can't rely on the HTTP authentication window for entering the username and password, we need to create an HTML Log-In form for entering them.



login.php

Rethinking the flow of log-ins

Using cookies instead of HTTP authentication for Mismatch log-ins involves more than just rethinking the storage of user data. What about the log-in user interface? The cookie-powered log-in must provide its own form since it can't rely on the authentication window for entering a username and password. Not only do we have to build this form, but we need to think through how it changes the flow of the application as users log in and access other pages.



login.php—now cookie-powered!

A cookie-powered log-in

The new version of the Log-In script that relies on cookies for log-in persistence is a bit more complex than its predecessor since it must provide its own form for entering the username and password. But it's more powerful in that it provides log-out functionality.

```
<?php
require_once('connectvars.php');

// Clear the error message
$error_msg = "";

// If the user isn't logged in, try to log them in
if (!isset($_COOKIE['user_id'])) {
    if (isset($_POST['submit'])) {
        // Connect to the database
        $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

        // Grab the user-entered log-in data
        $user_username = mysqli_real_escape_string($dbc, trim($_POST['username']));
        $user_password = mysqli_real_escape_string($dbc, trim($_POST['password']));

        if (!empty($user_username) && !empty($user_password)) {
            // Look up the username and password in the database
            $query = "SELECT user_id, username FROM mismatch_user WHERE username = '$user_username' AND ";
            $password = SHA('$user_password');
            $data = mysqli_query($dbc, $query);

            if (mysqli_num_rows($data) == 1) {
                // The log-in is OK so set the user ID and username cookies, and redirect to the home page
                $row = mysqli_fetch_array($data);
                setcookie('user_id', $row['user_id']);
                setcookie('username', $row['username']);
                $home_url = 'http://'. $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
                header('Location: ' . $home_url);
            }
            else {
                // The username/password are incorrect so set an error message
                $error_msg = 'Sorry, you must enter a valid username and password to log in.';
            }
        }
        else {
            // The username/password weren't entered so set an error message
            $error_msg = 'Sorry, you must enter your username and password to log in.';
        }
    }
}

?>

<html>
<head>
    <title>Mismatch - Log In</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
    <h3>Mismatch - Log In</h3>
```

Error messages are now stored in a variable and displayed, if necessary, later in the script.

Check the user_id cookie to see if the user is logged in.

If the user isn't logged in, see if they've submitted log-in data.

The user-entered data now comes from form POST data instead of an authentication window.

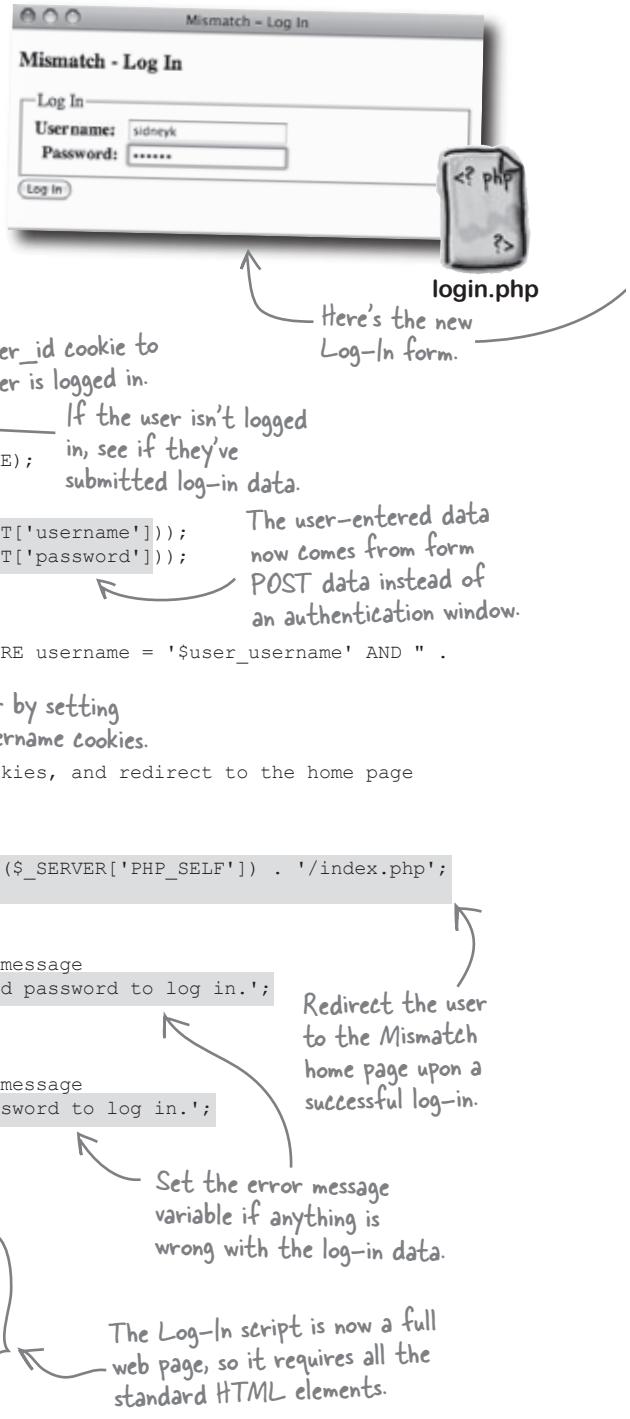
Log in the user by setting user_id and username cookies.

Set the error message variable if anything is wrong with the log-in data.

Redirect the user to the Mismatch home page upon a successful log-in.

The Log-In script is now a full web page, so it requires all the standard HTML elements.

continues on the facing page...



```

<?php
    // If the cookie is empty, show any error message and the log-in form; otherwise confirm the log-in
    if (empty($_COOKIE['user_id'])) {
        echo '<p class="error">' . $error_msg . '</p>'; ←
    }
?>

    {<form method="post" action=<?php echo $_SERVER['PHP_SELF']; ?>>
        <fieldset>
            <legend>Log In</legend>
            <label for="username">Username:</label>
            <input type="text" id="username" name="username"
                value=<?php if (!empty($user_username)) echo $user_username; ?>><br />
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" />
        </fieldset>
        <input type="submit" value="Log In" name="submit" />
    </form>
} ← Everything prior to this curly brace
    is still part of the first if clause.

<?php
    } ←
    else {
        // Confirm the successful log in
        echo('<p class="login">You are logged in as ' . $_COOKIE['username'] . '.</p>');
    }
?>

</body>
</html> ← Finish the HTML code to
            complete the Log-In web page.

```

If the user still isn't logged in at this point, go ahead and show the error message.

These two form fields are used to enter the username and password for logging in.

If the user is logged in at this point, just tell them so.

there are no Dumb Questions

Q: Why is it necessary to store both the user ID and username in cookies?

A: Since both pieces of information uniquely identify a user within the Mismatch user database, you could use either one for the purpose of keeping up with the current user. However, `user_id` is a better (more efficient) user reference with respect to the database because it is a numeric primary key. On the other hand, `user_id` is fairly cryptic and doesn't have any meaning to the user, so `username` comes in handy for letting the user know they are logged in, such as displaying their name on the page. Since multiple people sometimes share the same computer, it is important to not just let the user know they are logged in, but also who they are logged in as.

Q: Then why not also store the password in a cookie as part of the log-in data?

A: The password is only important for initially verifying that a user is who they claim to be. Once the password is verified as part of the log-in process, there is no reason to keep it around. Besides, passwords are very sensitive data, so it's a good idea to avoid storing them temporarily if at all possible.

Q: It looks as if the form in the Log-In script is actually inside the if statement? Is that possible?

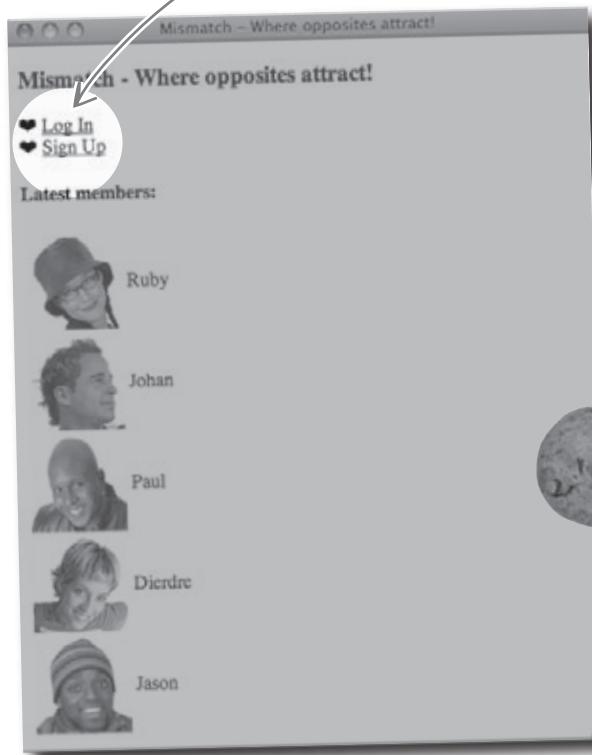
A: Yes. In fact it's quite common for PHP code to be "broken up" around HTML code, as is the case with the Log-In script. Just because you close a section of PHP code with `?>`, doesn't mean the logic of the code is closed. When you open another section of PHP code with `<?php`, the logic continues right where it left off. In the Log-In script, the HTML form is contained within the first `if` branch, while the `else` branch picks up after the form code. Breaking out of PHP code into HTML code like this keeps you from having to generate the form with a bunch of messy `echo` statements.

Navigating the Mismatch application

The new Log-In script changes the flow of the Mismatch application, requiring a simple menu that appears on the home page (`index.php`). This menu is important because it provides access to the different major parts of the application, currently the View Profile and Edit Profile pages, as well as the ability for users to log in, sign up, and log out depending on their current log-in state. The fact that the menu changes based on the user's log-in state is significant and is ultimately what gives the menu its power and usefulness.

A different menu is shown depending on whether the username cookie is set.

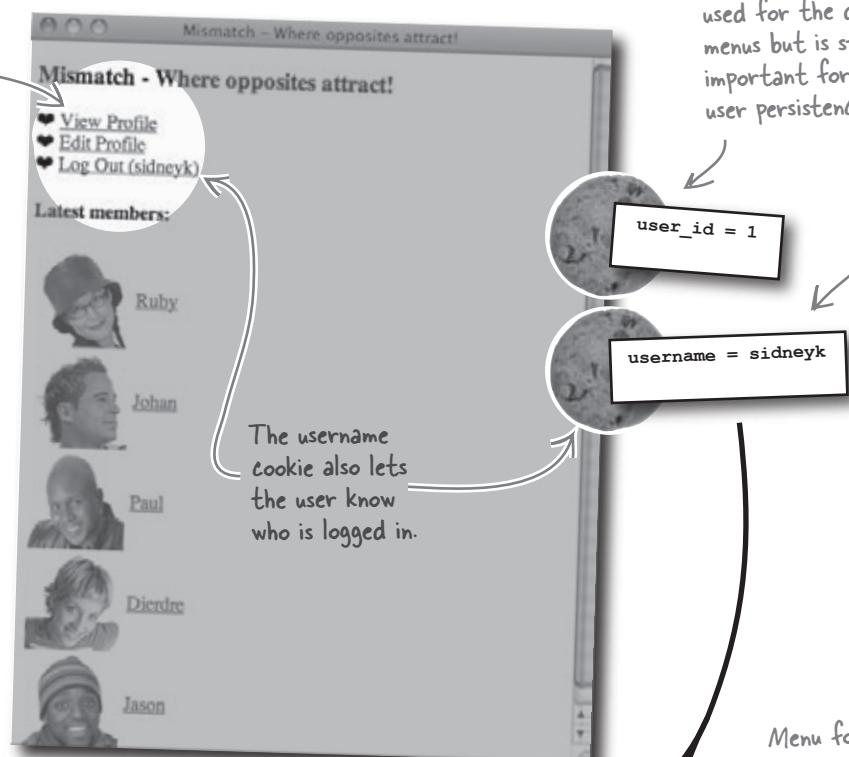
This menu appears when a user is not logged in, giving them an opportunity to either log in or sign up.



username = ?

The `index.php` script knows to show the limited menu when it can't find the username cookie.

The menu is generated by PHP code within the `index.php` script, and this code uses the `$_COOKIE` superglobal to look up the username cookie and see if the user is logged in or not. The user ID cookie could have also been used, but the username is actually displayed in the menu, so it makes more sense to check for it instead.



The user_id cookie isn't used for the different menus but is still important for Mismatch user persistence.

The username cookie determines which menu is displayed

The username cookie also lets the user know who is logged in.

Menu for logged in users



```
// Generate the navigation menu
if (isset($_COOKIE['username'])) {
    echo '&#10084; <a href="viewprofile.php">View Profile</a><br />';
    echo '&#10084; <a href="editprofile.php">Edit Profile</a><br />';
    echo '&#10084; <a href="logout.php">Log Out (' . $_COOKIE['username'] . ')</a>';
}
else {
    echo '&#10084; <a href="login.php">Log In</a><br />';
    echo '&#10084; <a href="signup.php">Sign Up</a>';
}
```

The little heart symbols next to each menu item are made possible by this HTML entity, which is supported on most browsers.

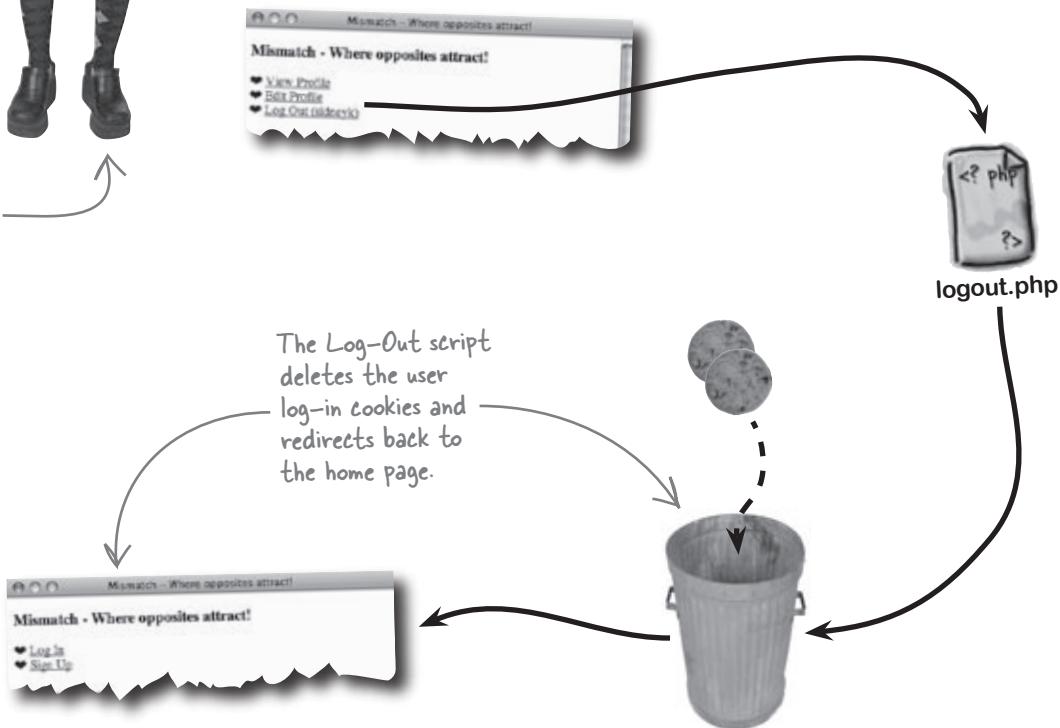
Menu for visitors (users who aren't logged in)



log out users by deleting cookies**We really need to let users log out.**

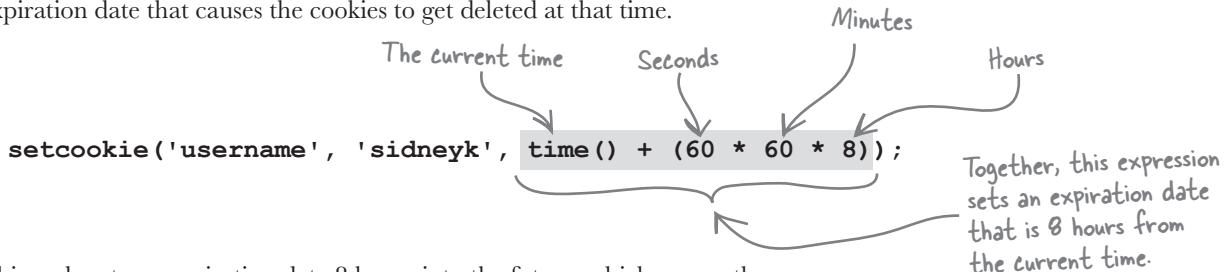
Cookies have made logging into Mismatch and navigating the site a bit cleaner, but the whole point of switching from HTTP authentication to cookies was to allow users to log out. We need a new Log-Out script that deletes the two cookies (user ID and username) so that the user no longer has access to the application. This will prevent someone from getting on the same computer later and accessing a user's private profile data.

Since there is no user interface component involved in actually logging out a user, it's sufficient to just redirect them back to the home page after logging them out.



Logging out means deleting cookies

Logging out a user involves deleting the two cookies that keep track of the user. This is done by calling the `setcookie()` function, and passing an expiration date that causes the cookies to get deleted at that time.



This code sets an expiration date 8 hours into the future, which means the cookie will be automatically deleted in 8 hours. But we want to delete a cookie immediately, which requires setting the expiration date to a time in the past. The amount of time into the past isn't terribly important—just pick an arbitrary amount of time, such as an hour, and subtract it from the current time.

```
setcookie('username', 'sidneyk', time() - 3600);
```

*60 seconds * 60 minutes = 3600 seconds, which is 1 hour into the past.*

To delete a cookie, just set its expiration date to a time in the past.



The Log-Out script for Mismatch is missing a few pieces of code. Write the missing code, making sure that the log-in cookies get deleted before the Log-Out page is redirected to the home page.

```
<?php
    // If the user is logged in, delete the cookie to log them out
    if ( ..... ) {
        // Delete the user ID and username cookies by setting their expirations to an hour ago (3600)
        .....
        .....
    }

    // Redirect to the home page
    $home_url = 'http://'. $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '.....';
    header('Location: ' . $home_url);
?>
```

the complete `logout.php` script



The Log-Out script for Mismatch is missing a few pieces of code. Write the missing code, making sure that the log-in cookies get deleted before the Log-Out page is redirected to the home page.

```
<?php
    // If the user is logged in, delete the cookie to log them out
    if ( !isset($_COOKIE['user_id']) ) {
        // Delete the user ID and username cookies by setting their expirations to an hour ago (3600)
        setcookie('user_id', '', time() - 3600);
        setcookie('username', '', time() - 3600);
    }

    // Redirect to the home page
    $home_url = 'http://'. $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
    header('Location: ' . $home_url);
?>
```

Only log out a user if they are already logged in.

Set each cookie to an hour in the past so that they are deleted by the system.

Redirect to the Mismatch home page, which is constructed as an absolute URL.

A location header results in the browser redirecting to another page.

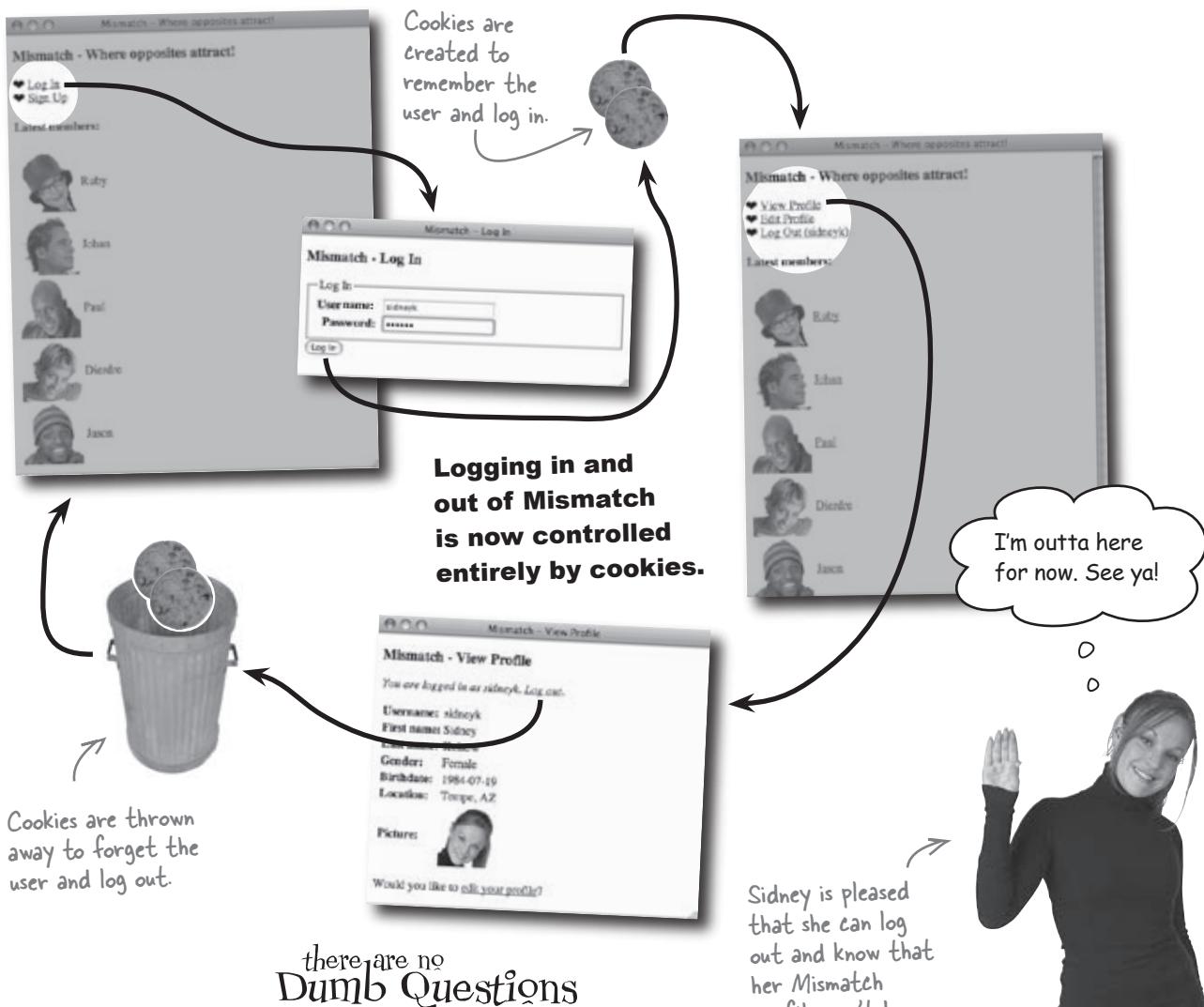


Test Drive

Use cookies to add Log-Out functionality to Mismatch.

Modify the Mismatch scripts so that they use cookies to allow users to log in and out (or download the scripts from the Head First Labs site at www.headfirstlabs.com/books/hfphp. The cookie modifications involve changes to the `index.php`, `login.php`, `logout.php`, `editprofile.php`, and `viewprofile.php` scripts. The changes to the latter two scripts are fairly minor, and primarily involve changing `$user_id` and `$username` global variable references so that they use the `$_COOKIE` superglobal instead.

Upload the scripts to your web server, and then open the main Mismatch page (`index.php`) in a web browser. Take note of the navigation menu, and then click the “Log In” link and log in. Notice how the Log-In script leads you back to the main page, while the menu changes to reflect your logged in status. Now click “Log Out” to blitz the cookies and log out.

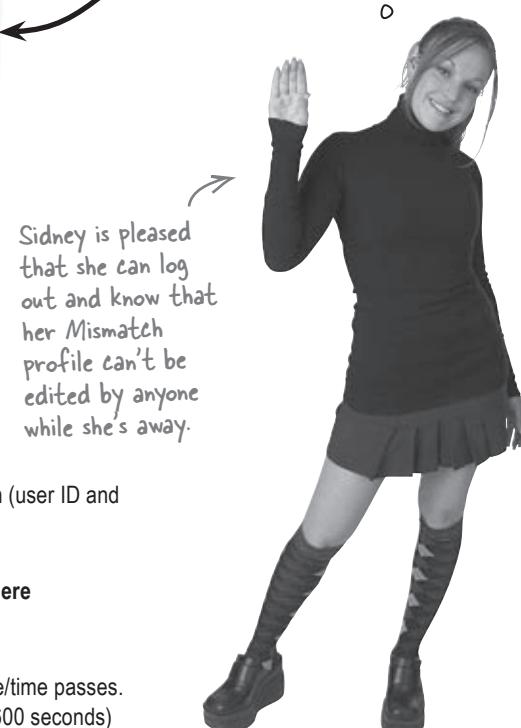


Q: So simply deleting the cookies is all that is required to log out?

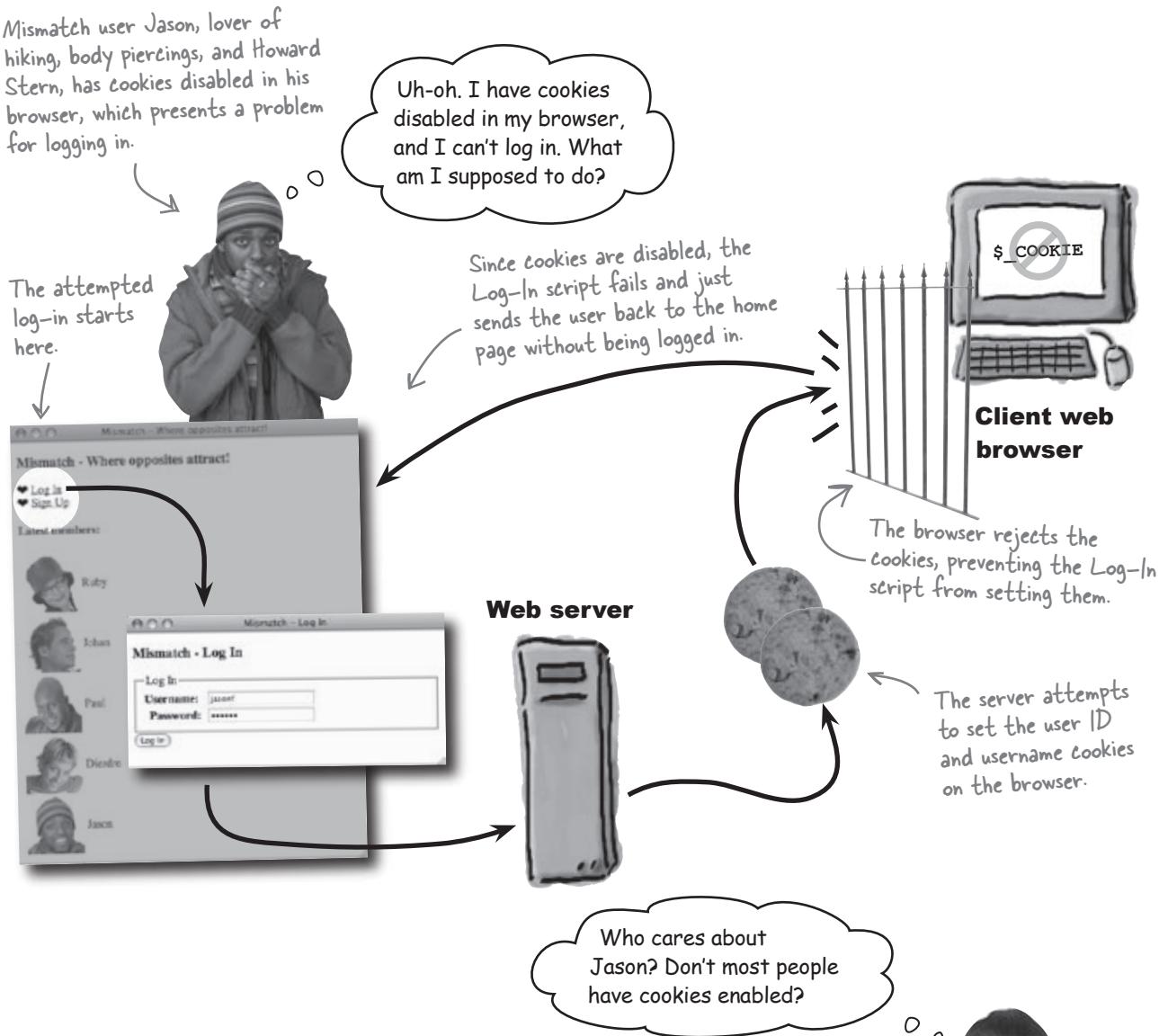
A: Yes. Cookies are responsible for storing all of the log-in information for Mismatch (user ID and username), so deleting them results in a complete log-out.

Q: Why are the cookies set to an hour in the past in order to be deleted? Is there something significant about an hour?

A: No. A cookie is automatically deleted by the web browser once its expiration date/time passes. So deleting a cookie involves setting the expiration to any time in the past. An hour (3600 seconds) is just an arbitrary amount of time chosen to consistently indicate that we're deleting a cookie.



storing user data on the server, instead of the client



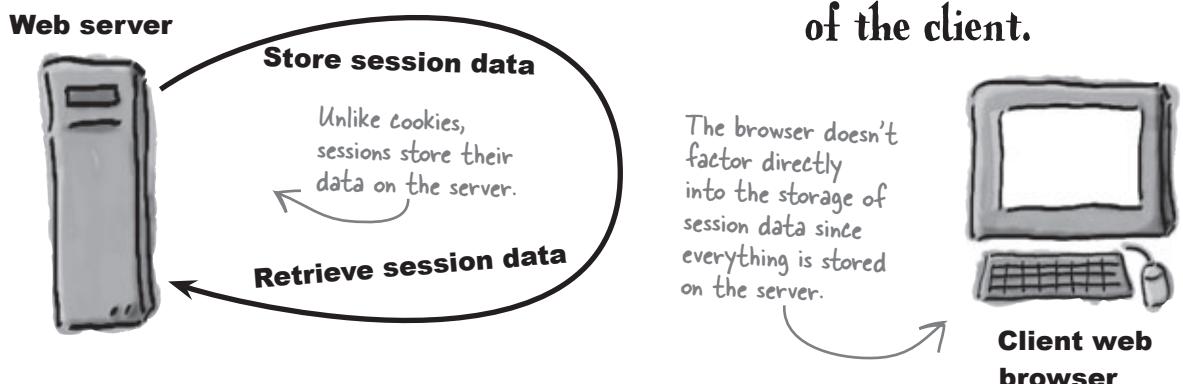
Yes, but web applications should be as accessible to as many people as possible.

Some people just aren't comfortable using cookies, so they opt for the added security of having them disabled. Knowing this, it's worth trying to accommodate users who can't rely on cookies to log in. But there's more. It turns out that there's another option that **uses the server to store log-in data**, as opposed to the client. And since our scripts are already running on the server, it only makes sense to store log-in data there as well.

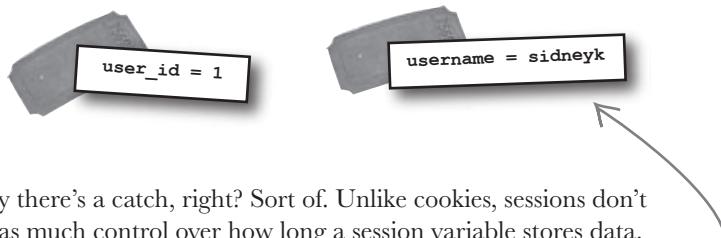


Sessions aren't dependent on the client

Cookies are powerful little guys, but they do have their limitations, such as being subject to limitations beyond your control. But what if we didn't have to depend on the browser? What if we could store data directly on the server? **Sessions** do just that, and they allow you to store away individual pieces of information just like with cookies, but the data gets stored on the server instead of the client. This puts session data outside of the browser limitations of cookies.



Sessions store data in **session variables**, which are logically equivalent to cookies on the server. When you place data in a session variable using PHP code, it is stored on the server. You can then access the data in the session variable from PHP code, and it remains persistent across multiple pages (scripts). Like with cookies, you can delete a session variable at any time, making it possible to continue to offer a log-out feature with session-based code.



Surely there's a catch, right? Sort of. Unlike cookies, sessions don't offer as much control over how long a session variable stores data. Session variables are **automatically destroyed as soon as a session ends**, which usually coincides with the user shutting down the browser. So even though session variables aren't stored on the browser, they are indirectly affected by the browser since they get deleted when a browser session ends.

Sessions allow you to persistently store small pieces of data on the server, independently of the client.

The browser doesn't factor directly into the storage of session data since everything is stored on the server.



A curved arrow points from the text above to the "Client web browser" illustration.

Since session data is stored on the server, it is more secure and more reliable than data stored in cookies.

A user can't manually delete session data using their browser, which can be a problem with cookies.

There isn't an expiration date associated with session variables because they are automatically deleted when a session ends.

The life and times of sessions

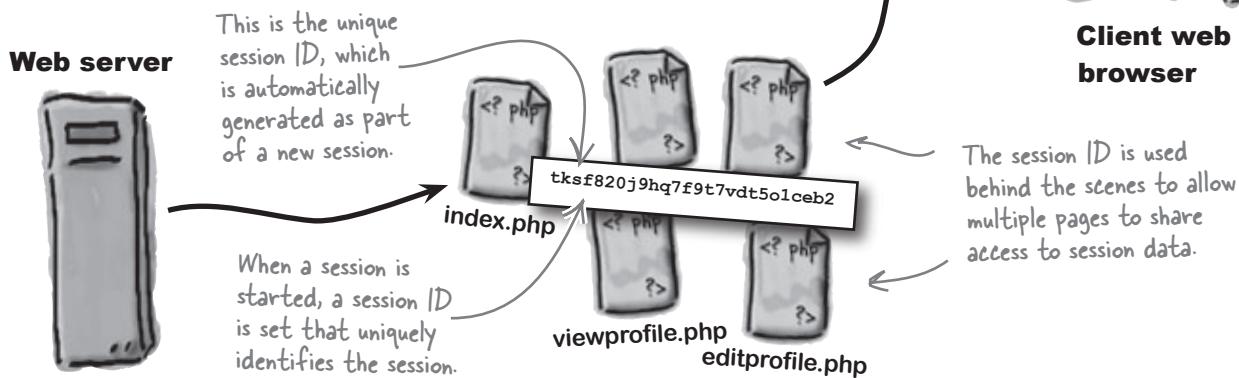
Sessions are called sessions for a reason—they have a very clear start and finish. Data associated with a session lives and dies according to the lifespan of the session, which you control through PHP code. The only situation where you don't have control of the session life cycle is when the user closes the browser, which results in a session ending, whether you like it or not.

You must tell a session when you're ready to start it up by calling the `session_start()` PHP function.

The `PHP session_start()` function starts a session and allows you to begin storing data in session variables.

`session_start();` ← This PHP function starts a session.

Calling the `session_start()` function doesn't set any data—its job is to get the session up and running. The session is identified internally by a unique session identifier, which you typically don't have to concern yourself with. This ID is used by the web browser to associate a session with multiple pages.



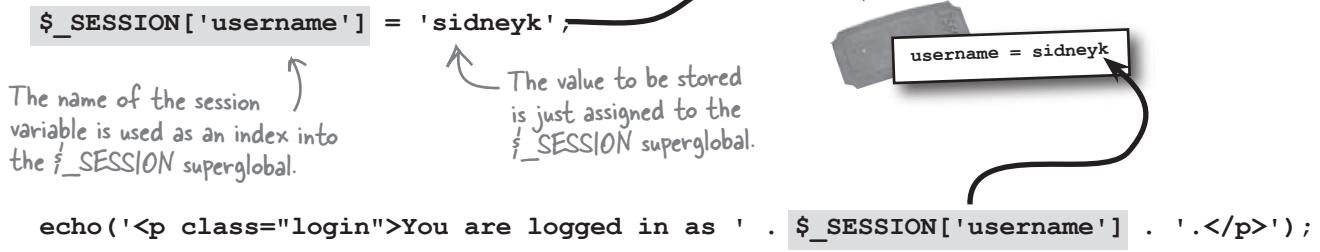
The session ID isn't destroyed until the session is closed, which happens either when the browser is closed or when you call the `session_destroy()` function.

`session_destroy();` ← This PHP function ends a session.

If you close a session yourself with this function, it doesn't automatically destroy any session variables you've stored. Let's take a closer look at how sessions store data to uncover why this is so.

Keeping up with session data

The cool thing about sessions is that they're very similar to cookies in terms of how you use them. Once you've started a session with a call to `session_start()`, you can begin setting session variables, such as Mismatch log-in data, with the `$_SESSION` superglobal.



Unlike cookies, session variables don't require any kind of special function to set them—you just assign a value to the `$_SESSION` superglobal, making sure to use the session variable name as the array index.

What about deleting session variables? Destroying a session via `session_destroy()` doesn't actually destroy session variables, so you must manually delete your session variables if you want them to be killed prior to the user shutting down the browser (log-outs!). A quick and effective way to destroy all of the variables for a session is to set the `$_SESSION` superglobal to an empty array.

`$_SESSION = array();`

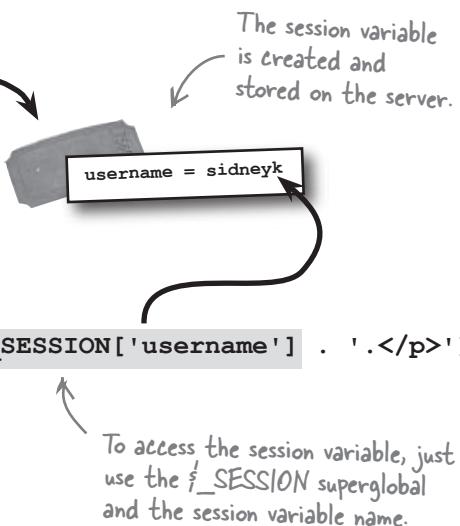
This code kills all of the session variables in the current session.

But we're not quite done. Sessions can actually use cookies behind the scenes. If the browser allows cookies, a session may possibly set a cookie that temporarily stores the session ID. So to fully close a session via PHP code, you must also delete any cookie that might have been automatically created to store the session ID on the browser. Like any other cookie, you destroy this cookie by setting its expiration to some time in the past. All you need to know is the name of the cookie, which can be found using the `session_name()` function.

```
if (isset($_COOKIE[session_name()])) {
    setcookie(session_name(), '', time() - 3600);
}
```

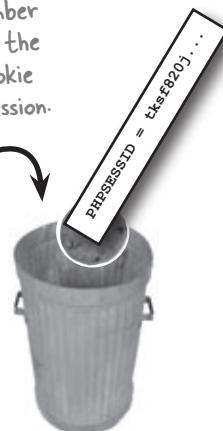
First check to see if a session cookie actually exists.

Destroy the session cookie by setting its expiration to an hour in the past.



Session variables are not automatically deleted when a session is destroyed.

If a session is using a cookie to help remember the session ID, then the ID is stored in a cookie named after the session.



how mismatch works with sessions

Start here!

**Renovate Mismatch with sessions**

Reworking the Mismatch application to use a session to store log-in data isn't as dramatic as it may sound. In fact, the flow of the application remains generally the same—you just have to take care of a little extra bookkeeping involved in starting the session, destroying the session, and then cleaning up after the session.

`session_start();`

The `session_start()` function gets things started by opening a session.

If cookies are enabled, the server creates one to hold the session ID – otherwise the ID is passed through the URL of each page.

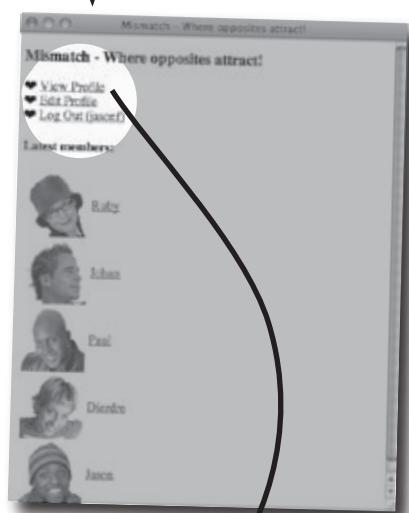
`session_destroy();`

The `session_destroy()` function ends the session, preventing it from being used in another page.

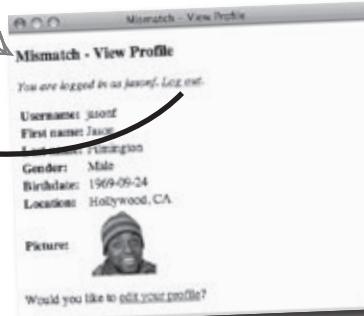
If a cookie was used to hold the session ID, it is destroyed.

The session variables are destroyed by clearing out the `'SESSION'` array.

Two session variables are created to store the user ID and username for the log-in.



Log-in data is now remembered using a session instead of cookies.



Log out with sessions

Logging a user out of Mismatch requires a little more work with sessions than the previous version with its pure usage of cookies. These steps must be taken to successfully log a user out of Mismatch using sessions.

- 1 Delete the session variables.**
- 2 Check to see if a session cookie exists, and if so, delete it.**
- 3 Destroy the session.**
- 4 Redirect the user to the home page.**

You don't know for certain if a session cookie is being used without checking.

OK, so this is a bonus step that isn't strictly required to log the user out, but is helpful nonetheless.



The Log-Out script for Mismatch is undergoing an overhaul to use sessions instead of pure cookies for log-in persistence. Write the missing code to “sessionize” the Log-Out script, and then annotate which step of the log-out process it corresponds to.

```
<?php
// If the user is logged in, delete the session vars to log them out
session_start();

if ( ..... ) {
    // Delete the session vars by clearing the $_SESSION array
    .....
}

// Delete the session cookie by setting its expiration to an hour ago (3600)
if (isset($_COOKIE[session_name()])) {

    .....
}

// Destroy the session
.....
}

// Redirect to the home page
$home_url = 'http://'. $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
header('Location: ' . $home_url);
?>
```

the "sessionized" logout.php



The Log-Out script for Mismatch is undergoing an overhaul to use sessions instead of pure cookies for log-in persistence. Write the missing code to "sessionize" the Log-Out script, and then annotate which step of the log-out process it corresponds to.

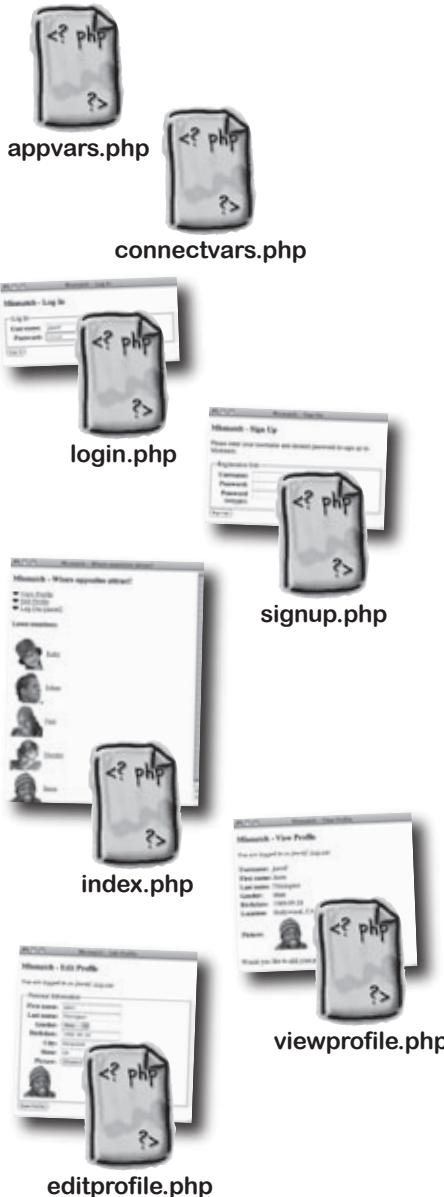
- ① Delete the session variables.
- ② Check to see if a session cookie exists, and if so, delete it.
- ③ Destroy the session.
- ④ Redirect the user to the home page.

Even when logging out, you have to first start the session in order to access the session variables.

```
<?php
    // If the user is logged in, delete the session vars to log them out
    session_start();
    if ( !isset($_SESSION['user_id']) ) {
        // Delete the session vars by clearing the $_SESSION array
        $_SESSION = array(); // ① To clear out the session variables, assign the
                            // $_SESSION superglobal an empty array.
    }
    // Delete the session cookie by setting its expiration to an hour ago (3600)
    if (isset($_COOKIE[session_name()])) {
        setcookie(session_name(), '', time() - 3600); // ②
    }
    // Destroy the session
    session_destroy(); // ③ Destroy the session with
                      // a call to the built-in
                      // session_destroy() function.
}
// Redirect to the home page
$home_url = 'http://'. $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
header('Location: ' . $home_url); // ④
?>
```

HOW DO I CHANGE?

The move from cookies to sessions impacts more than just the Log-Out script. Match the other pieces of the Mismatch application with how they need to change to accommodate sessions.



No change since the script has no direct dependence on log-in persistence.

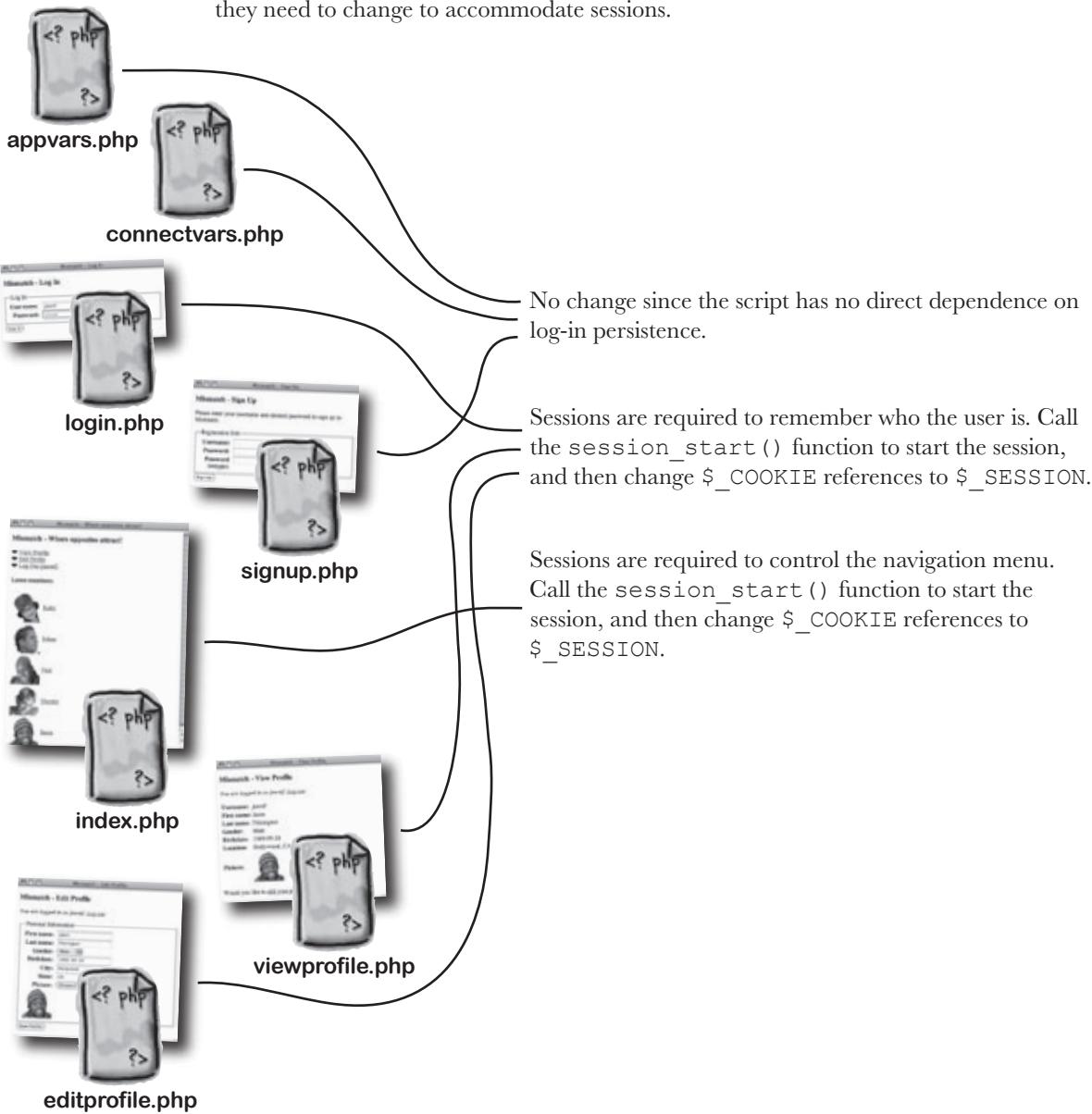
Sessions are required to remember who the user is. Call the `session_start()` function to start the session, and then change `$_COOKIE` references to `$_SESSION`.

Sessions are required to control the navigation menu. Call the `session_start()` function to start the session, and then change `$_COOKIE` references to `$_SESSION`.

how do i change solution

HOW DO I CHANGE? SOLUTION

The move from cookies to sessions impacts more than just the Log-Out script. Match the other pieces of the Mismatch application with how they need to change to accommodate sessions.





BULLET POINTS

- HTTP authentication is handy for restricting access to individual pages, but it doesn't offer a good way to "log out" a user when they're finished accessing a page.
- Cookies let you store small pieces of data on the client (web browser), such as the log-in data for a user.
- All cookies have an expiration date, which can be far into the future or as near as the end of the browser session.
- To delete a cookie, you just set its expiration to a time in the past.
- Sessions offer similar storage as cookies but are stored on the server and, therefore, aren't subject to the same browser limitations, such as cookies being disabled.
- Session variables have a limited lifespan and are always destroyed once a session is over (for example, when the browser is closed).

there are no Dumb Questions

Q: The `session_start()` function gets called in a lot of different places, even after a session has been started. Are multiple sessions being created with each call to `session_start()`?

A: No. The `session_start()` function doesn't just start a new session—it also taps into an existing session. So when a script calls `session_start()`, the function first checks to see if a session already exists by looking for the presence of a session ID. If no session exists, it generates a new session ID and creates the new session. Future calls to `session_start()` from within the same application will recognize the existing session and use it instead of creating another one.

Q: So how does the session ID get stored? Is that where sessions sometimes use cookies?

A: Yes. Even though session data gets stored on the server and, therefore, gains the benefit of being more secure and outside of the browser's control, there still has to be a mechanism for a script to know about the session data.

This is what the session ID is for—it uniquely identifies a session and the data associated with it. This ID must somehow persist on the client in order for multiple pages to be part of the same session. One way this session ID persistence is carried out is through a cookie, meaning that the ID is stored in a cookie, which is then used to associate a script with a given session.

Q: If sessions are dependent on cookies anyway, then what's the big deal about using them instead of cookies?

A: Sessions are not entirely dependent on cookies. It's important to understand that cookies serve as an **optimization** for preserving the session ID across multiple scripts, not as a necessity. If cookies are disabled, the session ID gets passed from script to script through a URL, similar to how you've seen data passed in a GET request. So sessions can work perfectly fine without cookies. The specifics of how sessions react in response to cookies being disabled are controlled in the `php.ini` configuration file on the web server via the `session.use_cookies`, `session.use_only_cookies`, and `session.use_trans_sid` settings.

Q: It still seems strange that sessions could use cookies when the whole point is that sessions are supposed to be better than cookies. What gives?

A: While sessions do offer some clear benefits over cookies in certain scenarios, they don't necessarily have an either/or relationship with cookies. Sessions certainly have the benefit of being stored on the server instead of the client, which makes them more secure and dependable. So if you ever need to store sensitive data persistently, then a session variable would provide more security than a cookie. Sessions are also capable of storing larger amounts of data than cookies. So there are clear advantages to using sessions regardless of whether cookies are available.

For the purposes of Mismatch, sessions offer a convenient server-side solution for storing log-in data. For users who have cookies enabled, sessions provide improved security and reliability while still using cookies as an optimization. And in the case of users who don't have cookies enabled, sessions can still work by passing the session ID through a URL, foregoing cookies altogether.

Complete the session transformation

Even though the different parts of Mismatch affected by sessions use them to accomplish different things, the scripts ultimately require similar changes in making the migration from cookies to sessions. For one, they all must call the `session_start()` function to get rolling with sessions initially. Beyond that, all of the changes involve moving from the `$_COOKIE` superglobal to the `$_SESSION` superglobal, which is responsible for storing session variables.

```
<?php
    session_start();
?>
```

All of the session-powered scripts start out with a call to `session_start()` to get the session up and running.

```
// If the user isn't logged in, try to log them in
if (!isset($_SESSION['user_id'])) {
    if (isset($_POST['submit'])) {
        // Connect to the database
        $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

        // Grab the user-entered log-in data
        $user_username = mysqli_real_escape_string($dbc, trim($_POST['username']));
        $user_password = mysqli_real_escape_string($dbc, trim($_POST['password']));

        if (!empty($user_username) && !empty($user_password)) {
            // Look up the username and password in the database
            $query = "SELECT user_id, username FROM mismatch_user WHERE username = '$user_username' AND " .
                "password = SHA('$user_password')";
            $data = mysqli_query($dbc, $query);

            if (mysqli_num_rows($data) == 1) {
                // The log-in is OK so set the user ID and username session vars, and redirect to the home page
                $row = mysqli_fetch_array($data);
                $_SESSION['user_id'] = $row['user_id'];
                $_SESSION['username'] = $row['username'];
                $home_url = "http://$_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
                header('Location: ' . $home_url);
            }
            else {
                // The username/password are incorrect so set an error message
                $error_msg = 'Sorry, you must enter a valid username and password to log in.';
            }
        }
    }
}
```



login.php

The Log-In script uses sessions to remember the user ID and username for log-in persistence, and it does so by relying on the `$_SESSION` superglobal instead of `$_COOKIE`.

```
// Generate the navigation menu
if (isset($_SESSION['username'])) {
    echo '&#10084; <a href="viewprofile.php">View Profile</a><br />';
    echo '&#10084; <a href="editprofile.php">Edit Profile</a><br />';
    echo '&#10084; <a href="logout.php">Log Out (' . $_SESSION['username'] . ')</a>';
} else {
    echo '&#10084; <a href="login.php">Log In</a><br />';
    echo '&#10084; <a href="signup.php">Sign Up</a>';
}

...
// Loop through the array of user data, formatting it as HTML
echo '<h4>Latest members:</h4>';
echo '<table>';
while ($row = mysqli_fetch_array($data)) {
    ...
    if (isset($_SESSION['user_id'])) {
        echo '<td><a href="viewprofile.php?user_id=' . $row['user_id'] . '">' . .
            $row['first_name'] . '</a></td></tr>';
    } else {
        echo '<td>' . $row['first_name'] . '</td></tr>';
    }
}
echo '</table>';
```

The Mismatch home page uses the `$_SESSION` superglobal instead of `$_COOKIE` to access log-in data while generating the menu and choosing whether or not to provide a link to the "latest members" profiles.



index.php

Similar to the Log-In and home pages, the Edit Profile script now uses `$_SESSION` to access log-in data instead of `$_COOKIE`.

Although not shown, the View Profile script uses sessions in much the same way as Edit Profile.



viewprofile.php

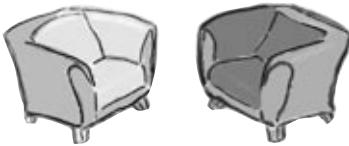
```
// Make sure the user is logged in before going any further.
if (!isset($_SESSION['user_id'])) {
    echo '<p>Please log in</p>' . '<a href="login.php">log in</a> to access this page.</p>';
    exit();
}
else {
    echo('<p class="login">You are logged in as ' . $_SESSION['username'] .
        '. <a href="logout.php">Log out</a>.</p>');
}

...
if (!empty($first_name) && !empty($last_name) && !empty($gender) && !empty($birthdate) &&
    !empty($city) && !empty($state)) {
    // Only set the picture column if there is a new picture
    if (!empty($new_picture)) {
        $query = "UPDATE mismatch_user SET first_name = '$first_name', last_name = '$last_name', " .
            "gender = '$gender', birthdate = '$birthdate', city = '$city', state = '$state', " .
            "picture = '$new_picture' WHERE user_id = '" . $_SESSION['user_id'] . "'";
    }
    else {
        $query = "UPDATE mismatch_user SET first_name = '$first_name', last_name = '$last_name', " .
            "gender = '$gender', birthdate = '$birthdate', city = '$city', state = '$state' " .
            "WHERE user_id = '" . $_SESSION['user_id'] . "'";
    }
    mysqli_query($dbc, $query);
```



editprofile.php

Fireside Chats

**Cookie:**

There's been a lot of talk around here among us cookies about what exactly goes on over there on the server. Rumor is you're trying to move in on our territory and steal data storage jobs. What gives?

That doesn't make any sense to me. The browser is a perfectly good place to store data, and I'm just the guy to do it.

Uh, well, that's a completely different issue. And if the user decides to disable me, then clearly they don't have any need to store data.

So I suppose your answer is to store the data on the server? How convenient.

Alright, Einstein. Since you seem to have it all figured out, why is it that you still sometimes use me to store your precious little ID on the browser?

Tonight's talk: **Cookie and session variable get down and dirty about who has the best memory**

**Session variable:**

Come on now, steal is a strong word. The truth is sometimes it just makes more sense to store data on the server.

What about when the user disables you?

Not true. The user often doesn't even know a web application is storing data because in many cases, it is behind-the-scenes data, like a username. So if you're not available, they're left with nothing.

Exactly. And the cool thing is that the user doesn't have the ability to disable anything on the server, so you don't have to worry about whether or not the data is really able to be stored.

Er, well, most people really don't know about that, so there's no need to get into it here. We can talk about that off the record. The important thing is that I'm always around, ready to store data on the server.

Cookie:

Come on, tell me how much you need me!

Oh I know you can, but the truth is you'd rather not.
And maybe deep down you really kinda like me.

Ah, so you're going to resort to picking on the little guy. Sure, I may not be able to store quite as much as you, and I'll admit that living on the client makes me a little less secure. But it sure is more exciting! And I have something you can only dream about.

Well, all that storage space and security you're so proud of comes at a cost... a short lifespan! I didn't want to be the one to have to tell you, but your entire existence is hinging on a single browser session. I think that's how you got your name.

It's simple. I don't die with a session, I just expire. So I can be set to live a long, full life, far beyond the whim of some click-happy web surfer who thinks it's cute to open and close the browser every chance he gets.

Problem is, those same scripters often set my expiration to such a short period that I don't really get to experience the long life I truly deserve. I mean, I...

Session variable:

Alright, I will admit that from time to time I do lean on you a little to help me keep up with things across multiple pages. But I can get by without you if I need to.

Look, I don't have any problem with you. I just wish you were a little more secure. And you have that size limitation. You know, not every piece of persistent data is bite-sized.

Is that so? Do tell.

You mean you can go on living beyond a single session? How is that possible?!

Wow. What a feeling that must be to experience immortality. My only hope is that some slacker scripter accidentally forgets to destroy me when he closes a session... but the browser will still do me in whenever it gets shut down.

Hello? Are you there? Geez, expiration is harsh.

test drive the "sessionized" mismatch



Test Drive

Change Mismatch to use sessions instead of cookies.

Modify the Mismatch scripts so that they use sessions instead of cookies to support log-in persistence (or download the scripts from the Head First Labs site at [www.headfirstlabs.com/books/hfphp](http://headfirstlabs.com/books/hfphp)). The session modifications involve changes to the index.php, login.php, logout.php, editprofile.php, and viewprofile.php scripts, and primarily involve starting the session with a call to the session_start() function and changing \$_COOKIE superglobal references to use \$_SESSION instead.

Upload the scripts to your web server, and then open the main Mismatch page (index.php) in a web browser. Try logging in and out to make sure everything works the same as before. Unless you had cookies disabled earlier, you shouldn't notice any difference—that's a good thing!

Very cool. It's nice being able to log in even without cookies turned on.

Thanks to sessions, users with cookies disabled can still log in and access their personal profiles.



Sessions without cookies may not work if your PHP settings in php.ini aren't configured properly on the server.

In order for sessions to work with cookies disabled, there needs to be another mechanism for passing the session ID among different pages. This mechanism involves appending the session ID to the URL of each page, which takes place automatically if the session.use_trans_id setting is set to 1 (true) in the php.ini file on the server. If you don't have the ability to alter this file on your web server, you'll have to manually append the session ID to the URL of session pages if cookies are disabled with code like this:

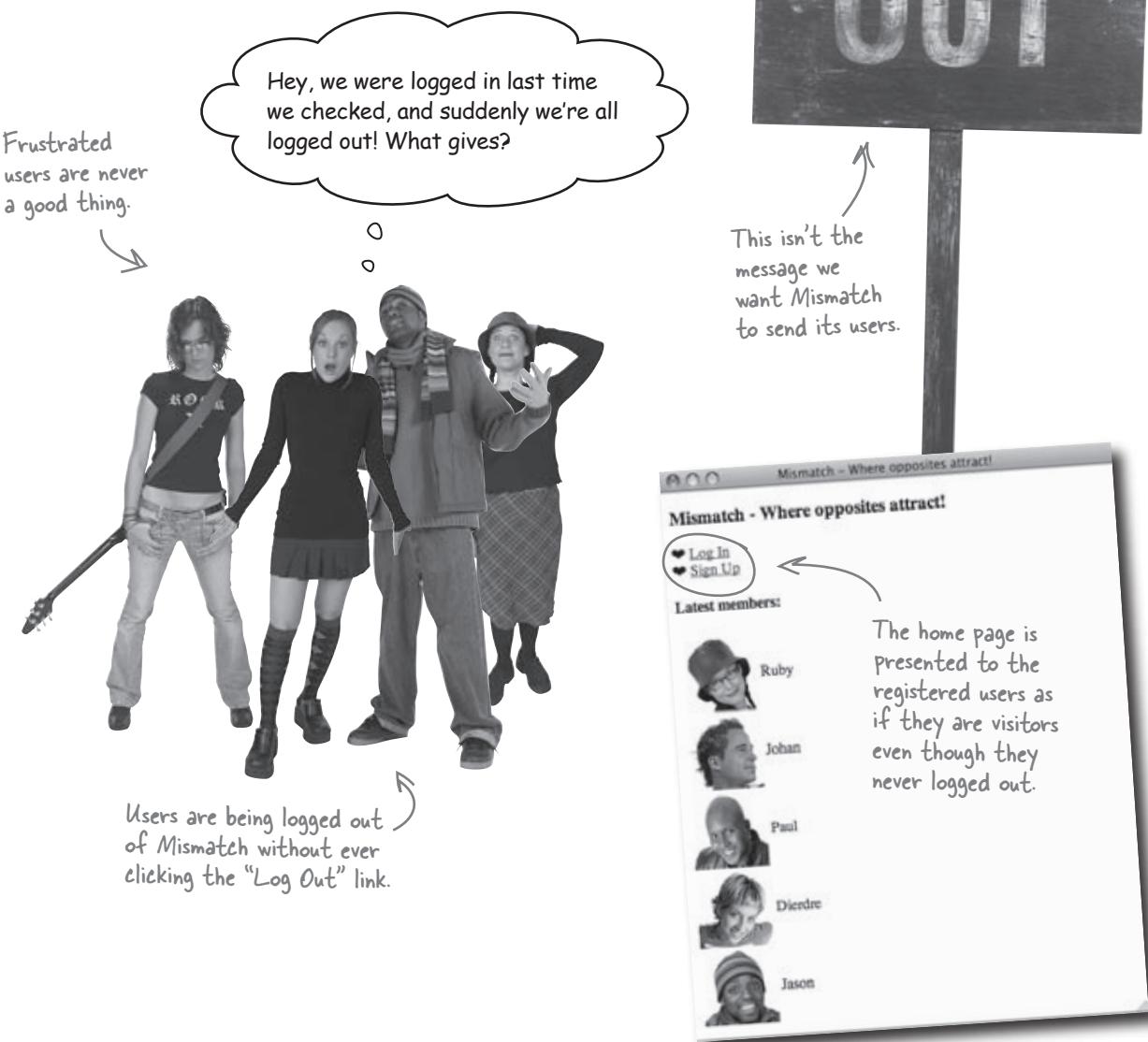
```
<a href="viewprofile.php?<?php echo SID; ?>">view your profile</a>
```

The \$ID superglobal holds the session ID, which is being passed along through the URL so that the View Profile page knows about the session.

why the automatic logout?

Users aren't feeling welcome

Despite serving as a nice little improvement over cookies, something about the new session-powered Mismatch application isn't quite right. Several users have reported getting logged out of the application despite never clicking the "Log Out" link. The application doesn't exactly feel personal anymore... this is a big problem.



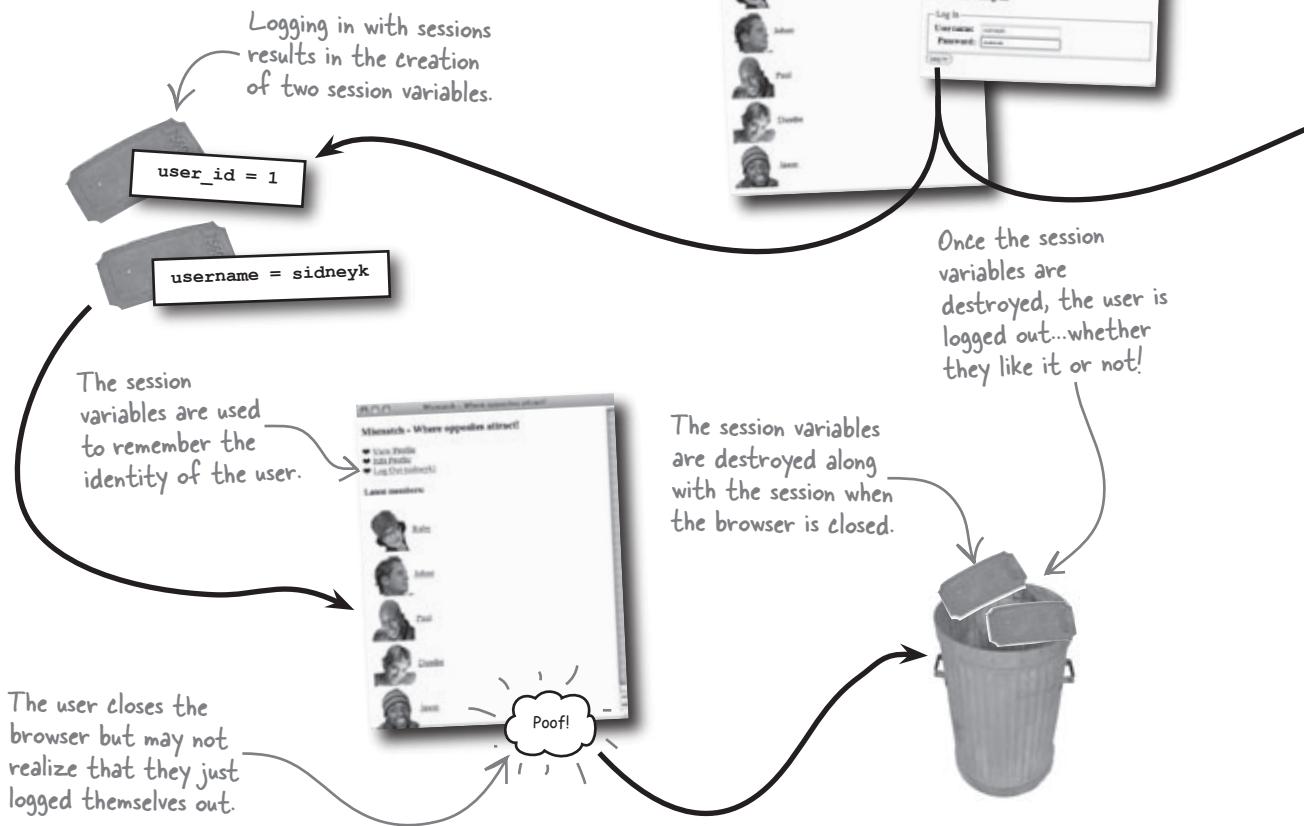


What do you think is causing users to be automatically logged out of Mismatch? Is it something they've done inadvertently?

Sessions are short-lived...

The problem with the automatic log-outs in Mismatch has to do with the limited lifespan of sessions. If you recall, sessions only last as long as the current browser instance, meaning that all session variables are killed when the user closes the browser application. In other words, closing the browser results in a user being logged out whether they like it or not. This is not only inconvenient, but it's also a bit confusing because we already have a log-out feature. Users assume they aren't logged out unless they've clicked the Log Out link.

Whether sessions or cookies are used, logging in is what sets the persistent wheels in motion.



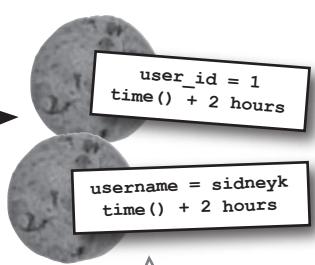
Even though you can destroy a session when you're finished with it, you can't prolong it beyond a browser instance. So sessions are more of a short-term storage solution than cookies, since cookies have an expiration date that can be set hours, days, months, or even years into the future. Does that mean sessions are inferior to cookies? No, not at all. But it does mean that sessions present a problem if you're trying to remember information beyond a single browser instance... such as log-in data!

Session variables are destroyed when the user ends a session by closing the browser.

... but cookies can last forever!

Unlike session variables, the lifespan of a cookie isn't tied to a browser instance, so cookies can live on and on, at least until their expiration date arrives. Problem is, users have the ability to destroy all of the cookies stored on their machine with a simple browser setting, so don't get too infatuated with the permanence of cookies—they're still ultimately only intended to store temporary data.

Maybe not forever,
but plenty long enough
to outlast a session.



Similar to sessions,
cookies are created
at log-in.

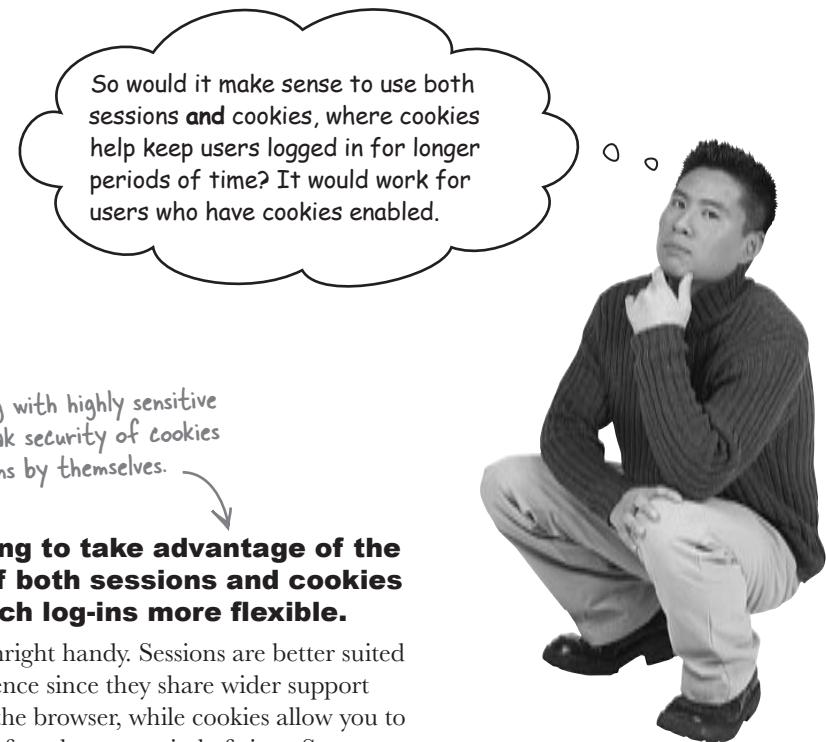
Cookies are only
destroyed when
they expire.



The lifespan of a cookie
is determined by its
expiration date/time.

**Cookies are destroyed
when they expire, giving
them a longer lifespan
than session variables.**





So would it make sense to use both sessions **and** cookies, where cookies help keep users logged in for longer periods of time? It would work for users who have cookies enabled.

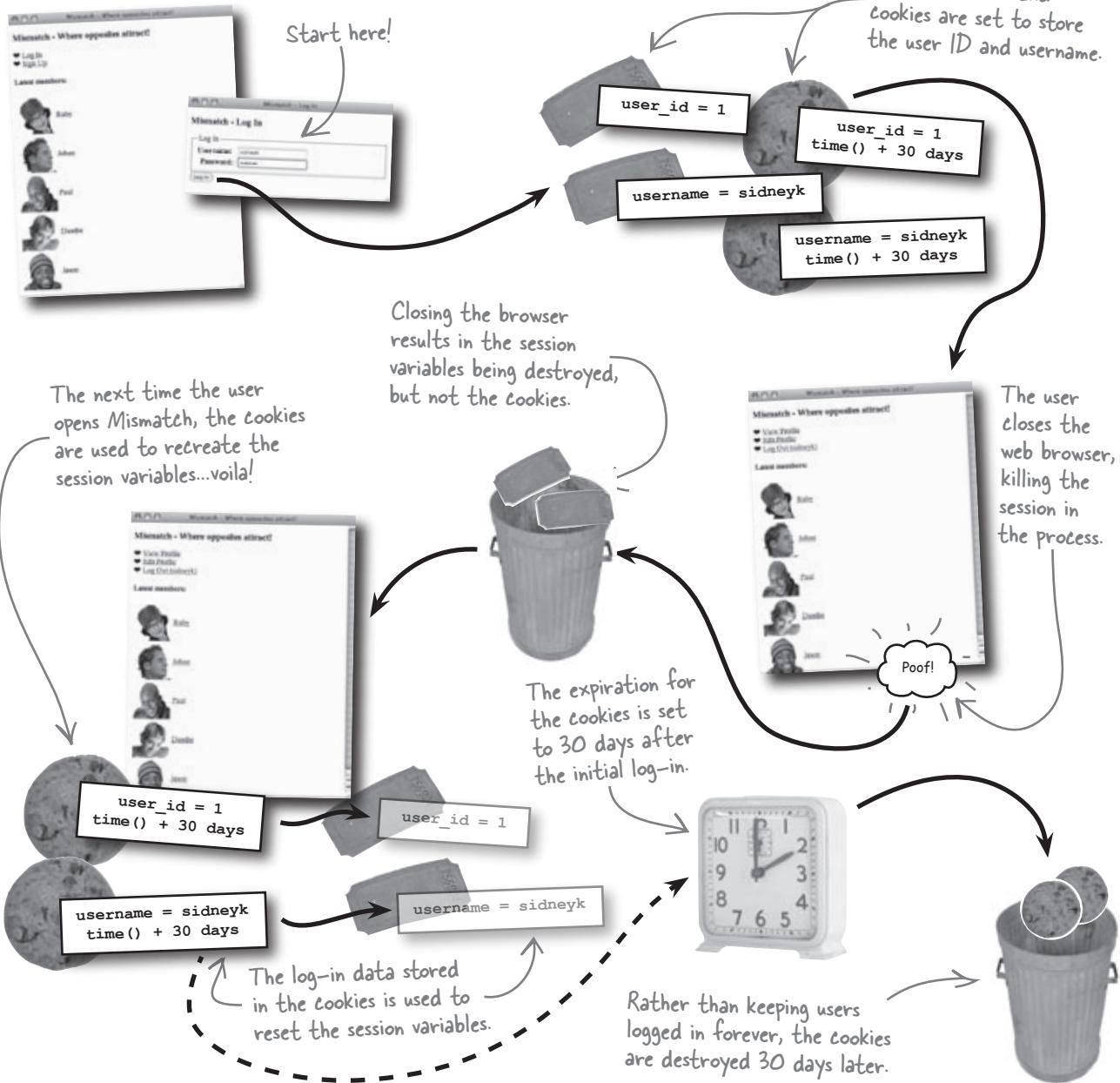
As long as you're not dealing with highly sensitive data, in which case, the weak security of cookies would argue for using sessions by themselves.

Yes, it's not wrong to take advantage of the unique assets of both sessions and cookies to make Mismatch log-ins more flexible.

In fact, it can be downright handy. Sessions are better suited for short-term persistence since they share wider support and aren't limited by the browser, while cookies allow you to remember log-in data for a longer period of time. Sure, not everyone will be able to benefit from the cookie improvement, but enough people will that it matters. Any time you can improve the user experience of a significant portion of your user base without detracting from others, it's a win.

Sessions + Cookies = Superior log-in persistence

For the ultimate in log-in persistence, you have to get more creative and combine all of what you've learned in this chapter to take advantage of the benefits of both sessions and cookies. In doing so, you can restructure the Mismatch application so that it excels at both short-term and long-term user log-in persistence.



there are no
Dumb Questions

Q: So is short-term vs. long-term persistence the reason to choose between sessions and cookies?

A: No. This happened to be the strategy that helped guide the design of the Mismatch application, but every application is different, and there are other aspects of sessions and cookies that often must be weighed. For example, the data stored in a session is more secure than the data stored in a cookie. So even if cookies are enabled and a cookie is being used solely to keep track of the session ID, the actual data stored in the session is more secure than if it was being stored directly in a cookie. The reason is because session data is stored on the server, making it very difficult for unprivileged users to access it. So if you're dealing with data that must be secure, sessions get the nod over cookies.

Q: What about the size of data? Does that play a role?

A: Yes. The size of the data matters as well. Sessions are capable of storing larger pieces of data than cookies, so that's another reason to lean toward sessions if you have the need to store data beyond a few simple text strings. Of course, a MySQL database is even better for storing large pieces of data, so make sure you don't get carried away even when working with sessions.

Q: So why would I choose a session or cookie over a MySQL database?

A: Convenience. It takes much more effort to store data in a database, and don't forget that databases are ideally suited for holding **permanent** data. Log-in data really isn't all that permanent in the grand scheme of things. That's where cookies and sessions enter the picture—they're better for data that you need to remember for a little while and then throw away.



PHP Magnets

The Mismatch application has been redesigned to use both sessions and cookies for the ultimate in user log-in persistence. Problem is, some of the code is missing. Use the session and cookie magnets to add back the missing code.



```
...
if (mysqli_num_rows($data) == 1) {
    // The log-in is OK so set the user ID and username session vars (and cookies),
    // and redirect to the home page
    $row = mysqli_fetch_array($data);

    ['user_id'] = $row['user_id'];
.....
    ['username'] = $row['username'];

    setcookie('user_id', $row['user_id'], time() + (60 * 60 * 24 * 30)); // expires in 30 days
    setcookie('username', $row['username'], time() + (60 * 60 * 24 * 30)); // expires in 30 days
    $home_url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
    header('Location: ' . $home_url);
}
...
```



login.php

```
<?php
// If the user is logged in, delete the session vars to log them out
session_start();

if (isset(.....['user_id'])) {

    // Delete the session vars by clearing the $_SESSION array
    ..... = array();

    // Delete the session cookie by setting its expiration to an hour ago (3600)
    if (isset(.....[session_name()])) {

        setcookie(session_name(), '', time() - 3600);
    }

    // Destroy the session
    session_destroy();
}

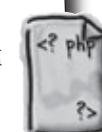
// Delete the user ID and username cookies by setting their expirations to an hour ago (3600)
setcookie('user_id', '', time() - 3600);
setcookie('username', '', time() - 3600);
...
```



logout.php

```
<?php
session_start();

// If the session vars aren't set, try to set them with a cookie
if (!isset(.....['user_id'])) {
    if (isset(.....['user_id']) && isset(.....['username'])) {
        ['user_id'] = .....['user_id'];
        ['username'] = .....['username'];
    }
}
?>
...
```



index.php

[here ▶](#)

411





PHP Magnets Solution

The Mismatch application has been redesigned to use both sessions and cookies for the ultimate in user log-in persistence. Problem is, some of the code is missing. Use the session and cookie magnets to add back the missing code.

```
...
if (mysqli_num_rows($data) == 1) {
    // The log-in is OK so set the user ID and username session vars (and cookies),
    // and redirect to the home page
    $row = mysqli_fetch_array($data);

    $_SESSION ['user_id'] = $row['user_id'];
    $_SESSION ['username'] = $row['username'];

    setcookie('user_id', $row['user_id'], time() + (60 * 60 * 24 * 30)); // expires in 30 days
    setcookie('username', $row['username'], time() + (60 * 60 * 24 * 30)); // expires in 30 days
    $home_url = 'http://'. $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
    header('Location: ' . $home_url);
}

...

```

The new cookies are set in addition to the session variables.



login.php

```
<?php
// If the user is logged in, delete the session vars to log them out
session_start();
```

```
if (isset($_SESSION ['user_id'])) {

    // Delete the session vars by clearing the $_SESSION array
    ... $_SESSION = array();

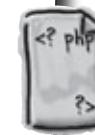
    // Delete the session cookie by setting its expiration to an hour ago (3600)
    if (isset($_COOKIE [session_name()])) {
        setcookie(session_name(), '', time() - 3600);
    }

    // Destroy the session
    session_destroy();
}

// Delete the user ID and username cookies by setting their expirations to an hour ago (3600)
setcookie('user_id', '', time() - 3600);
setcookie('username', '', time() - 3600);
...

```

Logging out now requires deleting both the session cookie and the new log-in cookies.



logout.php

```
<?php
session_start();

// If the session vars aren't set, try to set them with a cookie
```

```
if (!isset($_SESSION ['user_id'])) {
    if (isset($_COOKIE ['user_id']) && isset($_COOKIE ['username'])) {
        $_SESSION ['user_id'] = $_COOKIE ['user_id'];
        $_SESSION ['username'] = $_COOKIE ['username'];
    }
}
?>
...

```

Set the session variables using the cookies.

If the user isn't logged in via the session, check to see if the cookies are set.



index.php

This same cookie/session code must also go in editprofile.php and viewprofile.php.

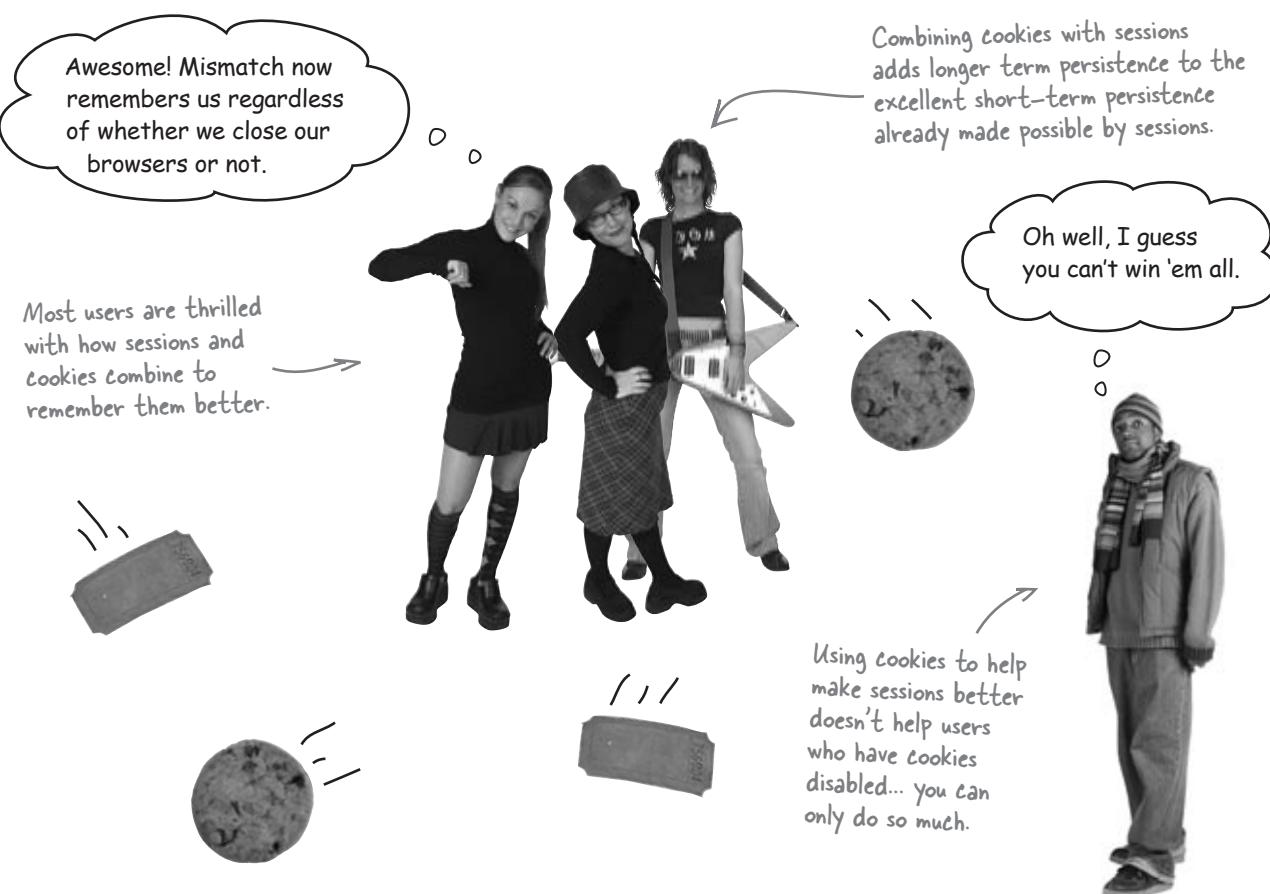


Test Drive

Change Mismatch to use both sessions and cookies.

Modify the Mismatch scripts so that they use both sessions and cookies to support log-in persistence (or download the scripts from the Head First Labs site at www.headfirstlabs.com/books/hfphp). This requires changes to the `index.php`, `login.php`, `logout.php`, `editprofile.php`, and `viewprofile.php` scripts.

Upload the scripts to your web server, and then open the main Mismatch page (`index.php`) in a web browser. Try logging in and then closing the web browser, which will cause the session variables to get destroyed. Re-open the main page and check to see if you're still logged in—cookies make this possible since they persist beyond a given browser session.





Your PHP & MySQL Toolbox

You've covered quite a bit of new territory in building a user management system as part of the Mismatch application. Let's recap some of the highlights.

`setcookie()`

This built-in PHP function is used to set a cookie on the browser, including an optional expiration date, after which the cookie is destroyed. If no expiration is provided, the cookie is deleted when the browser is closed.

`$_COOKIE`

This built-in PHP superglobal is used to access cookie data. It is an array, and each cookie is stored as an entry in the array. So accessing a cookie value involves specifying the name of the cookie as the array index.

`session_start()`

This built-in PHP function starts a new session or re-starts a pre-existing session. You must call this function prior to accessing any session variables.

`SHA(value)`

This MySQL function encrypts a piece of text, resulting in a string of 40 hexadecimal characters. This function provides a great way to encrypt data that needs to remain unrecognizable within the database. It is a one-way encryption, however, meaning that there is no "decrypt" function.

`session_destroy()`

This built-in PHP function closes a session, and should be called when you're finished with a particular session. This function does not destroy session variables; however, so it's important to manually clean those up by clearing out the `$_SESSION` superglobal.

`$_SESSION`

This built-in PHP superglobal is used to access session data. It is an array, and each session variable is stored as an entry in the array. So accessing the value of a session variable involves specifying the name of the variable as the array index.



Several pieces of code from the Mismatch application have been pulled out, and we can't remember what they do. Draw lines connecting each piece of code with what it does.

PHP/MySQL Code**Description**`empty($_COOKIE['user_id'])`

Use a session variable to determine if a user is logged in or not.

`setcookie(session_name(), '', time() - 3600);`

Use a cookie to determine if a user is logged in or not.

`SHA('$user_password')`

Destroy a session cookie by setting its expiration to an hour in the past.

`session_destroy()`

Encrypt a user's password into an unrecognizable format.

`setcookie('user_id', $row['user_id'])`

Store a user's unique ID in a cookie.

`$_SESSION = array()`

Start a new session.

`session_start()`

Close the current session.

`unset($_SESSION['user_id'])`

Destroy all session variables.

WHO DOES WHAT?

Several pieces of code from the Mismatch application have been pulled out, and we can't remember what they do. Draw lines connecting each piece of code with what it does.

PHP/MySQL Code

PHP/MySQL Code	Description
<code>empty(\$_COOKIE['user_id'])</code>	Use a session variable to determine if a user is logged in or not.
<code>setcookie(session_name(), '', time() - 3600);</code>	Use a cookie to determine if a user is logged in or not.
<code>SHA('\$user_password')</code>	Encrypt a user's password into an unrecognizable format.
<code>session_destroy()</code>	Destroy a session cookie by setting its expiration to an hour in the past.
<code>setcookie('user_id', \$row['user_id'])</code>	Store a user's unique ID in a cookie.
<code>\$_SESSION = array()</code>	Start a new session.
<code>session_start()</code>	Close the current session.
<code>isset(\$_SESSION['user_id'])</code>	Destroy all session variables.

7 ½ eliminate duplicate code

Sharing is caring

It's really quite simple, darling. By sharing one umbrella, we eliminate the need for two umbrellas, we both still stay dry... and you get to latch on to one handsome fella.

Handsome and...
Your shared umbrella...
theory is pure



Umbrellas aren't the only thing that can be shared. In any web application you're bound to run into situations where *the same code is duplicated* in more than one place. Not only is this wasteful, but it leads to **maintenance headache** since you will inevitably have to make changes, and these changes will have to be carried out in multiple places. The solution is to **eliminate duplicate code by sharing it**. In other words, you stick the duplicate code in one place, and then just reference that single copy wherever you need it. Eliminating duplicate code results in applications that are **more efficient, easier to maintain, and ultimately more robust**.

locate the duplicate code



The Mismatch application has evolved since you last saw it, with improved navigation and a more consistent look and feel. But these improvements have come at a cost. Just by looking at the pages themselves, see if you can figure out what parts of the application represent a duplicate code problem. Circle and annotate these application pages. Write down anything not visible that you think might also have code duplication issues.

Mismatch - Where opposites attract!

Mismatch - Where opposites attract!

Home • View Profile • Edit Profile • Log Out (jnettles)

Latest members:

	Ruby
	Johan
	Paul
	Dierdre
	Jason

Copyright ©2009 Mismatch Enterprises, Inc.

<? PHP
?>

index.php

Mismatch - View Profile

Mismatch - View Profile

[Home](#) [View Profile](#) [Edit Profile](#) [Log Out \(jnetties\)](#)

Username: jnetties
 First name: Johan
 Last name: Netties
 Gender: Male
 Birthdate: 1981-11-03
 Location: Athens, GA

Picture:

Would you like to [edit your profile?](#)

Copyright © 2009 Mismatch Enterprises, Inc.

viewprofile.php

Mismatch - Edit Profile

Mismatch - Edit Profile

[Home](#) [View Profile](#) [Edit Profile](#)

Personal Information

First name:	johan	Last name:	Netties
Gender:	Male		
Birthdate:	1981-11-03		
City:	Athens		
State:	GA		
Picture:	<input type="file"/> <input checked="" type="checkbox"/> johanned		
Save Profile			

Copyright © 2009 Mismatch

mismatch's duplicate code

The Mismatch application has evolved since you last saw it, with improved navigation and a more consistent look and feel. But these improvements have come at a cost. Just by looking at the pages themselves, see if you can figure out what parts of the application represent a duplicate code problem. Circle and annotate these application pages to point down anything not visible that you think might also have code duplication issues.

index.php

The "Mismatch" title appears on every page, with only the detailed page title varying from page to page.

Mismatch - View Profile

Username: johnt
First name: John
Last name: Nettles
Gender: Male
Birthdate: 1981-11-03
Location: Athens, GA
Picture:

Would you like to edit your profile?

Copyright © 2009 Mismatch Enterprises, Inc.

The navigation menu is identical across all three pages.

The page footer that holds the copyright information for the application is the same everywhere.

Mismatch - Edit Profile

Personal Information

First name:	John	Last name:	Nettles
Birthdate:	1981-11-03	Gender:	Male
City:	Athens	State:	GA
Picture:	<input type="file"/> johnt.jpg		

Save Profile

Copyright © 2009 Mismatch Enterprises, Inc.

```
<?php
session_start();

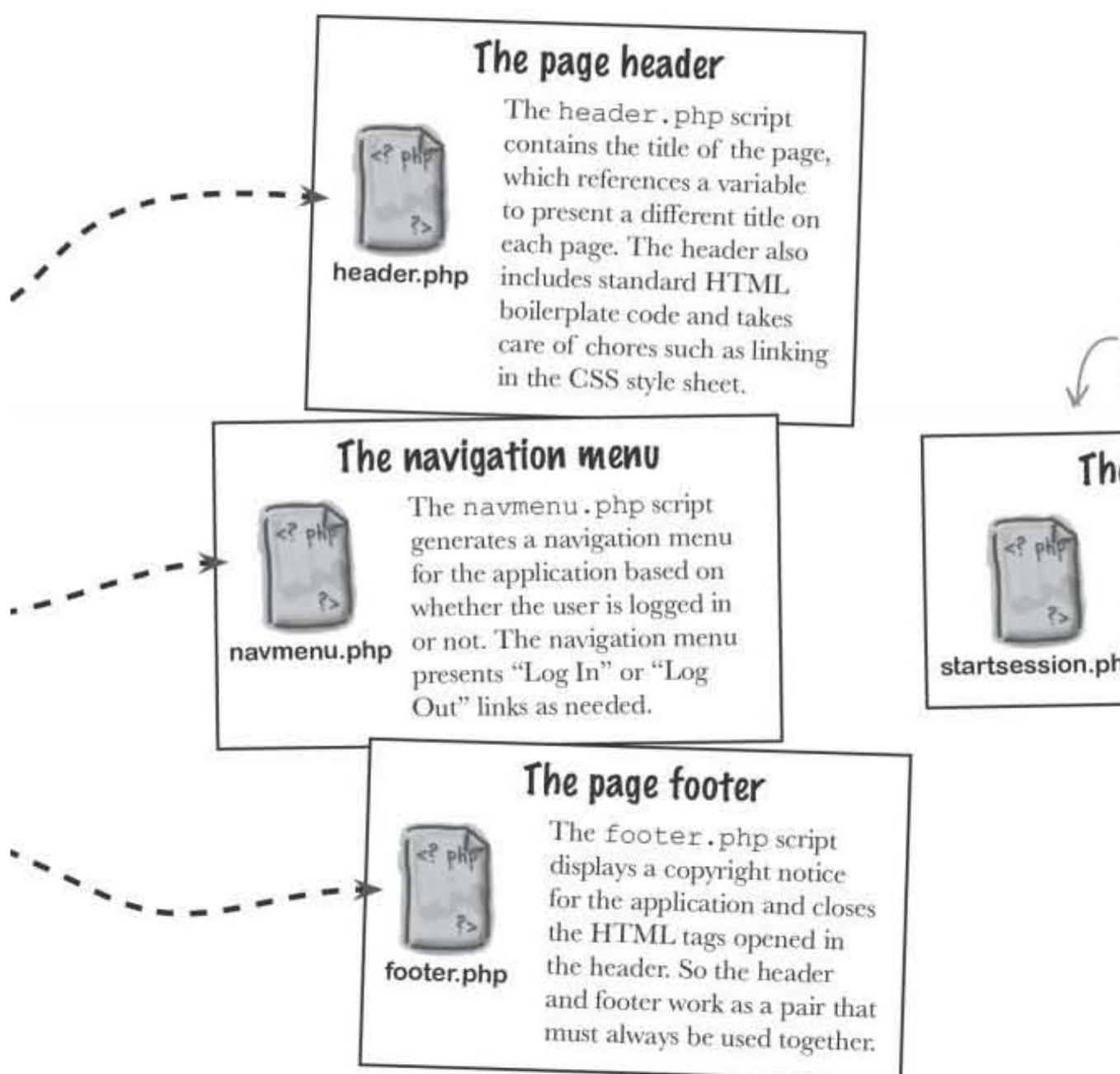
// If the session vars aren't set, try to set them with a cookie
if (!isset($_SESSION['user_id'])) {
    if (isset($_COOKIE['user_id'])) {
        $_SESSION['user_id'] = $_COOKIE['username'];
        $_SESSION['username'] = $_COOKIE['username'];
    }
}
```

All of the pages that rely on a user log-in require the exact same session start-up and log-in checking code.

Mismatch is in pieces

So the Mismatch application has some common elements that are duplicated in the main script files at the moment. Why is this a big deal? Because it makes the application difficult to maintain. What happens if you decide to add a new page that requires a new menu item? You have to go through and change the menu code in *every script file* to show the new menu item. The same thing applies to the copyright notice.

The solution to the problem is to only store any given piece of information once. Then if that code ever needs to change, you only change it in one place. With that in mind, it's possible to rethink the organization of Mismatch in terms of reusable script components.



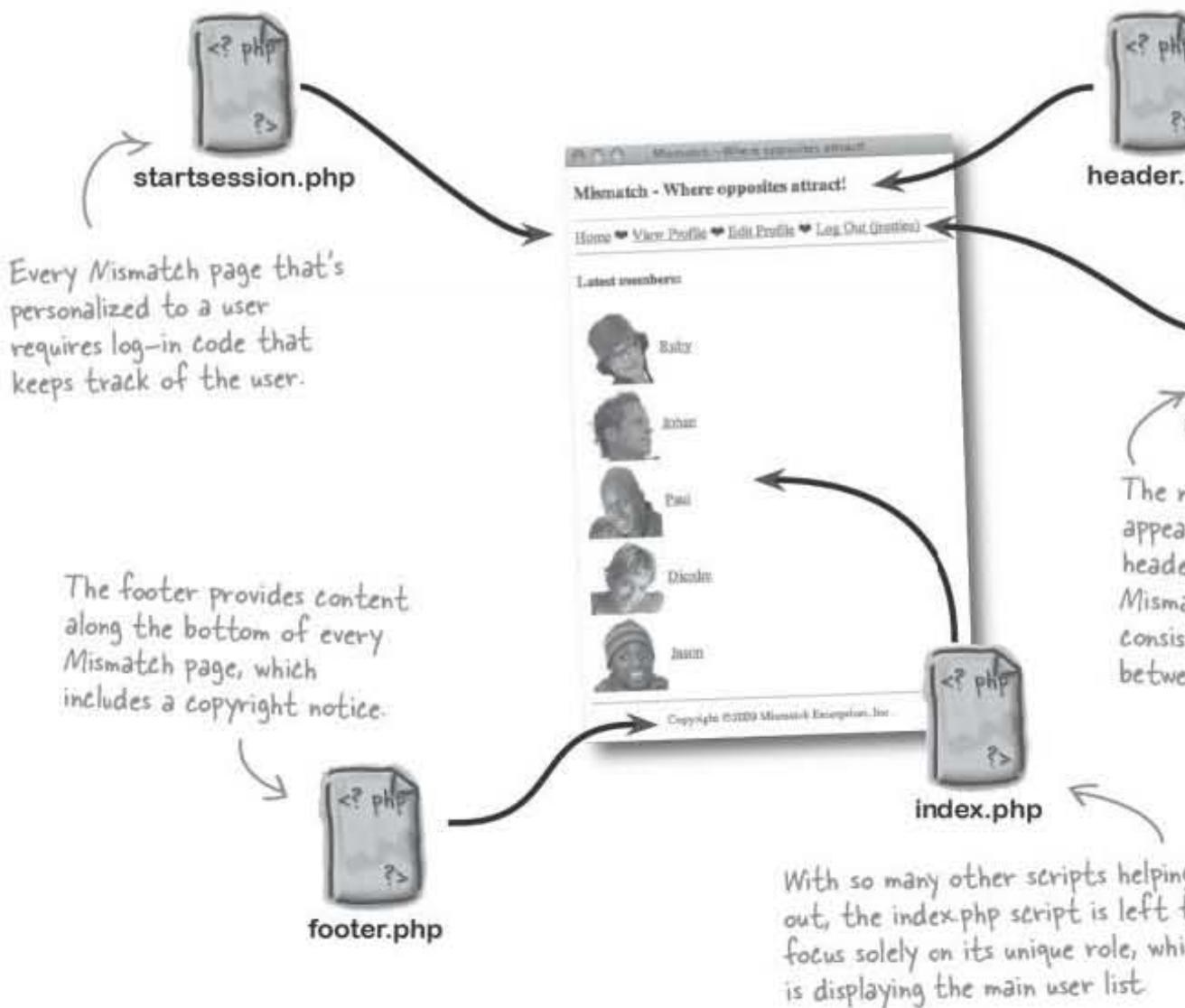
mismatch needs a template

Rebuilding Mismatch from a template

OK, so we break apart Mismatch into multiple scripts, but how do we put them back together? You're already familiar with how include files work, and those are part of the solution. But you have to think larger than include files... you have to think in terms of **templates**, which allow you to build a single page as a combination of multiple include files. A template is like a blueprint for a page within an application where everything but what is truly unique to that page comes from include files.

The template version of Mismatch involves breaking apart common code into scripts that each play a very specific role, some responsible for generating visual HTML code, some not. The idea is to distill as much common functionality as possible into template include files, and then only leave code in each application page that is completely unique to that page.

Templates
PHP applica
built out o
script com



there are no
Dumb Questions

Q: What is a template exactly? Isn't it just a bunch of include files?

A: Yes. A template is a collection of include files, but it's a collection designed specifically to separate an application into functional components. The goal is to reduce a page down to what is truly unique about that page, and only that page. So headers, footers, navigation menus, and any other application pieces and parts that are the same or similar among more than one page are ideal for inclusion in an application template. The end result is that you place template code in PHP include files that are referenced by other scripts that need them.

You can think of a template as a group of include files that go a step or two beyond just reducing duplicate code—they actually help organize the functionality of an application. Mismatch is a fairly simple example of how to employ templates—larger, more complex PHP applications often employ very sophisticated template systems.

Q: Doesn't template code have to be exactly the same in order to be shared across multiple scripts?

A: No. It's perfectly acceptable for template code to just be similar, not exact. The reason is because you can use variables to allow for some degree of customization as a template is applied to different pages. The page title in Mismatch is a perfect example of this. The page header template is similar in every page in that it has a title that always begins with "Mismatch - ." But the specific title is different, which is why a variable is needed to provide a means of varying the title slightly among different pages.

mismatch—now using templates!

Rebuild Mismatch with templates

The design work involved in breaking an application into template scripts is usually worth the effort. You end up with a collection of tightly focused, bite-size scripts, as well as dramatically simplified code in the main application script files that are now dependent on the template scripts.

```

<?php
    session_start();

    // If the session vars aren't set, try to set them with a cookie
    if (!isset($_SESSION['user_id'])) {
        if (isset($_COOKIE['user_id']) && isset($_COOKIE['username'])) {
            $_SESSION['user_id'] = $_COOKIE['user_id'];
            $_SESSION['username'] = $_COOKIE['username'];
        }
    }
?>

```

Start the session.


```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
    </head>
    <body>
        <?php
            echo '<title>Mismatch - ' . $page_title . '</title>';
        ?>

        <link rel="stylesheet" type="text/css" href="style.css"/>
    </body>
</html>

```

Start the official HTML code with a DOCTYPE and an <html> tag.


```

<?php
    echo '<h3>Mismatch - ' . $page_title . '</h3>';
?>

```

Build a custom page title using the \$page_title variable, which is provided by the script including this file.


```

<?php
    // Generate the navigation menu
    echo '<hr />';
    if (isset($_SESSION['username'])) {
        echo '<a href="index.php">Home</a> &#10084; ';
        echo '<a href="viewprofile.php">View Profile</a> &#10084; ';
        echo '<a href="editprofile.php">Edit Profile</a> &#10084; ';
        echo '<a href="logout.php">Log Out (' . $_SESSION['username'] . ')</a>';
    } else {
        echo '<a href="login.php">Log In</a> &#10084; ';
        echo '<a href="signup.php">Sign Up</a>';
    }
    echo '<hr />';
?>

```

See if the user is logged in, and then generate the appropriate navigation menu.


```

<hr />
<p class="footer">Copyright &copy; 2008 Mismat</p>

```

navmenu.php

The `startsession.php` script must be included first so that the session is started and the remainder of the script has access to session data.

The `$page_title` variable determines the title of the page that is displayed within the page header.

```

<?php
// Start the session
require_once('startsession.php');

// Insert the page header
$page_title = 'Where opposites attract!';
require_once('header.php');

require_once('appvars.php');
require_once('connectvars.php');

// Show the navigation menu
require_once('navmenu.php');

// Connect to the database
$dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

// Retrieve the user data from MySQL
$query = "SELECT user_id, first_name, picture FROM mismatch_user WHERE ";
    "ORDER BY join_date DESC LIMIT 5";
$data = mysqli_query($dbc, $query);

// Loop through the array of user data, formatting it as HTML
echo '<h4>Latest members:</h4>';
echo '<table>';
while ($row = mysqli_fetch_array($data)) {
    if (is_file(MM_UPLOADPATH . $row['picture']) && filesize(MM_UPLOADPATH . $row['picture']) > 0) {
        echo '<tr><td></td>';
    } else {
        echo '<tr><td></td>';
    }
    if (isset($_SESSION['user_id'])) {
        echo '<td><a href="viewprofile.php?user_id=' . $row['user_id'] . '">';
            '</a></td></tr>';
    } else {
        echo '<td>' . $row['first_name'] . '</td></tr>';
    }
}
echo '</table>';

mysqli_close($dbc);
?>
<?php
// Insert the page footer
require_once('footer.php');
?>
```

The connection variables and application variables are still included from separate script files like before.

The navigation menu is placed after the header but before the body of the page.

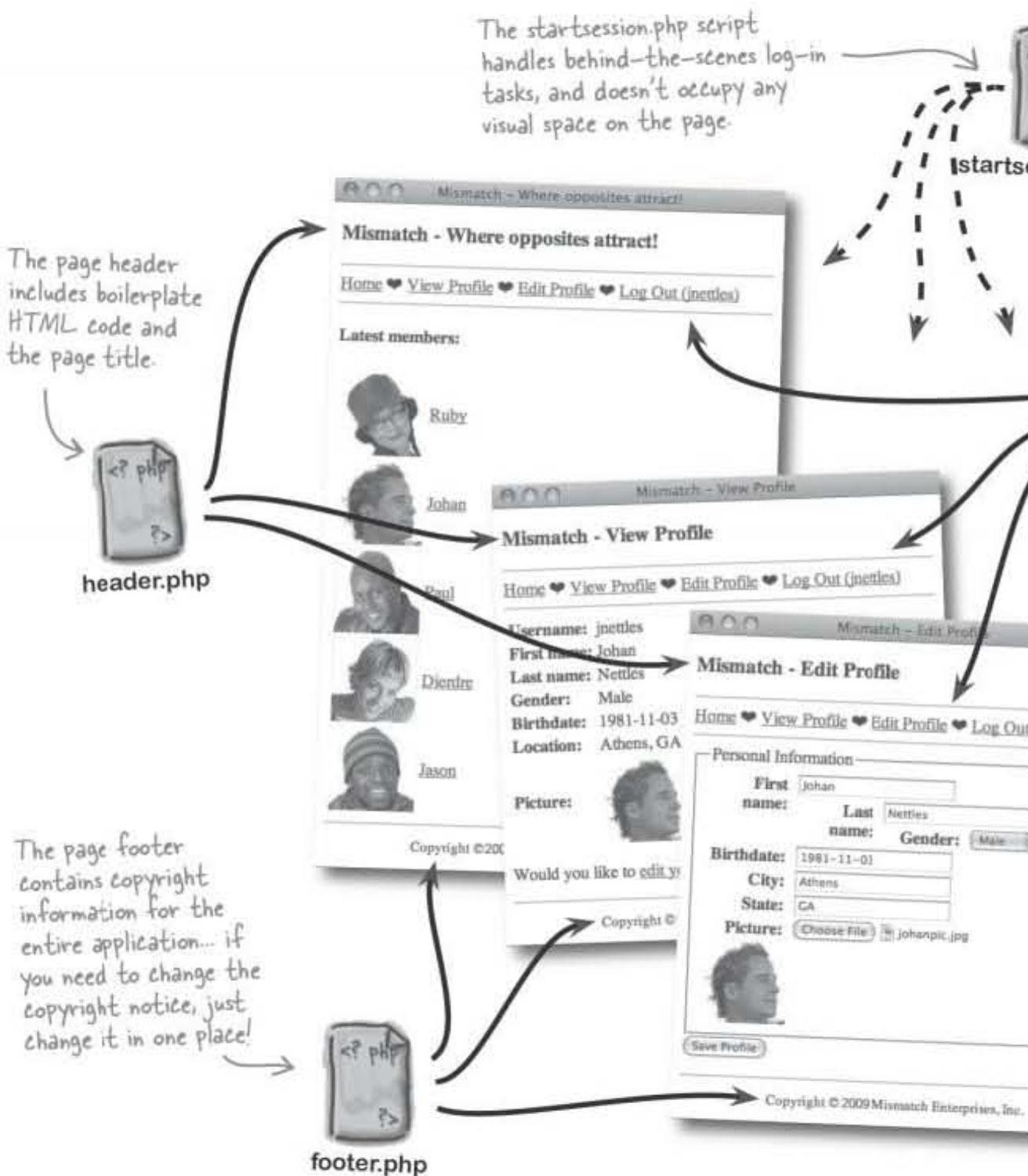
The footer finishes up the page, and must appear last since it closes up the HTML tags.

The true
the

a well-designed php app

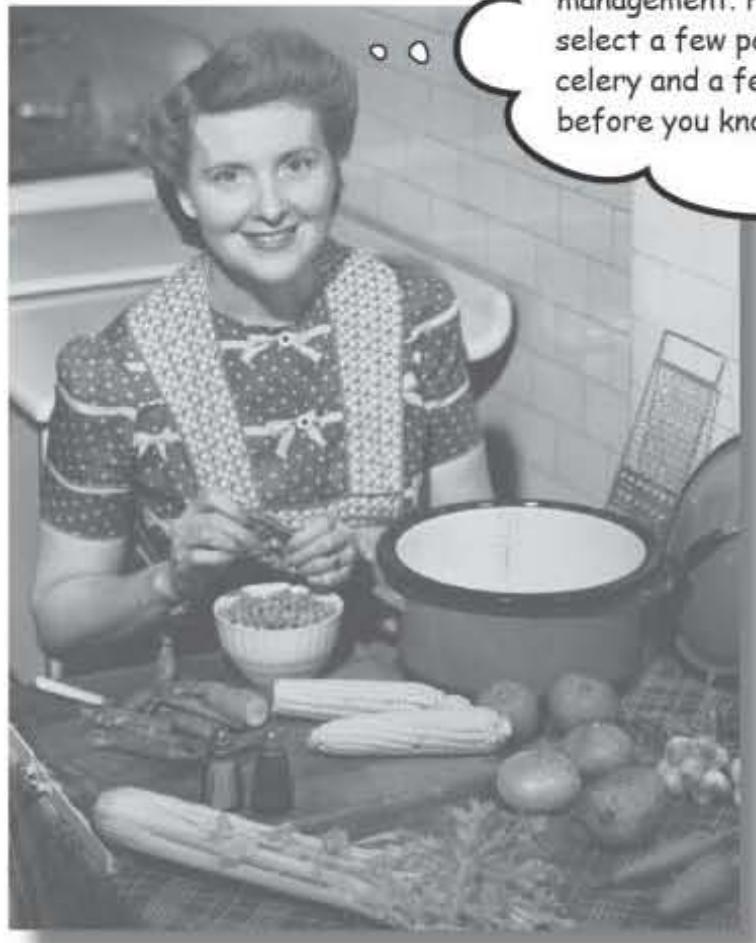
Mismatch is whole again... and better organized

While the thought of ripping apart the Mismatch application into tiny pieces might have been a bit unnerving, the end result is definitely worth the effort. The application's now spread across several new template (include) files, which offer much better organization and maximize the sharing of script code. If you need to change one of these pieces, just change one file and the effect cascades throughout the entire application... that's the power of templates!



8 control your data, control your world

Harvesting data



The way I see it, it's all about data management. First I sort the peas, then I select a few potatoes, join them with some celery and a few rows of corn kernels... before you know it there's a tasty stew!

There's nothing like a good fall data harvest. An abundance of information ready to be **examined, sorted, compared, combined**, and generally made to do whatever it is your killer web app needs it to do. Fulfilling? Yes. But like real harvesting, **taking control of data in a MySQL database** requires some hard work and a fair amount of expertise. Web users demand more than tired old wilted data that's dull and unengaging. They want data that enriches... data that fulfills... data that's relevant. So what are you waiting for? Fire up your MySQL tractor and get to work!

looking for a love-hate relationship

Making the perfect mismatch

The Mismatch application has a growing database of registered users but they're ready to see some results. We need to allow users to find their ideal opposite by comparing their loves and hates against other users and looking for mismatches. For every love mismatched against a hate, a couple is that much closer to being the perfect mismatch.

Sidney has yet to find her Mr. Right but she has a hunch he'll hate reality TV as much as she loves it.

I really hate horror movies. And Spam, blech! But I do love Barbara Streisand, and there's nothing better than a good hike...



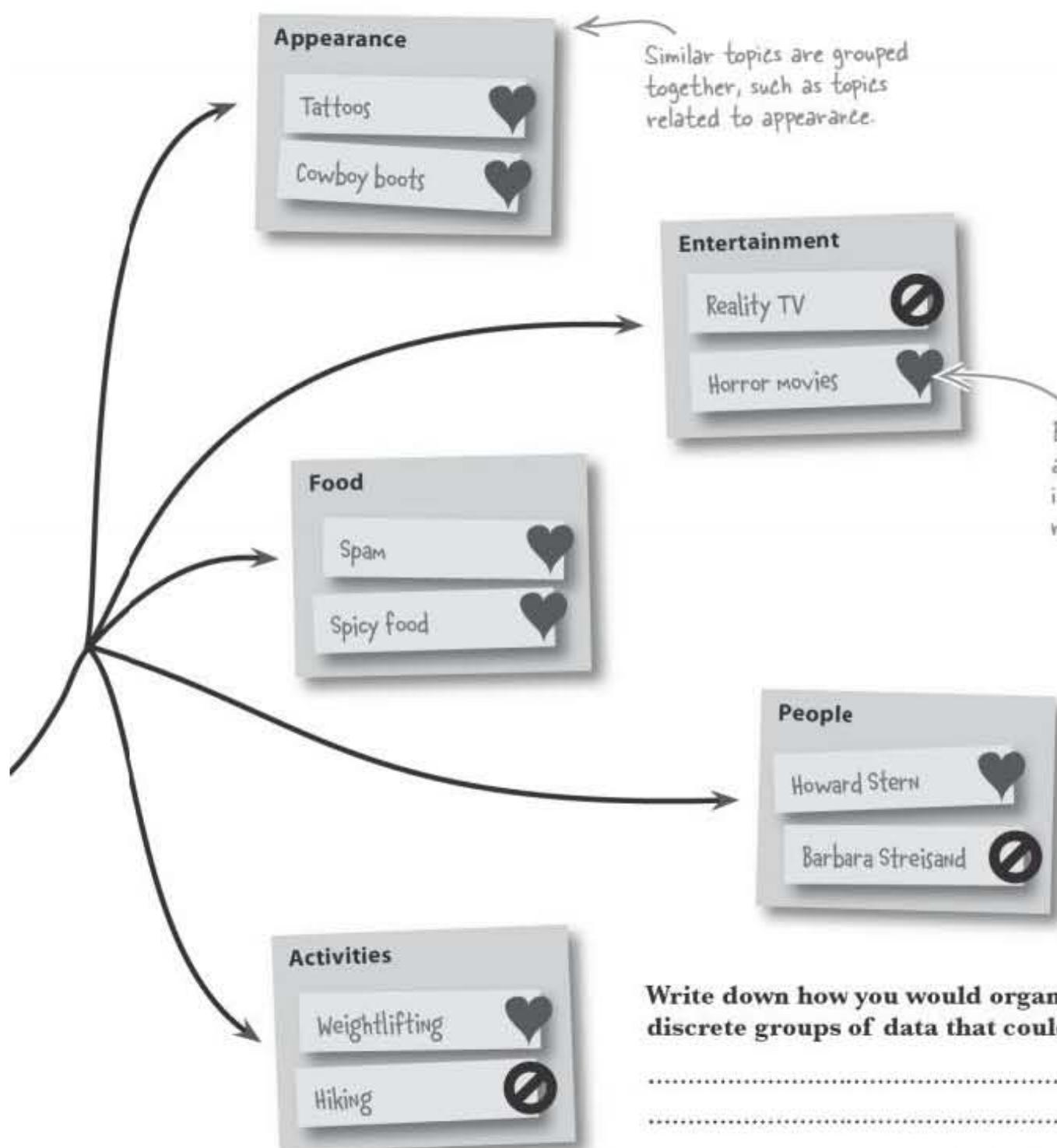
Nothing warms my heart like a good slasher flick coupled with a Spam sandwich. As long as Barbara Streisand doesn't show up in the movie hiking!



Sidney's list of loves and hates contrasts starkly with Johan's, making the couple quite an effective mismatch.

Mismatching is all about the data

In order to carry out Mismatches between users, we must first figure out how to organize the data that keeps up with what they love and hate. Knowing that it's going to be stored in a MySQL database isn't enough. We need to organize these love/hate topics so that they are more manageable, allowing users to respond to related topics, indicating whether they love or hate each one.



a data model for mismatch

Break down the Mismatch data

Coming up with a data model for an application such as Mismatch is an extremely important step, as it controls an awful lot about how the application is constructed. In the case of Mismatch, we can break down its data needs into three interrelated pieces of data.

Categories

Categories are used to help organize topics. Although they don't play a direct role in determining a mismatch, they will help make it easier for users to enter responses.

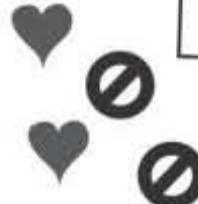
Entertainment

Activities

Categories are used to group together related Mismatch topics.

Responses

A user describes themselves for mismatching purposes by responding to topics. An individual response is just a love/hate answer to a topic.



Reality TV

Weightlifting

Horror movies

Hiking

Mismatch with resp...
as tattoos
each of v...
response

Responses are the individual hate answers to each topic, specific to each Mismatch user.

How exactly does this data lead to a mismatch between two users? We compare responses that users have made on each topic. For example, since Sidney and Johan have opposite responses to the topic "Horror movies," we have a successful mismatch on that particular topic. Figuring the best overall mismatch for a given user involves finding the user who has the most mismatched topics with them.

Sidney's dislike of horror movies leads to a mismatch.

Horror movies



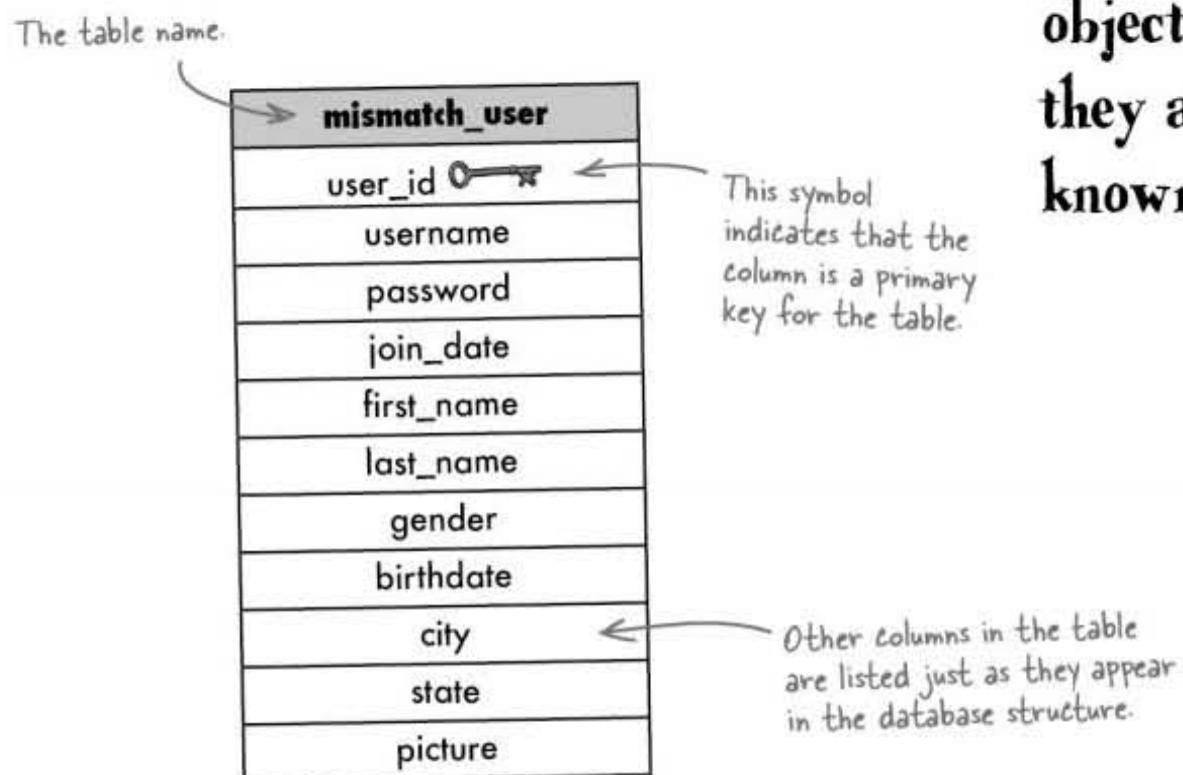
Hate 'em!

Horror movies

A mismatch!

Model a database with a schema

In order to translate the data requirements of the Mismatch application into an actual database design, we need a schema. A **schema** is a representation of all the structures, such as tables and columns, in your database, along with how they connect. Creating a visual depiction of your database can help you see how things connect when you're writing your queries, not to mention which specific columns are responsible for doing the connecting. As an example, let's take a look at the schema for the original Mismatch database from the previous chapter, which consists of only a single table, `mismatch_user`.



This way of looking at the structure of a table is a bit different than what you've seen up until now. Tables have normally been depicted with the column names across the top and the data below. That's a great way to look at individual tables and tables populated with data, but it's not very practical when we want to create a structural diagram of multiple tables and how they relate to one another. And Mismatch is already in need of multiple tables...

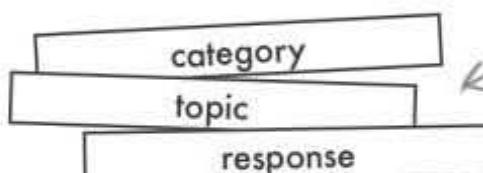
A descriptive
data (the
columns)
database,
any other
objects a
they all c
known as

Creating a
a table lets
the design
separate fr
that's insid

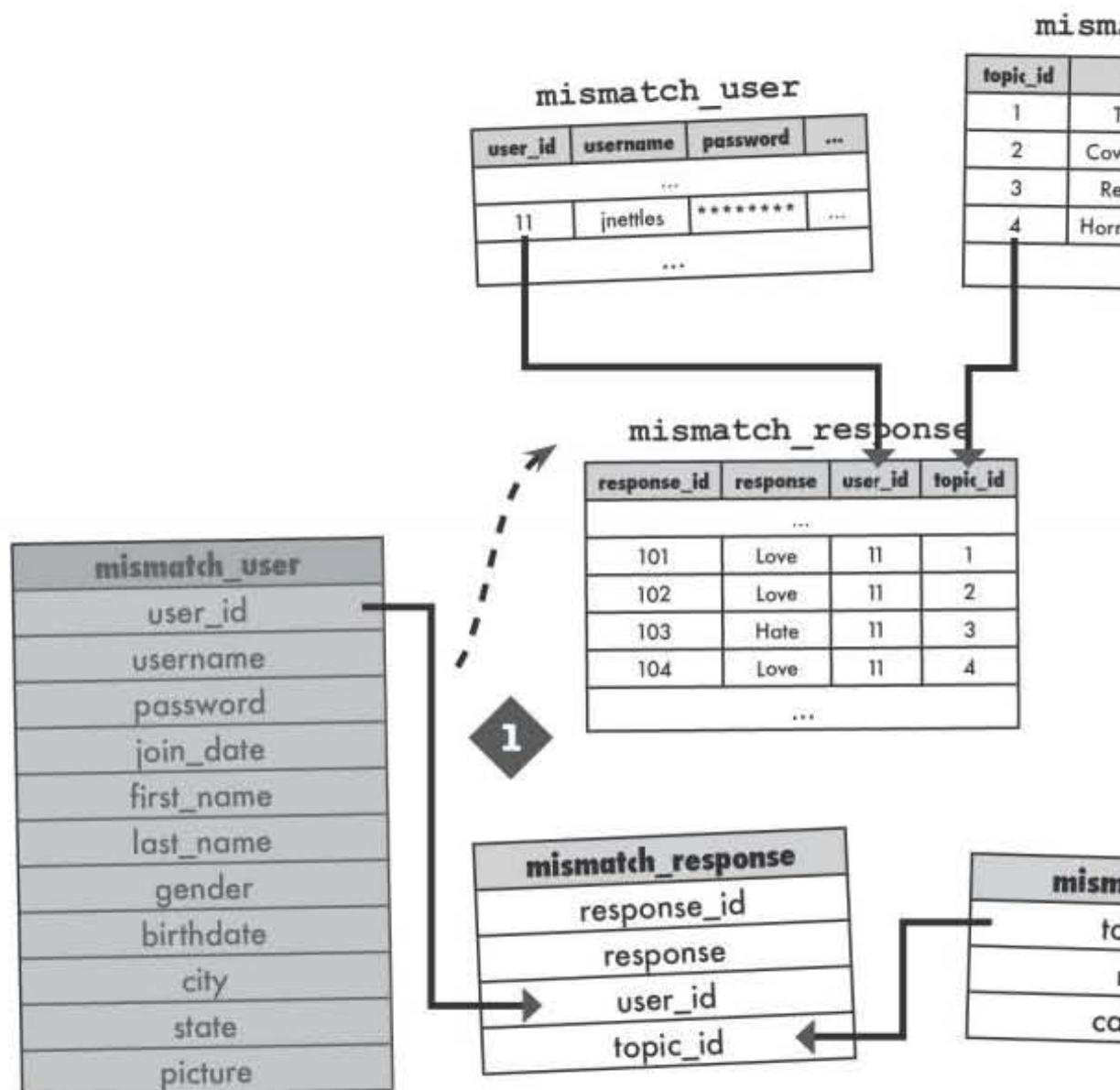
pick the best mismatch schema

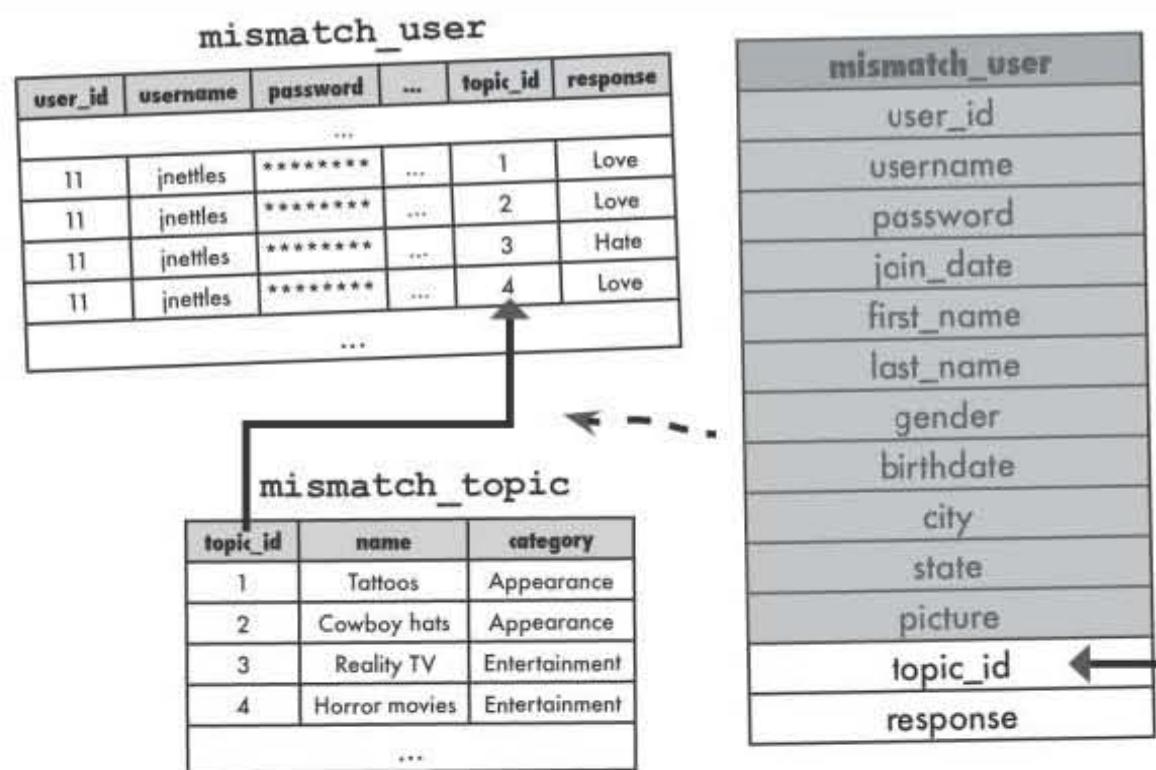
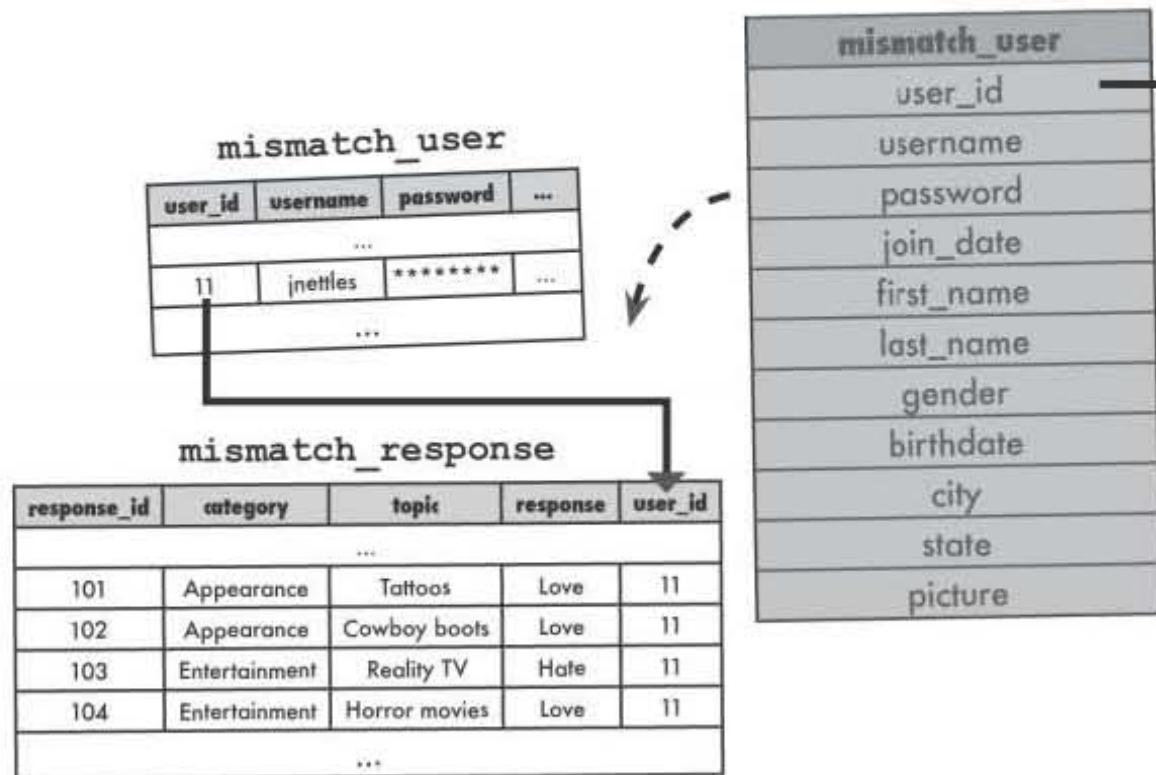


The Mismatch database is in need of storage for user responses to love/hate the topic names and their respective categories. Here are three different designs incorporating categories, topics, and responses into the Mismatch database. Please pick the best one that you think makes the most sense, and annotate why.



This is the new data introduced by the need to keep up with Mismatch users' loves and hates.





mismatch's best database schema

The Mismatch database is in need of storage for user responses to love/hate the topic names and their respective categories. Here are three different database schemas for incorporating categories, topics, and responses into the Mismatch database. Please choose one that you think makes the most sense, and annotate why.

First off, it's important to establish that the only new data involved in a user giving love/hate responses are the responses themselves – everything else in the database is fixed, at least from the user's perspective.

Who said simpler is always better? This database schema stores responses in their own table, separate from other data that isn't directly impacted by the responses. There's no duplication due to responses because users, categories, and topics are all outside of the mismatch_response table.

mismatch_user								
user_id	username	password
11	jnettles	*****
...

mismatch_user			
user_id	username	password	...
11	jnettles	*****	...
...

mismatch_response			
response_id	response	user_id	topic_id
101	Love	11	1
102	Love	11	2
103	Hate	11	3
104	Love	11	4
...

mismatch_response			
response_id	response	user_id	topic_id
101	Love	11	1
102	Love	11	2
103	Hate	11	3
104	Love	11	4
...

The original mismatch_user table remains unchanged.

mismat

topic_id	topic_name
1	T
2	Cow
3	Re
4	Horn
...	...

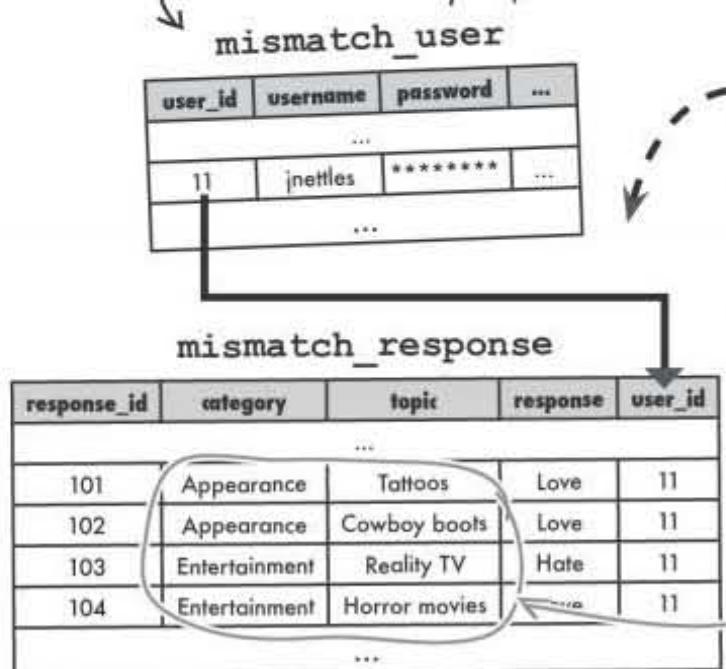
The ne
table s
and th
category

Ther
data
respo
very

mismat			
topic_id	topic_name
1	T
2	Cow
3	Re
4	Horn
...

The mismatch_response table links users and topics together via the user_id and topic_id columns.

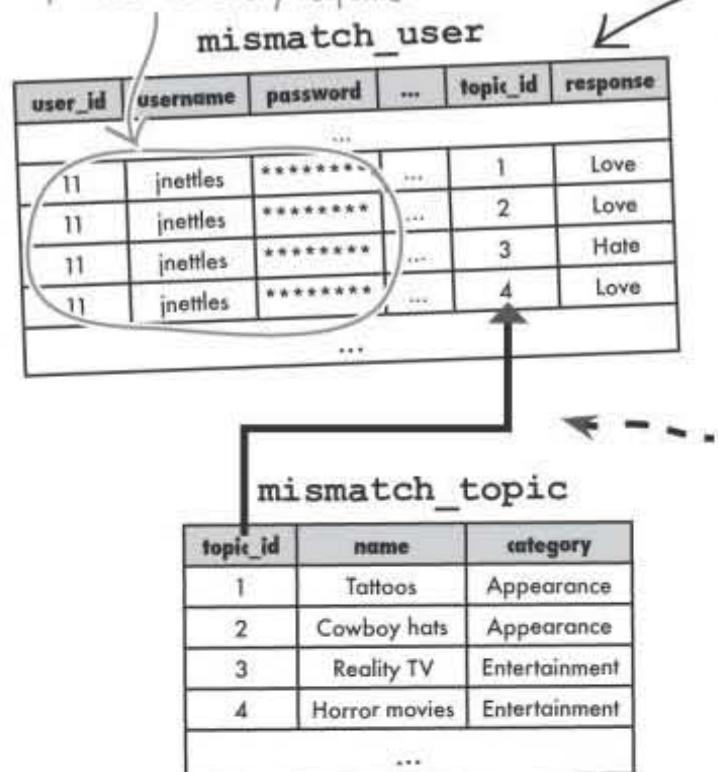
The responses aren't stored inside the user table, which is great. But there's a ton of duplicate data since the categories and topics are duplicated for each and every response.



mismatch_user
user_id
username
password
join_date
first_name
last_name
gender
birthdate
city
state
picture

Categories and topics are duplicated for each response, which is crazy wasteful.

The user data is horribly duplicated for every response.



mismatch_user
user_id
username
password
join_date
first_name
last_name
gender
birthdate
city
state
picture
topic_id
response

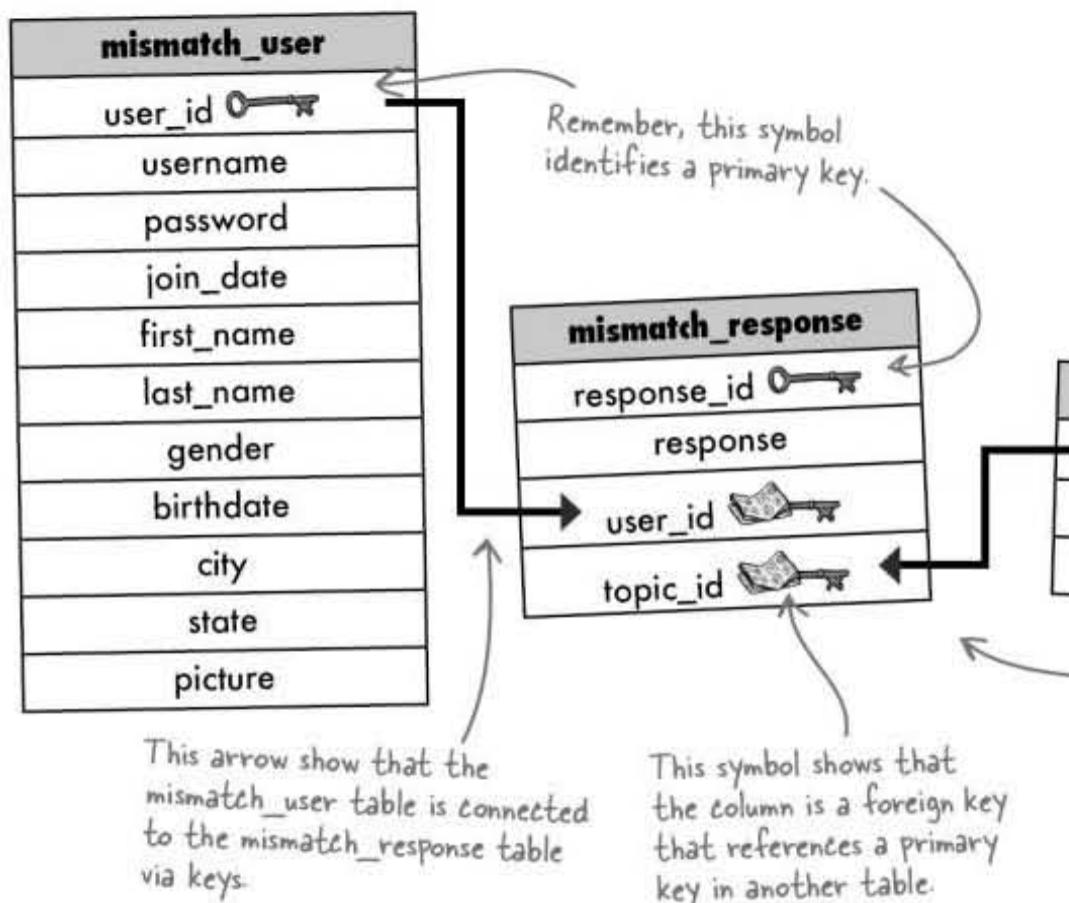
mismatch's schema uses foreign keys

Wire together multiple tables

Connecting tables together to form a cohesive system of data involves the use of **keys**. We've used **primary keys** to provide a unique identifier for data within a table, but we now need **foreign keys** to link a row in one table to a row in another table. A foreign key in a table references the primary key of another table, establishing a connection between the two tables that can be used in queries.

The Mismatch schema from the previous exercise relies on a pair of foreign keys in the `mismatch_response` table to connect response rows to user and topic rows in other tables.

A foreign column in that refers to the primary key in another table



Without foreign keys, it would be very difficult to associate data from one table with data in another table. And spreading data out across multiple tables is how we're able to eliminate duplicate data and arrive at an efficient database. So foreign keys play an important role in all but the most simplistic of database schemas.

Large arrows point from primary keys to foreign keys in the mismatch_response table

This primary key uniquely identifies the mismatch_user table and also connects to the mismatch_response table

mismatch_topic table also connects to the mismatch_response table via its topic_id column

The mismatch_topic table also connects to the mismatch_response table via its topic_id column

Foreign keys in action

It often helps to visualize data flowing into tables and connecting tables to one another through primary and foreign keys. Taking a closer look at the Mismatch tables with some actual data in them helps to reveal how primary keys and foreign keys relate to one another.

As a primary key, `user_id` must be unique within the `mismatch_user` table. In fact, that's its purpose – providing a unique reference to user rows.

The `user_id` foreign key serves as a reference to user rows in the `mismatch_user` table, allowing you to know which user is associated with a given response.

Remember, each row in this table corresponds to a Mismatch user.

Each row in this table is a single love/hate response by a certain user.

`mismatch_user`

<code>user_id</code>	<code>username</code>	<code>password</code>	...
...			
11	jnettles	*****	...
...			

`mismatch_response`

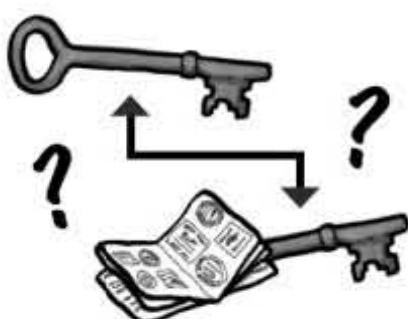
<code>response_id</code>	<code>response</code>	<code>user_id</code>	<code>topic_id</code>
...			
101	Love	11	1
102	Love	11	2
103	Hate	11	3
104	Love	11	4
...			

Within the `mismatch_response` table, you can find out more information about the user who entered a response by looking up the `user_id` in the `mismatch_user` table. Similarly, you can find out the name of the topic for the response, as well as its category, by looking up the `topic_id` in the `mismatch_topic` table.

Binding together tables with primary keys and foreign keys allows us to connect the data between them in a consistent manner. You can even structure your database so that primary keys and their respective foreign keys are **required** to match up. This is known as **referential integrity**, which is a fancy way of saying that all key references must be valid.

The `user_id` for a response row in the `mismatch_response` table isn't unique since several love/hate responses were made by the same user.

types of relationships between tables



I understand that primary keys and foreign keys connect multiple tables together, but does the direction of the arrows between keys in those diagrams mean anything?

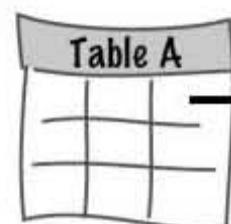
Yes, the direction of the arrows tells us how rows in each table relate to each other.

More specifically, they tell us how many rows in one table can have matching rows in another table, and vice-versa. This is a critical aspect of database schema design, and involves three different possible **patterns** of data: **one-to-one**, **one-to-many**, and **many-to-many**.

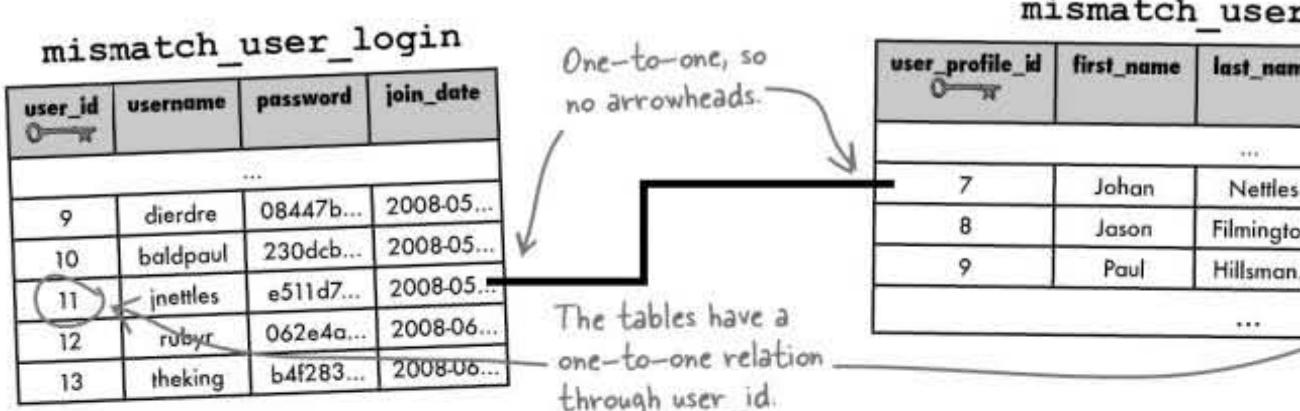
Tables can match row for row

The first pattern, **one-to-one**, states that a row in Table A can have at most ONE matching row in Table B, and vice-versa. So there is only one match in each table for each row.

As an example, let's say the Mismatch user table was separated into two tables, one for just the log-in information (Table A) and one with profile data (Table B). Both tables contain a user ID to keep users connected to their profiles. The `user_id` column in the log-in table is a primary key that ensures user log-in uniqueness. `user_id` in the profile table is a foreign key, and plays a different role since its job is just to connect a profile with a log-in.

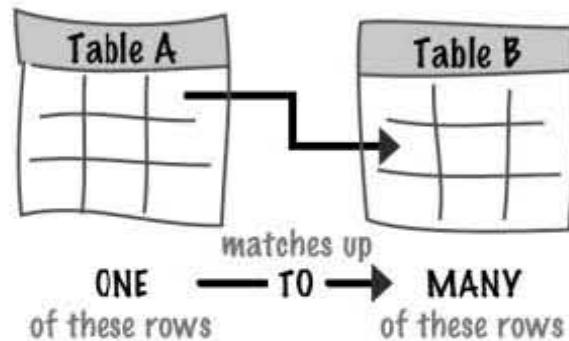


ONLY ONE
of these rows



With respect to the two `user_id` columns, the log-in table is considered a **parent** table, while the profile table is considered a **child** table—a table with a primary key has a parent-child relationship to the table with the corresponding foreign key.

One row leads to many



Primary key.

mismatch_user			
user_id	username	password	...
9	dierdre	08447b...	...
10	baldpaul	230dc...	...
11	jnettles	e511d7...	...
12	rubyr	062e4a...	...
13	theking	b4f283...	...

There is a
one-to-many
relationship
through
user_id.

mismatch_r		
response_id	response	user
101	Love	
102	Love	
103	Hate	
104	Love	

there are no
Dumb Questions



One-to-One:
exactly one
row of a
parent table
is related to
one row of
a child table.

Q: How do I know whether rows in two tables should have a one-to-one or one-to-many relationship?

A: There will be a tendency to use one-to-many patterns much more often than one-to-one, and rightly so. It's common to have a main (parent) table containing primary data, such as users in Mismatch, that connects to a secondary (child) table in a one-to-many arrangement. This happens twice in the Mismatch schema, where both users and topics have a one-to-many relationship to responses.

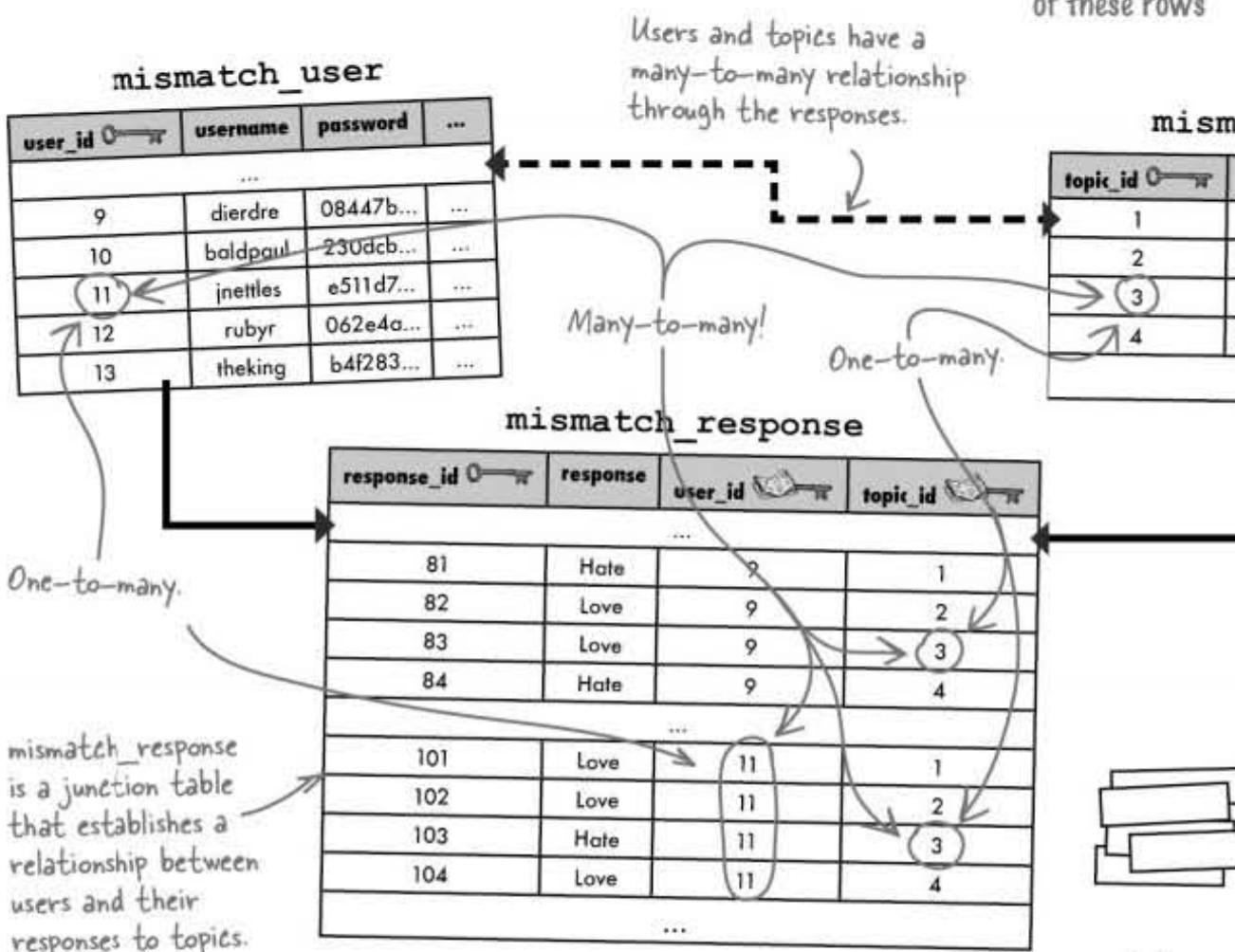
In many cases, rows with a one-to-one relationship in two tables can be combined into the same table. However, there are certainly situations where it makes sense to go with a one-to-one pattern, such as the hypothetical user profile example on the facing page, where there is a security motivation in moving a portion of the data into its own table.

On
exa
of
tal
to
row
ch

the many-to-many relationship

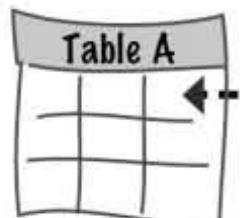
Matching rows many-to-many

The third and final table row relationship data pattern is the **many-to-many** relationship, which has multiple rows of data in Table A matching up with multiple rows in Table B... it's kinda like data overload! Not really. There are plenty of situations where a many-to-many pattern is warranted. Mismatch, perhaps? Let's have a look.



The many-to-many pattern in Mismatch is indirect, meaning that it takes place through the **mismatch_response** table. But the pattern still exists. Just look at how many of the same **user_ids** and **topic_ids** appear in **mismatch_response**.

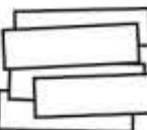
In addition to holding the response data, the **mismatch_response** table is acting as what's known as a **junction table** by serving as a convenient go-between for the users and topics. Without the junction table, we would have lots of duplicate data, which is a bad thing. If you aren't convinced, turn back to the schema exercise near the beginning of the chapter and take a closer look at Design 2. In that design, the **mismatch_topic** table is folded into the **mismatch_response** table, resulting in lots of duplicate data.



MANY
of these rows

mism

topic_id	...
1	
2	
3	
4	



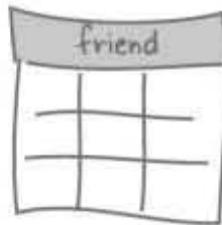
Man
Multi
a pa
are 1
mult
a ch

NAME THAT RELATIONSHIP

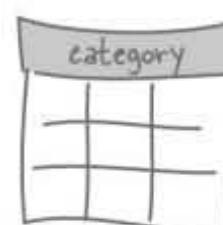
In each of the tables below, there are circled columns that could be moved out into their own tables. Write down if each of the columns is best represented by a one-to-one, one-to-many, or many-to-many relationship with its original table, and then draw a relationship as a line connecting the two tables with appropriate arrowheads.

RELATIONSHIP

mismatch_user	
user_id	key
...	
address	...
employer	...
friends	...



mismatch_topic	
topic_id	key
name	
category	



name that relationship solution

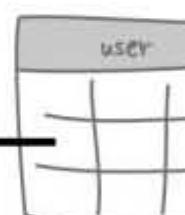
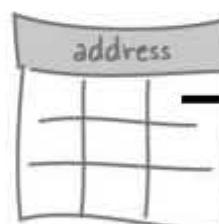
NAME THAT RELATIONSHIP SOLUTION

In each of the tables below, there are circled columns that could be moved out in their own tables. Write down if each of the columns is best represented by a one-to-one, one-to-many, or many-to-many relationship with its original table, and then draw a relationship as a line connecting the two tables with appropriate arrowheads.

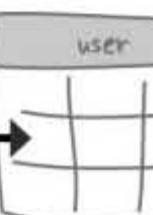
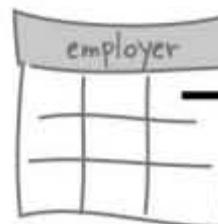
There's only one street address for any given user, which means an address row has a one-to-one relationship with a user row.

mismatch_user	
user_id	key
...	
address	oval
employer	oval
friends	oval

RELATIONSHIP

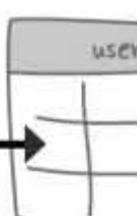
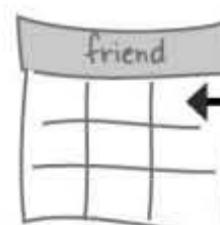


..... one-to-one



..... one-to-many

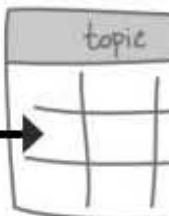
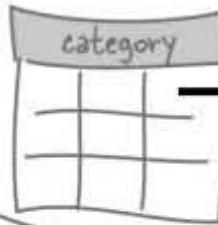
Multiple users can have multiple friends, meaning that friend rows have a many-to-many relationship with user rows.



..... many-to-many

Multiple topics can belong to the same category, resulting in a one-to-many relationship between a category row and topic rows. But a topic can't belong to more than one category.

mismatch_topic	
topic_id	key
name	
category	oval



..... one-to-many



Hold it right there! Take a second to get the Mismatch database in order so that we can make mismatches.

Download the .sql files for the Mismatch application from the Head First Labs web site at www.headfirstlabs.com/books/hfphp. These files contain SQL statements that build the necessary Mismatch tables: mismatch_user, mismatch_topic, and mismatch_response. Make sure to run the statement in each of the .sql files in a MySQL tool so that you have the initial Mismatch tables to get started with.

When all that's done, run a DESCRIBE statement on each of the new tables (mismatch_topic and mismatch_response) to double-check their structures. These tables factor heavily into the Mismatch PHP scripts we're about to put together.

```

File Edit Window Help Logout
mysql> DESCRIBE mismatch_topic;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra
+-----+-----+-----+-----+-----+
| topic_id | int(11) | NO | PRI |          | auto_increment
| name | varchar(48) | NO |     |          |
| category | varchar(48) | NO |     |          |
+-----+-----+-----+-----+-----+
3 rows in set (0.04 sec)

File Edit Window Help Logout
mysql> DESCRIBE mismatch_response;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra
+-----+-----+-----+-----+-----+
| response_id | int(11) | NO | PRI |          | auto_incr
| user_id | int(11) | NO |     |          |
| topic_id | int(11) | NO |     |          |
| response | tinyint(4) | NO |     |          |
+-----+-----+-----+-----+-----+
4 rows in set (0.05 sec)

```

The topic_id foreign key ties back to the primary key in the mismatch_topic table.

using the database to build a questionnaire

OK, so we have this wonderfully designed database of users, categories, topics, and responses. How is that going to actually help us make a mismatch?

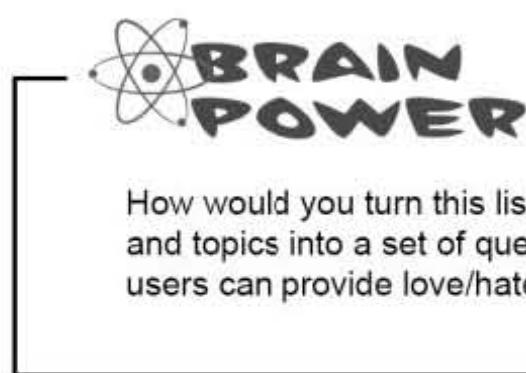
If you start with a well-designed other piece of the application puzzle, it's much easier to build and assemble.

Getting the database right when initially designing the application is perhaps the best thing you can do to make the application run smoothly. It may seem like a lot of work up front, but it's worth scheming about how best to store the data, because it will pay off in the long run. Think about how much more difficult it would be to rework the Mismatch database schema with an application built on top of it.

That's the big picture benefit of a good database design. In the case of the Mismatch database specifically, we have a database populated by the users themselves through signups, edits, and we have a new topic table that contains 25 topics and topics to give some decent insight into a person's interests. Still missing to make mismatches is a way to associate responses, and then store them away in the responses table.

topic_id	name	category
1	Tattoos	Appearance
2	Gold chains	Appearance
3	Body piercings	Appearance
4	Cowboy boots	Appearance
5	Long hair	Appearance
6	Reality TV	Entertainment
7	Professional wrestling	Entertainment
8	Horror movies	Entertainment
9	Easy listening music	Entertainment
10	The opera	Entertainment
11	Sushi	Food
12	Spam	Food
13	Spicy food	Food
14	Peanut butter & banana sandwiches	Food
15	Martini's	Food
16	Howard Stern	People
17	Bill Gates	People
18	Barbara Streisand	People
19	Hugh Hefner	People
20	Martha Stewart	People
21	Yoga	Activities
22	Weightlifting	Activities

The full mismatch_topic table contains 25 topics broken up across 5 categories...it's our "5 dimensions of opposability!"



How would you turn this list of topics and topics into a set of questions that users can provide love/hate responses to?

Build a Mismatch questionnaire

So how exactly do we get love/hate responses from users for each Mismatch topic? The answer is a questionnaire form that allows the user to choose “Love” or “Hate” for each topic in the `mismatch_topic` table. This form can be generated directly from the responses in the database, with its results getting stored back in the database. In fact, the design of the questionnaire form involves reading and writing responses from and to the `mismatch_response` table. Here’s a peek at the questionnaire, along with the steps involved in building it.

How do you feel about each topic?	
Appearance	
Tattoos:	<input type="radio"/> Love <input type="radio"/> Hate
Gold chains:	<input type="radio"/> Love <input type="radio"/> Hate
Body piercings:	<input type="radio"/> Love <input type="radio"/> Hate
Cowboy boots:	<input type="radio"/> Love <input type="radio"/> Hate
Long hair:	<input type="radio"/> Love <input type="radio"/> Hate
Entertainment	
Reality TV:	<input type="radio"/> Love <input type="radio"/> Hate
Professional wrestling:	<input type="radio"/> Love <input type="radio"/> Hate
Horror movies:	<input type="radio"/> Love <input type="radio"/> Hate

1 Use INSERT to add empty response rows to the database the first time a user accesses the form.

We’re going to generate the questionnaire form from data in the `mismatch_response` table. Since the user has never entered any responses, this means we need to “seed” the `mismatch_response` table with empty responses the first time a user accesses the questionnaire. Since the responses are empty, neither the “Love” or “Hate” radio buttons are checked when the form is first displayed.

2 Use UPDATE to change response rows based on user responses.

When users submit the questionnaire form, we must commit their personal responses to the database. Then, only responses with checked radio buttons should be updated. In other words, the database needs to know about the responses that **have** been answered.

3 Use SELECT to retrieve the response data required to generate the questionnaire form.

In order to generate the questionnaire form, we need all of the responses for the logged-in user. We need to look up the topic and category name for each response so that they can be displayed. These responses are stored in the `mismatch_topic` table, not `mismatch_response`.

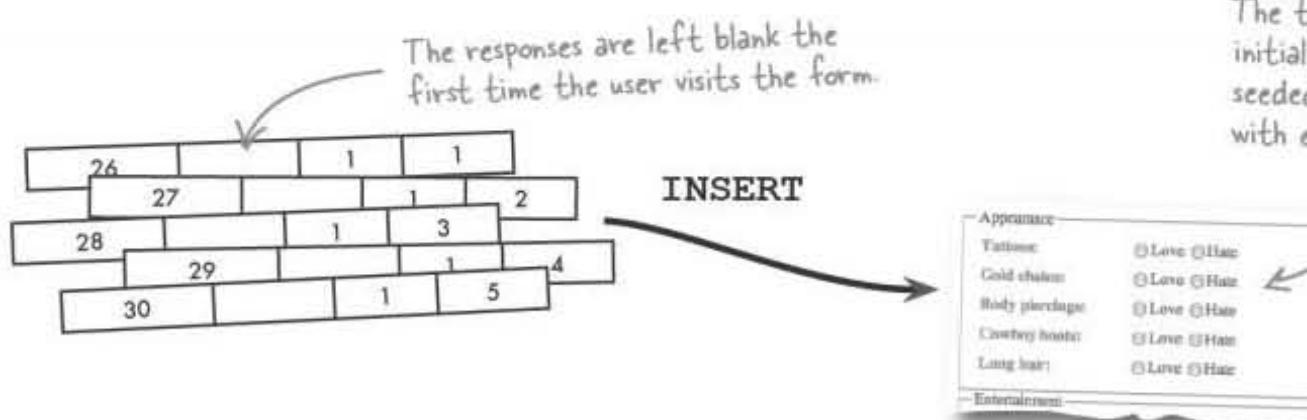
4 Generate the HTML questionnaire form from response data.

With the response data in hand, we can generate the HTML questionnaire form as a string of HTML code, making sure to check the appropriate “Love” or “Hate” radio buttons based on the user’s responses.

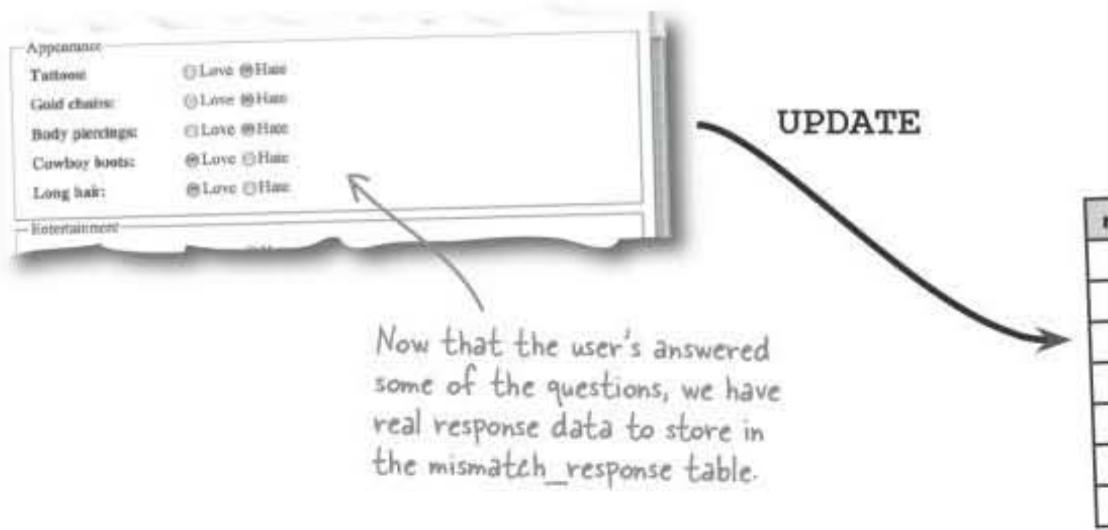
putting responses into mismatch_response

Get responses into the database

Although it might seem as if we should start out by generating the questionnaire form, the form's dependent on response data existing in the `mismatch_response` table. So first things first: we need to "seed" the `mismatch_response` table with rows of unanswered responses the first time a user accesses the questionnaire. This will allow us to generate the questionnaire form from the `mismatch_response` table without having to worry about whether the user has actually made any responses.



So from the perspective of the questionnaire form, there's always a row of data in the `mismatch_response` table for each question in the form. This means that when the user submits the questionnaire form, we just update the rows of data for each response in the form.



Although storing responses in the Mismatch database is ultimately a two-step process, the first step (INSERT) only takes place once for each user. Once the empty responses are added initially, all future changes to the questionnaire are handled by the second step via SQL UPDATEs.



PHP & MySQL Magnets

The following code takes care of inserting empty responses into the mismatch_response table the first time a user visits the questionnaire form. It also updates the responses when the user makes changes and submits the form. Unfortunately, some of the code has fallen off and needs to be replaced. Use the magnets to fix the missing code.

```
... // If this user has never answered the questionnaire, insert empty responses
$query = "SELECT * FROM mismatch_response WHERE user_id = '" . $_SESSION['user_id'] . "'";
$data = mysqli_query($dbc, $query);

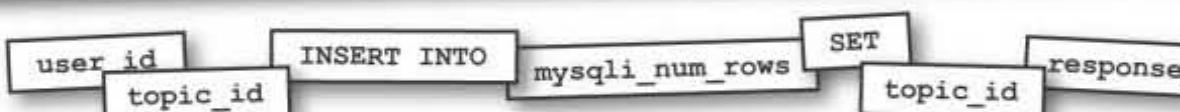
if (!.....) { // First grab the list of topic IDs from the topic table
    $query = "SELECT ..... FROM mismatch_topic ORDER BY category_id, topic_id";
    $data = mysqli_query($dbc, $query);
    $topicIDs = array();
    while ($row = mysqli_fetch_array($data)) {
        array_push($topicIDs, $row['topic_id']);
    }

    // Insert empty response rows into the response table, one per topic
    foreach ($topicIDs as $Topic_id) {
        $query = "..... mismatch_response ";
        "..... ) VALUES ('" . $_SESSION['user_id'] . "' , ";
        mysqli_query($dbc, $query);
    }
}

// If the questionnaire form has been submitted, write the form responses to the response table
if (isset($_POST['submit'])) {
    // Write the questionnaire response rows to the response table
    foreach ($_POST as $response_id => $response) {
        $query = "..... mismatch_response ..... response = '$response'";
        "WHERE ..... = '$response_id'";

        mysqli_query($dbc, $query);
    }
    echo '<p>Your responses have been saved.</p>';
}

...
```



php & mysql magnets solution

PHP & MySQL Magnets

The following code takes care of inserting empty responses into the mismatch_response table the first time a user visits the questionnaire form. It also updates the responses when the user makes changes and submits the form. Unfortunately, some of the code has fallen off and needs to be replaced. Use the magnets to fix the missing code.

```
... // If this user has never answered the questionnaire, insert empty responses
$query = "SELECT * FROM mismatch_response WHERE user_id = '" . $_SESSION['user_id'] . "'";
$data = mysqli_query($dbc, $query);

if (!...mysqli_num_rows ($data) == 0) { ← Check to see if the query
    // First grab the list of topic IDs from the topic table
    $query = "SELECT ... topic_id ... FROM mismatch_topic ORDER BY category_id, topic_id";
    $data = mysqli_query($dbc, $query);
    $topicIDs = array();
    while ($row = mysqli_fetch_array($data)) {
        array_push($topicIDs, $row['topic_id']);
    }

    // Insert empty response rows into the response table, one per topic
    foreach ($topicIDs as $topic_id) {
        $query = "... INSERT INTO mismatch_response ...";
        "... user_id ... topic_id ... VALUES ('" . $_SESSION['user_id'] . "'...'";
        mysqli_query($dbc, $query);
    }
}

// If the questionnaire form has been submitted, write the form responses to the response table
if (isset($_POST['submit'])) {
    // Write the questionnaire response rows to the response table
    foreach ($_POST as $response_id => $response) {
        $query = "... UPDATE mismatch_response ... SET ... response = '$response'";
        "... WHERE response_id = '$response_id'";
        mysqli_query($dbc, $query);
    }
    echo '<p>Your responses have been saved.</p>';
}

...
```

In order to generate an empty response, we first need to get the topics in the topic table.

All that changes when the user submits the form is the response of the response that's all we want.

there are no
Dumb Questions

Q: What's the deal with the `array_push()` function? I don't think we've used that one before.

A: We haven't. And that's because we haven't needed to build an array dynamically one element at a time. The `array_push()` function tacks a new element onto the end of an array, causing the array to grow by one. In the Mismatch code on the facing page, we're using `array_push()` to build an array of topic IDs from the `mismatch_topic` table. This array is then used to insert blank responses into the `mismatch_response` table... one for each topic.

1 **Use INSERT to add empty response rows to the database the first time the user accesses the form.**

DONE

Bam! We just killed two virtual birds with one stone, and now have the questionnaire script already half built.

2 **Use UPDATE to change response rows based on user responses on the form.**

DONE

But two steps are left before we can start making love connections with the Mismatch questionnaire...

3 **Use SELECT to retrieve the response data required to generate the questionnaire for...**

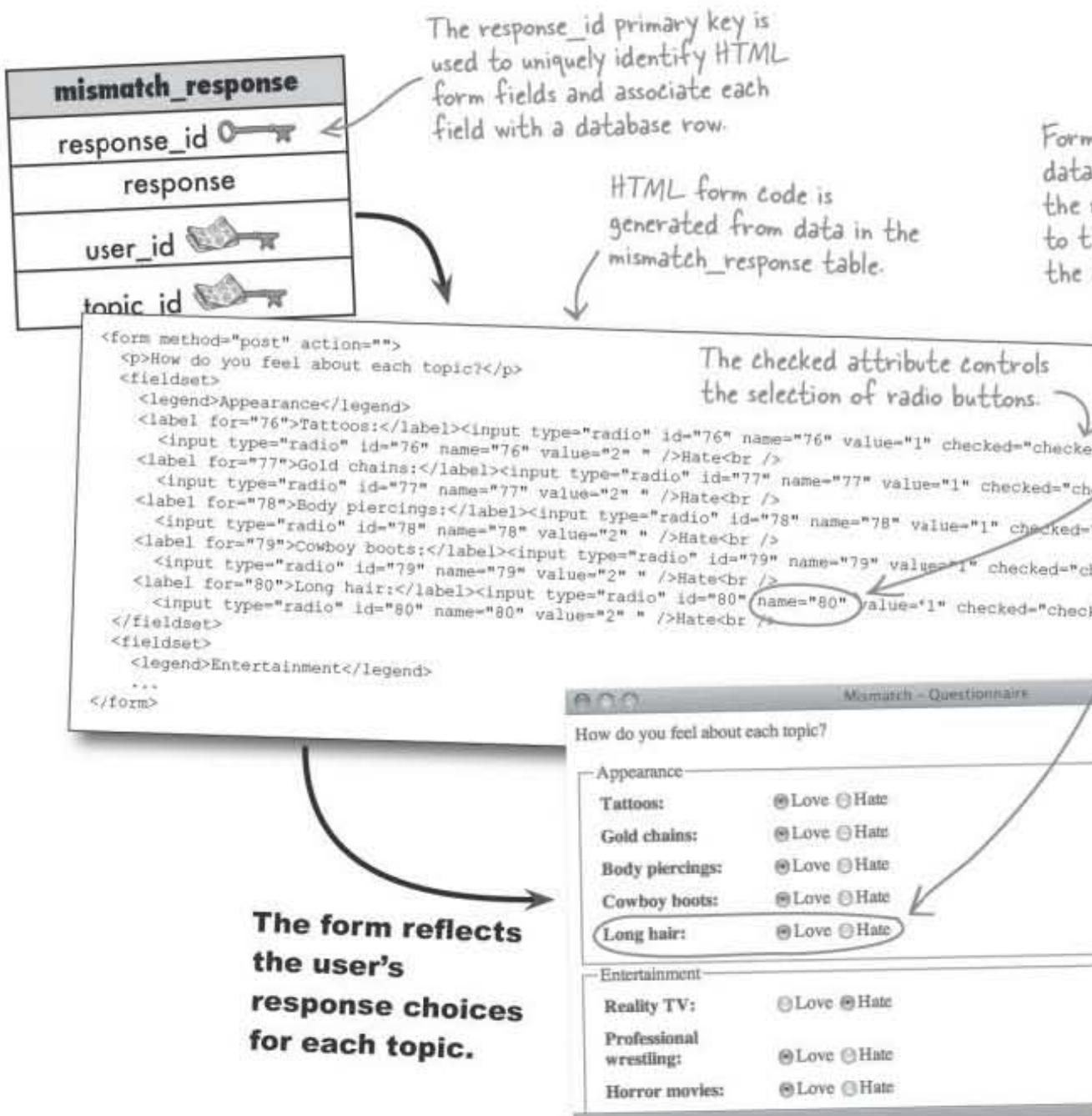
4 **Generate the form from...**

using data-driven forms

We can drive a form with data

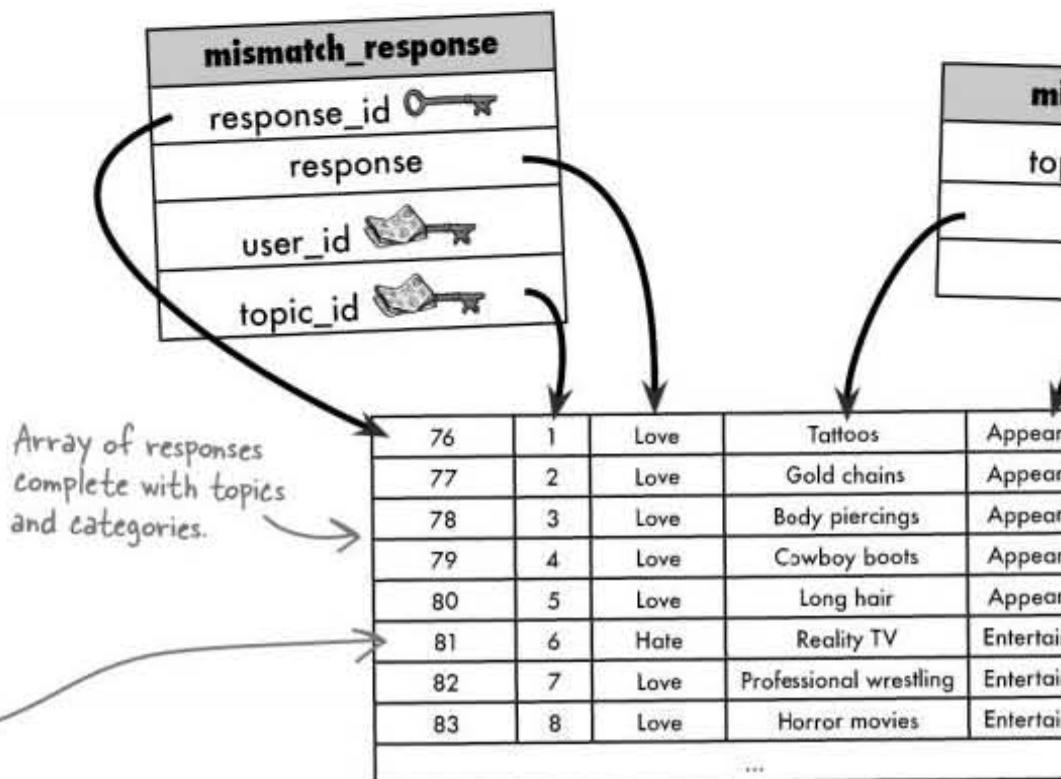
It's nothing new that web forms are used to retrieve data from users via text fields, selection lists, radio buttons, etc., but it may not be all that obvious that you can generate HTML forms from database data using PHP. The idea with Mismatch is to dynamically generate an HTML questionnaire form from response data. The Mismatch questionnaire script makes the assumption that response data already exists, which allows it to generate the form from data in the `mismatch_response` table. We know this assumption is a safe one because we just wrote the code to add empty responses the first time a user visits the form.

Data-driven forms rely on MySQL to generate form fields





The Mismatch response questionnaire is generated from user responses in the `mismatch_response` table. In order to generate the code for this exercise, you will need to read these responses, making sure to look up the name of the topic for each response from the `mismatch_topic` table. The following code shows how to query the database to get the responses along with topics and categories by performing two queries: the first query looks up the responses for a specific user, while the second query looks up the topic and category names for each response. Note that there is some missing code; fill in the blanks to get it working correctly.



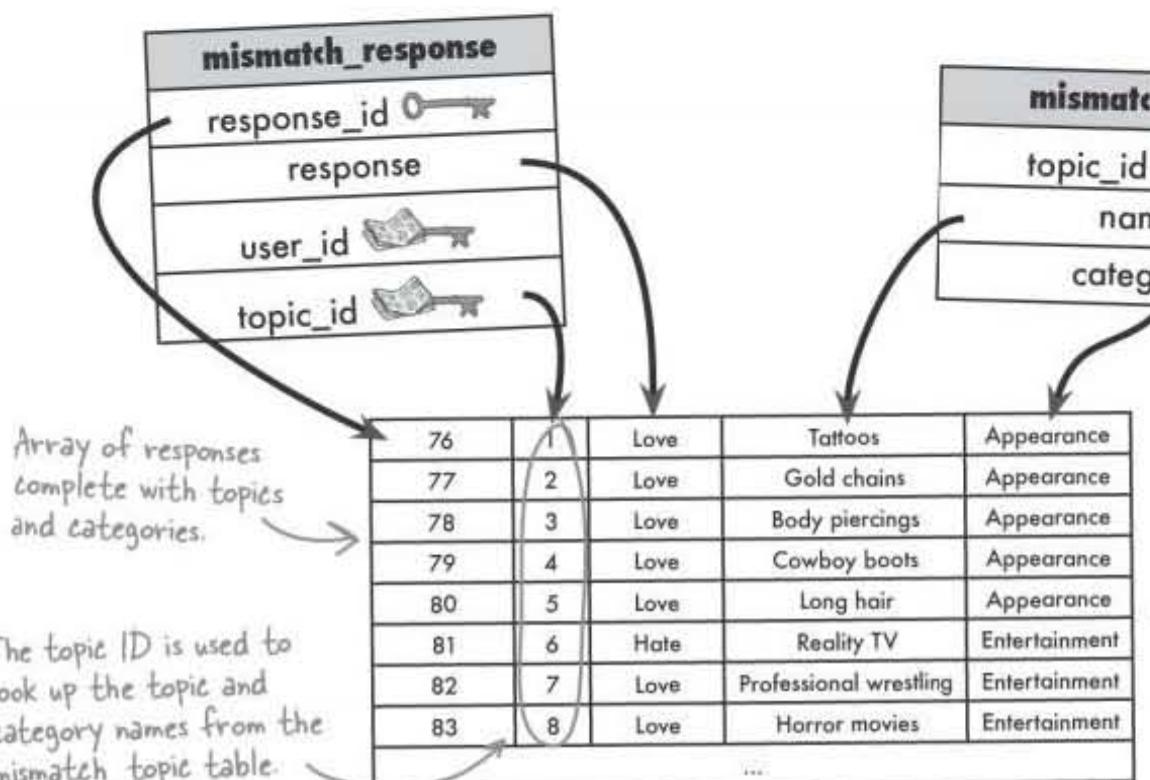
```
// Grab the response data from the database to generate the questionnaire
$query = "SELECT response_id, topic_id, response FROM mismatch_response
          WHERE user_id = '" . $_SESSION['user_id'] . "'";
$data = mysqli_query($dbc, $query);
$responses = array();
while ($row = mysqli_fetch_array($data)) {
    // Look up the topic name for the response from the mismatch_topic table
    $query2 = "
        SELECT topic_name, category_name
        WHERE topic_id = '" . $row['topic_id'] . "'";
    $data2 = mysqli_query($dbc, $query2);
    if (mysqli_num_rows($data2) == 1) {
        $row2 = mysqli_fetch_array($data2);
        $row['topic_name'] = $row2['topic_name'];
        $row['category_name'] = $row2['category_name'];
        array_push($responses, $row);
    }
}
```

This PHP function tells you how many rows of data were returned as query results.

exercise solution



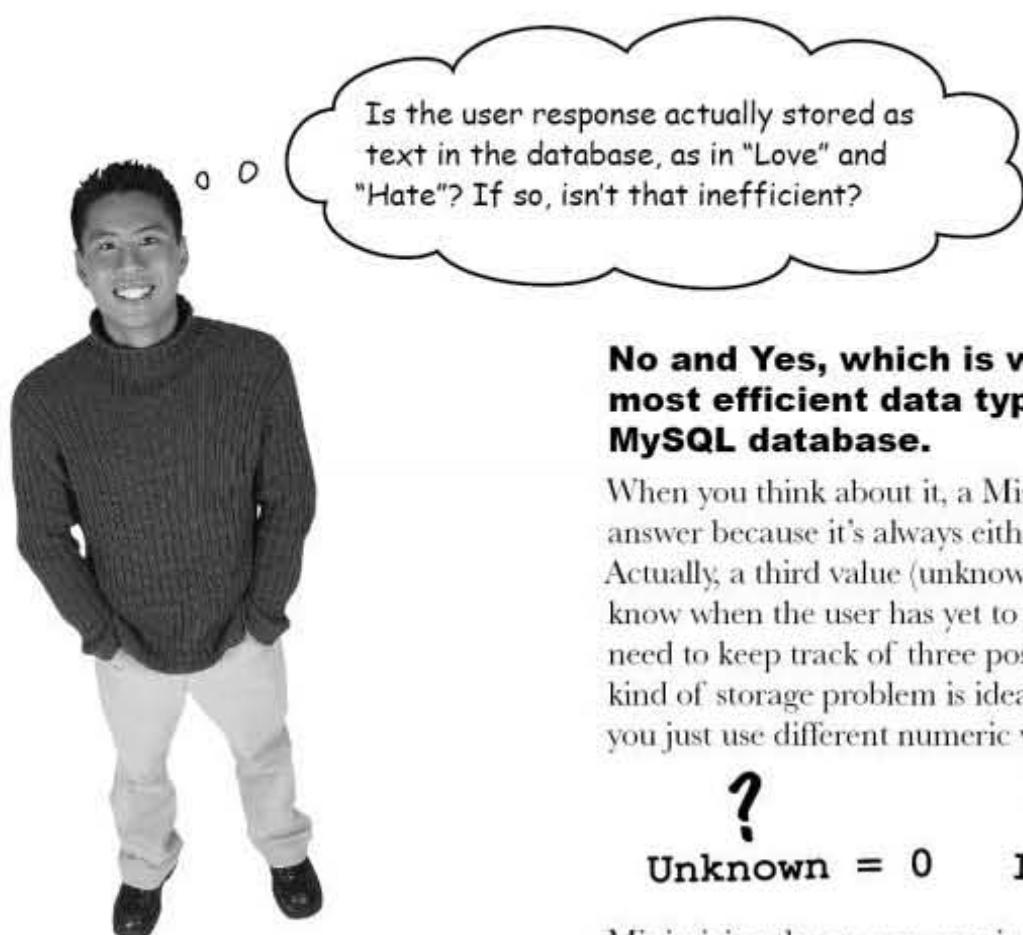
The Mismatch response questionnaire is generated from user responses that mismatch_response table. In order to generate the code for the HTML to read these responses, making sure to look up the name of the topic and response from the mismatch_topic table. The following code builds an array with topics and categories by performing two queries: the first query grabs the user, while the second query looks up the topic and category name for each. As is, some of the code is missing... fill in the blanks to get it working!



```
// Grab the response data from the database to generate the form
$query = "SELECT response_id, topic_id, response FROM mismatch_response
          WHERE user_id = '" . $_SESSION['user_id'] . "'";
$data = mysqli_query($dbc, $query);
$responses = array();
while ($row = mysqli_fetch_array($data)) {
    // Look up the topic name for the response from the topic table
    $query2 = "...SELECT name, category FROM mismatch_topic....."
              ."WHERE topic_id = '" . $row['topic_id'] . "'";
    $data2 = mysqli_query($dbc, $query2);
    if (mysqli_num_rows($data2) == 1) {
        $row2 = mysqli_fetch_array($data2);
        $row['topic_name'] = ...$row2['name'];
        $row['category_name'] = ...$row2['category'];
        array_push($responses, $row);
    }
}
```

Annotations for the code:

- A callout box says: "Make sure there actually is response data."
- A callout box says: "The array_push() function adds (pushes) an item onto the end of an array."
- A callout box says: "The topic name and category name are added to the response array from the data from the database."



No and Yes, which is why it is important to choose the most efficient data type possible to store in MySQL database.

When you think about it, a Mismatch response is not really an answer because it's always either one value (love) or the other. Actually, a third value (unknown) can be useful in lots of cases. You know when the user has yet to respond to a particular question. You need to keep track of three possible values for any given question. This kind of storage problem is ideal for a number, such as 0, 1, or 2. You just use different numeric values to represent each response.



Unknown = 0



Love = 1

Minimizing the storage requirements of data is an important part of database design, and in this case a subtle but important part of a Mismatch application. These numeric responses play a key role in generating the HTML code for the form fields for the Mismatch questions.



Don't worry about the "Hate" radio buttons for now - they're generated exactly the same way.

The following code loops through the Mismatch responses that you just created, generating an HTML form field for each "Love" radio button. Fill in the missing code so that the radio button is initially checked if the response is set to love (1). Make sure the value of the `<input>` tag is set according to the response value.

```

foreach ($responses as $response) {
    ...
    if (.....) {
        echo '<input type="radio" name="' . $response['response_id'] . '" value="....." checked="....." />Love ';
    }
    else {
        echo '<input type="radio" name="' . $response['response_id'] . '" value="....." />Love ';
    }
}

```

sharpen your pencil solution

Sharpen your pencil Solution

The "Love" radio button is checked based on the value of the response (1 represents love in the database).

```
foreach ($responses as $response) {
    ...
    if ($response['response'] == 1) {
        echo '<input type="radio" name="' . $response['response_id'] .
            '" value="1" checked="checked" />Love ';
    }
    else {
        echo '<input type="radio" name="' . $response['response_id'] .
            '" value="1" />Love ';
    }
}
```

The value of the <input> tag is set to "1" so that it will be easier to store the response in the database when the form is submitted.

If this response is set to (1), check the radio button setting its checked attribute to "checked".

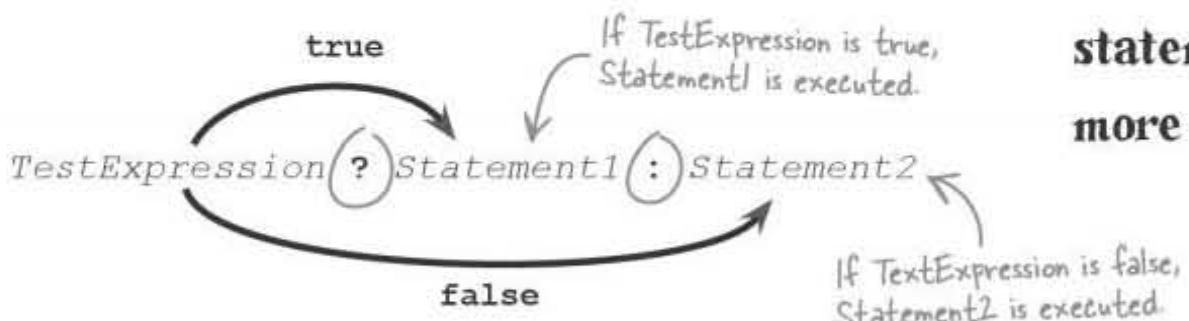
Leaving off checked="checked" results in the radio button being unchecked if the response isn't set to love (1).

```
foreach ($responses as $response) {
    ...
    if ($response['response'] == 2) {
        echo '<input type="radio" name="' . $response['response_id'] .
            '" value="2" checked="checked" />Hate ';
    }
    else {
        echo '<input type="radio" name="' . $response['response_id'] .
            '" value="2" />Hate ';
    }
}
```

In case you're curious, the code to generate the "Hate" radio buttons works exactly the same way – it just looks for a different response... but there's actually a cleaner way to generate both the "Love" and "Hate" radio buttons with less code.

Speaking of efficiency...

Database efficiency isn't the only kind of efficiency worth considering. There's also **coding efficiency**, which comes in many forms. One form is taking advantage of the PHP language to simplify if-else statements. The **ternary operator** is a handy way to code simple if-else statements so that they are more compact.



The ternary operator is really just a shorthand way to write an if-else statement. It can be helpful for simplifying if-else statements, especially when you're making a variable assignment or generating HTML code in response to the if condition. Here's the same "Love" radio button code rewritten to use the ternary operator:

```
echo '<input type="radio" name="' . $response['response_id'] . '" value="1"' . ($response['response'] == 1 ? ' checked="checked"' : '') . ' />Love'
```

This true/false test controls the outcome of the ternary operator.

If the response value stored in `$response['response']` is equal to 1, then the `checked` attribute will get generated as part of the `<input>` tag, resulting in the following **checked** "Love" radio button:

```
<input type="radio" name="279" value="1" checked="checked" />Love
```

This part of the `<input>` tag's code is controlled by the ternary operator.

Long hair: @ Love @ Hate

The `checked` of the `<input>` now generates ternary operator of an if-else

On the other hand, a response value of anything other than 1 will prevent the `checked` attribute from being generated, resulting in an `<input>` tag for the "Love" radio button that is unchecked.

The ternary operator used to code if-else statements more compactly.

the full questionnaire.php script

Generate the Mismatch questionnaire form

We now have enough pieces of the Mismatch questionnaire form puzzle to use the response array (`$responses`) we created earlier to generate the entire HTML form. If you recall, this array was built by pulling out the current user's responses from the `mismatch_response` table. Let's go ahead and see the questionnaire generation code in the context of the full `questionnaire.php` script.

```
<?php
    // Start the session
    require_once('startsession.php');

    // Insert the page header
    $page_title = 'Questionnaire';
    require_once('header.php');

    require_once('appvars.php');
    require_once('connectvars.php');

    // Make sure the user is logged in before going any further.
    if (!isset($_SESSION['user_id'])) {
        echo '<p class="login">Please <a href="login.php">log in</a> to access this
            exit();
    }

    // Show the navigation menu
    require_once('navmenu.php');

    // Connect to the database
    $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

    // If this user has never answered the questionnaire, insert empty responses
    $query = "SELECT * FROM mismatch_response WHERE user_id = '" . $_SESSION['use
    $data = mysqli_query($dbc, $query);
    if (mysqli_num_rows($data) == 0) {
        // First grab the list of topic IDs from the topic table
        $query = "SELECT topic_id FROM mismatch_topic ORDER BY category_id, topic_i
        $data = mysqli_query($dbc, $query);
        $topicIDs = array();
        while ($row = mysqli_fetch_array($data)) {
            array_push($topicIDs, $row['topic_id']);
        }

        // Insert empty response rows into the response table, one per topic
        foreach ($topicIDs as $topic_id) {
            $query = "INSERT INTO mismatch_response (user_id, topic_id) VALUES ('"
                "', '$topic_id')";
            mysqli_query($dbc, $query);
        }
    }

    // If the questionnaire form has been submitted, write the form responses to
    if (isset($_POST['submit'])) {
        // Write the questionnaire response rows to the response table
        foreach ($_POST as $response_id => $response) {
            $query = "UPDATE mismatch_response SET response = '$response' ";
    }
}
```

Include the template files that start the session and display the page header.

Rest users

1

2

```

        "WHERE response_id = '$response_id'";
        mysqli_query($dbc, $query);
    }
    echo '<p>Your responses have been saved.</p>';

    // Grab the response data from the database to generate the form
    $query = "SELECT response_id, topic_id, response FROM mismatch_response WHERE
        $_SESSION['user_id'] . '"';
    $data = mysqli_query($dbc, $query);
    $responses = array();
    while ($row = mysqli_fetch_array($data)) {
        // Look up the topic name for the response from the topic table
        $query2 = "SELECT name, category FROM mismatch_topic WHERE topic_id = '" .
            '"';
        $data2 = mysqli_query($dbc, $query2);
        if (mysqli_num_rows($data2) == 1) {
            $row2 = mysqli_fetch_array($data2);
            $row['topic_name'] = $row2['name'];
            $row['category_name'] = $row2['category'];
            array_push($responses, $row);
        }
    }

    mysqli_close($dbc);

    // Generate the questionnaire form by looping through the response array
    echo '<form method="post" action="' . $_SERVER['PHP_SELF'] . '">';
    echo '<p>How do you feel about each topic?</p>';
    $category = $responses[0]['category_name'];
    echo '<fieldset><legend>' . $responses[0]['category_name'] . '</legend>';
    foreach ($responses as $response) {
        // Only start a new fieldset if the category has changed
        if ($category != $response['category_name']) {
            $category = $response['category_name'];
            echo '</fieldset><fieldset><legend>' . $response['category_name'] . '</legend>';
        }

        // Display the topic form field
        echo '<label>' . ($response['response'] == NULL ? 'class="error"' : '') . ' ' .
            $response['response_id'] . '">' . $response['topic_name'] . ':</label>';
        echo '<input type="radio" id="' . $response['response_id'] . '" name="' .
            $response['response_id'] . '" value="1"' . ($response['response'] == 1 ? 'checked="checked"' : '') . ' />Love ';
        echo '<input type="radio" id="' . $response['response_id'] . '" name="' .
            $response['response_id'] . '" value="2"' . ($response['response'] == 2 ? 'checked="checked"' : '') . ' />Hate<br />';

        echo '</fieldset>';
        echo '<input type="submit" value="Save Questionnaire" name="submit" />';
        echo '</form>';

        // Insert the page footer
        require_once('footer.php');
    ?>

```

Each of these echo statements generates a radio button - one for "Love" and one for "Hate".

Grab the category of the first response to get started before entering the loop.

Remember,
1 = love,
2 = hate.

4 Generate the form from re

test out questionnaire.php

Test Drive

Try out the new Mismatch questionnaire.

Modify Mismatch to use the new Questionnaire script (or download the application from the Head First Labs site at www.headfirstlabs.com/books/hfphp). This involves creating a new `questionnaire.php` script, as well as adding a “Questionnaire” menu item to the `navmenu.php` script so that users can access the questionnaire.

Upload the scripts to your web server, and then open the main Mismatch page (`index.php`) in a web browser. Make sure you log in, and then click the “Questionnaire” menu item to access the questionnaire. Notice that none of the topics have answers since this is your first visit to the questionnaire. Answer the responses and submit the form. Return to the main Mismatch page, and then go back to the questionnaire once more to make sure your responses were properly loaded from the database.

The Questionnaire script allows users to answer love/hate questions and store the results in the database.

How do you feel about each topic?

Appearance	
Tattoos:	<input type="radio"/> Love <input type="radio"/> Hate
Gold chains:	<input type="radio"/> Love <input type="radio"/> Hate
Body piercings:	<input type="radio"/> Love <input type="radio"/> Hate
Cowboy boots:	<input type="radio"/> Love <input type="radio"/> Hate
Long hair:	<input type="radio"/> Love <input type="radio"/> Hate

Entertainment	
Reality TV:	<input type="radio"/> Love <input type="radio"/> Hate
Professional wrestling:	<input type="radio"/> Love <input type="radio"/> Hate
Horror movies:	<input type="radio"/> Love <input type="radio"/> Hate

The topic questions in the form are dynamically generated from the database – if you add new topics, the form changes.

Download It!



The complete source code for the application is available for download from the Head First Labs website.

www.headfirstlabs.com/

there are no
Dumb Questions

Q: How does the “Love” radio button code know that the ternary operator result is a string?

A: The ternary operator always evaluates to one of the two statements on each side of the colon based on the value (`true` or `false`) of the test expression. If these statements are strings, then the result of the ternary operator will be a string. That’s what makes the operator so handy—you can insert it right into the middle of an assignment or concatenation.

Q: Does the ternary operator make my script run faster?

A: No, probably not. The ternary operator is more about adding stylistic efficiency to your code than performance efficiency, meaning that it literally requires less script code. Sometimes it’s more concise to use the ternary operator rather than a full-blown `if-else` statement, even though the two are logically equivalent. Even so, don’t get too carried away with the ternary operator because it can make some code more difficult to understand if you’re attempting to recast a complex `if-else` statement. The idea is to use the ternary operator in places where eliminating an `if-else` can actually help simplify the code, as opposed to making it more complicated. This usually involves using the ternary operator to selectively control a value being assigned to a variable or inserted into an expression. In the case of the Mismatch radio buttons, the latter approach was used to selectively control the insertion of an HTML attribute (`checked`).

Q: How is it possible to generate the form from the `mismatch_response` table since it has yet to respond to anything?

A: Excellent question. The questionnaire needs to generate two possible scenarios the user is answering for the first time or the user has already answered. In the first scenario, no responses have been added to the `mismatch_response` table. But we still need to dynamically generate the `mismatch_topic` table for this scenario. This won’t work for the second scenario, since the form must be generated based on the user’s responses; remember, the radio buttons are generated as part of the form. So we have to generate the form completely differently depending on whether the users have answered the questionnaire or not. They only answered a few questions? That’s fine. The solution taken by Mismatch is to pre-populate the `mismatch_response` table with the first time the user accesses the questionnaire. This allows us to always generate the questionnaire from the `mismatch_response` table, and the complication of generating the form differently for the users who have responded before, or who haven’t responded to. Sure, the form generation is a bit more complex, but it’s simpler than if we had not taken this approach.



1 Add a new topic to your own `mismatch_topic` table with this SQL statement:

```
INSERT INTO mismatch_topic
  (name, category) VALUES
  ('Virtual guitars', 'Activities')
```

2 Empty all the data from the `mismatch_response` table with this SQL statement:

```
DELETE FROM mismatch_response
```

3 View the questionnaire in the Mismatch application to see the new topic.

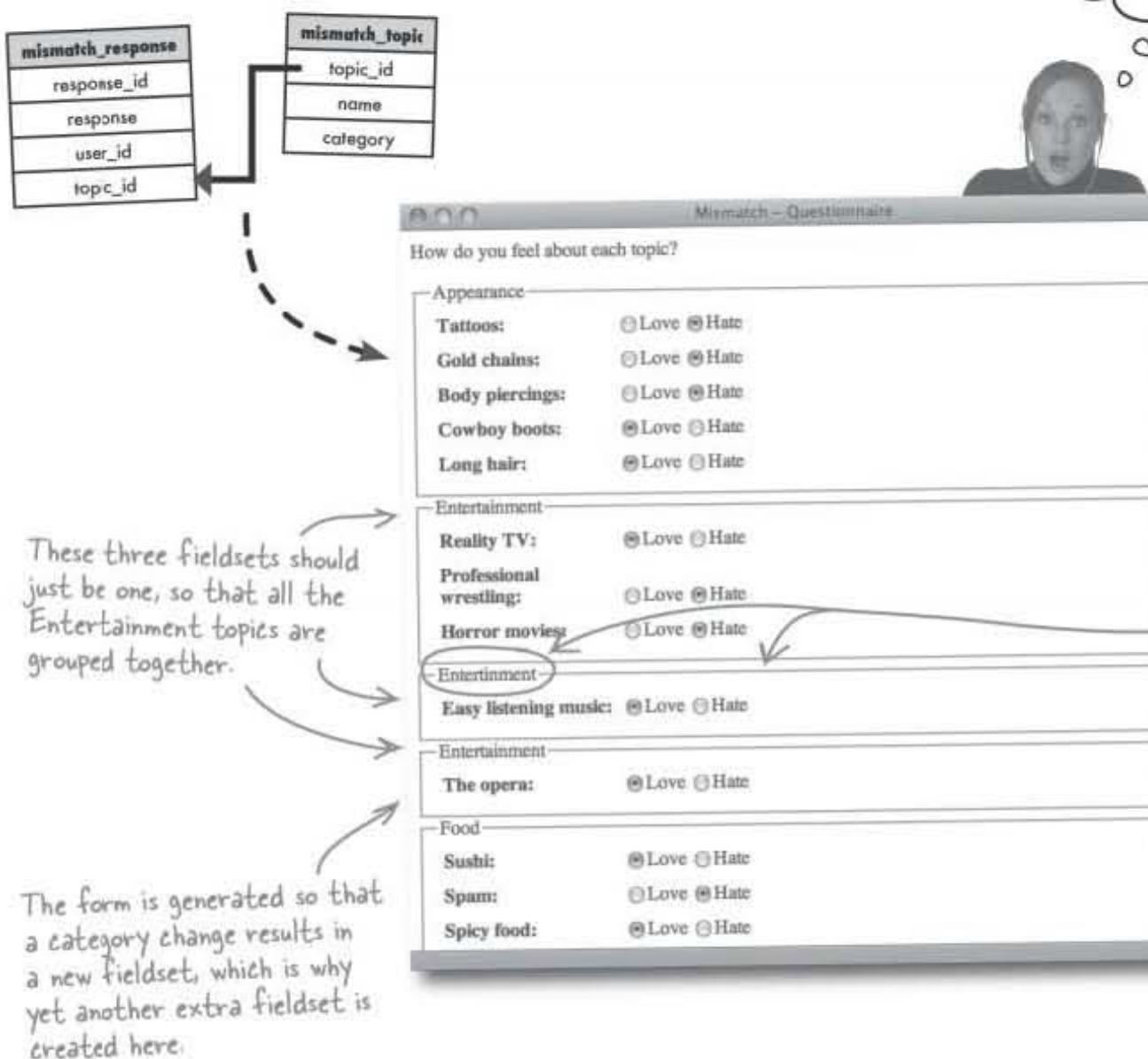
4 Respond to the new topic, submit the form, and check out your saved responses.

To simplify the code, Mismatch does not automatically, at least not when it’s first created, add a new row to the `mismatch_response` table. Instead, it adds a new row to the `mismatch_topic` table, which allows us to always generate the questionnaire from the `mismatch_response` table, and the complication of generating the form differently for the users who have responded before, or who haven’t responded to. Sure, the form generation is a bit more complex, but it’s simpler than if we had not taken this approach.

dealing with bad database data

The data is now driving the form

It took some doing, but the Mismatch application dynamically generates the questionnaire from responses stored in the database. This means that any changes made to the database will automatically be reflected in the form—that's the whole idea of driving the user interface of a web application from a database. But what happens when we have bad data?



The data is driving the form's fine, but something's that one of the categories has been misspelled in the PHP code to generate a separate fieldset for it, because it ruins the effect of using fieldsets to help make it easier to respond to topics.

OK, so one of the categories in the database is misspelled, which is screwing up our form. How do you guys think we should fix it?

topic_id	name	category
8	Horror movies	Entertainment
9	Easy listening music	Entertainment
10	The opera	Entertainment
11	Sushi	Food
12	Spam	Food
13	Spicy food	Food
14	Peanut butter & banana sandwiches	Food
15	Martinis	Food
16	Howard Stern	People
17	Bill Gates	Peopel
18	Barbara Streisand	People

We already have category names in the database:

- People
- Howard Stern
- Peopel
- Bill Gates:
- People
- Barbara Streisand

...and here they are in the same table, with the question:

Frank: That's easy. Just change the name of the category in the `mismatch_topic` table to the correct one.

Joe: But there's more than one category misspelled. And now that I think about it, I'm not really sure why we need to store category names in the database at all. After all, the category names have to be stored more than once.

Jill: I agree. We went to the trouble of eliminating duplicate data in designing the database schema, so we might as well keep it consistent with a bunch of duplicate category names. Not only that, but we have a couple that aren't even correct.

Frank: OK, what about just getting rid of category names and maybe referring to categories by their ID numbers? That way, we wouldn't run the risk of typos.

Joe: True, but we still need the category names as headings in the questionnaire form.

Jill: Maybe we can refer to categories by number without throwing out the names. That's sort of what we did with topics already with the `mismatch_topic` table, right?

Joe: That's right! We didn't want to store a bunch of duplicate topic names in the `mismatch_topic` table, so instead we put the topic names into the `mismatch_topic` table, and tied topics to responses with numbers.

Frank: Are you saying we could solve the duplicate category name problem by creating a new table?

Jill: That's exactly what he's saying. We can create a new `mismatch_category` table where each category name is stored exactly one time. And then connect categories with topics using primary and foreign keys between the `mismatch_topic` and `mismatch_category` tables. Brilliant!

normalizing your data

Strive for a bit of **normalcy**

The process of redesigning the Mismatch database to eliminate duplicate data and break apart and connect tables in a logical and consistent manner is known as **normalization**. Normalization is a fairly deep database design topic that can be intimidating. But it doesn't have to be. There are enough simple database design techniques we can graft from the basics of normalization to make our MySQL databases much better than if we had just guessed at how data should be laid out.

Here are some very broad steps you can take to begin the database design process that will naturally lead to a more “normal” database:

1. **Pick your thing, the **one thing** you want a table to describe.**
2. **Make a **list of the information** you need to know about your one thing when you're using the table.**
3. **Using the list, **break down the information** about your thing **into pieces** you can use for **organizing the table**.**

What's
you wa
be abo

How w
this t

How c
query

One fundamental concept in normalization is the idea of **atomic** data, which is data broken down into the smallest form that makes sense given the usage of a database. For example, the `first_name` and `last_name` columns in the Mismatch database are atomic in a sense that they break the user's name down further than a single name column would have. This is necessary in Mismatch because we want to be able to refer to a user by first name alone.

It might not always be necessary for an application to break a full name down into separate first and last columns, however, in which case name by itself might be atomic enough. So as you're breaking down the “one thing” of a table into pieces, think about **how** the data is going to be used, not just what it represents.

Atomic data
data that h
broken dow
smallest fo
for a given

When normalizing, think in atoms

To help turn your database design brainstorms into actions, it's helpful to ask targeted questions of your data. This will help determine how the data fits into a table, and if it has truly been broken down into its appropriate atomic representation. No one ever said splitting the atom was easy, but this list of questions can help.



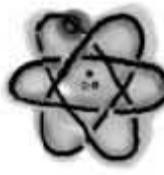
1. What is the **one thing** your table describes?

Does your table describe alien sightings, email list subscriptions, video game high scores, hopeless romantics?



2. How will you **USE** the table to get at the **one thing**?

Design Y be easy



3. Do your **columns** contain **atomic data** to make your queries short and to the point?

Make su small as

there are no
Dumb Questions

Q: Should I try to break my data down into the tiniest pieces possible?

A: Not necessarily. Making your data atomic means breaking it down into the smallest pieces that you need to create an efficient table, not just the smallest possible pieces you can.

Don't break down your data any more than you have to. If you don't need extra columns, don't add them just for the sake of it.

Q: How does atomic data help me?

A: It helps you ensure that the data is atomic. For example, if you have a column for the address of an alien sighting, you might want to break that column into two columns: the number and the street. Then the only numbers end up in the number column.

Atomic data also lets you perform queries faster. The queries are easier to write, and take less time to run, which adds up when you have a massive database.

Makin
atomic
step in
norma

virtues of normalization

Why be normal, really?

If all this talk about nuclear data and normalcy seems a bit overkill for your modest database, consider what might happen if your web application explodes and becomes the next “big thing.” What if your database grows in size by leaps and bounds in a very short period of time, stressing any weaknesses that might be present in the design? You’d rather be out shopping for your new dot-com trophy car than trying to come up with a retroactive Band-aid fix for your data, which is increasingly spiraling out of control. You yearn for some normalcy.

If you still aren’t convinced, or if you’re stuck daydreaming of that canary yellow McLaren, here are two proven reasons to normalize your databases:

**Normalize
its benefit
improve
database
speed.**

1. Normal tables won't have duplicate data, which will reduce the size of your database.

Huge bloated database bad...



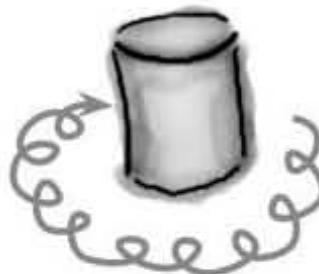
Normalized databases tend to be much smaller than databases with inferior designs.



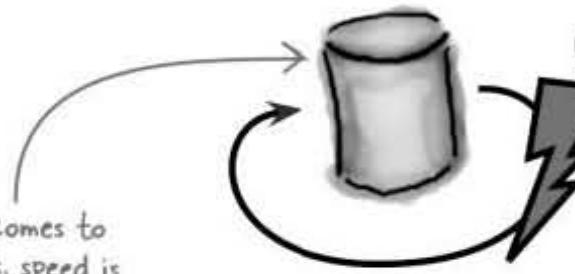
...small, efficient database good...

2. With less data to search through, your queries will be faster.

Slow queries mired in duplicate data bad...



When it comes to databases, speed is always a good thing.



...speedy queries good...

Three steps to a normal database

You've pondered your data for a while and now have a keen appreciation for why it should be normalized, but general ideas only get you so far. What you really need is a concise list of rules that can be applied to any database to ensure normalcy... kinda like a checklist you can work through and use to make sure a database is sufficiently normal. Here goes:

1 Make sure your columns are atomic.

For a column to truly be atomic, there can't be several values of the same type of data in that column. Similarly, there can't be multiple columns with the same type of data.



love	hate
cowboy boots, long hair, reality TV, easy listening music, the opera	tattoos, gold chains, body piercings, professional wrestling, horror movies
tattoos, gold chains, body piercings, cowboy boots, long hair, professional wrestling, horror movies	reality TV, easy listening music, the opera

Several values of the same type of data are in the same column, and there are also multiple columns with the same data... big problem!

2 Give each table its own primary key.

A primary key is critical for assuring that data table can be accessed uniquely. A primary key must be a single column, and ideally be a numeric or type so that queries are as efficient as possible.

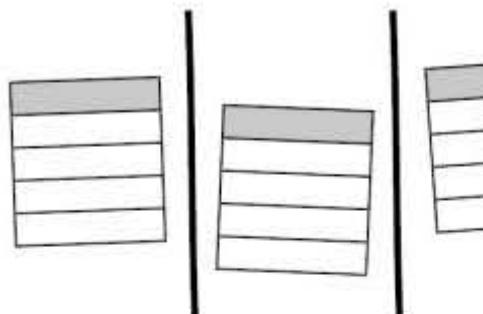
username	password	...
dierdre	08447b...	...
baldpaul	230dc9...	...
jnettles	e511d7...	...
rubyr	062e4a...	...
theking	b4f283...	...



3 Make sure non-key columns aren't dependent on each other.

This is the most challenging requirement of normal databases, and one that isn't always worth adhering to strictly. It requires you to look a bit closer at how columns of data within a given table relate to each other. The idea is that changing the value of one column shouldn't necessitate a change in another column.

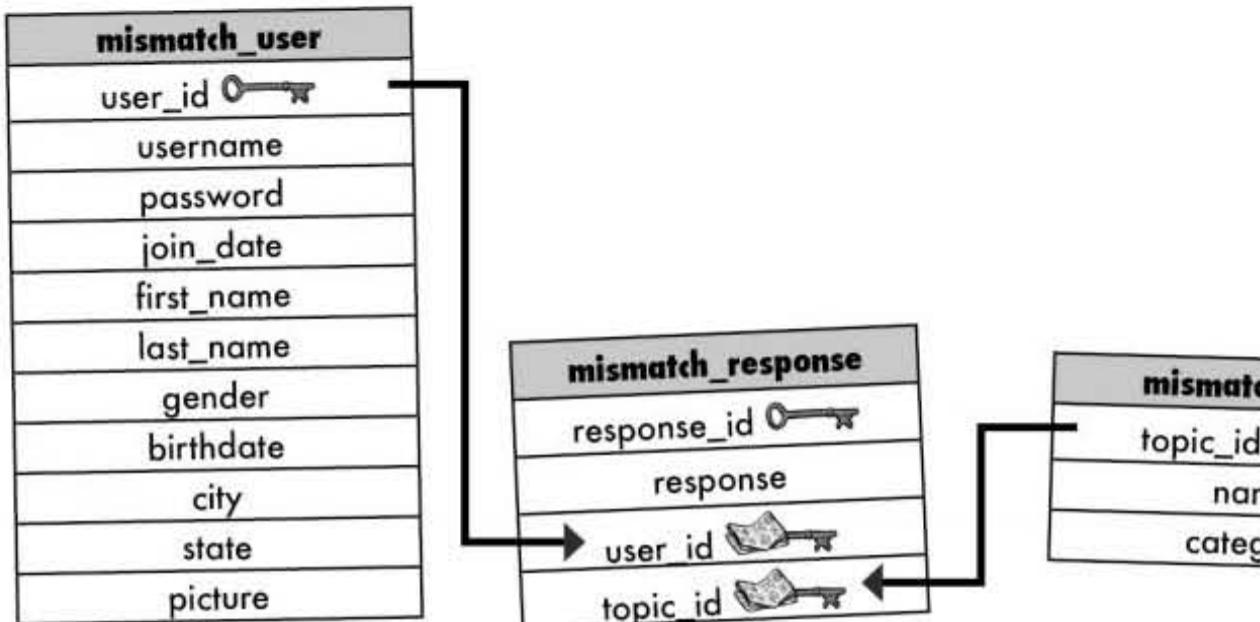
username	password	...	city	state	zip	picture
dierdre	08447b...	...	Cambridge	MA	02138	dierdrepic.jpg
baldpaul	230dc9...	...	Charleston	SC	29401	paulpic.jpg
jnettles	e511d7...	...	Athens	GA	30601	johanpic.jpg
rubyr	062e4a...	...	Conundrum	AZ	85399	rubypic.jpg
theking	b4f283...	...	Tupelo	MS	38801	elmerpic.jpg



The hypothetical ZIP code example shows that changing the city and state columns, which are dependent on the zip code, requires changing the others. We'll need to break out the zip code into its own table with the ZIP code as the primary key.

normalize the mismatch database

The Mismatch database is in need of a normalization overhaul to solve the duplicate category names. Given the existing database structure, sketch a modified design to explain how it works.



there are no Dumb Questions

Q: How do I go about applying the third normalization step to Mismatch to fix the hypothetical city/state/ZIP problem?

A: The solution is to break out the location of a user into its own table, and then connect the `mismatch_user` table to the new table via a foreign key. So you might create a table called `mismatch_location` that has a primary key named `location_id`, along with columns to store the city and state for a user, for example. Then the `city` and `state` columns are removed from `mismatch_user` and replaced with a `location_id` foreign key. Problem solved! What makes this design work is that the `location_id` column actually uses the ZIP code as the primary key, alleviating the non-key dependency problem.

Q: Geez, that seems like a lot of work just to meet a picky database design requirement. Is that really necessary?

A: Yes and no. The first two normalization steps really are non-negotiable because atomic data and primary keys are critical to any good database design. It's the third step where you enter the realm of weighing the allure of an impeccable database design against the practical realities of what an application really needs. In the case of the Mismatch city/state/ZIP problem, it's probably worth accepting for the sake of simplicity. This isn't a decision that should be taken lightly, and many database purists would argue that you should rigidly adhere to all three normalization steps. The good news is that the ZIP code column is purely hypothetical, and not actually part of the `mismatch_user` table, so we don't really have to worry about it.

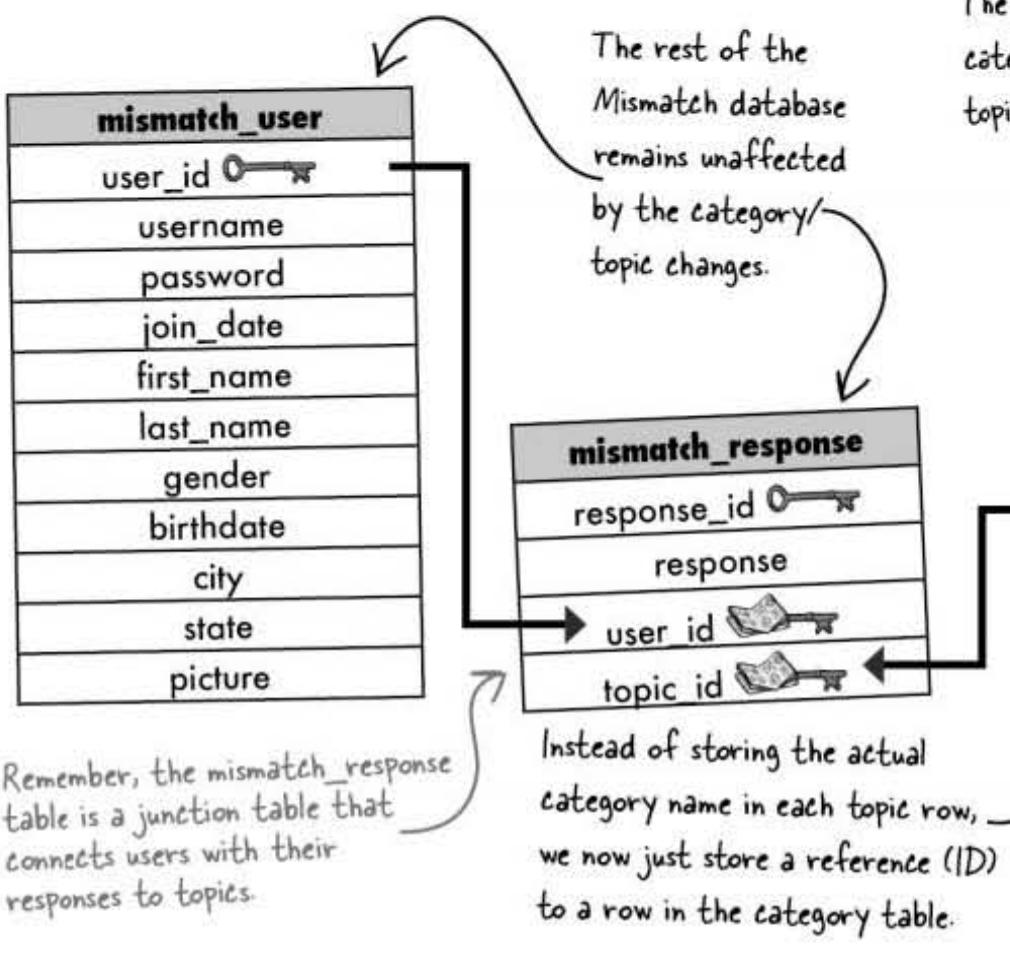
Q: Even without the ZIP code column, wouldn't city and state need to be moved into their own tables to meet the third normalization step?

A: Possibly. It's certainly true that you will end up with duplicate city/state data in the `mismatch_user` table. The problem with breaking out the city and state without a ZIP code is that you would have to somehow populate those tables with every city and state in existence. Otherwise users would no doubt misspell some cities and you'd still end up with problematic data. This is a good example of where you have to seriously weigh the benefits of strict normalization against the realities of a practical application. An interesting possible solution that solves all of the problems is to use a ZIP code in the `mismatch_user` table instead of a city and state, and then look up the city and state from a static table or some other web service as needed. That's more complexity than we need at the moment, so let's just stick with the `city` and `state` columns.

the mismatch database—now normalized!



The Mismatch database is in need of a normalization overhaul to solve the category names. Given the existing database structure, sketch a modified design to explain how it works.



The new category table separates category names from topics, eliminating redundancy.

mismatch
category

mismatch
topic_id
name
category

Each category has a new category ID, one-to-many topics in the mismatch_topic table.

there are no Dumb Questions

Q: How exactly does the new mismatch_category table solve the duplicate data problem?

A: The new table separates category names from the mismatch_topic table, allowing them to be stored by themselves. With categories stored in their own table, it's no longer necessary to duplicate their names—you just have a row for each category, and these rows are then referenced by rows in the mismatch_topic table. This means that category rows in the mismatch_category table have a one-to-many relationship with topic rows in the mismatch_topic table.

Q: So does that mean the mismatch_topic table only has five rows, one for each category?

A: Indeed it does:

category_id
1
2
3
4
5

Each category name is only stored once!

Altering the Mismatch database

In order to take advantage of the new schema, the Mismatch database requires some structural changes. More specifically, we need to create a new `mismatch_category` table, and then connect it to a new foreign key in the `mismatch_topic` table. And since the old `category` column in the `mismatch_topic` table with all the duplicate category data is no longer needed, we can drop it.

```
CREATE TABLE mismatch_category (
    category_id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(48) NOT NULL,
    PRIMARY KEY (category_id)
)
```

Create the new category table that will hold category names by themselves.

```
ALTER TABLE mismatch_topic
DROP COLUMN category
```

Drop the old category column since we're now going to reference categories from the category table.

```
ALTER TABLE mismatch_topic
ADD COLUMN category_id INT NOT NULL
```

mismatch_topic
topic_id
name
topic_id
name
category
category_id

mismatch_topic		
topic_id	name	category_id
...		
8	Horror movies	2
9	Easy listening music	2
10	The opera	2
11	Sushi	3
12	Spam	3
13	Spicy food	3
14	Peanut butter & banana sandwiches	3
15	Martinis	3
16	Howard Stern	4
17	Bill Gates	4
18	Barbara Streisand	4
...		

The new `mismatch_category` data, which is accomplished by these statements.

```
INSERT INTO mismatch_category
VALUES ('Horror movies');
INSERT INTO mismatch_category
VALUES ('Easy listening music');
INSERT INTO mismatch_category
VALUES ('The opera');
INSERT INTO mismatch_category
VALUES ('Sushi');
INSERT INTO mismatch_category
VALUES ('Spam');
```

The new `category_id` column data to correctly wire the category in the `mismatch_topic` table.

```
UPDATE mismatch_topic SET category_id = 3
WHERE name = 'Martinis';
```

test drive the normalized mismatch tables



Test Drive

Create and populate the new mismatch_category database

Using a MySQL tool, execute the CREATE TABLE SQL command on the previous page to add a new table named mismatch_category to the Mismatch database. Then add the INSERT statements to populate the table with category data. Now run the UPDATE statements to modify the mismatch_topic table so that it has a category column. Finally, UPDATE each row in the mismatch_topic table so that its category column points to the correct category in the mismatch_category table.

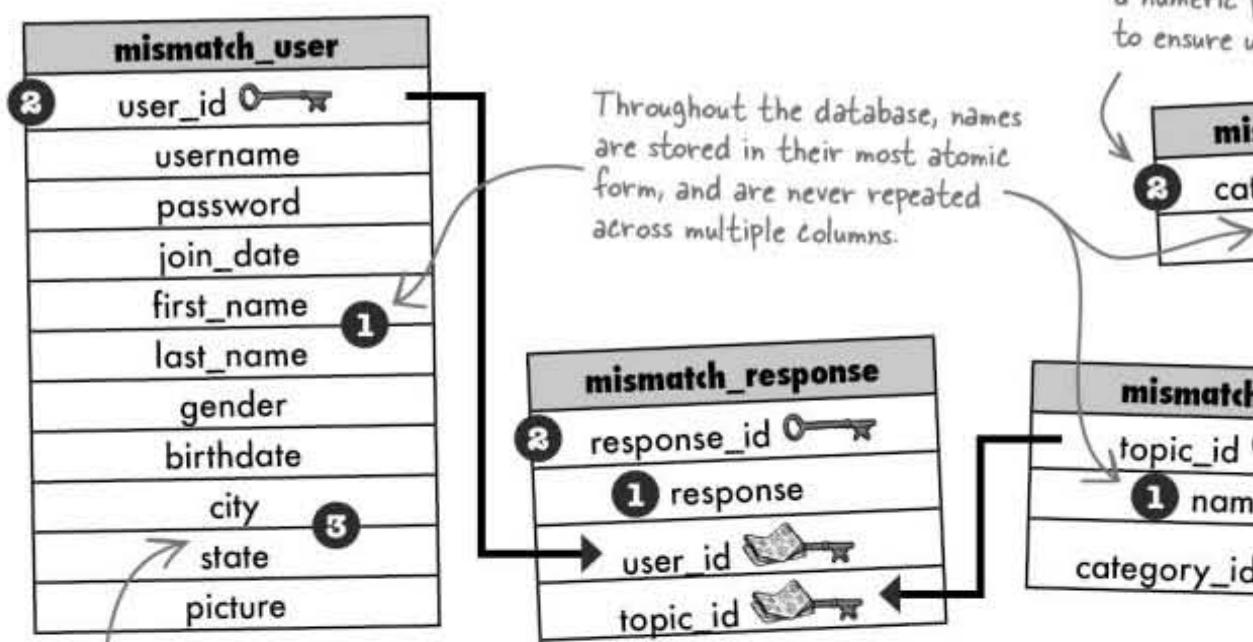
Now run a SELECT on each of the tables just to make sure everything checks out.

So is Mismatch really normal?

Yes, it is. If you apply the three main rules of normalcy to each of the Mismatch tables, you'll find that it passes with flying colors. But even if it didn't, all would not be lost. Just like people, there are **degrees** of normalcy when it comes to databases. The important thing is to attempt to design databases that are completely normal, only accepting something less when there is a very good reason for skirting the rules.

- ➊ Make sure every column has a unique name
- ➋ Give each table a primary key
- ➌ Make sure dependent columns point to primary keys

All of the names are stored in their most atomic form, and are never repeated across multiple columns.



Without the hypothetical ZIP code dependency, the user location columns no longer have dependency problems.

Doesn't the new Mismatch table design affect the queries that are being made in the questionnaire script code?

mismatch_category	
category_id	0 → n
	name

```

...
// Grab the response data from the database to generate the report
$query = "SELECT response_id, topic_id, response FROM mismatch_response
          WHERE user_id = '" . $_SESSION['user_id'] . "'";
$data = mysqli_query($dbc, $query);
$responses = array();
while ($row = mysqli_fetch_array($data)) {
    // Look up the topic name for the response from the topic table
    $query2 = "SELECT name, category FROM mismatch_topic
              WHERE topic_id = '" . $row['topic_id'] . "'";
    $data2 = mysqli_query($dbc, $query2);
    if (mysqli_num_rows($data2) == 1) {
        $row2 = mysqli_fetch_array($data2);
        $row['topic_name'] = $row2['name'];
        $row['category_name'] = $row2['category'];
        array_push($responses, $row);
    }
}
...

```

Yes. In fact, most structural database changes require us to tweak any queries involving affected tables.

In this case, changing the database design to add the new `mismatch_category` table affects any query involving the `mismatch_topic` table. This is because the previous database design had categories stored directly in the `mismatch_topic` table. With categories broken out into their own table, which we now know is a great idea thanks to normalization, it becomes necessary to revisit the queries and code them to work with an additional table (`mismatch_category`).

there must be a better way to perform queries!

A query within a query within a query...

One problem brought on by normalizing a database is that queries often require subqueries since you're having to reach for data in multiple tables. This can get messy. Consider the new version of the query that builds the response array to generate the Mismatch questionnaire form:

```
// Grab the response data from the database to generate the form
$query = "SELECT response_id, topic_id, response FROM mismatch_response "
    "WHERE user_id = '" . $_SESSION['user_id'] . "'";
$data = mysqli_query($dbc, $query);
$responses = array();
while ($row = mysqli_fetch_array($data)) {
    // Look up the topic name for the response from the topic table
    $query2 = "SELECT name, category_id FROM mismatch_topic "
        "WHERE topic_id = '" . $row['topic_id'] . "'";
    $data2 = mysqli_query($dbc, $query2);
    if (mysqli_num_rows($data2) == 1) {
        $row2 = mysqli_fetch_array($data2);
        $row['topic_name'] = $row2['name'];

        // Look up the category name for the topic from the category table
        $query3 = "SELECT name FROM mismatch_category "
            "WHERE category_id = '" . $row2['category_id'] . "'";
        $data3 = mysqli_query($dbc, $query3);
        if (mysqli_num_rows($data3) == 1) {
            $row3 = mysqli_fetch_array($data3);
            $row['category_name'] = $row3['name'];
            array_push($responses, $row);
        }
    }
}
```

Remember, this function tells you how many rows of data were returned by the query.

mismatch_response	
response_id	PK
response	
user_id	FK
topic_id	FK

mismatch_topic	
topic_id	PK
name	
category_id	FK



26	1	Love	Tattoos	Appearance
27	2	Love	Gold chains	Appearance
28	3	Love	Body piercings	Appearance
29	4	Love	Cowboy boots	Appearance
30	5	Love	Long hair	Appearance
31	6	Hate	Reality TV	Entertainment
32	7	Love	Professional wrestling	Entertainment
33	8	Love	Horror movies	Entertainment

This is the temporary response array, which is used to generate the Mismatch questionnaire form.

More tables lead to more

This new query uses category_id key to the category name in the category table.

tables

Let's all join hands

Yikes! Can anything be done about all those nested queries? The solution lies in an SQL feature known as a **join**, which lets us retrieve results from more than one table in a single query. There are lots of different kinds of joins, but the most popular join, an **inner join**, selects rows from two tables based on a condition. In an inner join, query results only include rows where this condition is matched.

```
SELECT mismatch_topic.topic_id, mismatch_category.name
FROM mismatch_topic
```

INNER JOIN mismatch_category

```
ON (mismatch_topic.category_id = mismatch_category.category_id)
```

The category table is joined to the topic table via an INNER JOIN.

The topic ID and category name are selected by the query. These columns are in two different tables.

The condition for the join is that the category ID must match for each row of data returned.

mismatch_topic		
topic_id	name	category_id
1	Tattoos	1
2	Gold chains	1
3	Body piercings	1
4	Cowboy boots	1
5	Long hair	1
6	Reality TV	2
7	Professional wrestling	2
8	Horror movies	2
9	Easy listening music	2
10	The opera	2
11	Sushi	3
...		

This column controls the join!

mismatch_category	
category_id	category_name
1	Appearance
2	Appearance
3	Appearance
4	Appearance
5	Appearance
6	Entertainment
7	Entertainment
8	Entertainment
...	

The first column of results consists of topic IDs from the topic table.

topic_id	category_name
1	Appearance
2	Appearance
3	Appearance
4	Appearance
5	Appearance
6	Entertainment
7	Entertainment
8	Entertainment
...	

This inner join successfully merges data from two tables that would've previously required two separate queries. The query results consist of columns of data from both tables.

The result table has three columns: topic_id, name, and category_name.

A join gets data from multiple tables in a single query.

using dot notation

with Connect the dots

Since joins involve more than one table, it's important to be clear about each column referenced in a join. More specifically, you must identify the table for each column so that there isn't any confusion—tables often have columns with the same names, especially when it comes to keys. Just preface the column name with the table name, and a dot. For example, here's the previous INNER JOIN query that builds a result set of topic IDs and category names:

```
SELECT mismatch_topic.topic_id, mismatch_category.name
FROM mismatch_topic
INNER JOIN mismatch_category
ON mismatch_topic.category_id = mismatch_category.category_id
```

This is the name of the table.

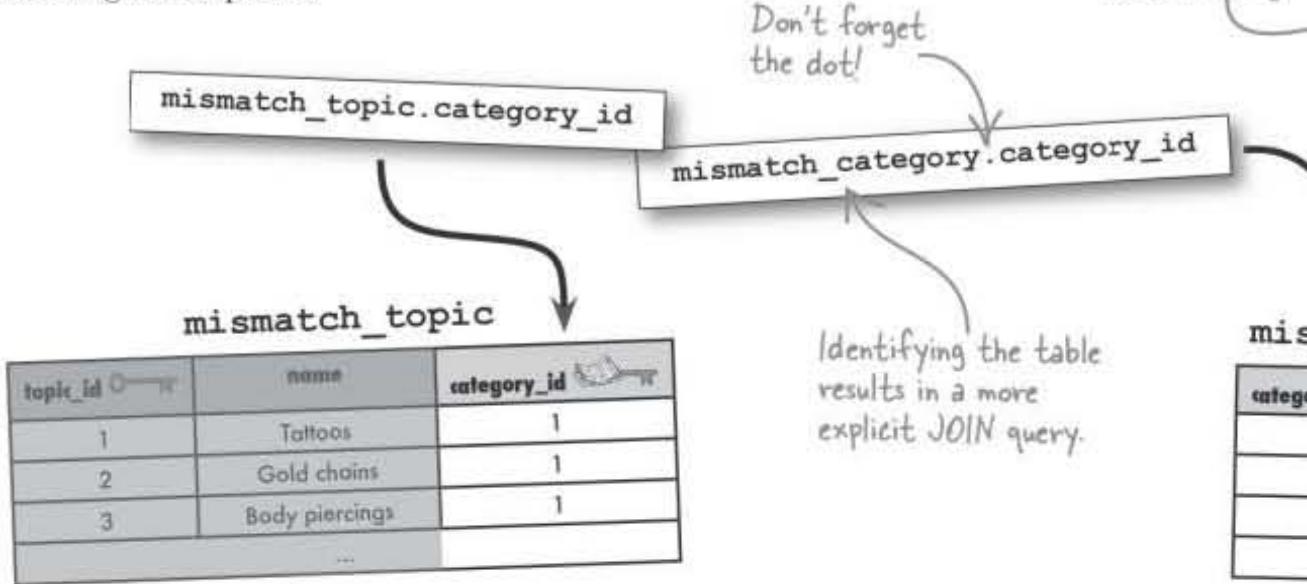
The dot!

This is the name of the column within the table, separated from the table name by a dot (period).

Here's another table/column reference that uses the dot notation.

This is where the dot really pays off – the column names are identical, resulting in total ambiguity without the table names.

Without the ability to specify the tables associated with the columns in this query, we'd have quite a bit of ambiguity. In fact, it would be impossible to understand the ON part of the query because it would be checking to see if the category_id column equals itself, presumably within the mismatch_topic table. For this reason, it's always a good idea to be very explicit about identifying the tables associated with columns when building JOIN queries.



category_id

Within a JOIN query, this column name is ambiguous

mis

categ

Surely we can do more with inner joins

Inner joins don't stop at just combining data from two tables. Since an inner join is ultimately just a query, you can still use normal query constructs to further control the results. For example, if you want to grab a specific row from the set of joined results, you can hang a WHERE statement on the INNER JOIN query to isolate just that row.

```
SELECT mismatch_topic.topic_id, mismatch_category.name
  FROM mismatch_topic
  INNER JOIN mismatch_category
    ON (mismatch_topic.category_id = mismatch_category.category_id)
   WHERE mismatch_topic.name = 'Horror movies'
```

So what exactly does this query return? First remember that the WHERE clause serves as a refinement of the previous query. In other words, it **further constrains** the rows returned by the original INNER JOIN query. As a recap, here are the results of the inner join **without** the WHERE clause:

	topic_id	name	category_id
1	Appearance	1	
2	Appearance	1	
3	Appearance	1	
4	Appearance	1	
5	Appearance	1	
6	Entertainment	2	
7	Entertainment	2	
8	Entertainment	2	
...			

The WHERE clause has the effect of whittling down this result set to a single row, the row whose topic name equals 'Horror movies'. We have to look back at the mismatch_topic table to see which row this is:

mismatch_topic		
topic_id	name	category_id
7	Professional wrestling	2
8	Horror movies	2
9	Easy listening music	2
...		

This row matches the WHERE clause.

This column refines the results as per a WHERE statement.

mismatch
topic_id
name
8
Horror movies

Category names are pulled from the mismatch_category table.

The original INNER JOIN result set is chosen by this solitary row in the WHERE clause.

The WHERE clause constrains the results of the join to a single row.

simplifying queries with the USING statement

*there are no
Dumb Questions*

Q: So a WHERE clause lets you constrain the results of a JOIN query based on rows in one of the joined tables?

A: That's correct. Keep in mind that the actual comparison taking place inside a WHERE clause applies to the **original tables**, not the query results. So in the case of the Mismatch example, the query is retrieving data from two different tables that match on a certain column that appears in both tables (`category_id`), and then only selecting the row where the `name` column in `mismatch_topic` is a certain value ('Horror movies'). So the INNER JOIN takes place with respect to the `category_id` column in, **both tables** but the WHERE clause refines the results using only the `name` column in the `mismatch_topic` table.

Q: Could the WHERE clause in the query be based on the `mismatch_category` table?

A: Absolutely. The WHERE clause can be based on either of the tables involved in the query. The WHERE clause could be changed to look like this:

```
... WHERE mismatch_category.name = ...
```

This WHERE clause limits the result set to rows that fall under the Entertainment category. So it doesn't affect the manner in which the tables are joined, but it does affect the specific rows returned by the query.

Simplifying ON with USING

Remember that our goal is to simplify the messy Mismatch queries with INNER JOIN. When an inner join involves **matching columns with the same name**, we can further simplify the query with the help of the USING statement. The USING statement takes the place of ON in an INNER JOIN query, and requires the name of the column to be used in the match. Just make sure the column is named exactly the same in both tables. As an example, here's the Mismatch query again:

```
SELECT mismatch_topic.topic_id, mismatch_category.name
  FROM mismatch_topic
  INNER JOIN mismatch_category
    ON (mismatch_topic.category_id = mismatch_category.category_id)
   WHERE mismatch_topic.name = 'Horror movies'
```

The name of each of the columns is the same - only the tables are different.

Since the ON part of the query relies on columns with the same name (`category_id`), it can be simplified with a USING statement:

```
SELECT mismatch_topic.topic_id, mismatch_category.name
  FROM mismatch_topic
  INNER JOIN mismatch_category
    USING (category_id)
   WHERE mismatch_topic.name = 'Horror movies'
```

All that is required is the name of the column... no need to specify equality with =.

Rewrite
USING
concise
queries
on a co

The co
must b
in order
USING
in an i

Nicknames for tables and columns

Our INNER JOIN query just keeps getting tighter! Let's take it one step further. When it comes to SQL queries, it's standard to refer to table and columns by their names as they appear in the database. But this can be cumbersome in larger queries that involve joins with multiple tables—the names can actually make a query tough to read. It's sometimes worthwhile to employ an **alias**, which is a temporary name used to refer to a table or column in a query. Let's rewrite the Mismatch query using aliases.

```

SELECT mt.topic_id, mc.name
FROM mismatch_topic AS mt
INNER JOIN mismatch_category AS mc
USING (category_id)
WHERE mt.name = 'Horror movies'
  
```

The code becomes a bit easier to read with the table names condensed into smaller aliases.

The AS keyword in SQL creates an alias, in this case for the mismatch_topic table.

The mismatch_category can now be referred to simply as "mc" thanks to this alias.

Any reference to the mismatch_topic table can now be shortened to "mt".

Are aliases only good for writing more compact queries? In some situations where they're downright essential!

It's quite handy in the Mismatch application is retrieving the topic and category name for a given topic ID. But the mismatch_topic and mismatch_category tables use the same column names for this data. This is a problem because the result of the join would have two columns with the same name. To fix this, we'll rename the result columns to be more descriptive while still using the alias.

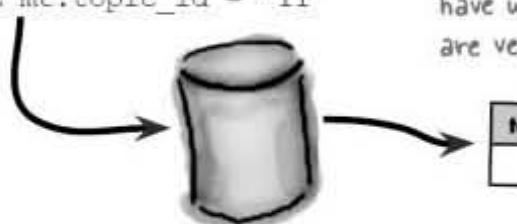
Selecting these two columns in a join yields result columns with the same name...not good!

When a column is renamed with an alias, the alias is what appears in the query results.

An alias
to rename
or column
a query
simplify
in some

```

SELECT mt.name AS topic_name, mc.name
FROM mismatch_topic AS mt
INNER JOIN mismatch_category AS mc
USING (category_id)
WHERE mt.topic_id = '11'
  
```



rewriting the queries with joins

Joins to the rescue

So joins make it possible to involve more than one table in a query, effectively pulling data from more than one place and sticking it in a single result table. The Mismatch query that builds a response array is a perfect candidate for joins since it contains no less than three nested queries for dealing with multiple tables. Let's start with the original code:

```
// Grab the response data from the database to generate the form
$query = "SELECT response_id, topic_id, response FROM mismatch_response " .
    "WHERE user_id = '" . $_SESSION['user_id'] . "'";
$data = mysqli_query($dbc, $query);
$responses = array();
while ($row = mysqli_fetch_array($data)) {
    // Look up the topic name for the response from the topic table
    $query2 = "SELECT name, category_id FROM mismatch_topic " .
        "WHERE topic_id = '" . $row['topic_id'] . "'";
    $data2 = mysqli_query($dbc, $query2);
    if (mysqli_num_rows($data2) == 1) {
        $row2 = mysqli_fetch_array($data2);
        $row['topic_name'] = $row2['name'];

        // Look up the category name for the topic from the category table
        $query3 = "SELECT name FROM mismatch_category " .
            "WHERE category_id = '" . $row2['category_id'] . "'";
        $data3 = mysqli_query($dbc, $query3);
        if (mysqli_num_rows($data3) == 1) {
            $row3 = mysqli_fetch_array($data3);
            $row['category_name'] = $row3['name'];
            array_push($responses, $row);
        }
    }
}
```

And here's the new version of the code that uses a join:

```
// Grab the response data from the database to generate the form
$query = "SELECT response_id, topic_id, response FROM mismatch_response " .
    "WHERE user_id = '" . $_SESSION['user_id'] . "'";
$data = mysqli_query($dbc, $query);
$responses = array();
while ($row = mysqli_fetch_array($data)) {
    // Look up the topic and category names for the response from the topic
    $query2 = "SELECT mt.name AS topic_name, mc.name AS category_name " .
        "FROM mismatch_topic AS mt " .
        "INNER JOIN mismatch_category AS mc USING (category_id) " .
        "WHERE mt.topic_id = '" . $row['topic_id'] . "'";
    $data2 = mysqli_query($dbc, $query2);
    if (mysqli_num_rows($data2) == 1) {
        $row2 = mysqli_fetch_array($data2);
        $row['topic_name'] = $row2['topic_name'];
        $row['category_name'] = $row2['category_name'];
        array_push($responses, $row);
    }
}
```

The last two
are responsible
the topic
name from
tables -

With a join
grab both
category

The topic ID is used
the basis for the
query, but the category
controls the join it



We don't still need two queries, at least no joins to their full potential.

It is possible to join more than two tables, which is what is done in the Mismatch response array code. We need a single query that accomplishes the following three things: retrieve all of the responses from the user, get the topic name for each response, and then get the category name for each response. The new and improved code on the next page accomplishes the last two steps in a single query involving joins between the mismatch_response, mismatch_topic, and mismatch_category tables. Ideally, a single query with two joins would kill all three birds with one big join-shaped stone.



Following is code that is capable of retrieving response data from the database via joins. Be clever and write the SQL query that does the joins between mismatch_response, mismatch_topic, and mismatch_category.

```
// Grab the response data from the database to generate the form
$query = .....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
$data = mysqli_query($dbc, $query);
$responses = array();
while ($row = mysqli_fetch_array($data)) {
    array_push($responses, $row);
}
```

exercise solution and no dumb questions

Following is code that is capable of retrieving response data from the database thanks to the clever usage of joins. Be clever and write the SQL query that joins between the mismatch_response, mismatch_topic, and mismatch_category tables.

```
// Grab the response data from the database to generate the form
$query = "SELECT mr.response_id, mr.topic_id, mr.response,
....."mt.name AS topic_name, mc.name AS category_name";
"FROM mismatch_response AS mr"
"INNER JOIN mismatch_topic AS mt USING(topic_id)"
"INNER JOIN mismatch_category AS mc USING(category_id)"
"WHERE mr.user_id = '".$_SESSION['user_id']."'";

$data = mysqli_query($dbc, $query);
$responses = array();
while ($row = mysqli_fetch_array($data)) {
    array_push($responses, $row);
}

Store all the query
result data in the
$responses array.
```

Aliases are used to help simplify the query and make it a bit easier to read.

The second join uses the category table using the category_id column to access the category name.

there are no Dumb Questions

Q: What other kinds of joins are there?

A: Other types of inner joins include equijoins, non-equijoins, and natural joins. Equijoins and non-equijoins perform an inner join based on an equality or inequality comparison, respectively. You've already seen several examples of equijoins in the Mismatch queries that check for matching `topic_id` and `category_id` columns. Since the matching involves looking for "equal" columns (the same ID), these queries are considered equijoins.

Another kind of inner join is a natural join, which involves comparing all columns that have the same name between two tables. So a natural join is really just an equijoin, in which the columns used to determine the join are automatically chosen. This automatic aspect of natural joins makes them a little less desirable than normal inner joins because it isn't obvious by looking at them what is going on—you have to look at the database structure to know what columns are being used in the join.

Q: So all SQL joins are really just

A: No, there are lots of other joins at your disposal. The major classification of joins is collectively known as outer joins. There are `left` outer joins, `right` outer joins, and the seldom used but awe-inspiring triple outer join. Note that last one isn't a real join, but it should be mentioned. The point of an outer join is that rows in the joined table are included in order to make it into the join. So it's possible to have outer joins that always result in rows from a table because there are no matching conditions.

Outer joins can be just as handy as inner joins, depending on the specific needs of a database application. In fact, there are many different kinds of joins and how they are used depends on the context. For example, in Head First SQL,



Test Drive

Revamp the Questionnaire script to grab the user's response with a single query.

Modify the questionnaire.php script to use inner joins so that the queries for the user's response data are handled in a single query. Upload the new script to your server and then navigate to the questionnaire in a web browser. If all goes well, you should see no difference... but deep down you know the script code is much better built!

The normalized Mismatch table is much less error-prone than the old category table.

Mismatch - Questionnaire

How do you feel about each topic?

Appearance	
Tattoos:	<input type="radio"/> Love <input checked="" type="radio"/> Hate
Gold chains:	<input type="radio"/> Love <input checked="" type="radio"/> Hate
Body piercings:	<input type="radio"/> Love <input checked="" type="radio"/> Hate
Cowboy boots:	<input checked="" type="radio"/> Love <input type="radio"/> Hate
Long hair:	<input type="radio"/> Love <input checked="" type="radio"/> Hate

Entertainment	
Reality TV:	<input checked="" type="radio"/> Love <input type="radio"/> Hate
Professional wrestling:	<input type="radio"/> Love <input checked="" type="radio"/> Hate
Horror movies:	<input type="radio"/> Love <input checked="" type="radio"/> Hate
Easy listening music:	<input checked="" type="radio"/> Love <input type="radio"/> Hate
The opera:	<input checked="" type="radio"/> Love <input type="radio"/> Hate

Food	
Sushi:	<input checked="" type="radio"/> Love <input type="radio"/> Hate
Spam:	<input type="radio"/> Love <input checked="" type="radio"/> Hate
Spicy food:	<input type="radio"/> Love <input checked="" type="radio"/> Hate
Peanut butter & banana sandwiches:	<input checked="" type="radio"/> Love <input type="radio"/> Hate
Martini:	<input type="radio"/> Love <input checked="" type="radio"/> Hate

Now that data isn't duplicated in the database, the form is much more consistent, and therefore less confusing to users.

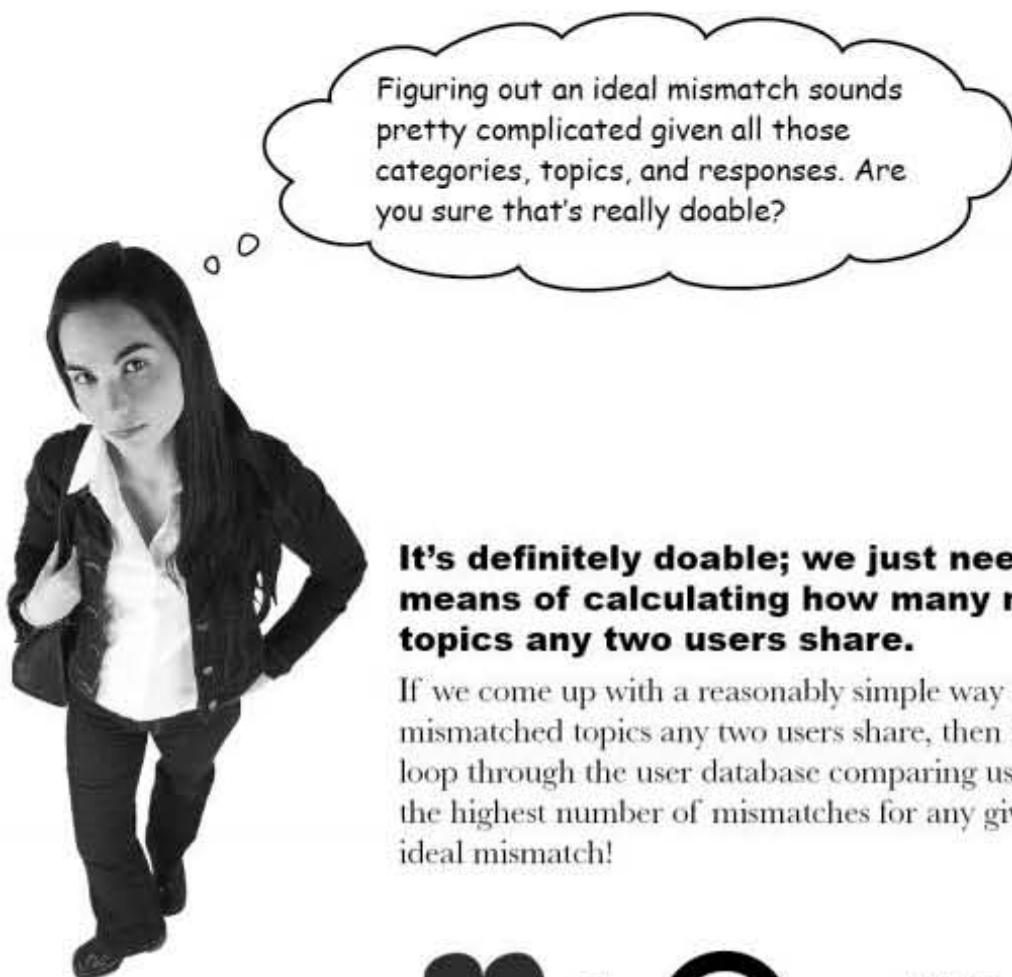
The Mismatch table dynamically generates response tables of join

time to play matchmaker



Mismatch now remembers user responses but it doesn't yet do anything with them...like finding a mismatch!

The collection of user response data only gets us halfway to a successful mismatch. The Mismatch application is still missing a mechanism for firing Cupid's arrow into the database to find a love connection. This involves somehow examining the responses for all the users in the database to see who matches up as an ideal mismatch.



It's definitely doable; we just need a consistent means of calculating how many mismatched topics any two users share.

If we come up with a reasonably simple way to calculate how many mismatched topics any two users share, then it becomes possible to loop through the user database comparing users. The person with the highest number of mismatches for any given user is that user's ideal mismatch!

 +  = **Mismatch!**

Write down how you would go about calculating the “mismatches” of two users using data stored in the Mismatch database:

.....
.....
.....

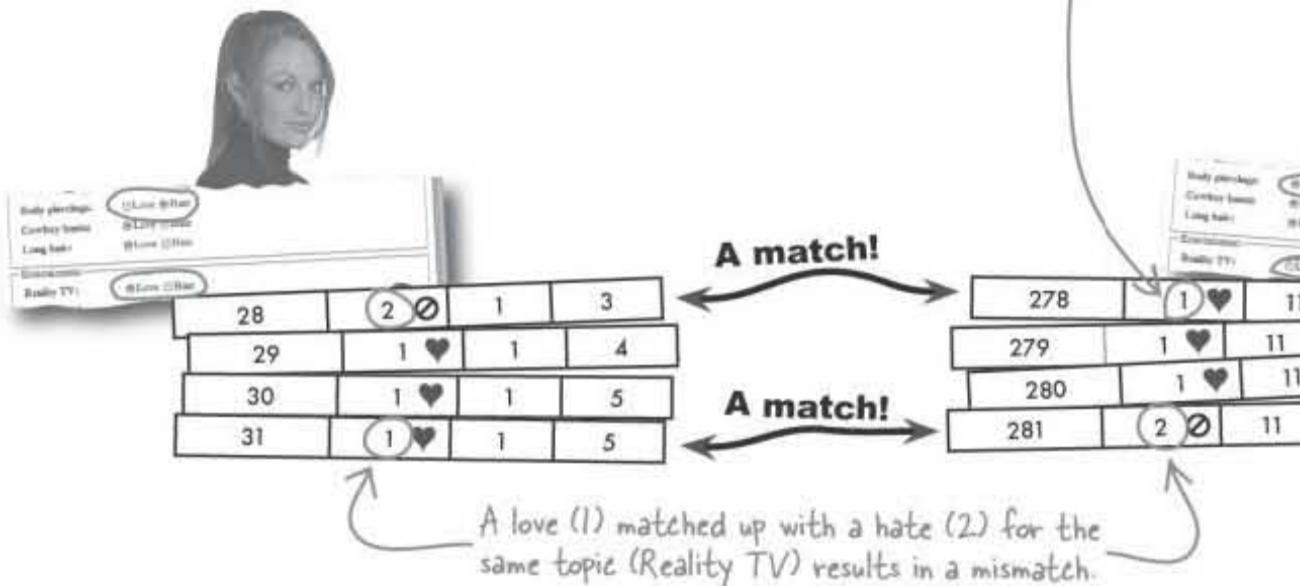
the (mis)matchmaking logic

Love is a numbers game

If you recall, mismatch responses are stored in the mismatch response table as numbers, with 0, 1, and 2 all having special meaning in regard to a specific response.

? Unknown = 0	 Love = 1	 Hate = 2
--------------------------------	--	--

This is the data used to calculate a mismatch between two users, and what we're looking for specifically is a love matching up with a hate or a hate matching up with a love. In other words, we're looking for response rows where response is either a 1 matched against a 2 or a 2 matched against a 1.



	28	29	30	31		278	279	280	281
	(2)	1	3						
	28	1	1	4					
	29	1	1	4					
	30	1	1	5					
	31	1	1	5					

A love (1) matched up with a hate (2) for the same topic (Reality TV) results in a mismatch.

We're still missing a handy way in PHP code to determine when a mismatch takes place between two responses. Certainly a couple of `if-else` statements could be hacked together to check for a 1 and a 2, and then a 2 and a 1, but the solution can be more elegant than that. In either scenario, adding together the two responses results in a value of 3. So we can use a simple equation to detect a mismatch between two responses.

If ResponseA + ResponseB = 3, we have a mismatch!

So finding a love connection really does boil down to simple math. That solves the specifics of comparing individual matches, but it doesn't address the larger problem of how to actually build the My Mismatch script.

Five steps to a successful mismatch

Finding that perfectly mismatched someone isn't just a matter of comparing response rows. The My Mismatch script has to follow a set of carefully orchestrated steps in order to successfully make a mismatch. These steps hold the key to finally satisfying users and bringing meaning to their questionnaire responses.

1 Grab the user's responses from the mismatch_response table, making sure to join the topic names with the results.

2 Initialize the mismatch search including the variables that go with the "best mismatch."

3 Loop through the user table comparing other people's responses to the user's responses. This involves comparing responses for every person in the database to the user's corresponding responses. A "score" keeps up with how many opposite responses the user shares with each person.

4 After each cycle through the loop, see if current mismatch is better than the best mismatch so far. If so, store this one as new "best mismatch," making sure to store away the mismatched topics as well.

5 Make sure a "best mismatch" was found, then query to get more information about the mismatched user, and show the results.

get user responses and initialize search results

Prepare for the mismatch search

Step 1 falls under familiar territory since we've already written some queries that perform a join like this. But we need to store away the user's responses so that we can compare them to the responses of other users later in the script (Step 3). The following code builds an array, \$user_responses, that contains the responses for the logged in user.

```
$query = "SELECT mr.response_id, mr.topic_id, mr.response, mt.name
AS topic_name " .
"FROM mismatch_response AS mr " .
"INNER JOIN mismatch_topic AS mt " .
"USING (topic_id) " .
"WHERE mr.user_id = '" . $_SESSION['user_id'] . "'";
```

\$data = mysqli_query(\$dbc, \$query);
 \$user_responses = array();
 while (\$row = mysqli_fetch_array(\$data)) {
 array_push(\$user_responses, \$row);
 }
 When this loop finishes, the
 \$user_responses array will hold
 all of the user's responses.

A while loop is
 through each r
 results, buildin
 user responses

1 Grab the user's responses from the mismatch_response table to join the topic names.

Step 2 of the My Mismatch script construction process involves setting up some variables that will hold the results of the mismatch search. These variables will be used throughout the My Mismatch script as the search for the best mismatch is carried out:

This is the user ID of the person who is being checked as a potential mismatch... when the search is complete, this variable holds the ID of the best mismatch.

\$mismatch_score = 0;

\$mismatch_user_id = -1;

\$mismatch_topics = array();

This array holds the topics that are mismatched between two users.

This variable holds the score between two highest score ultimately a mismatch.

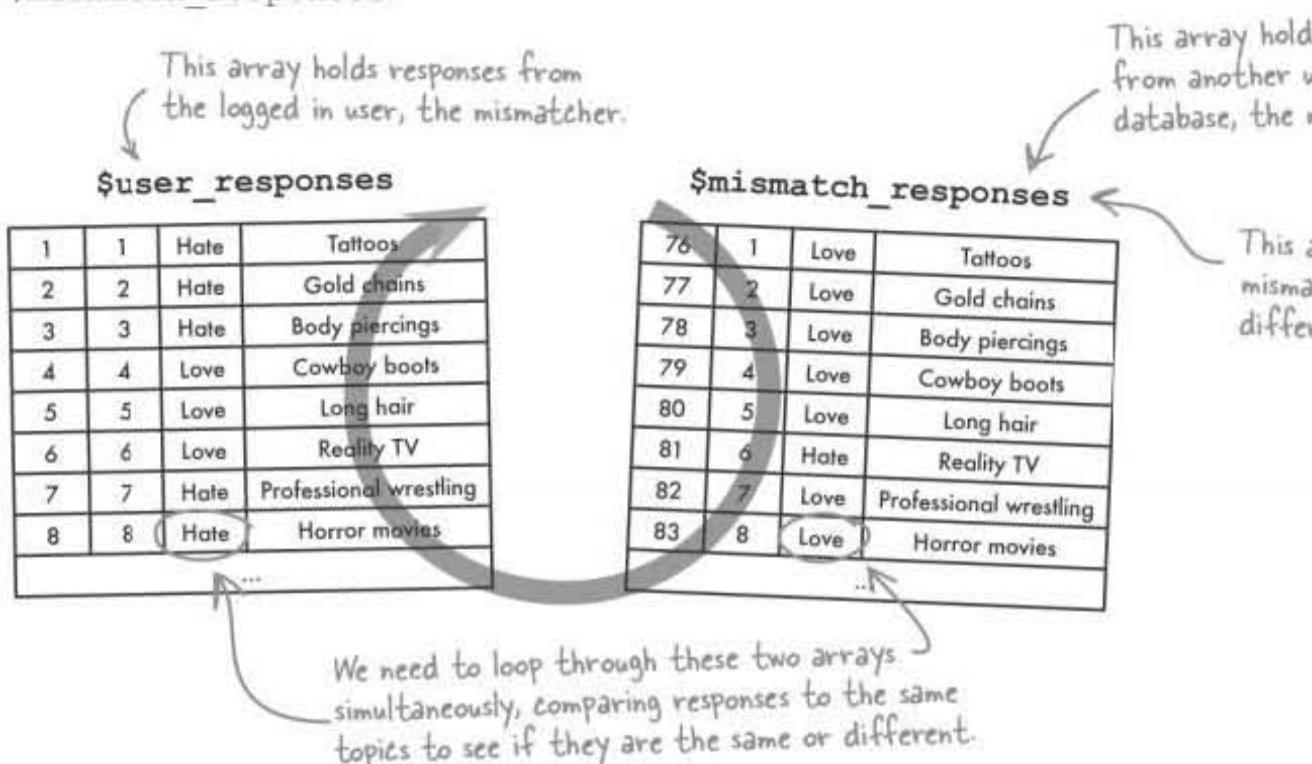
If this variable holds the search, we have a mismatch - this is no other users' questionnaire, v

2 Initialize the mismatch search results, including the variables that keep up with the "best mismatch."

Compare users for "mismatchiness"

The next mismatching step requires looping through every user, and comparing their responses to the responses of the logged in user. In other words, we're taking the logged in user, or **mismatcher** (Sidney, for example), and going through the entire user table comparing her responses to those of each **mismatchee**. We're looking for the mismatchee with the most responses that are opposite of the mismatcher.

Where to begin? How about a loop that steps through the `$user_responses` array (mismatcher responses)? Inside the loop we compare the value of each element with comparable elements in another array that holds the mismatchee responses. Let's call the second array `$mismatch_responses`.



The challenge here is that we need a loop that essentially loops through two arrays at the same time, comparing respective elements one-to-one. A `foreach` loop won't work because it can only loop through a single array, and we need to loop through two arrays **simultaneously**. A `while` loop could work, but we'd have to create a counter variable and manually increment it each time through the loop. Ideally, we need a loop that automatically takes care of managing a counter variable so that we can use it to access elements in each array.

~~foreach (...){~~

Won't work!

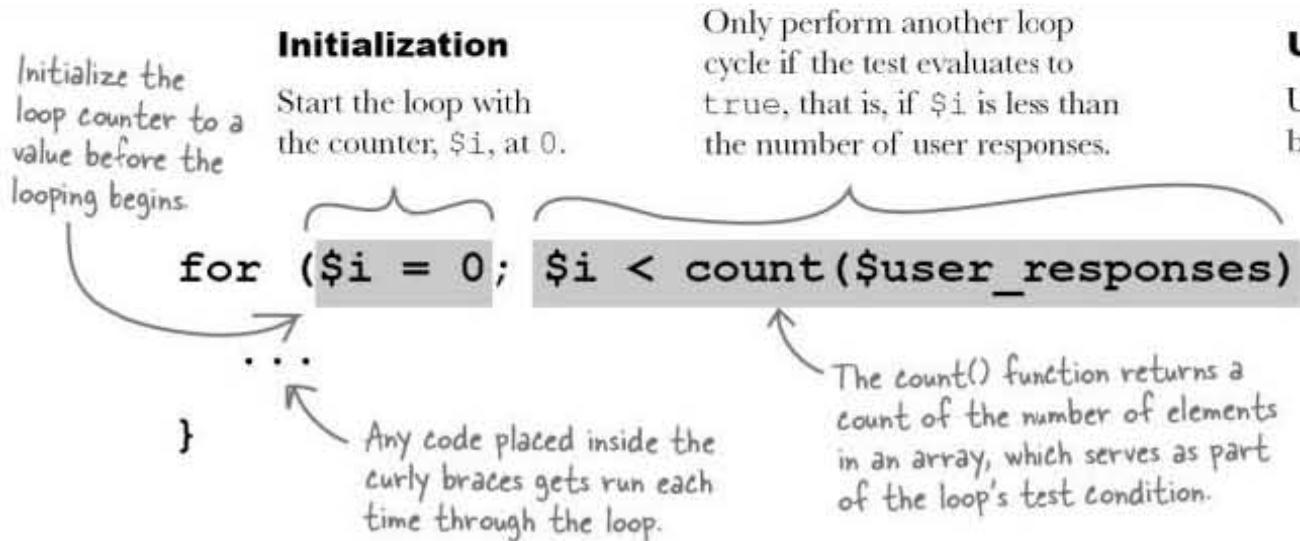
~~while (...){~~

Could work, but not exactly ideal.

introducing the for loop

All we need is a FOR loop

PHP offers another type of loop that offers exactly the functionality we need for the My Mismatch response comparisons. It's called a `for` loop, and it's great for repeating something a certain amount of **known** times. For example, `for` loops are great for counting tasks, such as counting down to zero or counting up to some value. Here's the structure of a `for` loop, which reveals how a loop can be structured to step through an array using a loop counter variable (`$i`).



Exercise

Step 3 of the My Mismatch script involves comparing two users by looping their responses, and calculating a "score" based on how many responses are mismatched. Given the following pieces of data, finish writing the `for` loop that calculates this score.

`$user_responses`

The array of responses for the user.

`$mismatch_responses`

The array of responses for the potential mismatched user.

`$score`

The mismatched score calculated.

```

for ($i = 0; $i < count($user_responses); $i++) {
    if (..... + ..... == .....)
        .....
    array_push($topics, $user_responses[$i]['topic_name']);
}
    
```

^{there are no} Dumb Questions

Q: Why not just use a `foreach` loop to calculate the score instead of a `for` loop?

A: Although a `foreach` loop would work perfectly fine for looping through all of the different responses, it wouldn't provide you with an index (`$i`) at any given iteration through the loop. This index is important because the code is using it to access both the array of user responses and the array of mismatch responses. A `foreach` loop would eliminate the need for an index for one of the two arrays, but not both. So we need a regular `for` loop with an index that can be used to access similar elements of each array.

Q: What's the purpose of storing the mismatched responses in their own array?

A: The array of mismatched responses is important in letting the users know exactly how they compared topically with their ideal mismatch. It's not enough to just share the identity of the ideal mismatch person—what is even better is also sharing the specific topics the user mismatched against that person. This helps give the mismatch result a little more context, and lets the users know a bit more about why this particular person is truly such a great mismatch for them.

Q: In Step 5 of the Mismatch script, how would there ever not be a best mismatch for any given user?

A: Although unlikely, you have to consider the scenario where there is only one user in the entire system, in which case there would be no one else to mismatch that user against.

exercise solution

Step 3 of the My Mismatch script involves comparing two users by looping through their responses, and calculating a "score" based on how many responses are the same. Given the following pieces of data, finish writing the `for` loop that calculates this score.

\$user_responses

1	1	Hate	Tattoos
2	2	Hate	Gold chains
3	3	Hate	Body piercings
4	4	Love	Cowboy boots
5	5	Love	Long hair
6	6	Love	Reality TV
7	7	Hate	Professional wrestling
8	8	Hate	Horror movies
...			

\$mismatch_responses

76	1	Love	Tattoos
77	2	Love	Gold chains
78	3	Love	Body piercings
79	4	Love	Cowboy boots
80	5	Love	Long hair
81	6	Hate	Reality TV
82	7	Love	Professional wrestling
83	8	Love	Horror movies
...			

The score is incremented for each mismatched response found.

Remember, the number of user responses is the same as the number of total topics since the responses come straight from the questionnaire.

```
for ($i = 0; $i < count($user_responses); $i++) {
    if ($user_responses[$i]['response'] + $mismatch_responses[$i]['response'] > $score) {
        $score += 1;
        array_push($topics, $user_responses[$i]['topic_name']);
    }
}
```

The loop counter is used to step through each user response.

Each mismatched topic is added to an array so that it can be displayed to the user when revealing the mismatch.

- DONE**
- 3 Loop through the user table comparing other people's responses to the user's responses.**

Finishing the mismatching

The shiny new loop that calculates a mismatch score is part of a larger script (`mymismatch.php`) that takes care of finding a user's ideal mismatch in the Mismatch database, and then displaying the information.

```
// Only look for a mismatch if the user has questionnaire responses stored
$query = "SELECT * FROM mismatch_response WHERE user_id = '" . $_SESSION['user_id'] . "'";
$data = mysqli_query($dbc, $query);
if (mysqli_num_rows($data) != 0) {
    // It's only possible to find a mismatch for a user who has responded to the questionnaire.

    // First grab the user's responses from the response table (JOIN to get the topic name)
    $query = "SELECT mr.response_id, mr.topic_id, mr.response, mt.name AS topic_name
              FROM mismatch_response AS mr
              INNER JOIN mismatch_topic AS mt
              USING (topic_id)
              WHERE mr.user_id = '" . $_SESSION['user_id'] . "'";

    $data = mysqli_query($dbc, $query);
    $user_responses = array();
    while ($row = mysqli_fetch_array($data)) {
        array_push($user_responses, $row);
    }
    // The $user_responses array holds all of the responses for the user.

    // Initialize the mismatch search results
    $mismatch_score = 0;
    $mismatch_user_id = -1;
    $mismatch_topics = array();
    // These variables keep track of the mismatch search as it progresses.
```

mymis

hang on,
more -

the full mymismatch.php script

```

// Loop through the user table comparing other people's responses to
$query = "SELECT user_id FROM mismatch_user WHERE user_id != '' . $s_";
$data = mysqli_query($dbc, $query);
while ($row = mysqli_fetch_array($data)) {
    // Grab the response data for the user (a potential mismatch)
    $query2 = "SELECT response_id, topic_id, response FROM mismatch_respo
3    "WHERE user_id = '" . $row['user_id'] . "'";
    $data2 = mysqli_query($dbc, $query2);
    $mismatch_responses = array();
    while ($row2 = mysqli_fetch_array($data2)) {
        array_push($mismatch_responses, $row2);
    }
}

This curly brace marks the end of the main while loop.
// Compare each response and calculate a mismatch total
$score = 0;
$topics = array();
for ($i = 0; $i < count($user_responses); $i++) {
    if ((int)$user_responses[$i]['response'] + ((int)$mismatch_respo
        $score += 1;
        array_push($topics, $user_responses[$i]['topic_name']);
    }
}

// Check to see if this person is better than the best mismatch so
if ($score > $mismatch_score) {
    // We found a better mismatch, so update the mismatch search resu
    $mismatch_score = $score;
    $mismatch_user_id = $row['user_id'];
    $mismatch_topics = array_slice($topics, 0);
}

DONE 4 This function extracts a "slice" of an array
    In this case we're just using it to copy the $topics array into $mismatch_topics.
    ...
    We're still inside that first if statement from the previous page, and there's still more code...

```

For each user, this code grabs the question responses for comparison as a potential mismatch.

Here's how the calculation for a potential mismatch is done.

If this user is a better mismatch than the best mismatch so far, then save him as the best mismatch.

DONE

5

```

// Make sure a mismatch was found
if ($mismatch_user_id != -1) {
    $query = "SELECT username, first_name, last_name, city, state, picture
              WHERE user_id = '$mismatch_user_id'";
    $data = mysqli_query($dbc, $query);
    if (mysqli_num_rows($data) == 1) {
        // The user row for the mismatch was found, so display the user
        $row = mysqli_fetch_array($data);
        echo '<table><tr><td class="label">';
        if (!empty($row['first_name']) && !empty($row['last_name'])) {
            echo $row['first_name'] . ' ' . $row['last_name'] . '<br />';
        }
        if (!empty($row['city']) && !empty($row['state'])) {
            echo $row['city'] . ', ' . $row['state'] . '<br />';
        }
        echo '</td><td>';
        if (!empty($row['picture'])) {
            echo '</tr></table>';

        // Display the mismatched topics
        echo '<h4>You are mismatched on the following ' . count($mismatch_topics) . '<br />';
        foreach ($mismatch_topics as $topic) {
            echo $topic . '<br />';
        }

        // Display a link to the mismatch user's profile
        echo '<h4>View <a href=viewprofile.php?user_id=' . $mismatch_user_id . '>' .
             $row['first_name'] . '\'s profile</a>.</h4>';
    }
}
else {
    echo '<p>You must first <a href="questionnaire.php">answer the questionnaire</a> to
          be mismatched.</p>';
}

```

Before displaying the mismatch results, make sure a "best mismatch" was actually found.

control

Query for user's information to display it.

Show city

Don't forget to generate an `` with the user's picture.

It's important to show what topics actually resulted in the mismatch.

Finally, we provide a link to the mismatched user's profile so that the logged in user can find out more about him.

make a mismatch!



Test DRIVE

Find your perfect Mismatch!

Modify Mismatch to use the new My Mismatch script (or download the application from the Head First Labs site at www.headfirstlabs.com/books/hfphp). This involves creating a new `mymismatch.php` script, as well as adding a “My Mismatch” link to the `navmenu.php` script so that users can access the script.

Upload the scripts to your web server, and then open the main Mismatch page (`mismatch/mismatch.php`) in a web browser. Make sure you log in and have filled out the questionnaire, and click the “My Mismatch” menu item to view your mismatch.

Johan's My Mismatch page reveals Sidney as his perfect opposite.

You are mismatched on:

- Tattoos
- Gold chains
- Body piercings
- Reality TV
- Professional wrestling
- Horror movies
- Easy listening music
- The opera
- Sushi
- Spam
- Peanut butter & banana sandwiches
- Martini
- Howard Stern
- Bill Gates
- Barbara Streisand
- Hugh Hefner
- Martha Stewart
- Yoga
- Weightlifting
- Cube puzzles
- Karaoke
- Hiking

[View Sidney's profile](#)

You are mismatched on the following 22 topics:

- Tattoos
- Gold chains
- Body piercings
- Reality TV
- Professional wrestling
- Horror movies
- Easy listening music
- The opera
- Sushi
- Spam
- Peanut butter & banana sandwiches
- Martini
- Howard Stern
- Bill Gates
- Barbara Streisand
- Hugh Hefner
- Martha Stewart
- Yoga
- Weightlifting
- Cube puzzles
- Karaoke
- Hiking

[View Johan's profile](#)

When Sidney visits the My Mismatch page, she sees Johan, her ideal mismatch.

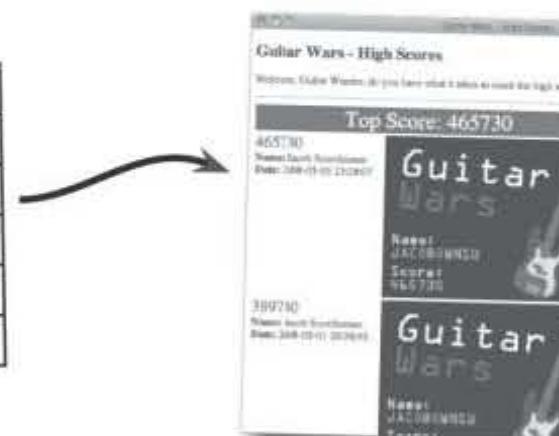


Database Schema Magnets

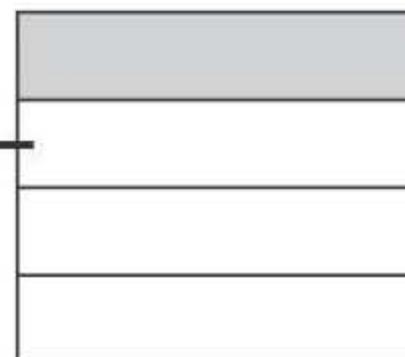
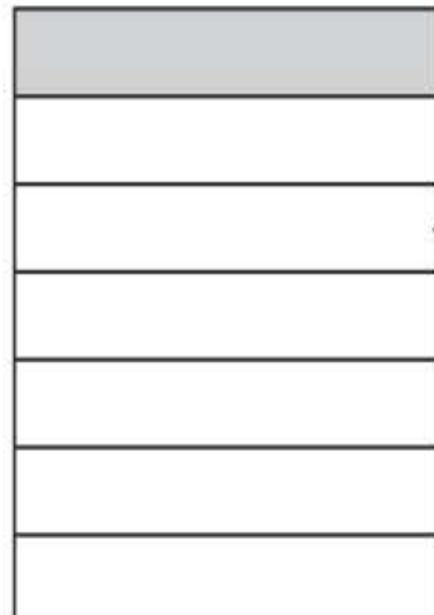
Remember the Guitar Wars application from eons ago? Your job is to study the Guitar Wars database, which could use some normalization help, and come up with a better schema. Use all of the magnets below to flesh out the table and column names, and also identify the primary and foreign keys.

Here's the original
Guitar Wars database,
which stores high
scores that are
submitted by users.

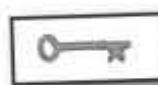
guitarwars
date
name
score
screenshot
approved



Here's the new and improved
schema you need to build out
with the magnets... good luck!



screenshot

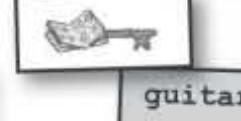


last_name

score_id

guitarwars_player

player_id



score



guitar

date

approved

database schema magnets solution



Database Schema Magnets Solution

Remember the Guitar Wars application from eons ago? Your job is to study the Guitar Wars database, which could use some normalization help, and come up with a better schema. Use all of the magnets below to flesh out the table and column names, and also identify the primary and foreign keys.

The table is missing a primary key, which is an important part of any normalized database.

guitarwars	
date	
name	name
score	score
screenshot	
approved	

Since the same user is capable of posting multiple high scores, the name column results in redundant data... not good!

The new score_id column serves as a much needed primary key for the score table.

guitarwars_score	
3 score_id	key
player_id	key
date	
score	
screenshot	
approved	

The score table references players via a new foreign key.

The tables have new names since they serve more specific purposes.

guitarwars_player	
3 player_id	key
first_name	
last_name	

A one-to-many relationship between players and high scores!

Reduced data
a new player
connection table

- 1 Make sure your columns are atomic.
- 2 Give each table its own primary key.
- 3 Make sure non-key columns aren't dependent on each other.

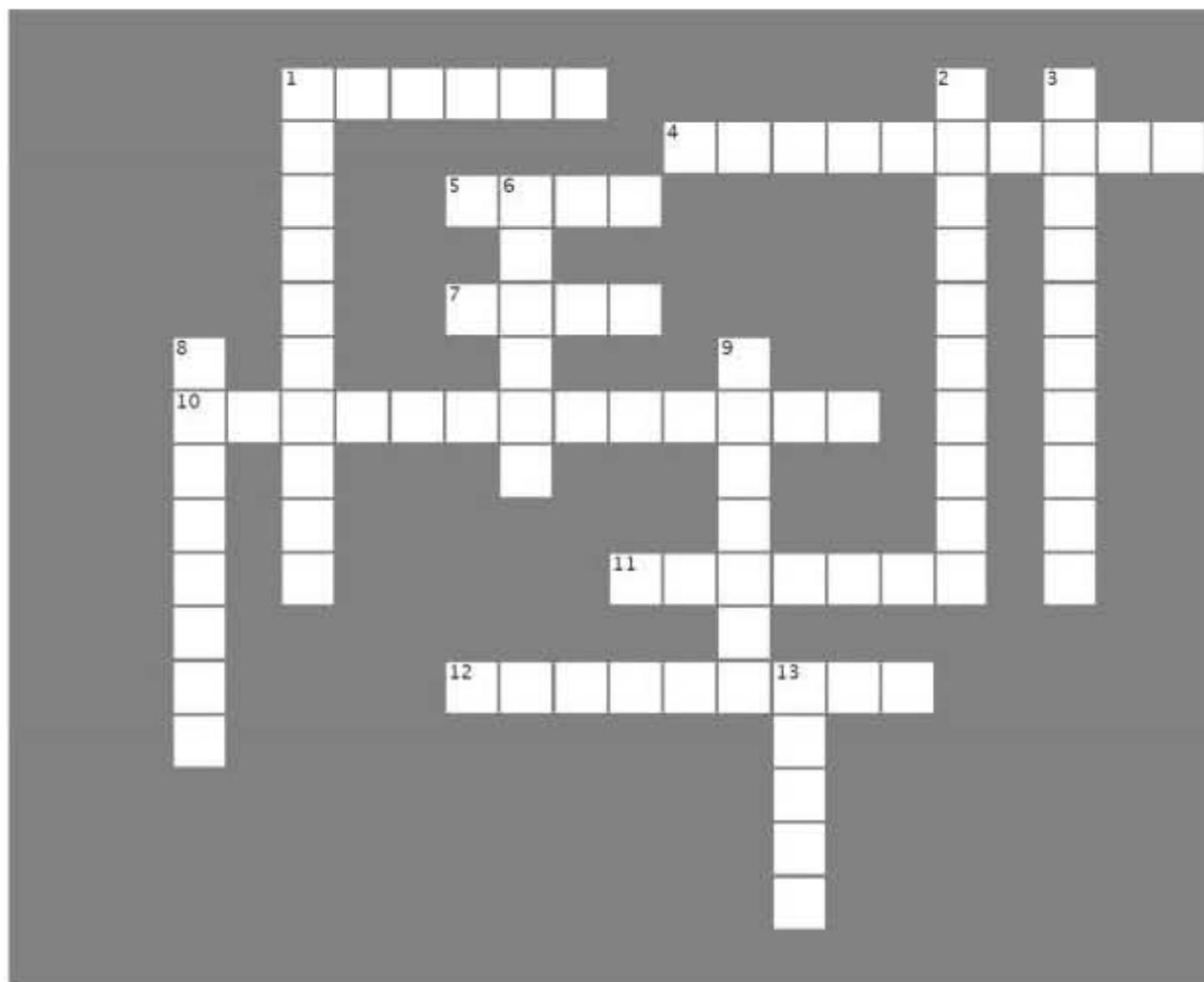
Here are the rules one more time, just to make sure you didn't forget!

The Guitar Wars database does not have any dependent columns.



PHP&MySQLcross

Concerned that your own perfect mismatch is still out there waiting to be found? Take your mind off it by completing this crossword puzzle.



Across

1. A representation of all the structures, such as tables and columns, in your database, along with how they connect.
4. This happens when multiple rows of a table are related to multiple rows of another table.
5. This allows you to convert between different PHP data types.
7. Use this to combine results from one table with the results of another table in a query.
10. The process of eliminating redundancies and other design problems in a database.
11. You can shorten some if-else statements with this handy little operator.
12. When one row of a table is related to multiple rows in another table.

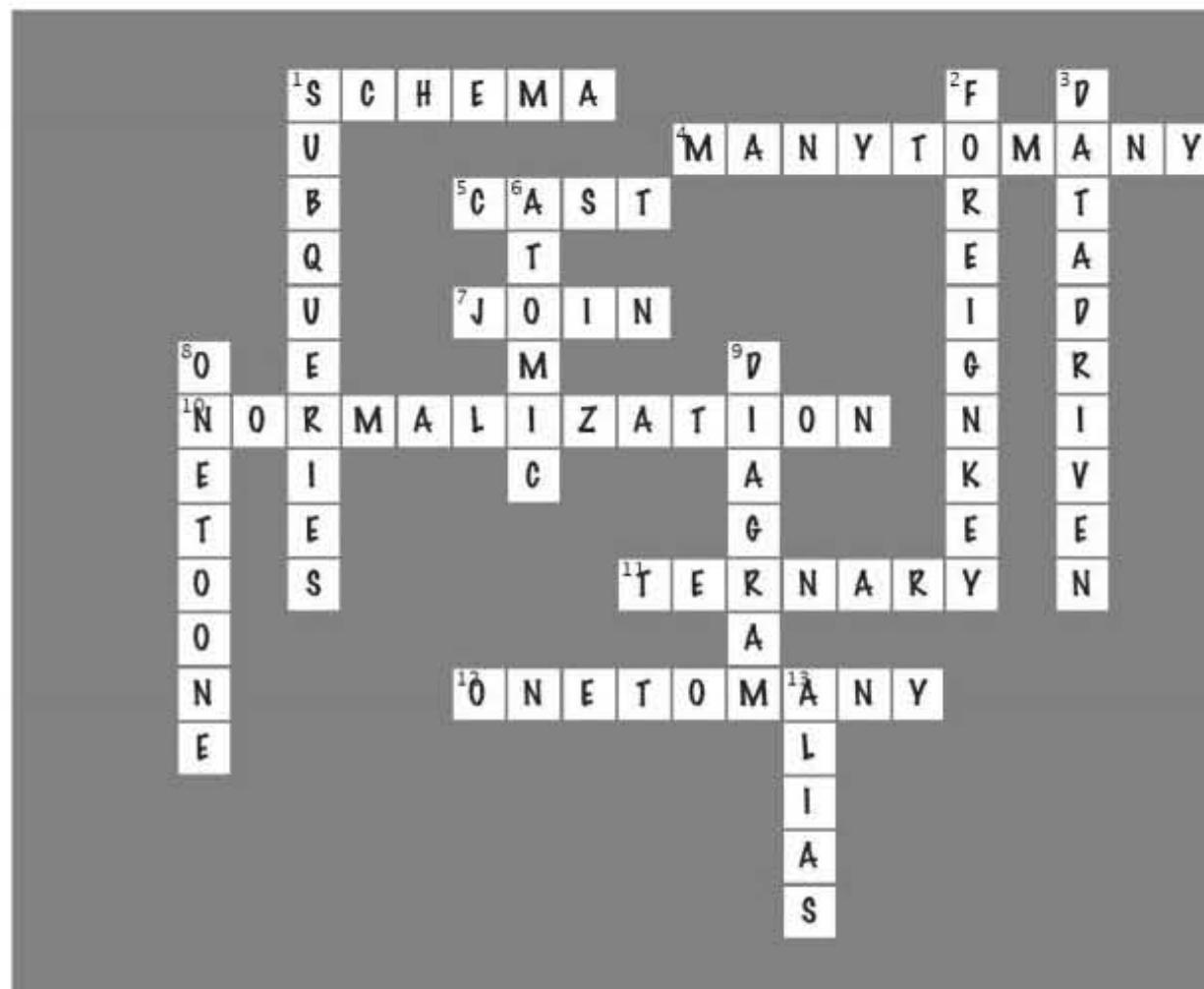
Down

1. A join makes it possible to get rid of these.
2. A column in a table that references the primary table.
3. When a form is generated from a database, -----.
6. It's not nuclear, it's just data in the smallest sense for a given database.
8. Rows in two tables have this relationship when exactly one row in one table for every row in the other.
9. One of these can help greatly in figuring out a table.
13. A temporary name used to reference a piece of data in a query.

php&mysqlcross solution



PHP&MySQLcross Solution



control your



Your PHP & MySQL Toolbox

Quite a few new MySQL database techniques were uncovered in this chapter, not to mention a few new PHP tricks. Let's do a quick recap!

Schemas and Diagrams

A schema is a representation of all the structures (tables, columns, etc.) in your database, along with how they connect. A diagram is a visual depiction of your database, including details about the specific columns responsible for connecting tables.

Foreign Key

A column in a table that is used to link the table to another table. A foreign key in a child table typically connects to a primary key in a parent table, effectively linking rows between the two tables.

? :

The ternary operator is a PHP construct that works like a really compact if-else statement. It is handy for performing simple choices based on a true/false expression.

INNER JOIN

This kind of join combines data from two tables that have matching rows. Unlike a normal query, a join allows you to grab data from more than one table, which is extremely helpful when a database consists of multiple tables.

Normalization

Normalization is the process of altering the design of a database to reduce duplicate data and improve the placement of data. The goal is to produce a database that holds up well to scaling.

for (...)

A loop that is identical to looping based on a fixed number of iterations. A for loop by itself establishes a template for specifying how many times a block of code should be updated after each iteration.

AS name

This SQL statement creates an alias, which identifies a piece of data in a query. Aliases can simplify queries by replacing a long table and column name with a shorter one that can also be used to refer to the original column when the original column name isn't specific enough.

9 string and custom functions

Better living through functions



Functions take your applications to a whole new level.

You've already been using PHP's built-in functions to accomplish things. Now it's time to take a look at a few more really useful **built-in functions**. And then you'll learn to build your very own **custom functions** to take you farther than you ever imagined it was possible to go. Well, maybe not to the point of raising laser sharks, but custom functions will streamline your code and make it reusable.

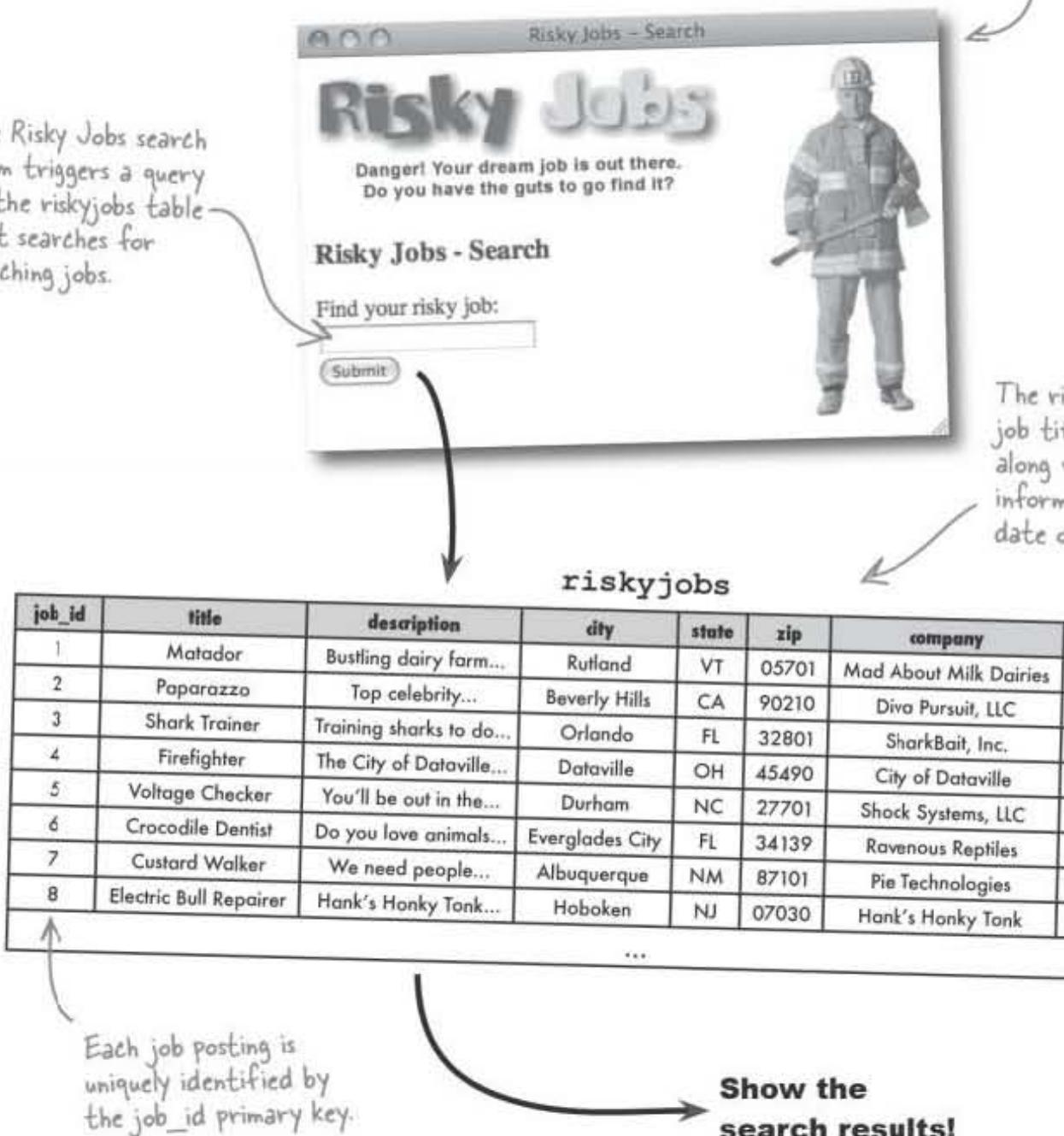
riskyjobs needs search functionality on its site

A good risky job is hard to find

The Internet startup, RiskyJobs.biz, is designed to help companies find the right people to fill their riskiest jobs. The business model is simple: for each risky job we can fill with the right candidate, we get a commission. The more successful matches, the bigger our profit.

Risky Jobs needs help improving its site's job-search functionality. Right now, there's a database full of risky jobs just waiting to be discovered by the right people. Let's take a look at the Risky Jobs search form and the underlying database of available jobs.

This si
form o
that s
riskyjo





Ernesto, our fearless bullfighter, is seeing red because his job search isn't producing any results.

Sharpen your pencil



When the Risky Jobs form is submitted, the search variable \$user_search is used in the SQL query to do the actual searching. Write down the result of Ernesto's search.

```
$search_query = "SELECT job_id, title, state, description FROM riskyjobs
                WHERE title = '$user_search'";
$result = mysqli_query($dbc, $search_query);
```

Write your answer here!

we need our queries to be more flexible



Sharpen your pencil Solution

A WHERE clause with an = means that the two strings being compared must match exactly.

```
$search_query = "SELECT job_id, title, state, description FROM riskyjobs
    WHERE title = '$user_search'";
```

\$result = mysqli_query(\$dbc, \$search_query);

This variable contains whatever is entered into the text box.

When the Risky Jobs form is submitted, the search variable \$user_search, which is plugged into the SQL query to do the actual searching. Write down in the riskyjobs database on page 502 will be the result of Ernesto's search.

Zip, zero, nada! The problem is that our query is too exact—the exact text entered by the user must match.

The search leaves no margin for error

The SELECT query in the Risky Jobs script is very rigid, only resulting in a match if the two strings being compared are **identical**. This presents a problem for our job search because people need to be able to enter search terms that match job listings even if the job title isn't an exact match.

Let's go back to Ernesto's search, which results in a query that searches the title column of the riskyjobs table for the text "Bull Fighter Matador":

The case of the string doesn't matter in MySQL WHERE clauses because it's case-insensitive.

```
SELECT job_id, title, description FROM riskyjobs
    WHERE title = 'Bull Fighter Matador'
```

The = operator requires an exact match when comparing two strings.

See the problem? This query is only going to match rows in the table where the title column contains the exact text "Bull Fighter Matador". The job with the title "Matador" isn't matched, and neither are "Firefighter" or "Electric Bull Repairer". OK, maybe it's good those last two were missed, but the search still isn't working as intended. And it's not the mixed case that presents the problem (MySQL searches are by default **case-insensitive**), it's the fact that the entire search string must be an exact match due to the equality (=) operator in the WHERE clause.

SQL queries can be flexible with LIKE

What we really need is a way to search the database for a match on any portion of a search string. SQL lets us do just that with the `LIKE` keyword, which adds flexibility to the types of matches returned by a `WHERE` clause. You can think of `LIKE` as a more forgiving version of the `=` operator. Take a look at the following query, which uses `LIKE` to match rows where the word “fighter” appears **anywhere** in the `title` column:

```
SELECT job_id, title, description FROM riskyjobs
WHERE title LIKE '%fighter%'
```

The keyword `LIKE` lets you look for matches that aren't exactly the same as the word in quotes... and still case-insensitive.

The `%` signs are wildcards; they stand in for any other characters before or after the word.

`LIKE` makes it much easier to find matches, especially when you need to match the search string as part of a larger word or phrase. Check out these examples of strings that match up with the above query:

Fire**fighter**

Prize **Fighter**

FightErnestoPlease

LIKE clauses typically work in conjunction with **wildcard characters**, which are stand-ins for characters in the data we're matching. In SQL, the percent sign (%) wildcard stands for any group of zero or more characters. Placing this wildcard in a query before and after a search term, as in the `SELECT` statement above, tells SQL to return results whenever the term appears somewhere in the data, no matter how many characters appear before or after it.



Geek Bits

SQL has another wildcard character that can be used with `LIKE`. It's the underscore character (_), which represents a single character. For example, the following `LIKE` clause:

`LIKE '_fig' + fighter + '%'`

It's saying: “Find the string “fig” followed by four characters in front of it, and then any number of characters after it.” This would match “bulldogfighter” but not “streetfighter”.

riskyjobs code setup

Time out! Take a moment to familiarize yourself with the Risky Jobs database... and try out a few searches.

Download the `riskyjobs.sql` file for the Risky Jobs application from the Head First Labs web site at www.headfirstlabs.com/books/hfphp. This file contains statements that build and populate the `riskyjobs` table with sample data.

After you've executed the statements in `riskyjobs.sql` in a MySQL tool, try out queries to simulate job searches. Here are some to get you started.

```
SELECT * FROM riskyjobs
```

This query selects all columns for all jobs in the `riskyjobs` table.

```
SELECT job_id, title, description FROM riskyjobs  
WHERE title = 'Bull Fighter Matador'
```

This query finds the job title, description where the title is "Bull Fi

```
SELECT job_id, title, description FROM riskyjobs  
WHERE description LIKE '%animals%'
```

This query uses `LIKE` to find jobs with the word "animals" anywhere in the job description.

Download It!

The complete source code for this application is available for download on the Head First Labs web site:

www.headfirstlabs.com/books/hfphp



LIKE Clause Magnets

A bunch of LIKE clauses are all scrambled up on the fridge.
Can you match up the clauses with their appropriate results? ←
Which magnets won't be matched by any of the LIKE clauses?

Some magnets
multiple

LIKE '%er'

LIKE

LIKE 'c%'

LIKE '%test %'

LIKE '%Ti%

LIKE '%do_'

LIKE '%m%

Human Cannonball

Cliff Diver

Team Mascot

Crash Test Dummy

Pet Food Tester

Rodeo Clown

Cat F...

Matador

Cow Tipper

Politician

Shark Fin...

LIKE clause magnets solution

LIKE Clause Magnets Solution

A bunch of LIKE clauses are all scrambled up on the fridge. Can you match up the clauses with their appropriate results? (Some may have multiple answers.) Which magnets won't be matched by any of the LIKE clauses?

LIKE '%er'

Anything ending in "er" matches this.

Pet Food Tester

Shark Finder

Cow Tipper

Cliff Diver

Snake Charmer

Cat Herder

LIKE '%

Pet Food Teste

Crash Test Dummy

LIKE '%test %'

Crash Test Dummy

SQL queries are case-insensitive, so words starting with either lowercase or uppercase 'c' are matched by this query.

LIKE 'c%'

Cow Tipper

Cliff Diver

Cat Herder

Only one ch after "do".

The letters "ma" appearing anywhere give us a match.

LIKE '%ma%'

Human Cannonball

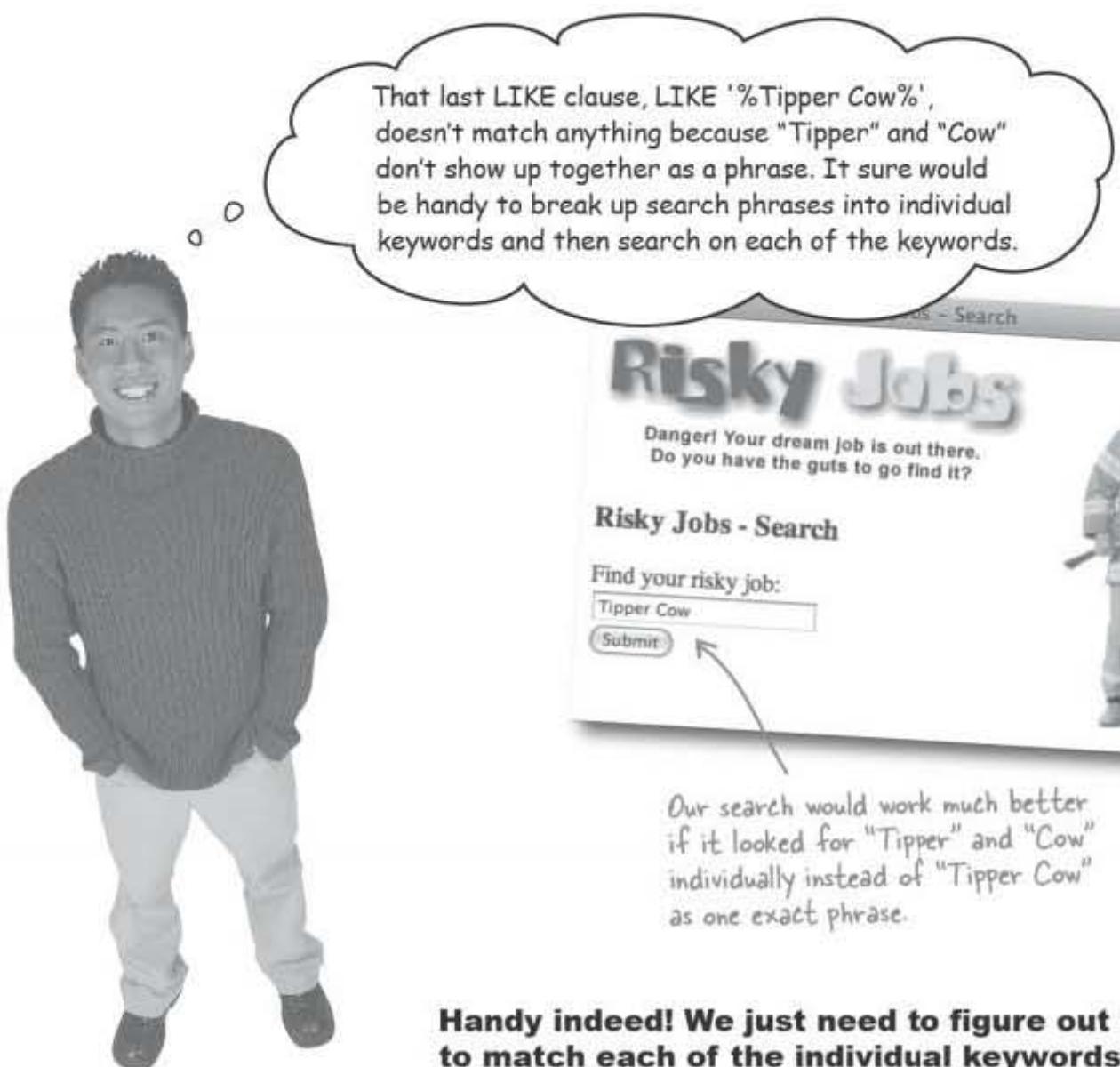
Team Mascot

These are the unmatched phrases.

Rodeo Clown

Politician

LIKE '%Ti



Handy indeed! We just need to figure out how to match each of the individual keywords in search phrase.

Taking what people type in the Risky Jobs search field and matching it exactly won't always work. The search would be much more accurate if we searched on each search term entered, as opposed to searching on the entire search phrase. But how do you search on multiple terms? We could store each of the search terms in an array, and then run the SELECT query to search for each keyword individually.

the php explode() function

Explode a string into individual words

To make Risky Jobs search functionality more effective, we need a way to break apart users' search strings when they enter multiple words in the form field. The data our risky job seekers enter into the search form is text, which means we can use any of the built-in PHP string functions to process it. One extremely powerful function is `explode()`, and it breaks apart a string into an array of separate strings. Here's an example:

The `explode()` function chops a string into an array of substrings based on a common separator, also known as a delimiter.

```
$search_words = explode(' ', 'Tipper Cow');
```

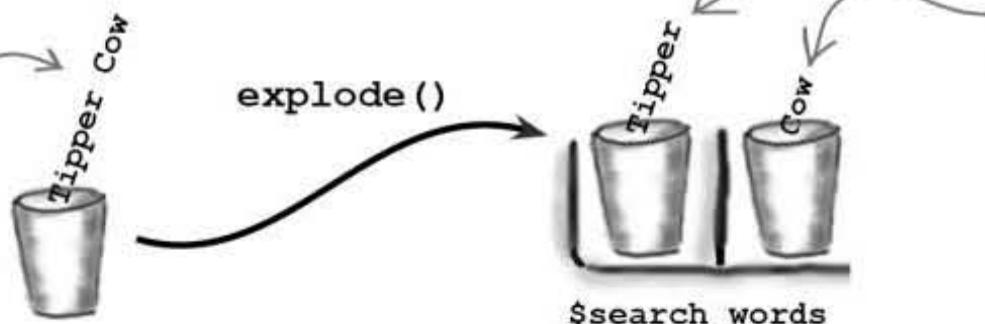
The `$search_words` variable now stores the array of search terms that we'll then feed into an SQL query.

This parameter is the text we want "exploded".

This parameter tells `explode()` what separates the substrings within the string: in this case a space. You can specify one or more characters, which are called the delimiter.

The `explode()` function requires two parameters. The first is the **delimiter**, which is a character or characters that indicates **where** to break up the string. We're using a space character as our delimiter, which means the search string is broken everywhere a space appears. The delimiter itself is not included in the resulting substrings. The second parameter is the string to be exploded.

The space delimiter controls how the string is exploded.



`$search_words`

Incorporating the array of search terms into Risky Jobs involves adding a line of code before we run the query on the Risky Jobs database. Now, if someone enters "Tipper Cow" into the search field, this code breaks it into two words and stores each word in an array (`$search_words`).

```
$user_search = $_GET['usersearch'];
```

```
$search_words = explode(' ', $user_search);
```

The `explode()` function takes each word in `$user_search` and stores it in an array called `$search_words`.



In order to incorporate the exploded search terms into the Risky Jobs application, plug each of the terms into an SQL SELECT query using LIKE and OR. For example, what a query would look like for Ernesto's earlier search on "Bull Fighter Matador":

```
SELECT * FROM riskyjobs
WHERE description LIKE '%Bull%' OR description LIKE '%Fighter%'
      description LIKE '%Matador%'
```

We're now searching the job description instead of the title since the description has more information to match.

Now suppose we used the following PHP code to attempt to assemble this query from the search terms entered by the user into the Risky Jobs search form:

```
$search_query = "SELECT * FROM riskyjobs";
$where_clause = '';
$user_search = $_GET['usersearch'];
$search_words = explode(' ', $user_search);
foreach ($search_words as $word) {
    $where_clause .= " description LIKE '%$word%' OR ";
}

if (!empty($where_clause)) {
    $search_query .= " WHERE $where_clause";
}
```

Write down the SQL query generated by this code when Ernesto enters "Bull Fighter Matador" as his search, and then annotate any problems you think it might have.

.....
.....
.....



Risky Jobs - Search

Find your risky job:

Bull Fighter Matador

exercise solution

In order to incorporate the exploded search terms into the Risky Jobs application, plug each of the terms into an SQL SELECT query using LIKE and OR. For example, what a query would look like for Ernesto's earlier search on "Bull Fighter Matador":

```
SELECT * FROM riskyjobs
WHERE description LIKE '%Bull%' OR description LIKE '%Fighter%'
      description LIKE '%Matador%'
```

We're now searching the job description instead of the title since the description has more information to match.

Now suppose we used the following PHP code to attempt to assemble this query from the search terms entered by the user into the Risky Jobs search form:

```
$search_query = "SELECT * FROM riskyjobs";
$where_clause = '';
$user_search = $_GET['usersearch'];
$search_words = explode(' ', $user_search);
foreach ($search_words as $word) {
    $where_clause .= " description LIKE '%$word%' OR ";
}
if (!empty($where_clause)) {
    $search_query .= " WHERE $where_clause";
}
```

Make sure the WHERE clause isn't empty before appending it to the search query.

This operator concatenates one string onto the end of another string.

...oon

Risky

Danger! Your dream job
Do you have the guts?

Risky Jobs - Search

Find your risky job:
Bull Fighter Matador

Write down the SQL query generated by this code when Ernesto enters "Bull Fighter Matador" as his search, and then annotate any problems you think it might have.

SELECT * FROM riskyjobs

WHERE description LIKE '%Bull%' OR description LIKE '%Fighter%' OR

description LIKE '%Matador%'(OR)

There's an extra OR at the end of the query, which will make it invalid.

`implode()` builds a string from substrings

What we really need to do is only put OR between the LIKEs in our WHERE clause, **but not after the last one**. So how exactly can that be done? How about a special case inside the loop to see if we're on the last search term, and then not include OR for that one? That would work but it's a little messy. A much cleaner solution involves a function that does the reverse of the `explode()` function. The `implode()` function takes an array of strings and builds a single string out of them.

This is the de
is wedged bet
the strings wh
stuck together

```
$where clause = implode(' OR ', $where list);
```

 The `implode()` function returns a single string.

5

This must
- of strings
want to

But how does that help the dangling OR problem in our query? Well, `implode()` lets you specify a delimiter to stick between the strings when combining them together. If we use ' `OR` ' as the delimiter, we can construct a `WHERE` clause that only has OR **between** each `LIKE` clause.



Sharpen your pencil

Rewrite the PHP code that generates the Risky Jobs table so that it fixes the dangling OR problem by using the `IN` operator.

sharpen your pencil solution

Sharpen your pencil Solution

Rewrite the PHP code that generates the Risky Jobs query so that it fixes the dangling OR problem by using the implode function.

```

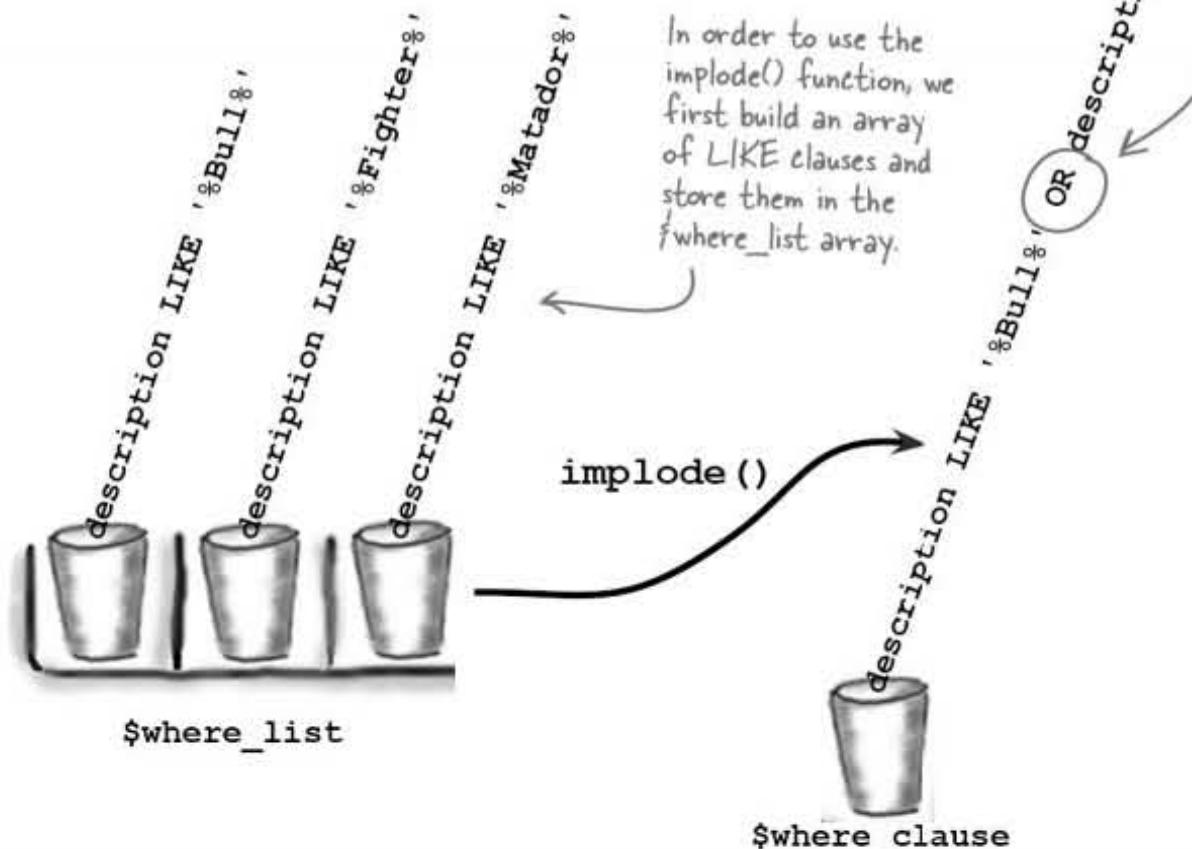
$search_query = "SELECT * FROM riskyjobs";
$where_list = array();
$user_search = $_GET['usersearch'];
$search_words = explode(' ', $user_search);
foreach ($search_words as $word) {
    $where_list[] = "description LIKE '%{$word}%'";
}
$where_clause = implode(' OR ', $where_list);
if (!empty($where_clause)) {
    $search_query = "WHERE $where_clause";
}

```

Since implode() accepts an array of strings to be joined together, we have to build an array of LIKE clauses.

When used like this, the [] operator acts the same as the array_push() function – it adds a new element onto the end of an array.

The delimiter passed to implode() is "OR" with a space on each side of it.





Test DRIVE

Take the Risky Jobs search form for a spin.

Download the Risky Jobs application from the Head First Labs site at www.headfirstlabs.com/books/hfphp. The `search.php` script contains the query generation code you just worked through, and is used to process the search data entered into the form in the `search.html` page.

Upload the script and other Risky Jobs files to your web server, and then open the search form (`search.html`) in a web browser. Try a few different searches to see how your query generation code fares. Make sure to try Ernesto's "Bull Fighter Matador" search, which is a good test of the new `implode()` code.

Risky Jobs – Search

Risky Jobs

Danger! Your dream job is out there.
Do you have the guts to go find it?

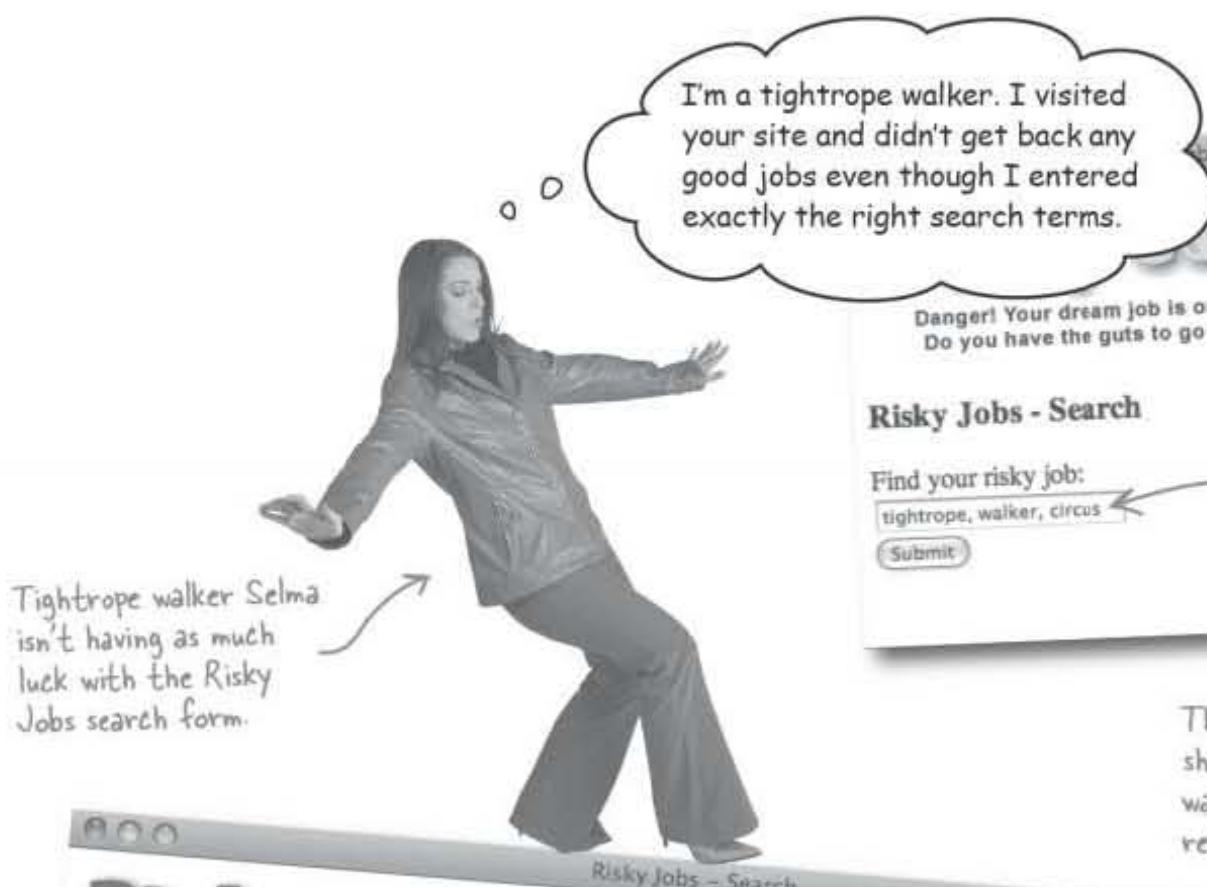
Risky Jobs - Search Results

Job Title	Description	State	Date Post
Prize Fighter	Up and coming super fly gnat weight boxer needs an opponent to help build his winning record. Slow feet, sloppy hands, and a glass jaw are preferred. No experience required. This is not a full-time salaried position. We'll meet you in the alley behind the rink to share the purse. Let Ron King make you a championship fighter, or at least help you lose to one!	MO	2008-11-14 21:49:31
Toreador	Lovely bovines waiting for your superior non-violent cape waving skills. Must pass basic bull fighting aptitude test.	ID	2008-11-14 21:49:31
Electric Bull Repairer	Hank's Honky Tonk needs an experienced electric bull repairer. Free rides (after you fix it) and half off hot wings are some of the benefits.	NJ	2008-11-14 21:49:31

So many jobs to choose from... I think I'm up for prize fighting! I'll start repairing electric bulls and find my dream job here.

Ernesto doesn't immediately see his perfect risky job, but he's definitely making progress now that the search script looks for each individual search term.

can the search be improved?



Tightrope walker Selma
isn't having as much
luck with the Risky
Jobs search form.

Risky Jobs - Search

Risky Jobs

Danger! Your dream job is out there.
Do you have the guts to go find it?

Risky Jobs - Search Results

Job Title	Description	State	Date Posted
Master Cat Juggler	<p>Are you a practitioner of the lost art of cat juggling? Banned in forty countries, only the Jim Ruiz Circus has refined cat juggling for the sophisticated tastes of the modern audience. Ply your trade with premiere cat jugglers at our circus, the only place on earth to master synchronized cat juggling. It's true, juggling them is even harder than herding them. We are an equal opportunity employer, and look forward to adding you to our team. Please be prepared to undergo a thorough battery of tests to prove your deft handling of felines. Only the cream of the crop will be accepted into our Master Cat Juggler program.</p>	AZ	2008-11-14 21:13:35
Tightrope Tester	<p>If the thought of dangling for hours on end from great heights is your idea of a good time, then this job just may be for you. Every one of our tightropes goes through rigorous a 43 point test, culminating in a real live human hanging for a prolonged period of time. That could be you! We do provide safety nets but you'll need to bring your own helmet and gloves. Here at our manufacturing and testing facility in Big Top, Montana, we offer an incredible employment package with benefits ranging from Bring Your Pet to Work Week and Formal Fridays. We will need three references, including your verified maximum hang time and number of past falls. We're the circus behind the circus!</p>	MT	2008-11-14 21:17:16



The sea
show a s
walker i
results a

Danger! Your dream job is out there.
Do you have the guts to go find it?

Risky Jobs - Search

Find your risky job:

tightrope, walker, circus

Submit

string

Sharpen your pencil



Write down the SQL query generated when Selma "tightrope, walker, circus" as her search, and then a few problems you think it might have.

.....
.....
.....

preprocessing the search strings



Sharpen your pencil Solution

Write down the SQL query generated when Selma "tightrope, walker, circus" as her search, and then answer the problems you think it might have.

```
SELECT * FROM riskyjobs
```

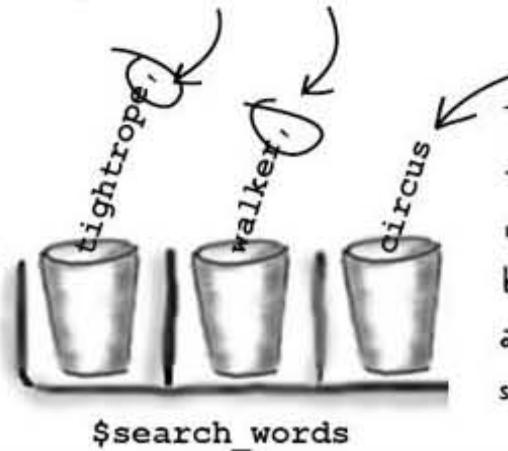
```
WHERE description LIKE '%tightrope,%' OR description LIKE '%walker,%' OR
      description LIKE '%circus%'
```

The explode()
function uses
spaces as
delimiters but
it totally misses
the commas.



`explode()`

The commas are considered part
of the search terms instead of
separators between the terms.



`$search_words`

I don't see what the big deal is. Just call
the explode() function twice - first to get
rid of spaces and then to get rid of commas.

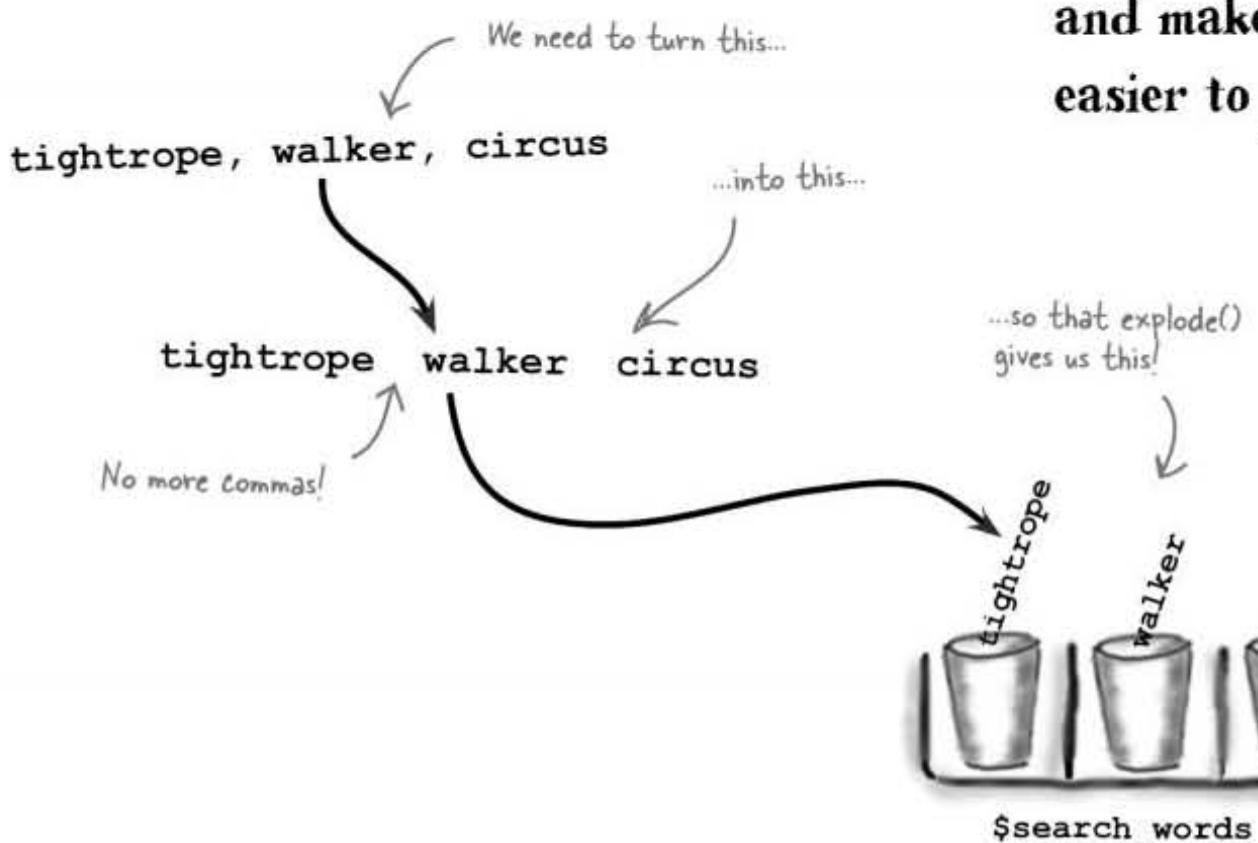


The explode() function lets you explode a string into substrings, but in this case we have substrings.

The first call to the `explode()` function leaves us with multiple strings stored in an array, so there's isn't a single string to explode. And attempting to explode each string in turn would likely just create more problems. Instead of trying to solve the delimiter problem with multiple calls to `explode()`, we can **preprocess** the search string to get it down to a single string with a single delimiter before we ever even call `explode()`. Then it can be easily broken apart using one delimiter.

Preprocess the search string

We want to hand the `explode()` function a string that it can break apart cleanly in one shot. How do we do that? By making sure that `explode()` only has to worry with a single delimiter, such as a space character. This means we need to **preprocess** the search string so that each search term is separated by a space, even if the user entered commas.



there are no
Dumb Questions

Q: Can we use more than one character as the delimiter when exploding the search string?

A: Yes, you can specify any number of characters to serve as your delimiter, but that's not the same as specifying **different** delimiters, and won't solve our problem here.

If we used `explode(' ', ', $user_search)` to break apart our string, it would use a combined comma and space as a delimiter, and would work if someone entered "tightrope, walker, circus". But it would fail if someone entered "tightrope walker circus". In that case, we'd be left with one long string—not good.

Q: Can we just delete the commas into spaces?

A: That will work only if users separate with both a comma and a space, which when deleted commas, we'd run the risk of turning "tightropewalker", which probably wouldn't be good for a Jobs database.

the php str_replace() function

Replace unwanted search characters

If you think about it, preprocessing the RiskyJobs search string is a lot like using find-and-replace in a word processor. In our case, we want to find commas and replace them with spaces. PHP's `str_replace()` lets you do just that by supplying it three parameters: the text you want to find, the text you want to replace it with, and the string you want to perform the find-and-replace on. Here's an example of `str_replace()` in action:

```
$clean_search = str_replace('thousands', 'hundreds',
    'Make thousands of dollars your very first month. Ap
    ↑
    This is the substring
    you want to replace...
    ↓
    ...and this
    gets inserted
    ↑
    The third parameter is the string
    that will be changed. We're adding
    a little truth to the advertising by
    replacing "thousands" with "hundreds".
```

But what about those commas in the search string? The `str_replace()` function works just as well at replacing individual characters:

```
$clean_search = str_replace(',', ' ', 'tightrope, walk
    ↑
    Remember, this is the
    substring you're replacing...
    ↓
    ...and this is the string
    you're replacing it with.
    ↑
    Everywhere that a comma appears in this
    string, it will be replaced with a space.
```

After this code runs, the variable `$clean_string` will contain the string "tightrope walker circus".



Do you see anything suspicious about the `str_replace()` function? Do you think commas with spaces will work like we w



Given the PHP code below, show what the output of the `$search_words` array will be for each of the following search strings. Make sure to write in data for the array elements, and scratch through elements if the `$search_words` array ends.

```
$clean_search = str_replace(',', ' ', $user_search);
$search_words = explode(' ', $clean_search);
```

`bull,matador cape`



`$search_words`

`3 spaces!` →
`bull matador cape`



`$search_words`

`bull , matador cape`



`$search_words`

`2 spaces!` →
`bull,matador, cape`



`$search_words`

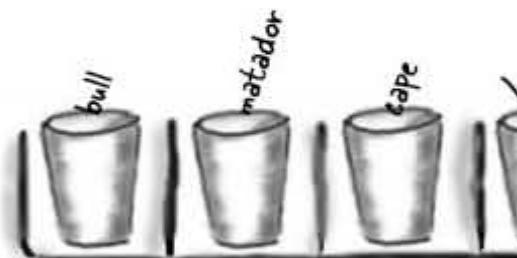
exercise solution

Given the PHP code below, show what the output of the \$search_words array will be for each of the following search strings. Make sure to write in data for the array elements, and scratch through elements if the \$search_words array ends.

```
$clean_search = str_replace(',', ' ', $user_search);
$search_words = explode(' ', $clean_search);
```

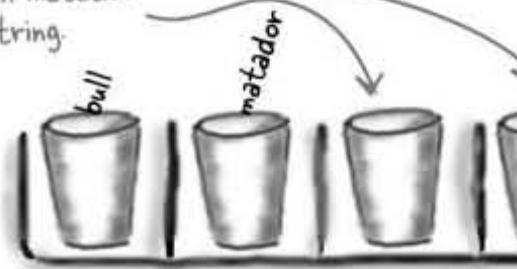
This
three

bull,matador cape

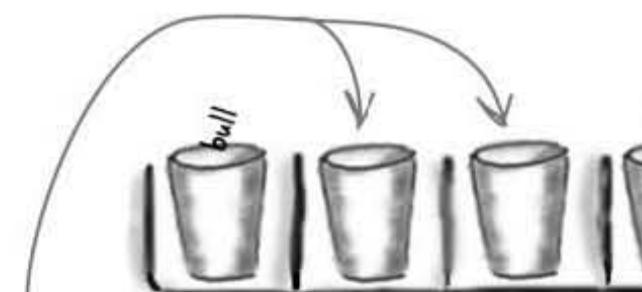


These two array elements are actually empty because of those two extra spaces between matador and cape in the search string.

3 spaces!
bull matador cape

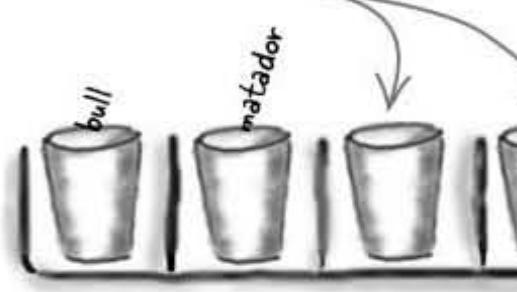


bull , matador cape



Again, two empty elements, because the comma is replaced with a space

2 spaces!
bull,matador, cape



\$search_words



So we're all set now that we're preprocessing the search string, right?

Uh, no. Preprocessing gets rid of unwanted characters but unfortunately, it doesn't result in an array containing only good search terms.

Remember, our goal is to end up with a string where each search term is separated by exactly the same delimiter, a space. Take another look at what happened in the last three examples on the facing page. Some of the elements in the \$search_words array are empty. If we try to add our WHERE clause with the empty search elements, we might end up with something like this:

```
SELECT * FROM riskyjobs
WHERE description LIKE '%bull%' OR
      description LIKE '%matador%' OR
      description LIKE '% %' OR
      description LIKE '% %' OR
      description LIKE '%cape%'
```

Those single spaces will match every single space in each job description. They are a real problem.

But those spaces won't match anything, right?

Wrong! They will match everything.

If there's a space anywhere in a job description (which is pretty much a given), this query will match it and return it as a result. So every job in the Risky Jobs database will be matched by this query. We need to get rid of those empty array elements before we construct the SQL query in order to make the search script useful again.

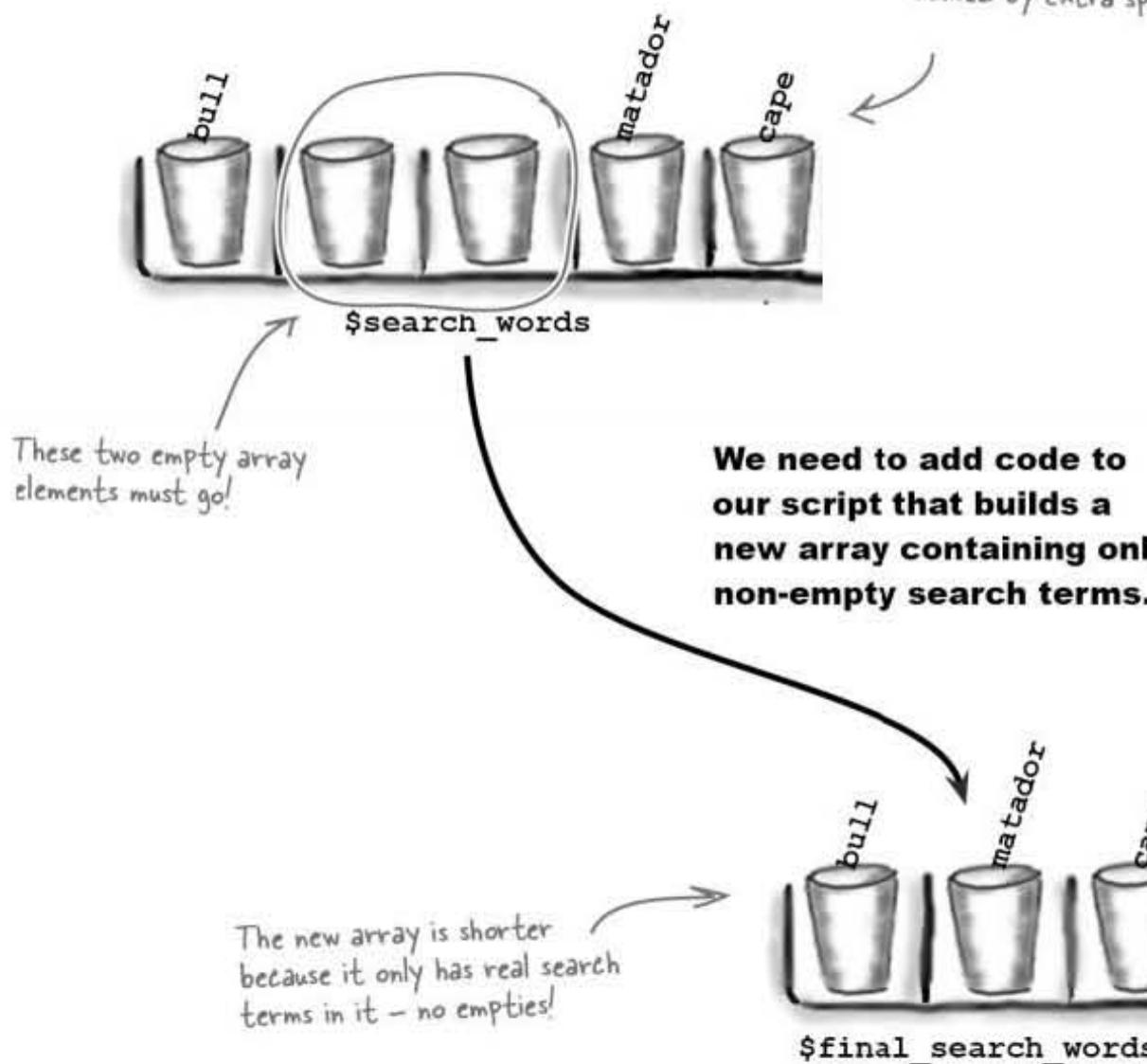
stripping out empty search terms

The query needs legit search terms

The good news is that it's not too difficult to clean up our search terms before using them in a query. We'll need to create a new array that only contains the real search terms. So we'll copy all the non-empty elements from our first array into the second array, and then use that array to construct the SELECT query.

To construct the new array, we can use a `foreach` loop to cycle through each element in the original array, and then use an `if` statement to find non-empty elements. When we find a non-empty element, we just add it to the new array. Here's what this process looks like:

Here's the original array that contains the search terms and empty elements caused by extra spaces.



Copy non-empty elements to a new array

Now let's look at the code that will copy the non-empty elements from our `$search_words` array to the new `$final_search_words` array.

```

$search_query = "SELECT * FROM riskyjobs";

// Extract the search keywords into an array
$clean_search = str_replace(',', ' ', $user_search);
$search_words = explode(' ', $clean_search);
$final_search_words = array();
if (count($search_words) > 0) {
    foreach ($search_words as $word) {
        if (!empty($word)) {
            $final_search_words[] = $word;
        }
    }
}

```

This is
comma
str_re

Loop through each element
the `$search_word` array
element is not empty, push
array named `$final_sear`

After checking to make sure there is at least one search term in the `$search_words` array, the `foreach` loop cycles through the array looking for non-empty elements. When it finds a non-empty element, it uses the `[]` operator to add the element onto the end of the `$final_search_words` array. This is how the new array is assembled.

Then what? Well, then we generate the `SELECT` query just as before, except now we use the `$final_search_words` array instead of `$search_words`:

```

// Generate a WHERE clause using all of the search keywords
$where_list = array();
if (count($final_search_words) > 0) {
    foreach ($final_search_words as $word) {
        $where_list[] = "description LIKE '%$word%'";
    }
}
$where_clause = implode(' OR ', $where_list);

// Add the keyword WHERE clause to the search query
if (!empty($where_clause)) {
    $search_query .= " WHERE $where_clause";
}

```

This is the same c
seen that builds t
clause of the sear
but this time it u
\$final_search_w
contains no empt

This code gives us a search query that no longer has empty elements. Here's the new query for the search "bull, matador, cape":

```

SELECT * FROM riskyjobs
WHERE description LIKE '%bull%' OR
description LIKE '%matador%' OR
description LIKE '%cape%'

```



Will this se
users the n
looking fo

test out search.php

Test Drive

Update the Search script to preprocess the user search string

Update the search.php script to use the explode() and implode() functions to preprocess the user search string and generate a more robust SELECT query. Upload the script to your web server and try out a few searches.

Risky Jobs - Search

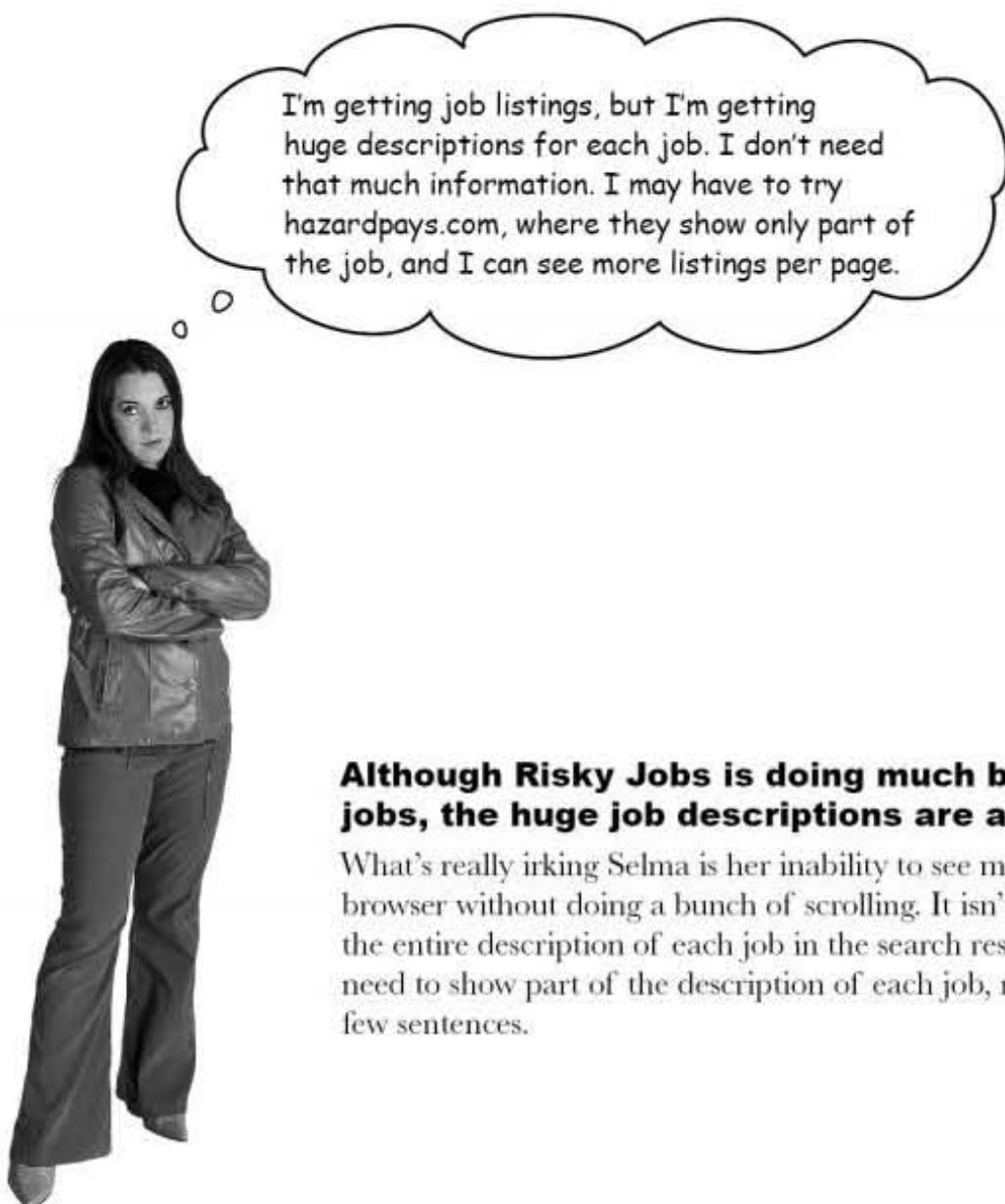
Risky Jobs

Danger! Your dream job is out there.
Do you have the guts to go find it?

Risky Jobs - Search Results

Job Title	Description	State	Date Posted
Tightrope Walker	Fledgling big top looking for three-ring professional with 1-3 years of experience to perform tightrope acrobatics with pudgy elephant. Willingness to sweep excrement a big plus. Excellent benefits including medical and dental plans, 401 (k), stock ownership and discount purchase plan, prescription coverage, merchandise discount, short and long term disability insurance, life and business travel insurance, vision discount plan, auto and home insurance discounts, medical care and dependent care reimbursement, educational assistance, paid vacation and holidays, and adoption assistance. Flexible starting salaries based on skills and abilities, experience and geographic market. Promotion opportunities based on performance. The only thing stopping you from the highest wire in the big tent is your desire and work ethic...and your balance! Other duties include planning & organizing wires, handling minor elephant administration, processing comment canis from children. Leading by example (don't fall!), showing initiative and a sense of urgency and being results-driven help acrobatic professionals become successful. If you want to be challenged and your talent needs mentoring and opportunity, Bingling Brothers can offer you a fast track to success!	TX	2008-11-14 21:16:19
Master Cat Juggler	Are you a practitioner of the lost art of cat juggling? Banned in forty countries, only the Jim Ruiz Circus has refined cat juggling for the sophisticated tastes of the modern audience. Ply your trade with premiere cat jugglers at our circus, the only place on earth to master synchronized cat juggling. It's true, juggling them is even harder than herding them. We are an equal opportunity employer, and look forward to adding you to our team. Please be prepared to undergo a thorough battery of tests to prove your deft handling of felines. Only the cream of the crop will be accepted into our Master Cat Juggler program.	AZ	2008-11-14 21:13:35
Tightrope Tester	If the thought of dangling for hours on end from great heights is your idea of a good time, then this job just may be for you. Every one of our tightropes goes through rigorous a 43 point test, culminating in a real	MT	2008-11-14 21:17:16

Selma's "tightrope, walker, circus" search now appears to be finding more relevant jobs.



Although Risky Jobs is doing much better at finding jobs, the huge job descriptions are a bit much.

What's really irking Selma is her inability to see more job listings in her browser without doing a bunch of scrolling. It isn't necessary to show the entire description of each job in the search results. Ideally, we really need to show part of the description of each job, maybe just the first few sentences.

Write down how you think we could trim the job description so that they aren't quite so huge in the search results:

.....

.....

.....

the php substr() function

Sometimes you just need part of a string

Since the lengths of the job descriptions in the Risky Jobs database vary quite a bit and some are quite long, we could clean up the search results by chopping all the descriptions down to a smaller size. And to keep from confusing users, we can just stick an ellipsis (...) on the end of each one to make it clear that they're seeing only part of each description.

The PHP `substr()` function is perfect for extracting part of a string. You pass the “substring” function the original string and two integers. The first integer is the index of where you want the substring to start, and the second integer is its length, in characters. Here’s the syntax:

`substr(string, start, length)`

This is the original string we want to extract a substring from.

This specifies where to start the substring...

...and this is how many characters to return.

When it comes to the `substr()` function, you can think of a string as being like an array where each character is a different element. Consider the following string:

```
$job_desc = 'Are you a practitioner of the lost art of cat juggling? '
```

Similar to elements in an array, each character in this string has an index, starting at 0 and counting up until the end of the string.

Are you a practitioner of the lost art of cat juggling?

0 1 2 3 4 5 6 7 8 9...

... 50 51 52

We can use these character indexes with the `substr()` function to grab portions of the string:

Start at 4, go for 3 characters. → `substr($job_desc, 4, 3)`

→ `substr($job_desc, 49)`

Start at 49, and since we left off the second argument, it means go to the end of the string.

→ `substr($job_desc, 0, 3)`

→ `substr($job_desc, 0, 9)`

**The PHP
function
you to ex-
portion**

Extract substrings from either end

The `substr()` function is not limited to grabbing substrings from the start of a string. You can also extract characters starting from the end of a string. The extraction still works left to right; you just use a negative index to identify the start of the substring.

Are you a practitioner of the lost art of cat juggling?

-53 -52 -51 -50 -3 -2

Here are a couple of examples:

Start at -53, then grab 7 characters.

Start at -9 and take the rest of the string.



Sharpen your pencil

Below is PHP code that generates an HTML table for Jobs search results. Finish the missing code, whose job description text to 100 characters, and also date posted text to only show the month, day, and year.

```

echo '<table border="0" cellpadding="2">';
echo '<td>Job Title</td><td>Description</td><td>State</td><td>Date Posted</td>';
while ($row = mysqli_fetch_array($result)) {
    echo '<tr class="results">';
    echo '<td valign="top" width="20%>' . $row['title'] . '</td>';
    echo '<td valign="top" width="50%>' . .....;
    echo '<td valign="top" width="10%>' . $row['state'] . '</td>';
    echo '<td valign="top" width="20%>' . .....;
    echo '</tr>';
}
echo '</table>';

```

sharpen your pencil solution

Sharpen your pencil Solution

Below is PHP code that generates an HTML table for Jobs search results. Finish the missing code, whose the job description text to 100 characters, and also date posted text to only show the month, day, and year.

```
echo '<table border="0" cellpadding="2">';
echo '<td>Job Title</td><td>Description</td><td>State</td><td>Date Pos
while ($row = mysqli_fetch_array($result)) {
    echo '<tr class="results">';
    echo '<td valign="top" width="20%">' . $row['title'] . '</td>';
    echo '<td valign="top" width="50%">' . substr($row['description'], 0, 100) . '...';
    echo '<td valign="top" width="10%">' . $row['state'] . '</td>';
    echo '<td valign="top" width="20%">' . substr($row['date_posted'], 0, 10) . '...';
    echo '</tr>';
}
echo '</table>';
```

Stick an ellipsis on the end of the substr() function because this is only part of the job description.

All of the date_posted data starts with MM-DD-YYYY, which takes up exactly 10 characters.



Geek Bits

*there are
Dumb Q*

It's possible to skip the PHP `substr()` function and limit the job description data in the SQL query itself. You use a very similar MySQL function called `SUBSTRING()` that accepts the same arguments as `substr()`. The only difference is that the starting index starts at 1 instead of 0. So grabbing the first 100 characters of the job description looks like this:

```
SELECT SUBSTRING(job_description, 1, 100)
FROM riskyjobs;
```

The advantage of sticking with the PHP function is that we have both the partial job description and the full job description available to us. If we use MySQL, we only get the partial job description, and would have to make another query to get the full description.

Q: Does `substr()`**A:** No, it operates strictly on strings. If you have a number stored in a column of type `TEXT`, when you retrieve it, it will be returned by PHP as a string, not a number. That's why the `substr()` function on its own is useful.**Q:** What if your length argument is longer than the string? Will it return a string of length zero or the entire string?**A:** It will return the entire string. The `substr()` function does not trim off the end of the string with spaces. For example, the following code will return the string "dog".`substr('dog', 0, 10)`



Test Drive

Tweak the Search script to limit the text displayed for job descriptions and dates posted.

Modify the search.php script so that it uses the PHP substr() function to trim down the job description and date posted text for the search results. Then upload the script to your web server and test it out with a few searches.

Selma is pleased now that she can see job search results without having to scroll through humongous job descriptions.



Risky Jobs – Search

Risky Jobs

Danger! Your dream job is out there.
Do you have the guts to go find it?

Risky Jobs - Search Results

Job Title	Description	State	Date Posted
Tightrope Walker	Fledgling big top looking for three-ring professional with 1-3 years of experience to perform tightr...	TX	2008-11-14
Master Cat Juggler	Are you a practitioner of the lost art of cat juggling? AZ Banned in forty countries, only the Jim Ruiz...	AZ	2008-11-14
Tightrope Tester	If the thought of dangling for hours on end from great heights fills you with fear, then thi... MT	MT	2008-11-14

I'd really like to see the results sorted by date posted, or maybe by state. I really want to find a matador job in Vermont.



How can we change the page layout to allow us to sort by date posted, s...

The post easier to shows the date and...

adding sort functionality to search results

Multiple queries can sort our results

In order to allow visitors to sort their search results, we need a way for them to identify how they want their results ordered. Maybe a form... or a button? It's actually way simpler than that. We can use HTML to turn each of the column headings in the search result table into links. Users can click a link to indicate which column they want to sort the results by.

Risky Jobs - Search Results			
Job Title	Description	State	Date Posted
Prize Fighter	Up and coming super fly gnat weight boxer needs an opponent to help build his winning record. Slow f...	MO	2008-11-14
Toreador	Lovely bovines waiting for your superior non-violent cape waving skills. Must pass basic bull fighti...	ID	2008-11-14
Electric Bull Repairer	Hank's Honky Tonk needs an experienced electric bull repairer. Free rides (after you fix it) and hal...	NJ	2008-11-14

We can use these links to reload the same Search script but with a query that sorts the results according to the link clicked. We already know how to use ORDER BY to structure a query with sorted results. If we create different SQL queries to ORDER BY each individual column, we can allow the user to sort the search results alphabetically by job title, description, or state, or chronologically by date posted.

Here is the SQL query to sort results alphabetically by job description:

```
SELECT * FROM riskyjobs
WHERE description LIKE '%Bull%' OR description LIKE '%Fighter%'
      OR description LIKE '%Matador%'
ORDER BY description
```

This sorts the results of the query on job descriptions in ascending alphabetical order.

string



Sharpen your pencil

Write three different queries that sort the Risky Jobs according to job title, state, and date posted. Assume the user has typed in the search string "window, washer, sky".

How could we rewrite these queries if you wanted to see the job titles in reverse order? What about the newest jobs first?

ORDER BY sorts the search query results



Sharpen your pencil Solution

Write three different queries that sort the Risky Jobs according to job title, state, and date posted. Assume the user has typed in the search string "window, washer, sky

`SELECT * FROM riskyjobs`

`WHERE description LIKE '%window%' OR description LIKE '%was'`
`description LIKE '%skyscraper%'`

`ORDER BY job_title`

The default for
`ORDER BY` is
ASCending order, which
is the same as saying
`ORDER job_title ASC`.

`SELECT * FROM riskyjobs`

`WHERE description LIKE '%window%' OR description LIKE '%was'`
`description LIKE '%skyscraper%'`

`ORDER BY state`

`SELECT * FROM riskyjobs`

`WHERE description LIKE '%window%' OR description LIKE '%was'`
`description LIKE '%skyscraper%'`

`ORDER BY date_posted`

How could we rewrite these queries if you wanted to see the job titles in reverse order? What about the newest jobs first?

`SELECT * FROM riskyjobs`

`WHERE description LIKE '%window%' OR description LIKE '%was'`
`description LIKE '%skyscraper%'`

`ORDER BY job_title DESC`

`SELECT * FROM riskyjobs`

`WHERE description LIKE '%window%' OR description LIKE '%was'`
`description LIKE '%skyscraper%'`

`ORDER BY state DESC`

`SELECT * FROM riskyjobs`

`WHERE description LIKE '%window%' OR description LIKE '%was'`
`description LIKE '%skyscraper%'`

`ORDER BY date_posted DESC`

We might want
these if we've
already ordered
one of the
columns and the
user clicks on it
again to reverse
the order.

It seems like a lot of redundant code will be needed to generate all those queries. Can't we avoid having the same query generation code repeated three, or even six times?

Yes. While it's true that we'll need to run a different query when a user clicks a different link, it's possible construct a single query based on the link clicked.

The first time the results are displayed, no links have been clicked so we don't have to worry about sorting. We can just take the keywords submitted into our form and build a query without an ORDER BY. The results are displayed with clickable headings, each of which links back to the script, but with a different sort order. So each link consists of a URL with the original keywords and a parameter named `sort` that indicates which order the results should be in.

What would really help in pulling this off is if we create our very own **custom function** that takes information about how to sort the job data, and returns a string with the WHERE clause and ORDER BY in place. Our new custom function takes a look at the `sort` parameter to figure out how to sort the search results. Here are the steps the function has to follow:

- 1 Preprocess the search keywords, and store them in an array.
- 2 Optionally take a `sort` parameter that tells the function what column to sort by.
- 3 Get rid of any empty search keywords.
- 4 Create a WHERE clause containing all of the search keywords.
- 5 Check to see if the `sort` parameter has a value. If it does, tack on an ORDER BY clause.
- 6 Return the newly formed query.



This might look like a lot of work, but we already have most of the code written. We just need to turn it into a function. But before we do that, let's take a look at how to put together custom functions...

writing custom php functions

Functions let you reuse code

A **function** is a block of code separate from the rest of your code that you can execute wherever you want in your script. Until now, you've used **built-in functions** that PHP has already created. `explode()`, `substr()`, and `mysqli_query()` are all functions that are predefined by PHP and can be used in any script.

But you can also write your own **custom functions** to provide features not supplied by the language. By creating a custom function, you can use your own code again and again without repeating it in your script. Instead, you just call the function by name when you want to run its code.

Following is an example of a custom function called `replace_commas()` that replaces commas with spaces in a string:

```
function replace_commas($str) {
    $new_str = str_replace(',', ' ', $str);
    return $new_str;
}
```

To create a custom function, you begin it with the word "function".

This is whatever you decide you want to name your function - make it as descriptive as possible.

A set of parentheses follow the name. You can send one or more function as arguments, each separated by a comma - in this case, there's just one.

Curly braces indicate where the function should go, just like in a loop or if statement.

A function can return a value to the code that called it - in this case we return the altered string.

When you're ready to use a custom function, just call it by name and enter any values that it expects in parentheses. If the function is designed to return a value, you can assign it to a new variable, like this:

```
$clean_search = replace_commas('tightrope, walker, circus');
```

We pass in a string, "tightrope, walker, circus".

The function returns a new string with the commas replaced by spaces.

Build a query with a custom function

We've already written much of the code we need to create the custom function that generates a Risky Jobs search query. All that's left is dropping the code into a PHP function skeleton. Here's the custom `build_query()` function:

```
function build_query($user_search) {
    $search_query = "SELECT * FROM riskyjobs";

    // Extract the search keywords into an array
    $clean_search = str_replace(',', ' ', $user_search);
    $search_words = explode(' ', $clean_search);
    $final_search_words = array();
    if (count($search_words) > 0) {
        foreach ($search_words as $word) {
            if (!empty($word)) {
                $final_search_words[] = $word;
            }
        }
    }

    // Generate a WHERE clause using all of the search keywords
    $where_list = array();
    if (count($final_search_words) > 0) {
        foreach($final_search_words as $word) {
            $where_list[] = "description LIKE '%$word%'";
        }
    }
    $where_clause = implode(' OR ', $where_list);

    // Add the keyword WHERE clause to the search query
    if (!empty($where_clause)) {
        $search_query .= " WHERE $where_clause";
    }

    return $search_query;
}
```

We're passing in
the `$user_search`
created from the
entered into the

Actually, this is new. Here's
return the new query so that
called the function can use

The `build_query()` function returns a complete SQL query based on the search string passed into it via the `$user_search` argument. To use the function, we just pass along the search data entered by the user, and then store the result in a new string that we'll call `$search_query`:

```
$search_query = build_query($user_search);
```

This lets us capture the value
our function returns, in this
case our new search query.

This is the value from the
search form the user submitted

interview with a custom function

Custom Functions Exposed

This week's interview:
Custom functions: how custom are they really?

Head First: Look, we're all wondering one thing: what's so wrong with redundant code? I mean really, it's easy to create, you just copy and paste and boom. You're done.

Custom Function: Oh, don't get me started about redundant code. It's just plain ugly and makes your code more difficult to read. That's bad enough. But there is a much much more important reason to avoid redundant code.

Head First: Yes?

Custom Function: Well, what if something changes in your code? That happens pretty often.

Head First: So what? Things change all the time. You just go in and you fix it.

Custom Function: But what if the thing that changed was in your redundant code? And was in five, or maybe ten places throughout your application?

Head First: I don't see a problem. You'd just find them and change them all. Done.

Custom Function: Fine, okay. But what if you missed changing it in one place? You're only human, you programmers. If you missed it, you might have a very tough time tracking it down.

Head First: Sure, I guess that could happen. But how do you help?

Custom Function: Ah, but that's the beauty that is me. If that code had been in a function, you could have changed it once. Just once. Bim bam boom and done.

Head First: I have to admit, that's compelling. But I still don't see why I'd go out of my way to use you. I mean, I'm limited, right? You can only use str-

Custom Function: Whoa! Wait a buckaroo! I can take any data type you want to send me. As long as the code inside me handles that data the way it should, I can use whatever you want me to. Heck, I used an array as an example. That's pretty darn sophis-

Head First: But you returned a string.

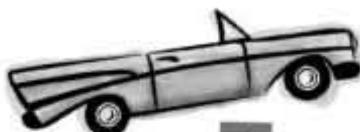
Custom Function: I can return whatever you want. It's all about making the most of me. Offer and using me correctly.

Head First: That's another thing I'm demanding. You have to have data.

Custom Function: Where are you going with this crazy ideas? You can call me with whatever data you want, and if I'm set up that way, I'll want to send me data, don't write me. I'll do the parentheses next to my name whatever you want me. Although I can't think of many situations where you wouldn't want to pass data to me and have me data back out again with a return statement.

Head First: We're all out of time. I have to go.

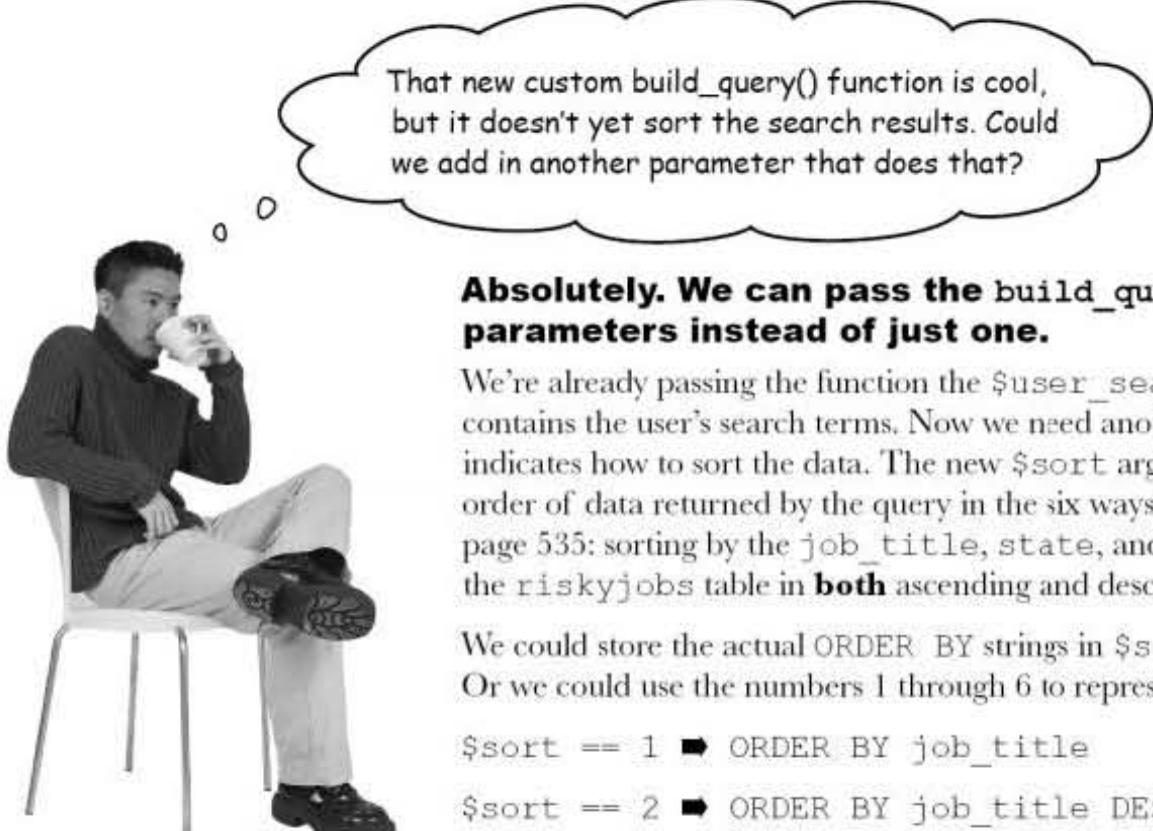
Custom Function: Don't mention time. Or is that serve to live? Or is that live to serve? Something like that.



Test Drive

Modify the Search script to use the build_query() function

Create the new `build_query()` function in the `search.php` script, making sure to replace the original code with a call to the new function. Upload the script to your web server and try out a search in a web browser to make sure it works correctly.



Absolutely. We can pass the `build_query()` parameters instead of just one.

We're already passing the function the `$user_search` argument, which contains the user's search terms. Now we need another argument that indicates how to sort the data. The new `$sort` argument controls the order of data returned by the query in the six ways we can see on page 535: sorting by the `job_title`, `state`, and `date_posted` columns in the `riskyjobs` table in **both** ascending and descending order.

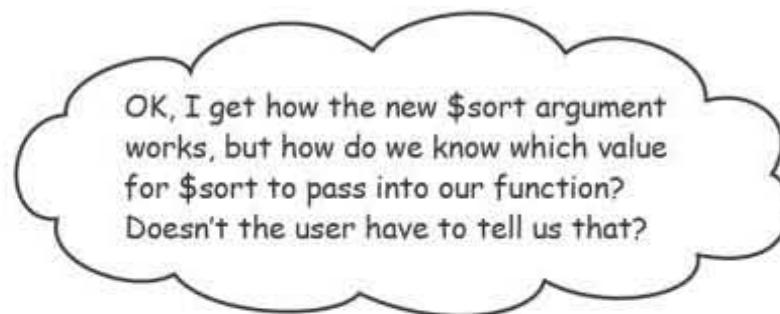
We could store the actual `ORDER BY` strings in `$sort` to represent each ordering. Or we could use the numbers 1 through 6 to represent each ordering.

```
$sort == 1 ➔ ORDER BY job_title
$sort == 2 ➔ ORDER BY job_title DESC
$sort == 3 ➔ ORDER BY state
$sort == 4 ➔ ORDER BY state DESC
$sort == 5 ➔ ORDER BY date_posted
$sort == 6 ➔ ORDER BY date_posted DESC
```

We just arbitrarily chose these numbers and the meaning that each one has. There are no special rules about it other than to use them consistently.

But aren't integers more cryptic when reading through your code? Yes, helpful comments, yes, but there's a more important reason here. If we used `ORDER BY` strings, our data would show the column names as part of each heading link. This would inadvertently expose the column names, which you'd rather not make public for security reasons.

letting users specify the type of sort



Yes, users must specify how to sort the search results they specify the search terms themselves.

The good news is we already know how we want to implement this functionality: we're going to turn the column headings on our results page into hyperlinks. When a user clicks on a given heading, like "State," we'll pass the number for sorting into our `build_query()` function.

But we still have to get the sort order from the link to the script. We can do this by generating custom links for the headings by tacking on a `sort` parameter.

The search results are generated as part of an HTML table, which is why there's a `<td>` tag here.

`$sort_links .= '<td>State' . '`

Our `build_query()` function needs the user's search keywords to display results, so we pass that in the URL.

We want to make sure that when a user clicks a column heading, we make this happen.

We pass along some information so that when the user clicks a column heading, the desired sorting order is passed along with it.

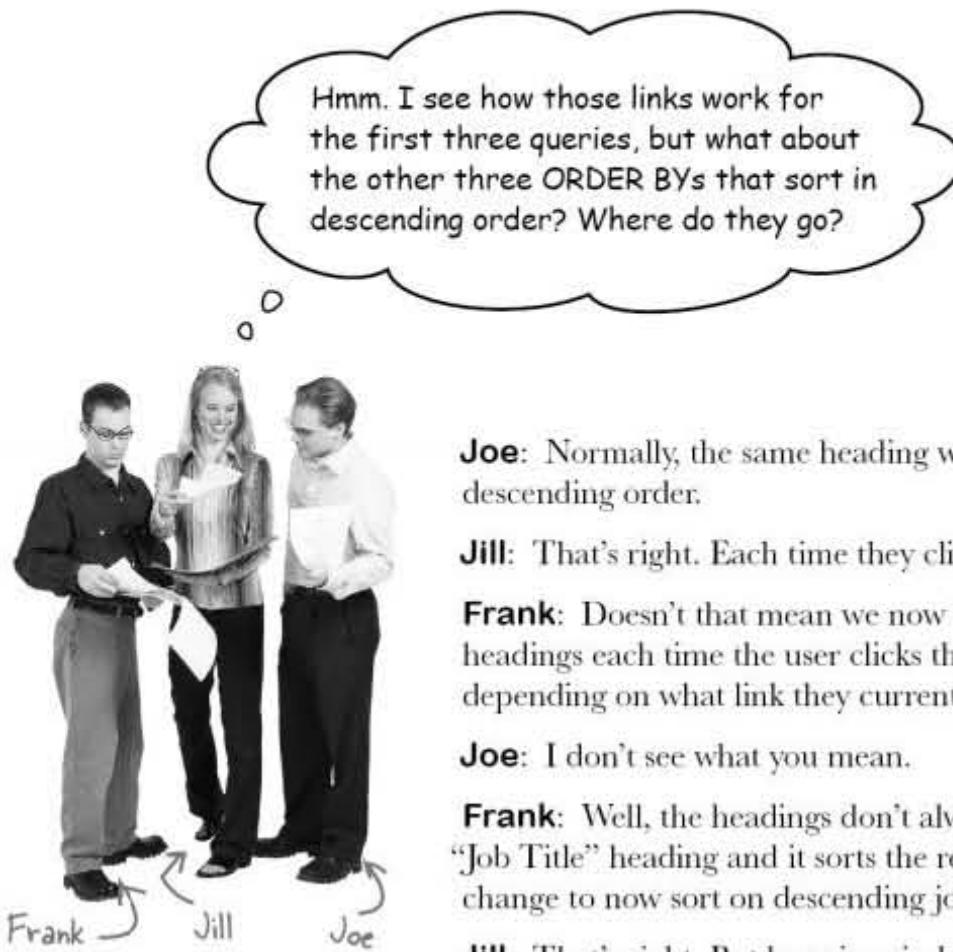
When the results page is generated, each heading link (except "Job Description") will generate its own customized URL, including a `sort` value for how the results should be sorted.

```
<a href="search.php?usersearch=bull fighter matador&sort=1">
```

The screenshot shows a search results page titled "Risky Jobs - Search Results" for the query "bull fighter matador". The results table has columns: Job Title, Description, State, and Date Posted. The "Description" column header is underlined, indicating it is a link. A callout points to this link with the text: "Ordering by description wouldn't tell us much so there's no reason to turn it into a sort link." Another callout points to the "Job Title" column with the text: "Risky Jobs - Search Results".

```
<a href="search.php?usersearch=bull fighter matador&sort=3">
```

```
<a href="search.php?usersearch=bull fighter matador">
```



Joe: Normally, the same heading would allow the user to sort in descending order.

Jill: That's right. Each time they click a heading it just reverses.

Frank: Doesn't that mean we now have to somehow keep up the headings each time the user clicks them because they now have to depend on what link they currently hold.

Joe: I don't see what you mean.

Frank: Well, the headings don't always do the same sort. For example, if you click the "Job Title" heading and it sorts the results by ascending job title, then if you click it again, it changes to descending job titles. If you change to now sort on descending job titles the next time it is clicked, it will be ascending again.

Jill: That's right. But keep in mind that each type of sort has a URL to let the script know what kind of sort to carry out. And with those links, we can control exactly what sort numbers get put in.

Joe: I see. So the challenge for us is to somehow structure our code to generate the correct link based on the current state of the sort, right?

Frank: Ah, I've got it! Isn't that something we can solve with a switch statement? I mean, that's the kind of decision making they're good for, right?

Joe: Yes, that would work but we're talking about several decisions for the same piece of data, the sort type. It would really be nice if we could find a better way to make those decisions other than a bunch of nested if statements.

Jill: That's a great point, and it's a perfect opportunity to try out something we've heard about. The switch statement lets you make multiple decisions between two, based solely on a single value.

Frank: That sounds great. Let's give it a try.

Joe: I agree. Anything to avoid complicated if-else statements that give me a headache!

Jill: Yeah, me too. I think the switch statement might just be the answer.

the php switch statement

SWITCH makes far more decisions than IF

The switch statement offers an efficient way to check a value and execute one of several different blocks of code depending on that value. This is something that would require a small army of if-else statements, especially in situations involving quite a few options.

Instead of writing nested if-else statements to check for each possible value, you instead write a switch statement that has a case label corresponding to each possible value. At the end of each case label, you put the statement break;, which instructs PHP to drop out of the entire switch statement and not consider any other cases. This ensures that PHP will execute the code in no more than one case.

Let's take a look at an example that uses switch:

```

switch ($benefit_code) { ← This is the value the switch
    case 1:           statement is checking - it
        $benefits = 'Major medical, 10 sick days'; ← controls the entire switch.
        break;          ← This code is only
    case 2:          ← executed when
        $benefits = 'Death and dismemberment only, one month paid life';
        break;          ← $benefit_code is 1.
    case 3:          ← If you need to do the same thing for two
        $benefits = 'Good luck!';                      or more values, just leave off the break
    case 4:          ← statement until you reach the last value.
        $benefits = 'None.';                          ← Any values stored in $benefit_code
    default:         ← other than 1, 2, 3, or 4 will cause
        $benefits = 'None.';                         the default code to execute.
    }                                         ← echo 'We offer four comprehensive benefits packages';
                                                ← echo 'The plan you have selected: ' . $benefits;
                                                ← Not really. There are only three
                                                packages since 3 and 4 are both the
                                                same thanks to 3 not having a break.

```

A SWITCH
contains a
CASE label
execute different
blocks depending
on the value



Risky Jobs has a new function called `generate_sort_links()` that search results by clicking on the result headings. Unfortunately, it's missing code. Finish the code for the function. And don't forget the numbers for each:

- 1 = ascending job title, 2 = descending job title, 3 = ascending state, 4 = descending state
- 5 = ascending date posted, and 6 = descending date posted.

```
..... generate_sort_links($user_search, $sort) {
    $sort_links = '';

    ..... ($sort) {
        case 1:
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= ..... "Job Title</a></td><td>Description</td>';
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= ..... "State</a></td>';
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= ..... "Date Posted</a></td>';

        .....
```

case 3:

```
$sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
    '&sort= ..... "Job Title</a></td><td>Description</td>';
$sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
    '&sort= ..... "State</a></td>';
$sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
    '&sort= ..... "Date Posted</a></td>';

.....
```

case 5:

```
$sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
    '&sort= ..... "Job Title</a></td><td>Description</td>';
$sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
    '&sort= ..... "State</a></td>';
$sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
    '&sort= ..... "Date Posted</a></td>';

.....
```

.....

```
$sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
    '&sort= ..... "Job Title</a></td><td>Description</td>';
$sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
    '&sort= ..... "State</a></td>';
$sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
    '&sort= ..... "Date Posted</a></td>';

}
```

return;

}


This is the default set of headings
appear when the user hasn't chosen

the completed `generate_sort_links()` function



Risky Jobs has a new function called `generate_sort_links()` that search results by clicking on the result headings. Unfortunately, it's missing code. Finish the code for the function. And don't forget the numbers for each 1 = ascending job title, 2 = descending job title, 3 = ascending state, 4 = descending state, 5 = ascending date posted, and 6 = descending date posted.

```

function generate_sort_links($user_search, $sort) {
    $sort_links = '';
    switch ($sort) {
        case 1:
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= 2.....">Job Title</a></td><td>Description</td>';
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= 3.....">State</a></td>';
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= 5.....">Date Posted</a></td>';
            break;
        case 3:
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= 1.....">Job Title</a></td><td>Description</td>';
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= 4.....">State</a></td>';
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= 3.....">Date Posted</a></td>';
            break;
        case 5:
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= 1.....">Job Title</a></td><td>Description</td>';
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= 3.....">State</a></td>';
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= 6.....">Date Posted</a></td>';
            break;
        default:
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= 1.....">Job Title</a></td><td>Description</td>';
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= 3.....">State</a></td>';
            $sort_links .= '<td><a href = "' . $_SERVER['PHP_SELF'] . '?usersearch='
                '&sort= 5.....">Date Posted</a></td>';
    }
    return $sort_links;
}

```

If \$sort is 1, it means we've already sorted by job title, so now we need to re-sort it in descending order.

If \$sort hasn't been set yet or if it's 2, 4, or 6, we should display the original link that sort the data in ascending order.

Give build_query() the ability to sort

We now have two functions to handle Risky Jobs searches. `build_query()` constructs an SQL query based on search terms entered by the user, and `generate_sort_links()` generates hyperlinks for the search result headings that allow the user to sort the results. But `build_query()` isn't quite finished since the query it generates doesn't yet sort. The function needs to append an `ORDER BY` clause to the query. But it has to be the **correct** `ORDER BY` clause, as determined by a new `$sort` argument:

```

function build_query($user_search, $sort) {
    $search_query = "SELECT * FROM riskyjobs";
    ...

    // Add the keyword WHERE clause to the search query
    if (!empty($where_clause)) {
        $search_query .= " WHERE $where_clause";
    }

    // Sort the search query using the sort setting
    switch ($sort) {
        // Ascending by job title
        case 1:
            $search_query .= " ORDER BY title";
            break;
        // Descending by job title
        case 2:
            $search_query .= " ORDER BY title DESC";
            break;
        // Ascending by state
        case 3:
            $search_query .= " ORDER BY state";
            break;
        // Descending by state
        case 4:
            $search_query .= " ORDER BY state DESC";
            break;
        // Ascending by date posted (oldest first)
        case 5:
            $search_query .= " ORDER BY date_posted";
            break;
        // Descending by date posted (newest first)
        case 6:
            $search_query .= " ORDER BY date_posted DESC";
            break;
        default:
            // No sort setting provided, so don't sort the query
    }

    return $search_query;
}

```

We're now passing in the `$sort` argument to our function, in addition to `$user_search`.

Here are the steps to build_query(). The switch statement checks the value of `$sort` and the corresponding `case` statement adds the search query.

When users land on this page without column headings, `$where_clause` is empty, so as long as there's no sort setting, we won't sort the results.

We return `$search_query` as before, only this time with an `ORDER BY` clause at the end.

test out the revised search.php script



Test Drive

Revamp the Search script to use the two new custom functions

Create the new generate_sort_links() function in the search.php script and then add the new code to the build_query() function so that it generates a query with sorted results. Don't forget to actually call the generate_sort_links() function in the script in place of the code that echoes the result headings.

Upload the script to your web server, open the search.html page in a browser, try doing a search. Now click the headings above the search results to sort the jobs on the different data. Make sure to click the same heading more than once to switch between ascending and descending order.



The build_query() function takes the user entered, exploded query, cleans out any empty strings in the query, builds an SQL query with the term BY corresponding to the sort value.



Danger! Your dream job is out there.
Do you have the guts to go find it?

The generate_sort_links() function generates the clickable column headings, including the packaging of sort options into the URL of each link.

Risky Jobs - Search Results

Job Title	Description	State	Date Posted
Matador	Busting dairy farm looking for part-time matador to entertain spirited bull with mild case of ADD...	VT	2008-03-15
Firefighter	The City of Dataville is hiring firefighters. No experience required - you will be trained. Non-smokers only.	OH	2008-05-01
Electric Bull Repairer	Hank's Honky Tonk needs someone to repair electric bull.	NY	2008-11-01

Now I can see the oldest jobs, the ones where they are getting truly desperate to hire a matador in Vermont.

But sometimes I try a broader search and the results are overwhelming.



Risky Jobs - Search

Risky Jobs

Danger! Your dream job is out there.
Do you have the guts to go find it?

Risky Jobs - Search Results

This is an awful lot
of job listings to try
and take in at once.



How do other sites avoid having lots
of search results on a single page?

Job Title	Description	State	Date
Custard Walker	We need people willing to test the theory that you can walk on custard. We're going to fill a swi...	NM	2008-0
Shark Trainer	Training sharks to do cute tricks for the audiences at our new water theme park. You'll spend tim...	FL	2008-0
Voltage Checker	You'll be out in the field checking a.c. and d.c. voltages in the range of 3 to 250 or more volts. Y...	NC	2008-0
Antenna Installer	You'll be installing antennas and other metallic broadcast receiving equipment on the roofs of Miami...	FL	2008-0
Elephant Proctologist	Needed: experienced proctologist willing to work with large animals. Elephants at our zoo (in San Pr...	CA	2008-0
Airplane Engine Cleaner	Jet airplanes needing engines cleaned. In need of clean-minded individuals willing to handle rust an...	TX	2008-0
Matador	Bustling dairy farm looking for part-time matador to entertain spirited bull with mild case of ADD.	VT	2008-0
Paparazzo	Top celebrity photography firm looking for seasoned paparazzo to stalk temperamental lip-syncing pop...	CA	2008-0
Tightrope Walker	Fledgling big top looking for three-ring professional with 1-3 years of experience to perform tightr...	TX	2008-0
	... love animals and hate plaque? Well, then this job for you! Our crocodile farm ... fresh new faces. Full health insurance provided. Must love kids....	FL	2008-0
	... gives on how good our pet food tastes. help make our products even better....	NY	2008-0
	... waiting for your superior non-violent kills. Must pass basic bull fighti...	ID	2008-0
	... Tonk needs an experienced electric bull rider (after you fix it) and hal...	NJ	2008-0
Firefighter	The City of Dataville is hiring firefighters. No ... you will be trained. Non-smo...	OH	2008-0

use pagination to display a subset of results

We can paginate our results

We're displaying all of our results on a single page right now, which is a problem when a search matches lots of jobs. Instead of forcing users to scroll up and down a huge page to see all the job matches, we can use a technique called **pagination** to display the search results. When you paginate results, you break the collection of job matches into groups, and then display each group on a separate web page, like this:

Each page scrolls along with other pages can easily be on a page and av

Risky Jobs - Search

Risky Jobs

Danger! Your dream job is out there. Do you have the guts to go find it?

Risky Jobs - Search Results

Job Title	Description
Custard Walker	We need people willing to walk on custard. We're going to be making custard.
Shark Trainer	Training sharks to do tricks in our new water theme park.
Voltage Checker	You'll be out in the field at night, in the range of 3 to 250 or more.
Antenna Installer	You'll be installing antenna on broadcast receiving equipment.
Elephant Pneuologist	Needed: experienced person to work with large animals. Elephants!

<- 1 2 3 4 ->

These links allow users to navigate through multiple pages.

Risky Jobs - Search Results

Job Title	Description
Airplane Engine Cleaner	Get airplanes nice and clean.
Matador	Bustling dairy farm.
Paparazzo	Entertain spinsters.
Tightrope Walker	Top celebrity photo paparazzo to stalk.
Crocodile Dentist	Fledgling big top with 1-3 years of experience.
Mime	Do you love animals? You might be the job for you.

<- 1 2 3 4 ->

The current page is not a link – this is the second page of job results.

Risky Jobs - Search Results

Job Title	Description	State
Mime	We need some fresh new faces. Full health insurance and skin pads provided. Must love kids...	NY
Pet Food Tester	We pride ourselves on how good our pet food tastes. Now you can help make our products even better...	MO
Tomador	Lovely bovines waiting for your superior non-violent cape waving skills. Must pass basic bull fight...	ID
Electric Bull Repairer	Hank's Honky Tonk needs an experienced electric bull repairer. Free rides (after you fix it) and beer...	NJ
Firefighter	The City of Denerville is hiring firefighters. No experience required - you will be trained. Non-smo...	OH

<- 1 2 3 4 ->

That's great, but how do we break up our results into groups like that? Our SQL query returns all the results that match the search string.

Pagination breaks query results into sets, and displays each set on its own web page.

We need a query that will return just a subset of the results, not all of them.

Luckily, SQL already gives us a way to do that: the `LIMIT` clause. Let's revisit `LIMIT` and see how we can use it to split our results up into groups of five...

Get only the rows you need with LIMIT

The key to controlling which rows we display on any given page is to add another clause to our search query, a `LIMIT` clause. To get a maximum of five rows, we add `LIMIT 5` to the end of our query, like this:

```
SELECT * FROM riskyjobs
    ORDER BY job_title
        LIMIT 5
```

Without a `WHERE` clause, this query returns all the jobs in the database, which is equivalent to searching with no search terms.

Only return the first five matches no matter how many matches are actually found.

If you recall, we use the custom `build_query()` function to create our Risky Jobs query. To force it to only display the first five matches, we just concatenate `LIMIT 5` to the end of the query string after it's built:

```
$query = build_query($user_search, $sort);
$query = $query . " LIMIT 5";
```

Adding a `LIMIT` clause to the end of a query limits the number of rows returned by the query, in this case to five rows.

This works well for getting the first five rows of results, but what about the next five rows, and the five rows after that? To pull out rows deeper in the result set, we have to change our `LIMIT` up a bit. But how? `LIMIT 10` would get the first 10 rows, so that wouldn't work. We need to get rows 6 through 10, and to do that we use `LIMIT` with different syntax. When you add two arguments to `LIMIT`, the first argument controls how many rows you skip, and the second argument controls how many rows you get back. For example, here's how you get rows 11 through 25, which would be the third page of results:

```
$query = build_query($user_search, $sort);
$query = $query . " LIMIT 10, 5";
```

The first argument tells `LIMIT` how many rows to skip – the first ten.

The second argument controls how many rows are returned – five, same as earlier.

LIMIT controls what and rows are returned by an SQL query

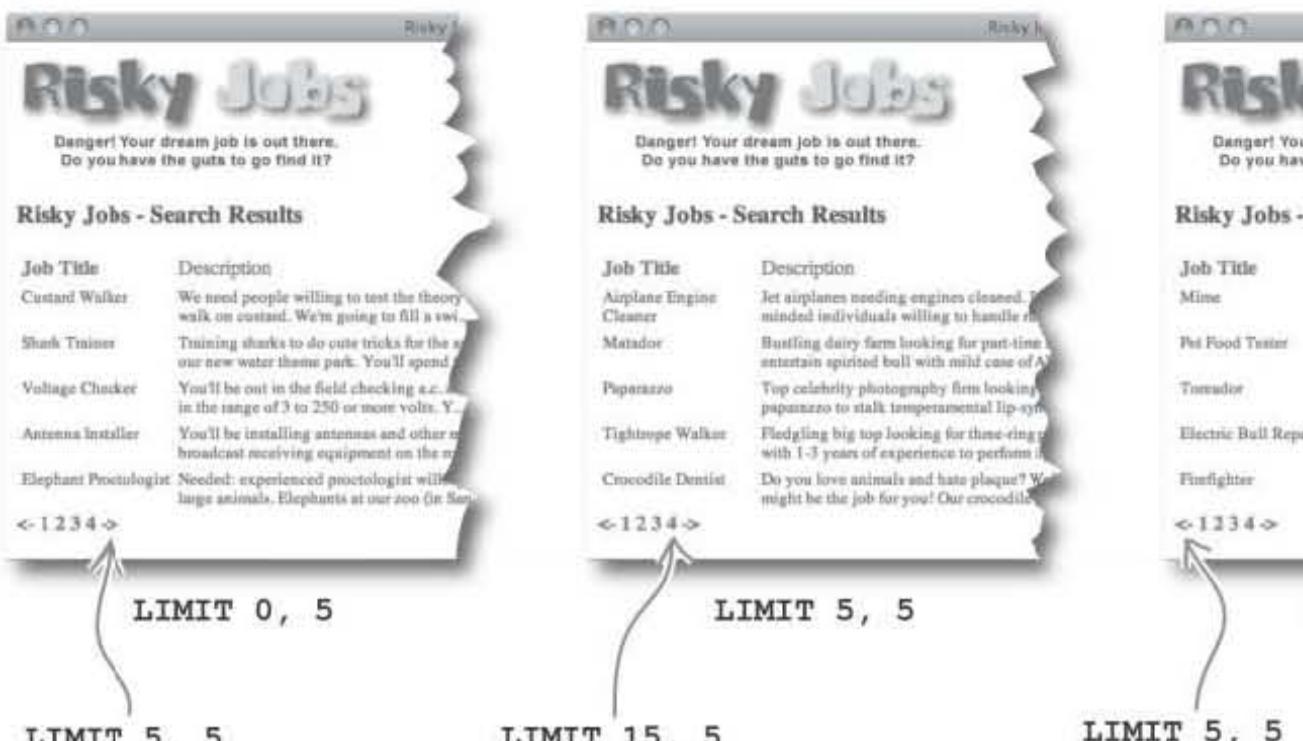
Custard
Shark
Voltage
Antenna
Elephant
Airplane Engine
Mata
Papar
Tightrope
Crocodile
Mim
Pet Food
Toreas
Electric Bull
Firefig

use LIMIT to help paginate results

Control page links with LIMIT

An important part of pagination is providing links that allow the user to move back and forth among the different pages of results. We can use the `LIMIT` clause to set up the navigation links for the bottom of each page of results. For example, the “next” and “previous” links each have their own `LIMIT`. The same thing applies to the numeric links that allow the user to jump straight to a specific page of results.

Here are the `LIMIT` clauses for the first three pages of search results, along with `LIMIT`s for some of the page links:



No problemo. We just need to write a bunch of queries with a different `LIMIT` on each one, right?

Sort of. We need a different `LIMIT` depending on the page and link, but we can generate it instead of writing multiple queries.

All we need to do is modify our `build_query()` function further to add the correct `LIMIT` at the end of the query.

Keep track of the pagination data

In order to add the new pagination functionality to `build_query()`, we need to set up and keep track of some variables that determine which search results to query and show on a given page. These variables are also important in determining how the navigation links at the bottom of the page are generated.

\$cur_page

Get the current page, `$cur_page`, from the script URL via `$_GET`. If no current page is passed through the URL, set `$cur_page` to the first page (1).

\$results_per_page

This is the number of results per page, which is based on the look and feel of the page, and the results fit nicely on the page with the layout. Set the second argument to the `LIMIT` clause of

\$skip

Compute the number of rows to skip before the rows on the current page begin, `$skip`. This variable is what controls where each page begins in terms of results, providing the first argument to the `LIMIT` clause.

\$total

Run a query that retrieves all the rows with `$_GET['q']` and then count the results and store it in `$total`. In other words, this is the total number of search results.

\$num_pages

Compute the number of pages, `$num_pages`, using `$total` divided by `$results_per_page`. So for any given search, there is a total of `$total` matching rows, but they are displayed a page at a time, with each page containing `$results_per_page` matches. There are `$num_pages` pages, and the current page is identified by `$cur_page`.

setting up variables needed for pagination

Set up the pagination variables

Most of the pagination variables can be set up purely through information provided via the URL, which is accessible through the `$_GET` superglobal. For example, the `$sort`, `$user_search`, and `$cur_page` variables all flow directly from GET data. We can then use these variables to calculate how many rows to skip to get to the first row of data, `$skip`. The `$results_per_page` variable is a little different in that we just set it to however many search results we want to appear on each page, which is more of a personal preference given the layout of the results page.

```

Get the current page, // Grab the sort setting and search keywords from the URL
$fcur_page          $sort = $_GET['sort'];
from the URL via GET. $user_search = $_GET['usersearch']; ← Grab the search string
If there's no current page, // Calculate pagination information
set $fcur_page to 1.    $cur_page = isset($_GET['page']) ? $_GET['page'] : 1;
                        $results_per_page = 5; // number of results per page
                        $skip = (($cur_page - 1) * $results_per_page); ← This calculation results
Set the number of results per page. ← Compute the number of the first row on the page, $skip.

```

Get the sort order, which is an integer in the range 1 to 6.

Grab the search string that the user entered into the form.

This calculation results for page 1, 5 for page 2, 15 for page 3, etc.

We're still missing a couple of important variables: `$total` and `$num_pages`. These variables can only be set after performing an initial query to find out how many matches are found in the database. Once we know how many matches we have, it's possible to set these variables and then `LIMIT` the results...

Revise the query for paginated results

Now that we've got our variables set up, we need to revise the Search script so that instead of querying for all results, it queries for just the subset of results we need for the page the user is currently viewing. This involves first doing a query so that the `$total` variable can be set and the `$num_pages` variable can be calculated. Then we follow up with a second query that uses `$skip` and `$results_per_page` to generate a LIMIT clause that we add to the end of the query. Here's the revised section of the `search.php` script with these new additions highlighted:

```

    mysqli_num_rows() returns a
    count of how many total rows
    were returned by the query.

This query retrieves all
the rows with no LIMIT.

// Query to get the total results
$query = build_query($user_search, $sort);
$result = mysqli_query($dbc, $query);
$total = mysqli_num_rows($result);
$num_pages = ceil($total / $results_per_page);

Store away the total number
of rows with a call to the
mysqli_num_rows() function.

Compute number of pages
number of rows by the
page, and then rounding

The ceil() function rounds
a number up to the next
integer - the "ceiling"

// Query again to get just the subset of results
$query = $query . " LIMIT $skip, $results_per_page";
$result = mysqli_query($dbc, $query);
while ($row = mysqli_fetch_array($result)) {
    echo '<tr class="results">';
    echo '<td valign="top" width="20%">' . $row['title'] . '</td>';
    echo '<td valign="top" width="50%">' . substr($row['description'], 0,
        echo '<td valign="top" width="10%">' . $row['state'] . '</td>';
        echo '<td valign="top" width="20%">' . substr($row['date_posted'], 0,
            echo '</tr>';
        }
    echo '</table>';

Issue a second query, but
this time LIMIT the
results to the current page.

```

creating the navigation links

Generate the page navigation links

So we've set up some variables and built a new SQL query that returns a subset of results for the page. All that's left to do is to generate the page navigation links for the bottom of the search results page: the "previous" link, numerical links for each page of results, and the "next" link. We already have all the information we need to put together the links. Let's go over it again to make sure it's clear how it will be used.

\$user_search

Every page link still has to know what the user is actually searching for, so we have to pass along the search terms in each link URL.

\$num_pages

We need to know how many pages there are in order to generate links for each of them.

\$cur_page

The page navigation links are entirely on the current page, so it's very important to get packaged into every link URL.

\$sort

The sort order also factors into the page links because the order has to be maintained; else the pagination wouldn't make any sense.

OK, we know what information we need in order to generate the page navigation links, so we're ready to crank out the PHP code to make it happen. This code could just be dropped into the `search.php` script, but what if we put it in its own custom function? That way the main script code that generates the search results can be much simpler, requiring only a single line of code to generate the page links—a call to this new function, which we'll call `generate_page_links()`.

The only catch is that we don't want this function to get called if there is only one page of results. So we need to do a check on the number of pages before calling the new `generate_page_links()` function. Here's how we can perform the check and call the function, making sure to pass along the required information as function arguments:

```
if ($num_pages > 1) {
    echo generate_page_links($user_search, $sort, $cur_page, $num_pages);
}
```

First check to make sure there is more than one page of search results; otherwise, don't generate the page links.

Pass along the search terms, current page, and sort order to use in generating the page links.



PHP & MySQL Magnets

The `generate_page_links()` function is almost finished, but it's missing a few magnets to plug in the missing code and give Risky Jobs the ability to generate page links.

```
function generate_page_links($user_search, $sort, $cur_page, $num_pages) {
    $page_links = '';

    // If this page is not the first page, generate the "previous" link
    if ($cur_page != 1) {
        .....  

        $page_links .= '<a href="' . $_SERVER['PHP_SELF'] . '?usersearch=' . $user_search . '&sort=' . $sort . '&page=' . ($cur_page - 1) . '"><-</a> ';
    }  

    else {  

        $page_links .= '<- ';  

    }

    // Loop through the pages generating the page number links
    for ($i = 1; $i <= $num_pages; $i++) {  

        if ($i == $cur_page) {  

            $page_links .= ' ' . $i;  

        }  

        else {  

            $page_links .= '<a href="' . $_SERVER['PHP_SELF'] . '?usersearch=' . $user_search . '&sort=' . $sort . '&page=' . $i . '">' . $i . '</a>';  

        }
    }

    // If this page is not the last page, generate the "next" link
    if ($cur_page != $num_pages) {
        .....  

        $page_links .= '<a href="' . $_SERVER['PHP_SELF'] . '?usersearch=' . $user_search . '&sort=' . $sort . '&page=' . ($cur_page + 1) . '">-></a> ';
    }  

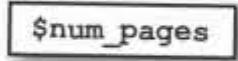
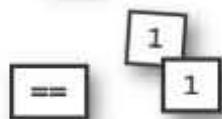
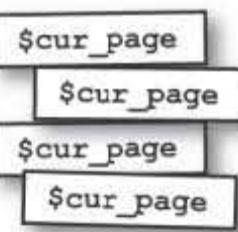
    else {  

        $page_links .= '->';
    }
}

return $page_links;
}
```

Annotations:

- A callout arrow points from the text "The 'previous' link appears as a left arrow, as in "<-"." to the line `&page=' . ($cur_page - 1) . '"><- ';`.
- A callout arrow points from the text "The 'next' link appears as a right arrow, as in "->"." to the line `&page=' . ($cur_page + 1) . '">-> ';`.



the completed `generate_page_links()` function



PHP & MySQL Magnets Solution

The `generate_page_links()` function is almost finished, but it's missing a few magnets to plug in the missing code and give Risky Jobs the ability to generate page links.

```
function generate_page_links($user_search, $sort, $cur_page,
    $page_links = '');

// If this page is not the first page, generate the "Previous" link
if ( $cur_page > 1 ) {
    $page_links .= '<a href="' . $_SERVER['PHP_SELF'] .
        '?usersearch=' . $user_search .
        '&sort=' . $sort .
        '&page=' . ($cur_page - 1) . '"><-</a> ';
}

else {
    $page_links .= '<-';
} // The "previous" link appears as a left arrow, as in "<-".

// Loop through the pages generating the page number links
for ($i = 1; $i <= $num_pages; $i++) {

    if ( $cur_page == $i ) {
        $page_links .= ' ' . $i;
    }
    else {
        $page_links .= '<a href="' . $_SERVER['PHP_SELF'] .
            '?usersearch=' . $user_search .
            '&sort=' . $sort .
            '&page=' . $i . '">' . $i . '</a>';
    } // Make sure each link goes back to the same URL by just passing a page number with each link.
}

// If this page is not the last page, generate the "Next" link
if ( $cur_page < $num_pages ) {
    $page_links .= '<a href="' . $_SERVER['PHP_SELF'] .
        '?usersearch=' . $user_search .
        '&sort=' . $sort .
        '&page=' . ($cur_page + 1) . '">-></a> ';
}
else {
    $page_links .= '->';
} // The "next" link appears as a right arrow, as in "->".

return $page_links;
}
```

Putting together the complete Search script

And finally we arrive at a complete Risky Jobs Search script that displays the appropriate search results based on the user's search terms, generates clickable result heading links for sorting, paginates those results, and generates page navigation links along the bottom of the page.

```
<?php
// This function builds a search query from the search keywords and so
function build_query($user_search, $sort) {
    ...
    return $search_query;
}

// This function builds heading links based on the specified sort setti
function generate_sort_links($user_search, $sort) {
    ...
    return $sort_links;
}

// This function builds navigational page links based on the current p
// the number of pages
function generate_page_links($user_search, $sort, $cur_page, $num_page
    ...
    return $page_links;
}

// Grab the sort setting and search keywords from the URL using GET
$sort = $_GET['sort'];
$user_search = $_GET['usersearch'];

// Calculate pagination information
$cur_page = isset($_GET['page']) ? $_GET['page'] : 1;
$results_per_page = 5; // number of results per page
$skip = (($cur_page - 1) * $results_per_page);

// Start generating the table of results
echo '<table border="0" cellpadding="2">';

// Generate the search result headings
echo '<tr class="heading">';
echo generate_sort_links($user_search, $sort);
echo '</tr>';

// We've already built these functions,
// so there's no need to rehash every
// line of their code here.

// Grab the sort order and search
// string that were passed through
// the URL as GET data.

// Initialize the pagination
// since we'll need them
// to LIMIT the query
// and to create the
// navigation links.

// Call the generate_sort_links()
// function to create the links for the
// result headings, and then echo them

Hang on,
```

the final search.php

The complete Search script, continued...

```
// Connect to the database
require_once('connectvars.php');
$dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

// Query to get the total results
$query = build_query($user_search, $sort);
$result = mysqli_query($dbc, $query);
$total = mysqli_num_rows($result);
$num_pages = ceil($total / $results_per_page);

// Query again to get just the subset of results
$query = $query . " LIMIT $skip, $results_per_page";
$result = mysqli_query($dbc, $query);
while ($row = mysqli_fetch_array($result)) {
    echo '<tr class="results">';
    echo '<td valign="top" width="20%">' . $row['title'] . '</td>';
    echo '<td valign="top" width="50%">' . substr($row['description'], 0, 150) . '</td>';
    echo '<td valign="top" width="10%">' . $row['state'] . '</td>';
    echo '<td valign="top" width="20%">' . substr($row['date_posted'], 0, 10) . '</td>';
    echo '</tr>';
}
echo '</table>';

// Generate navigational page links if we have more than one page
if ($num_pages > 1) {
    echo generate_page_links($user_search, $sort, $cur_page, $num_pages)
}

mysqli_close($dbc);
?>
```

Call the `build_query()` to build the SQL job search query.

Here's the `LIMIT` we created to query a subset of job results.

And here's the code we wrote that cuts down the job description and date posted using the `substr()` function.

Keep things tidy by closing the database connection.

there are no
Dumb Questions

Q: Do we really have to pass the search, sort, and pagination information into `generate_page_links()`?

A: Yes. And the reason has to do with the fact that well-designed functions shouldn't manipulate data outside of their own code. So a function should only access data passed to it in an argument, and then only make changes to data that it returns.

Q: OK, so what about echoing data from `generate_page_links()` just like we did with `echo $row['title']`?

A: Same problem. By echoing data to the page, you would be effectively reaching beyond its boundaries somewhere else. It's much harder to debug when it isn't clear what data they change. Instead, return the data affected by a function, and then echo it with the data returned by the function, or use `print_r()`.



Test Drive

Finish the Risky Jobs Search script.

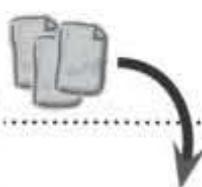
Add the new `generate_page_links()` function to the `search.php` script, also add the code that calls it after checking to see if there is more than one page to create and initialize the variables used as arguments to the function. And don't forget the query code so that it uses `LIMIT` to pull out the correct subset of results for each page.

When all that's done, upload the new `search.php` script to your web server, and open the `search.html` page in a web browser. Try a few searches, making sure to search for something that will end up with lots of results so that the new pagination features kick in. For example, on the first result pages, do a search with an empty search form.

Job Title	Description	State	Date Posted
Matador	Bustling dairy farm looking for part-time matador to entertain spirited bull with mild case of ADD. —	VT	2008-03-11
Paparazzo	Top celebrity photography firm looking for seasoned paparazzo to stalk temperamental lip-syncing pop stars. —	CA	2008-03-24
Shark Trainer	Training sharks to do cute tricks for the audiences at our new water theme park. You'll spend time in the water. —	FL	2008-04-28
Firefighter	The City of Dataville is hiring firefighters. No experience required - you will be trained. Non-smokers only. —	OH	2008-05-22
Voltage Checker	You'll be out in the field checking a.c. and d.c. voltages in the range of 3 to 250 or more volts. You must be able to climb trees. —	NC	2008-06-28

< 1 2 3 4 >

Download It!



Ernesto has found a job with the perfect amount of risk!

Don't forget, the complete source code for the Risky Jobs application is available for download from the Head First Labs web site:

www.headfirstlabs.com/books/hfphp

CHAPTER 9

php & mysql toolbox

Your PHP & MySQL Toolbox

The Risky Jobs Search script required quite a few new PHP and MySQL techniques. Let's recap some of the most important ones.

LIKE

Use LIKE to look for data within an SQL query without necessarily requiring an identical match. Put a % in front of and/or after a search term to let LIKE know that the term can have other characters surrounding it.

`explode()`, `implode()`

The PHP `explode()` function breaks a string apart into an array of substrings that are separated by a common delimiter, such as a space or comma. `implode()` does the opposite – it builds a single string from an array of strings, inserting a delimiter between them.

`str_replace()`

Call this PHP function to do a find-and-replace on a string of text, replacing one character or sequence of characters with another.

switch-case

A PHP decision-making construct that allows you to execute one of many groups of code based on a single value. If you find yourself with a bunch of nested if-else statements, you may find that you can code it more efficiently as a switch statement.

LIMIT

The LIMIT clause specifies exactly how many rows are returned by an SQL query. It does more than just return only that, but also lets you limit the number of rows in the result set, which can be useful when you want to isolate a specific section of data.

10 regular expressions

Rules for replacement



String functions are kind of lovable. But at the same time, they're limited. Sure, they can tell the length of your string, truncate it, and change certain characters to other certain characters. But sometimes you need to break free and tackle more complex text manipulations. This is where **regular expressions** can help. They can precisely modify strings based on a **set of rules** rather than a single criterion.

risky jobs is getting bad data

Risky Jobs lets users submit resumes

Riskyjobs.biz has grown. The company now lets job seekers enter their resumes and contact information into a web form so that our Risky Jobs employers can find them more easily. Here's what the form looks like:

In addition to normal contact information, a Risky Jobs candidate must also enter their desired job, as well as their resume.

Risky Jobs - Registration

Register with Risky Jobs, and post your resume.

First Name:

Last Name:

Email:

Phone:

Desired Job:

Paste your resume here:

The new Risky Jobs Registration form allows job candidates to enter information about themselves that potential employers can find.

Our job seeker information is stored in a table that can be searched by employers, recruiters, and headhunters to identify potential new employees. But there's a problem... the data entered into the form apparently can't be trusted!

First Name: Jimmy
Last Name: Swift
Email: JS@sim-u-duck.com
Phone: 636 4652
Desired Job: Ninja

First, I couldn't get a ninja because his phone number is missing, and now my email to this knife juggler has bounced. I've pretty much had it with the bad data in the Risky Jobs resume bank.



Employers can search the Risky Jobs candidate database and then contact people to possibly hire them... assuming enough contact information has been entered!

First Name: Four
Last Name: McCool
Email: four@greencards.com
Phone: 555-0987
Desired Job: Knifing





Below is some of the code for the registration.php script, which displays the user data entered into the form to register a new job candidate. Annotations show what is wrong with the code, and how it could be changed to resolve the bad data problem.

```
<?php
if (isset($_POST['submit'])) {
    $first_name = $_POST['firstname'];
    $last_name = $_POST['lastname'];
    $email = $_POST['email'];
    $phone = $_POST['phone'];
    $job = $_POST['job'];
    $resume = $_POST['resume'];
    $output_form = 'no';

    if (empty($first_name)) {
        // $first_name is blank
        echo '<p class="error">You forgot to enter your first name.</p>';
        $output_form = 'yes';
    }

    if (empty($last_name)) {
        // $last_name is blank
        echo '<p class="error">You forgot to enter your last name.</p>';
        $output_form = 'yes';
    }

    if (empty($email)) {
        // $email is blank
        echo '<p class="error">You forgot to enter your email address.</p>';
        $output_form = 'yes';
    }

    if (empty($phone)) {
        // $phone is blank
        echo '<p class="error">You forgot to enter your phone number.</p>';
        $output_form = 'yes';
    }

    ...  
Continuing validating non-empty  
job and resume fields.
} else {
    $output_form = 'yes';
}

if ($output_form == 'yes') {
?>
...  
Show the form.
...
```

exercise solution



Below is some of the code for the registration.php script, which displays the user data entered into the form to register a new job candidate. Annotations explain what's wrong with the code, and how it could be changed to resolve the bad data problem.

```
<?php
if (isset($_POST['submit'])) {
    $first_name = $_POST['firstname'];
    $last_name = $_POST['lastname'];
    $email = $_POST['email'];
    $phone = $_POST['phone'];
    $job = $_POST['job'];
    $resume = $_POST['resume'];
    $output_form = 'no';

    if (empty($first_name)) {
        // $first_name is blank
        echo '<p class="error">You forgot to enter your first name.</p>';
        $output_form = 'yes';
    }

    if (empty($last_name)) {
        // $last_name is blank
        echo '<p class="error">You forgot to enter your last name.</p>';
        $output_form = 'yes';
    }

    if (empty($email)) {
        // $email is blank
        echo '<p class="error">You forgot to enter your email address.</p>';
        $output_form = 'yes';
    }

    if (empty($phone)) {
        // $phone is blank
        echo '<p class="error">You forgot to enter your phone number.</p>';
        $output_form = 'yes';
    }

    ...
    ... ← Continuing validating non-empty
    job and resume fields.

} else {
    $output_form = 'yes';
}

if ($output_form == 'yes') {
?>
    Show the form.
    ...
}
```

The script checks for empty form fields, which is good, but some of the form fields have more specialized data that adhere to a certain format.

There isn't much else we can check in regard to first and last names, so this code is fine.

An email address has a very specific format that we should enforce before accepting form data from the user.

Four Fingers McGraw left off his email address near the end of the form, so this part of the code should catch those errors.

Same thing with a phone number. If the user's form submission doesn't contain a phone number, unless we can be certain that the user means no phone number is in the correct field, we should catch that error.

What we really need is to validate email addresses and phone numbers separately. We have two fields in the form, so we can demand a specific format. For example, it's OK to just make sure the email address is well-formed, but it's not OK to just make sure the phone number is well-formed.

Jimmy Swift didn't provide an area code with his phone number, which the form should've demanded.



Why don't we use some string functions to fix the bad data?
Can't we use `str_replace()` to add in the missing data?

You can fix some data with string functions but they help much when data must fit a very specific pattern

String functions are well suited to simple find-and-replace operations. For example, if users submitted their phone numbers using dots to separate the blocks of digits instead of hyphens, we could easily write some `str_replace()` code to substitute in hyphens in their place.

But for anything that we can't possibly know, like the area code of Jimmy's phone number, we need to ask the person who submitted the form to clarify. And the only way we can know that he's missing an area code is to understand the exact pattern of a phone number. What we really need is more advanced validation to ensure that things like phone numbers and email addresses are entered exactly right.

OK, but can't we still use string functions to do this validation?

String functions really aren't useful for more than the most primitive of data validation.

Think about how you might attempt to validate an email address using string functions. PHP has a function called `strlen()` that will tell you how many characters are in a string. But there's no predefined character length for data like email addresses. Sure, this could potentially help with phone numbers because they often contain a consistent quantity of numbers, but you still have the potential dots, dashes, and parentheses to deal with.

Getting back to email addresses, their format is just too complex for string functions to be of much use. We're really looking for specific **patterns of data** here, which requires a validation strategy that can check user data against a pattern to see if it's legit. Modeling patterns for your form data is at the heart of this kind of validation.

define what your data should look like

Decide what your data should look like

Our challenge is to clearly specify exactly what a given piece of form data should look like, right down to every character. Consider Jimmy's phone number. It's pretty obvious to a human observer that his number is missing an area code. But form validation isn't carried out by humans; it's carried out by PHP code. This means we need to "teach" our code how to look at a string of data entered by the user and determine if it matches the pattern for a phone number.

Coming up with such a pattern can be a challenge, and it involves really thinking about the range of possibilities for a type of data. Phone numbers are fairly straightforward since they involve 10 digits with optional delimiters. Email addresses are a different story, but we'll worry about them a bit later in the chapter.

It's easy for a human to look and see that Jimmy forgot his area code, but not so trivial making the same "observation" from PHP code.

First Name: Jimmy
Last Name: Swift
Email: JS@sim-u-duck.com
Phone: 636 4652
Desired Job: Ninja



there are no Dumb Questions

Q: I'm still not sure I see why I can't just stick with `isset()` and `empty()` for our form validation.

A: These two functions will tell you whether or not someone who submitted a form put data in a text field, but they won't tell you anything about the actual data they entered. As far as the `empty()` function is concerned, there's absolutely no difference between a user entering "(707) 827-700" or "4FG8SXY12" into the phone number field in our form. This would be a huge problem for sites like Risky Jobs, which depends on reliable data to get in touch with job candidates.

Q: If `isset()` and `empty()` won't work, can't we simply have someone check the data after it goes into the database?

A: You can, but by then it's often too late to fix the bad data. If a phone number is missing an area code, we need to ask the user to clarify things by resubmitting the data in that form field. If you wait until later and check the data once it's already in the database, you may have no way of contacting the user to let them know that some of their data was invalid. And since the user probably won't realize they made a mistake, they won't know anything's wrong either.

So, the best plan is to validate the users form data before they submit the form. We can add an error message to the form again.

Q: So then I can validate the data the user has entered.

A: That depends on what you mean by it is. Different types of data require different rules that define what kind of characters are allowed. For example, many characters are allowed in email addresses, but those characters have specific meanings when communicating through the network. Let's take a closer look at the rules governing phone numbers.



Sharpen your pencil

Write down all the different ways you can think of phone number.

.....

.....

.....

.....

.....

.....

.....

.....

.....

What are some rules that are reasonable to expect your users to follow when filling out your form? For example, phone numbers should not contain letters.

Here's a rule to get you started.

We could insist on rules such as only digits and all 10 digits must be run together

.....
.....
.....
.....

sharpen your pencil solution

Sharpen your pencil Solution

Write down all the different ways you can think of phone number.

Spaces, dashes, right and left parentheses, and sometimes periods can show up in phone numbers.

It's even possible to include letters in a phone number, although this is stretching the limits of what we should consider a valid number.

555 636 4652
 (555) 636-4652
 (555)636-4652
 (555) 6364652
 555636-4652
 555 636-4652
 555.636.4652
 5556364652
 555-636-4652
 555 ME NINJA

Here's a rule to get you started.

What are some rules that are reasonable to expect your users to filling out your form? For example, phone numbers should not

We could insist on rules such as only digits and all 10 digits must run together

We could break the number into three form fields, one for area for the first three digits, and the final one for the last four

Or we could tell the people filling out the form that their number must look like (555)636-4652. It's up to us to make the rules

There are so many possible patterns. How can we make rules to cover all of them?



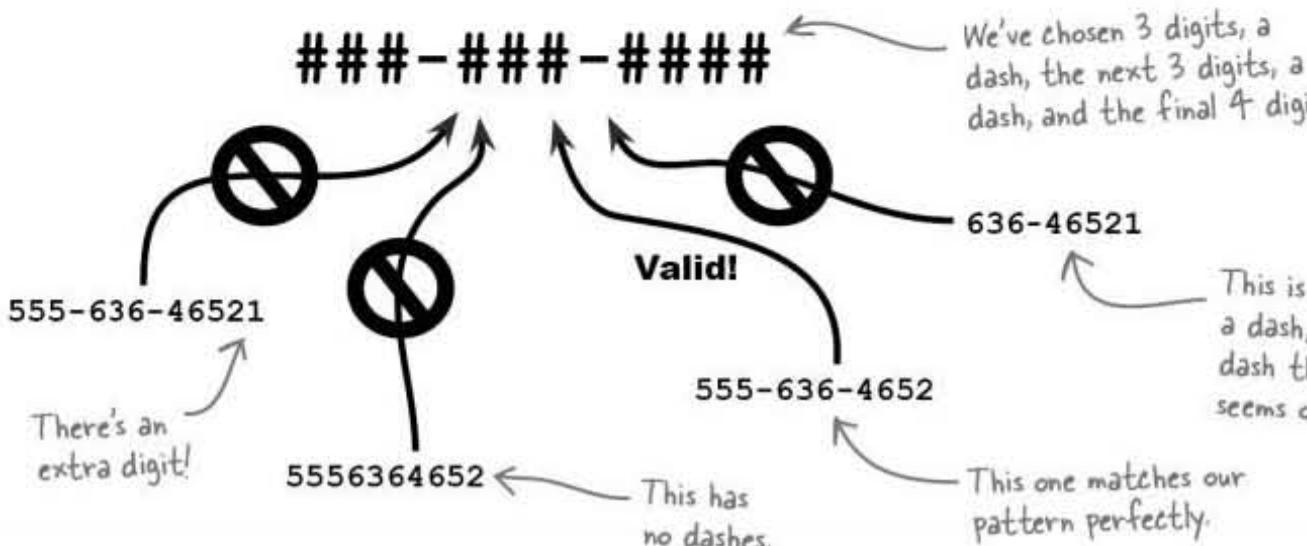
There are some things we know for sure about phone numbers, and we can use those things to our advantage.

First, they can't begin with 1 (long distance) or 0 (area code). They should be 10 digits. And even though some people like to include letters in their phone numbers, they're essentially numbers—10 digits when we include area codes and exchanges.

Formulate a pattern for phone numbers

To go beyond basic validation, such as `empty()` and `isset()`, we need to decide on a pattern that we want our data to match. In the case of a phone number, this means we need to commit to a single format that we expect to receive from the phone field in our form. Once we decide on a phone number format/pattern, we can validate against it.

Following is what is likely the most common phone number format in use today, at least for domestic U.S. phone numbers. Committing to this format means that if the phone number data users submit doesn't match this, the PHP script will reject the form and display an error message.



there are no Dumb Questions

Q: Do I have to use that pattern for matching phone numbers?

A: That's what we're using for Risky Jobs because it's pretty standard, but when you're designing your own forms you should pick one that makes sense to you. Just keep in mind that the more commonly accepted the pattern is, the more likely users will follow it.

Q: Couldn't I just tell users to enter a pattern like #####-#### and then use PHP's string functions to make sure the data contains 10 numeric characters?

A: You could, and that would be sufficient if that was the pattern your users expected. Unfortunately, it's not really a very good pattern because most people don't run their phone number together like that when filling out forms. It's a bit non-standard, which means users won't be used to it, and will be less likely to follow it.

Q: So? It's my pattern, I can do what I want, right?

A: Sure, but at the same time you want your users to have a good experience. Otherwise they'll quit visiting your site.

Q: OK, so can I just use a form with fields for the phone number, then validate the code, then the last four digits? I could use PHP's string functions to do that.

A: Yes, you can do that. But being a bit more flexible is useful for lots of things. Make sure your user experience is good in a phone number chapter.

introducing regular expressions

Match patterns with regular expressions

PHP offers a powerful way to create and match patterns in text. You can create rules that let you look for patterns in strings of text. These rules are referred to as **regular expressions**, or **regex** for short. A regular expression represents a pattern of characters to match. With the help of regular expressions, you can describe in your code the rules you want your strings to conform to in order for a match to occur.

As an example, here's a regular expression that looks for 10-digits in a row. This pattern will only match a string that consists of a 10 digit number. If the string is longer or shorter than that, it won't match. If the string contains anything but numbers, it won't match. Let's break it down.

The diagram shows the regular expression `/^\d\d\d\d\d\d\d\d\d\d$/`. Several annotations with arrows point to different parts of the expression:

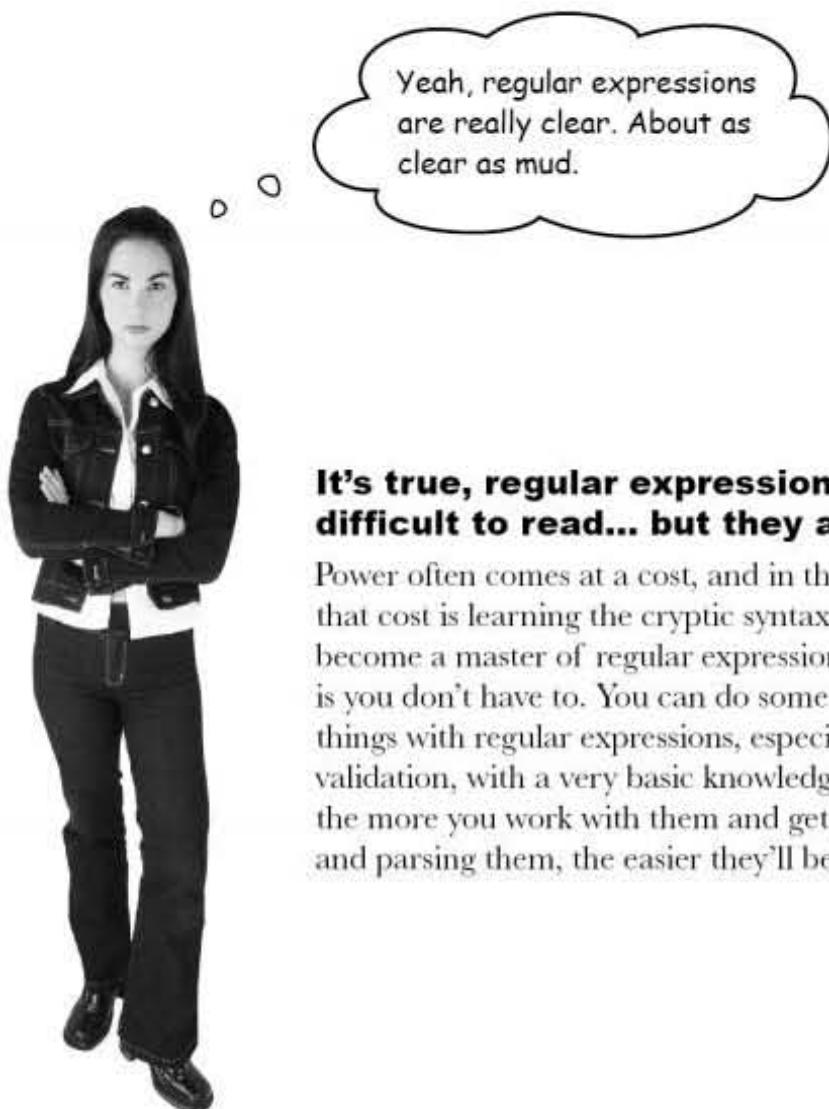
- An annotation at the top right states: "This part is easy. All regular expressions begin and end with forward slashes." It points to the outermost slashes.
- An annotation on the left points to the first character '^' and says: "The caret means to start matching at the beginning of the string."
- An annotation below the first '\d' character says: "\d stands for digit. The first character in the string must be a digit..."
- An annotation on the right points to the last '\d' character and says: "...and each one of these means the same thing, to look for another digit, 10 total."

There's also a more concise way of writing this same regular expression, which makes use of curly braces. Curly braces are used to indicate repetition:

`/^\d{10}\$/`

This means the same thing as the pattern above. `{10}` is a shorthand way to say 10 digits.

Regular expressions are rules used to match patterns one or more times.



It's true, regular expressions are cryptic and often difficult to read... but they are very powerful.

Power often comes at a cost, and in the case of regular expressions, that cost is learning the cryptic syntax that goes into them. You won't become a master of regular expressions overnight, but the good news is you don't have to. You can do some amazingly powerful and useful things with regular expressions, especially when it comes to form field validation, with a very basic knowledge of regular expressions. Besides, the more you work with them and get practice breaking them down and parsing them, the easier they'll be to understand.

common regex metacharacters

Build patterns using metacharacters

Being able to match digits in a text string using \d is pretty cool, but if that's all the functionality that regular expressions provided, their use would be sorely limited. Just matching digits isn't even going to be enough for Risky Jobs phone number validation functionality, as we're going to want to be able to match characters like spaces, hyphens, and even letters.

Luckily, PHP's regex functionality lets you use a bunch more special expressions like \d to match these things. These expressions are called **metacharacters**. Let's take a look at some of the most frequently used regex metacharacters.

\d

As you saw on the previous page, this metacharacter looks for a digit. It will match any number from 0 to 9. Keep in mind, on its own, \d matches just one digit, so if you wanted to match a two-digit number, you'd need to use either \d\d or \d{2}.

\s

Looks for whitespace. This doesn't mean just the space character you get on the screen when you hit the Space bar; \s also matches a tab character, or a newline or carriage return. Again, keep in mind that \s will only match one of these characters at a time. If you wanted to match exactly two space characters in a row, you'd need to use \s\s or \s{2}.

The period metacharacter, matches any one character, except a newline. It'll match a letter or digit, just like \w, as well as a space or tab, like \s.

Looks for any alphanumeric character, either a letter or a number. It includes the following: a-z and A-Z (uppercase and lowercase letters), as well as 0-9.

We saw the caret metacharacter as well. It looks for the beginning of a string. You can use it to indicate that a match must occur at the start of a text string, rather than anywhere in the string. For example, the regex / ^Nanny 411 / will match “Nanny 411” but not “I received 300 applications”.

\$

Looks for the end of a string. You can use the dollar sign (\$) metacharacter with ^ to bookmark a position exactly where it will start and finish. The regex / Nanny 411 \s\d{3}\\$/ will match “Nanny 411 is great” or “Call Nanny 411”.

These metacharacters are cool, but what if you really want a specific character in your regex? Just use that character in the expression. For example, if you wanted to match the exact phone number “707-827-7000”, you would use the regex /707-827-7000/.

Metacharacters
us describe
of text
regular

\w

^

\$

The logo features the words "WHO DOES WHAT?" in a stylized, hand-drawn font. A question mark is integrated into the letter "O". There are small star-like sparkles around the letters.

Match each different phone number regular expression with the phone number that it matches.

Regex

String it matches

5556364652

/^\d{3}\s\d{7}\$/

/^\d{3}\s\d{3}\s\d{4}\$/

555 636 4652

/^\d{3}\d{3}-\d{4}\$/

555636-4652

/^\d{3}-\d{3}-\d{4}\$/

555 ME NINJA

/^\d{3}\s\w\w\s\w{5}\$/

555 6364652

/^\d{10}\$/

555-636-4652

who does what solution**SOLUT**

Match each of the phone number regular expressions with the phone number that it matches.

`/^\d{3}\s\d{7}$/`

555636465

`/^\d{3}\s\d{3}\s\d{4}$/`

555 636 465

`/^\d{3}\d{3}-\d{4}$/`

555636-465

This is the pattern Risky Jobs
needs to match phone numbers in
the form ####-####-####.

`/^\d{3}-\d{3}-\d{4}$/`

555 ME NIN

`/^\d{3}\s\w\w\w\s\w{5}$/`

555 636465

The \w metacharacters in
this pattern match letters.

`/^\d{10}$/`

555-636-465

This pattern is all digits, so it
can only match a phone number
with no spaces or hyphens.



BE the Regular Expression

Your job is to play the role of regular expression, and either accept or reject phone numbers for Risky Jobs users. Check the box of phone numbers that you deem valid, and leave the others unchecked. Annotate why any invalid numbers are invalid.

This is the phone number regular expression - b

`/^\d{3}-\d{3}-\d{4}$/`



(555) 935-2659



(555)672-0953

555



Surfing can be quite
risky, especially
when you work as
professional shark bait!

555-441-9005



555.903.6386

555-61

be the regex solution

BE the Regular Expression Solution

Your job is to play the role of regular expression, and either accept or reject phone numbers for Risky Jobs users. Check the box of phone numbers that you deem valid, and leave the others unchecked. Annotate why any invalid numbers are invalid.

This is the phone number regular expression – b

`/^\d{3}-\d{3}-\d{4}$/`



Parentheses aren't allowed, and neither are spaces.

(555) 935-2659



(555)672-0953

555-3

No parentheses, please.



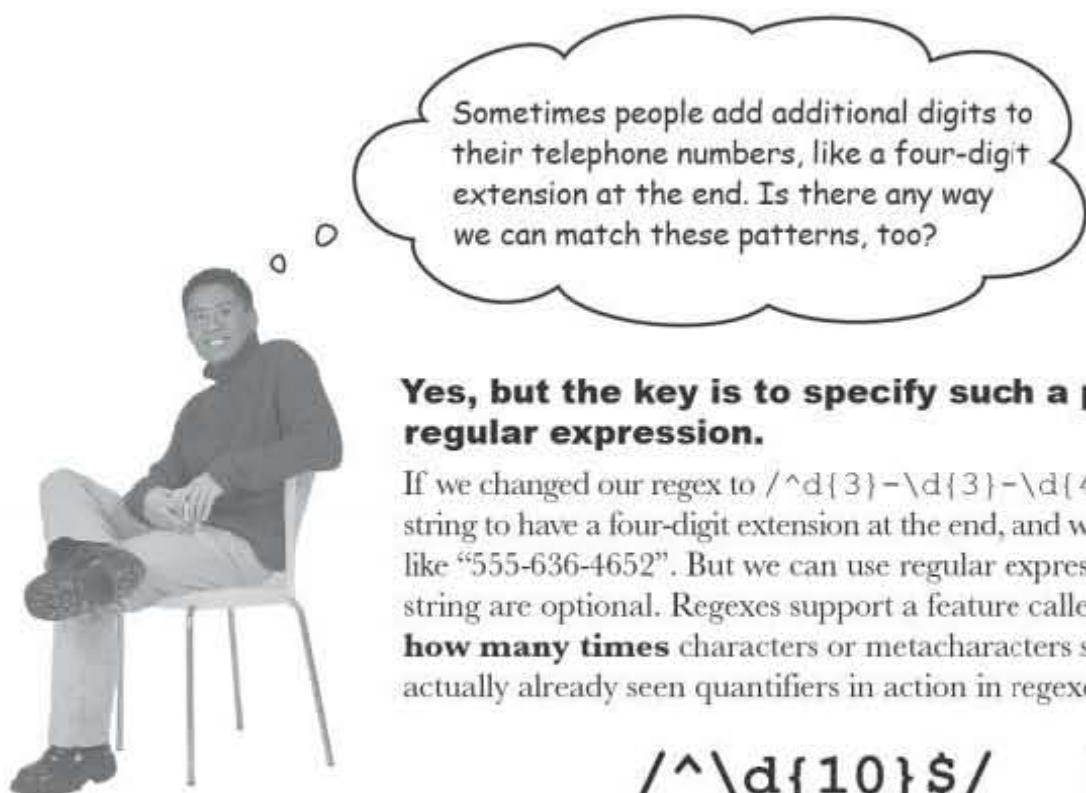
Our regular expression requires dashes, not dots.

555-441-9005



555.903.6386

555-612



Yes, but the key is to specify such a pattern as regular expression.

If we changed our regex to `/^\d{3}-\d{3}-\d{4}-\d{4}$`, we'd expect the string to have a four-digit extension at the end, and we'd no longer match strings like "555-636-4652". But we can use regular expressions to indicate that certain parts of the string are optional. Regexes support a feature called **quantifiers** that tell us how many times characters or metacharacters should appear. We've actually already seen quantifiers in action in regexes like this:

`/^\d{10}$`

This says "a digit must show up 10 times"

Here, curly braces act as a quantifier to say how many times the preceding character or metacharacter should appear. Let's take a look at some other frequently used quantifiers.

{min, max}

When there are two numbers in the curly braces, separated by a comma, this indicates a range of possible times the preceding character or metacharacter should be repeated. Here we're saying it should appear 2, 3, or 4 times in a row.

+

The preceding character or metacharacter must appear **one or more times**.

*

The character or metacharacter can appear **one or more times... or not at all**.

A quantifier specifies how many times a metacharacter should appear.

So, if we wanted to match those optional digits at the end of our telephone number, we could use the following pattern:

`/^\d{3}-\d{3}-\d{4}(-\d{4})$`

Surround the section the quantifier applies to in parentheses.

using character classes



You forgot one thing.
U.S. phone numbers
can't begin with 0 or 1.

You're absolutely right. 0 connects you to operator, and 1 dials long distance.

We simply want the area code and number. We need to make sure the first digit is not 1 or 0. And to do that, we need a **character class**.

Character classes let you match characters from a specific set of values. You can look for a range of digits with a character class. You can also look for a set of values. And you can add a caret to look for everything that **isn't** in the set.

To indicate that a bunch of characters or metacharacters belongs to a character class, all you need to do is surround them by square brackets, `[]`. Let's take a look at a few examples of character classes in action:

[0-2]

This matches a range of numbers. It will match 0, 1, or 2.

[A-D]

This will match A, B, C, or D.

In a character class, the caret (^) means "don't match".

[^b-f]

This caret has a different meaning when it's inside a character class. Instead of saying "the string must start with...", the caret means "everything except..."

This will match everything except b, c, d, e, and f.

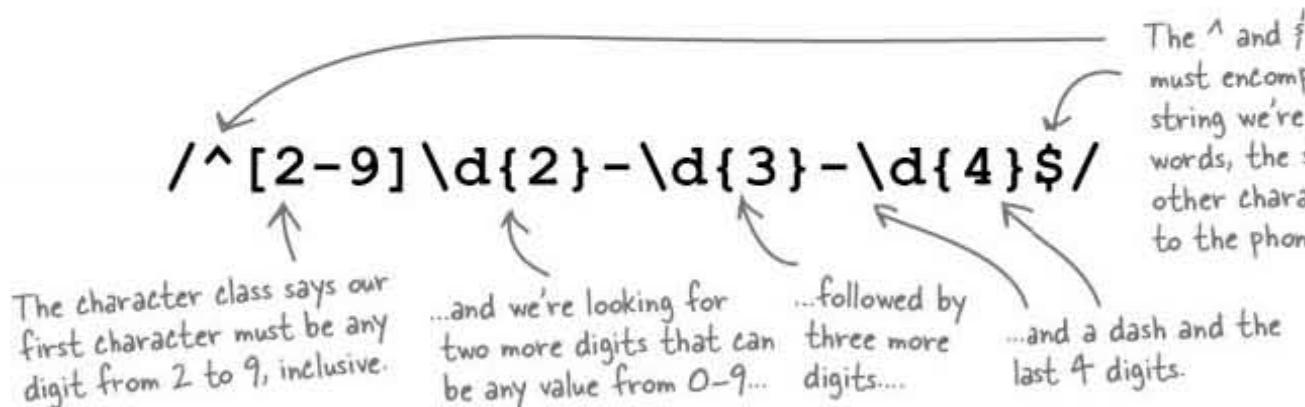
**A character class
is a set of rules
for matching a
single character.**

Write a regular expression that matches invalid numbers:

.....
.....
.....

Fine-tune patterns with character classes

With the help of character classes, we can refine our regular expression for phone numbers so that it won't match invalid digit combinations. That way, if someone accidentally enters an area code that starts with 0 or 1, we can throw an error message. Here's what our new-and-improved regex looks like:



there are no Dumb Questions

Q: So character classes let you specify a range of characters that will match the text string.

A: Yes, a character class lets you specify in your regular expression that any member of a specified set of characters will match the text string, instead of just one.

For example, the character class [aeiou] will match one instance of any lowercase vowel, and the class [m-zA-Z] will match one instance of any letter in the second half of the alphabet, lowercase or uppercase.

And the character class [0-9] is equivalent to the metacharacter \d, which is really just a shorthand way of saying the same thing.

Q: Don't I need to put spaces or commas in between the characters or ranges I specify in character classes?

A: No, if you do that, those extra characters will be interpreted as part of the set of characters that should match the text string. For example, the character class

[m-z, M-Z]

would match not only uppercase and lowercase letters from m to z, but also a comma or space, which is probably not what you want.

Q: What if I want to match a character more than once? Like one or more vowels.

A: Just add a quantifier after the character class. / [aeiouAEIOU] + / will match one or more vowels.

Q: I thought quantifiers only applied to the character immediately preceded them.

A: Usually that's the case, but if a quantifier appears after a character class, it applies to the whole class. And if you want to make a quantifier apply to individual characters that aren't in a character class, use parentheses to indicate the range together. As an example, the regular expression [a-zA-Z]{3} will match one or more consecutive instances of letters in the range a-zA-Z in the text string.

Q: What if I wanted to match two different words like "ketchup" or "catsup"?

A: You can use the vertical-pipe character class expression to indicate a set of options to choose from. So, the regular expression / (ketchup|catsup|barbecue) / will match any one of the three most common words.

escaping reserved characters

What about putting characters like periods or question marks in a regular expression. If I type those in, won't think they're metacharacters or quantifiers and screw processing my regex?

If you want to use reserved characters in your regular expression, you need to escape them.

In regular expression syntax, there are a small set of characters that have special meaning, because they are used to signify things like metacharacters, quantifiers, and character classes. These include the period (.), the question mark (?), the plus sign (+), the opening square bracket ([), opening and closing parentheses, the caret (^), the dollar sign (\$), the vertical pipe character (|), the backslash (\), the forward slash (/), and the asterisk (*).

If you want to use these characters in your regular expression to signify literal meaning instead of the metacharacters or quantifiers they usually represent, you need to “escape” them by preceding them with a backslash.

For example, if you wanted to match parentheses in a phone number, you couldn't just do this:

(555)636-4652



These will simply be treated like a group.

/^(\\d{3})\\d{3}-\\d{4}\$/

Instead, both the opening and closing parentheses need to be preceded by backslashes to indicate that they should be interpreted as actual parentheses:

(555)636-4652



Now PHP knows that these are literal parentheses.

/^\\\\(\\d{3}\\\\)\\d{3}-\\d{4}\$/



Come up with a string that will match each pattern shown.

`/^ [3-6] {4} /`

`/^ ([A-Z] \d)`

Suppose we want to expand the Risky Jobs validation scheme for phone numbers to allow submit their numbers in a few more formats. Write a single regular expression that will mate the following text strings, and won't allow a 0 or 1 as the first digit. Your pattern should only parentheses, spaces, and dashes.

555-636-4652 555 636-4652

(555)-636-4652 (555) 636-4652

exercise solution



Exercise Solution

Come up with a string that will match each pattern shown.

String must begin with...

3 through 6... and repeat that character class four times.

`/^ [3-6] {4} /`

String must begin with...

an uppercase letter... and a digit...

`/^ ([A-Z] \d) {`

Any string that starts with four digits in the range 3, 4, 5, or 6 will match. These will all match:

"5533", "3546 is a number.", "6533xyz"

Any string that starts with an uppercase letter and then a digit

"B5C9", "R2D2"

Suppose we want to expand the Risky Jobs validation scheme for phone numbers to allow submit their numbers in a few more formats. Write a single regular expression that will match the following text strings, and won't allow a 0 or 1 as the first digit. Your pattern should only parentheses, spaces, and dashes.

**555-636-4652 555 636-4652
(555)-636-4652 (555) 636-4652**

String must begin with...

2 through 9... twice... a hyphen or a space...

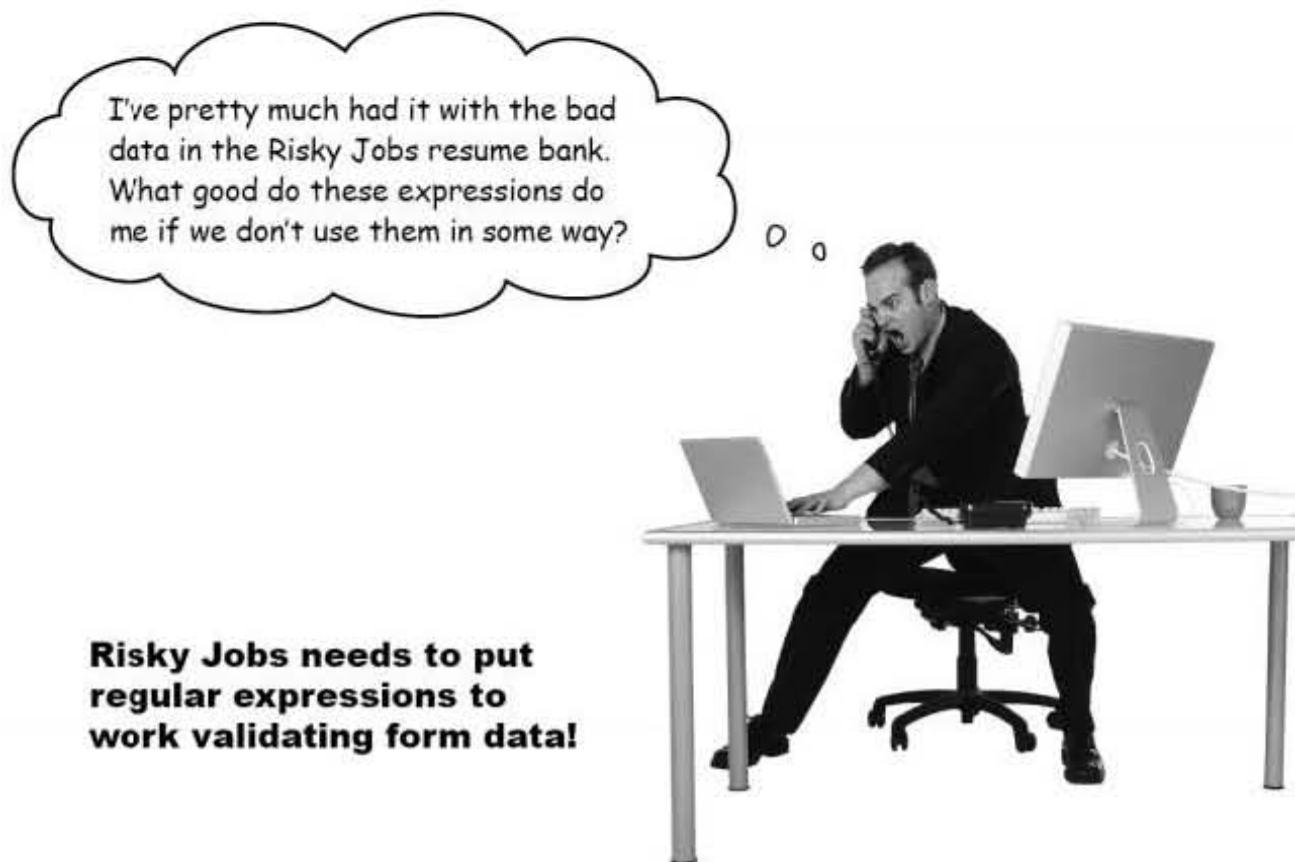
`/^ \((? [2-9] \d{2}) \)? [-\s] \d{3} - \d{4}`

an optional open parenthesis, which may appear 0 or 1 times...

and a digit...

an optional close parenthesis...

and a digit, three times...



the php preg_match() function

Check for patterns with preg_match()

We haven't been developing patterns just for the fun of it. You can use these patterns with the PHP function `preg_match()`. This function takes a regex pattern, just like those we've been building, and a text string. It returns `false` if there is no match, and `true` if there is.

`preg_match($regex, $my_string)`

Your regex goes here. The function expects a string, which means the regex should be surrounded by single quotes.

The string you are checking for a match goes here.

Here's an example of the `preg_match()` function in action, using a regex that searches a text string for a four-character pattern of alternating uppercase letters and digits:

When regexes are passed to `preg_match()`, they should be enclosed in quotes.

`preg_match('/^d{3}-d{2}-d{4}$/', '55`

>Returns an integer: 1 if the string matches the pattern, and 0 if it doesn't.

You can put the actual pattern in the function like this, but usually it's best to store it in a variable.

We can take advantage of the `preg_match()` function to enable more sophisticated validation functionality in PHP scripts by building an `if` statement around the return value.

The preg is nested so its re what co

```
if (preg_match('/^d{3}-d{2}-d{4}$/', '555-02-998
```

echo 'Valid social security number.';

} else {

echo 'That social security number is invalid!'

}

If the preg mo signifies is true.

If the match is not successful, `preg_match()` returns `false`, which makes the condition evaluate to `false`. So, this code is run.



Rewrite the highlighted portion of the Risky Jobs PHP script for checking the data below to validate the text entered into the phone field using `preg_match_empty()`. Use the regex you created earlier in the `preg_match()` function.

```
if (empty($phone)) {  
    // $phone is blank  
    echo '<p class="error">Your phone number is invalid</p>';  
    $output_form = 'yes';  
}
```

.....
.....
.....
.....

exercise solution

Rewrite the highlighted portion of the Risky Jobs PHP script for checking the data below to validate the text entered into the phone field using `preg_match()`. Use the regex you created earlier in the `preg_match()` function.

```
if (empty($phone)) {
    // $phone is blank
    echo '<p class="error">Your phone number is invalid.</p>';
    $output_form = 'yes';
}
```

Instead of `empty()`, we use a `preg_match` to validate the phone number. We precede it with the not operator (!), because we want to throw an error whenever the data entered DOESN'T match the pattern.

Our phone number regular expression from before.

```
if (!preg_match('/^(\d{2}-\d{2})\d{3}-\d{4}/', $phone)) {
    // $phone is not valid
    echo '<p class="error">Your phone number is invalid.</p>';
    $output_form = 'yes';
}
```

The echo needs a bit, since we're making sure it matches the phone number.

We set `$output_form` to 'yes', just as before.

I got an error and then entered my entire phone number. And then I got a ninja job!

First Name: Jimmy
Last Name: Swift
Email: JS@sim-u-duck.com
Phone: (555) 636 4652
Desired Job: Ninja





Test DRIVE

Not just empty
phone numbers!

Check for valid phone numbers in the Risky Jobs Registration

Download the registration.php script from the Head First Labs site at www.headfirstlabs.com/books/hfphp, along with the Risky Jobs style sheet and images (riskyjobs_title.gif and riskyjobs_fireman.png). Then edit the registration.php script so that it uses the preg_match() function to check phone numbers against the phone number regular expression. Make sure to tweak the message so that users know the phone number is invalid, not just empty.

Upload the changed script to your web server, and then open it in a web browser. Enter a few phone numbers with varying formats, and notice how the script catches them.

Now we can't accidentally enter bad phone numbers. That should keep us from missing job opportunities!



Risky Jobs - Registration

Risky Jobs

Danger! Your dream job is out there.
Do you have the guts to go find it?

Risky Jobs - Registration

Your phone number is invalid.

Register with Risky Jobs, and post your resume.

First Name:

Last Name:

Email:

Phone:

Desired Job:

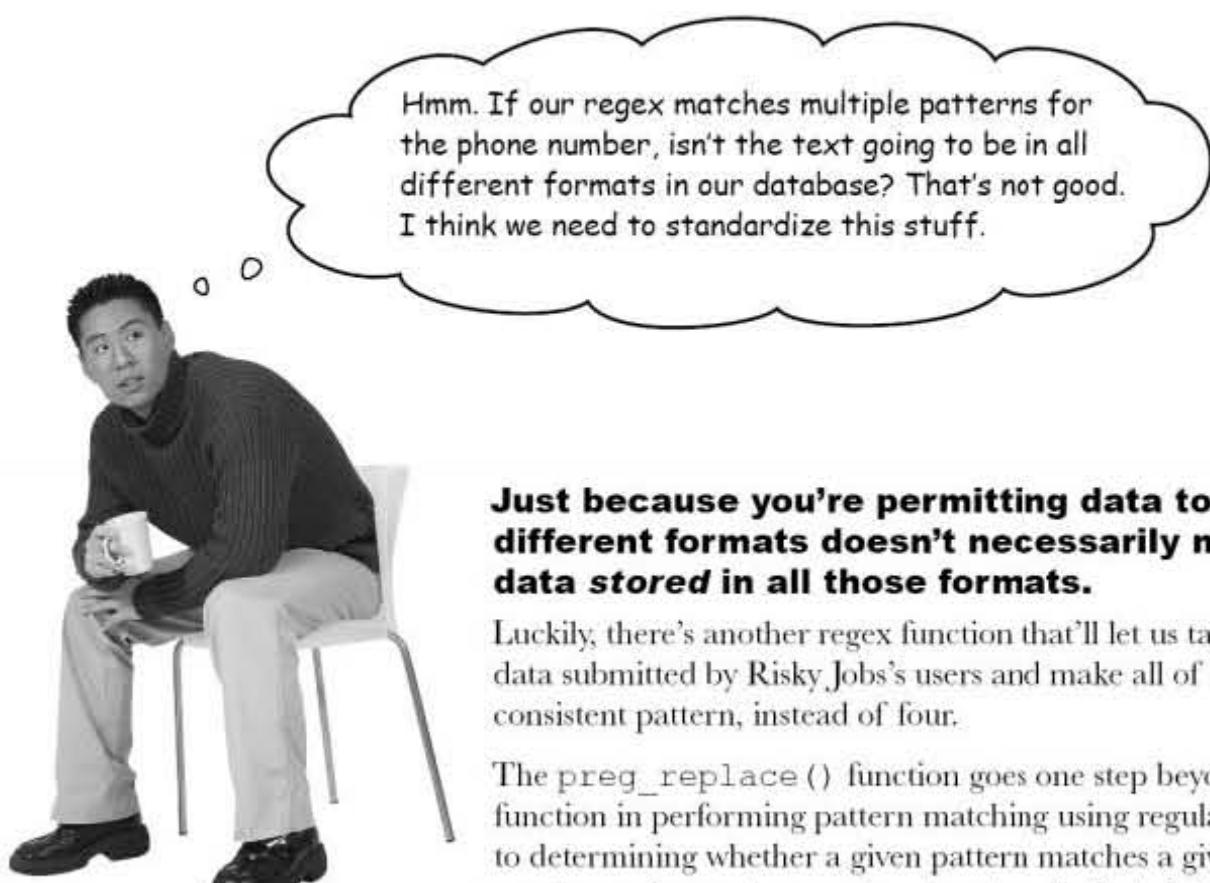
Paste your resume here:

I'm a swashbuckler, been doing it for years. I'm young with broomsticks but quickly learn swords. Now I'll fence anything. I'm certified from the Royal Fencing Institute.

Download It!

The complete source code application is available for download from the Head First Labs web site.

www.headfirstlabs.com

the php preg_replace() function

Just because you're permitting data to be in different formats doesn't necessarily mean data stored in all those formats.

Luckily, there's another regex function that'll let us take the data submitted by Risky Jobs's users and make all of it conform to a single, consistent pattern, instead of four.

The `preg_replace()` function goes one step beyond the `preg_match()` function in performing pattern matching using regular expressions. Instead of just telling you if a pattern matches a string, it lets you supply a replacement pattern to substitute into the string whenever a match is found. It's a lot like the `str_replace()` function, except that it matches using a regular expression instead of a simple string.

`preg_replace($pattern, $replacement, $subject)`

We need to find these unwanted characters.

When we find an unwanted character, we want to turn it into this.

The string we're replacing the first character of.

Here's an example of the `preg_replace()` function in action:

```
$new_year = preg_replace('/200[0-9]/', '2010', 'The year is 2009.'
```

The result of the `preg_replace()` function, our revised text string after the find-and-replace is complete, is stored in `$new_year`.

This regex tells `preg_replace` to look for a match for 2000 through 2009.

When a match is found, it will be replaced with 2010.

Even though 2009 is in our original string, it's not replaced.



Standardize the phone number data entered into the Risky Jobs form by writing the following numbers in the phone column of the database table below. Make sure that stores as little data as possible to represent a user's phone number.

(555) 935-2659

(555)672-0953

555-343-8263

555-441-9005

555.903.6386

555-612-8527-8724

Risky Jobs - Registration

Risky Jobs

Danger! Your dream job is out there.
Do you have the guts to go find it?

Risky Jobs - Registration

Register with Risky Jobs, and post your resume.

First Name:

Last Name:

Email:

Phone:

Desired Job:

Paste your resume here:

phone
...

exercise solution

Standardize the phone number data entered into the Risky Jobs form by writing a program that stores as little data as possible to represent a user's phone number.

(555) 935-2659

(555)672-0953

555-343-8263

555-441-9005

555.903.6386

555-612-8527-8724

The cleanest way to store a phone number is to strip out everything but the numeric digits.

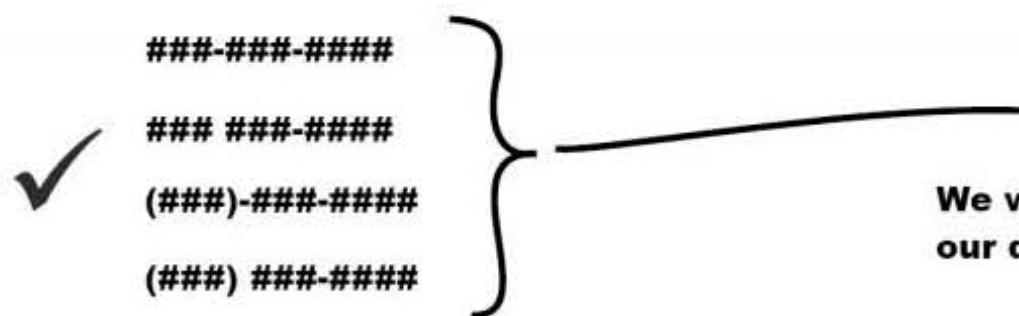
phone
...
5559352659
5556720953
5553438263
5554419005
5559036386
5556128527

Standardize the phone number data

Right now, Risky Jobs is using the following regular expression to validate the phone numbers users submit via their registration form:

```
/^\(?[2-9]\d{2}\)?[-\s]\d{3}-\d{4}$/
```

This will match phone numbers that fall into these four patterns:



While these formats are easily interpreted by people, they make it difficult for SQL queries to sort results the way we want. Those parentheses will most likely foil our attempts to group phone numbers by area code, for example, which might be important to Risky Jobs if we want to analyze how many of the site's users came from a specific geographical location.

To make these kinds of queries possible, we need to standardize phone numbers to one format using `preg_replace()` before we `INSERT` data into the database. Our best bet is to get rid of all characters except numeric digits. That way, we simply store 10 digits in our table with no other characters. We want our numbers to be stored like this in the table:

←

This leaves us with four characters to find and replace. We want to find and **remove** open and closing parentheses, spaces, and dashes. And we want to find these characters no matter where in the string they are, so we don't need the starting carat (^) or ending dollar sign (\$). We know we're looking for any one of a set, so we can use a character class. The order of the search doesn't matter. Here's the regex we can use:

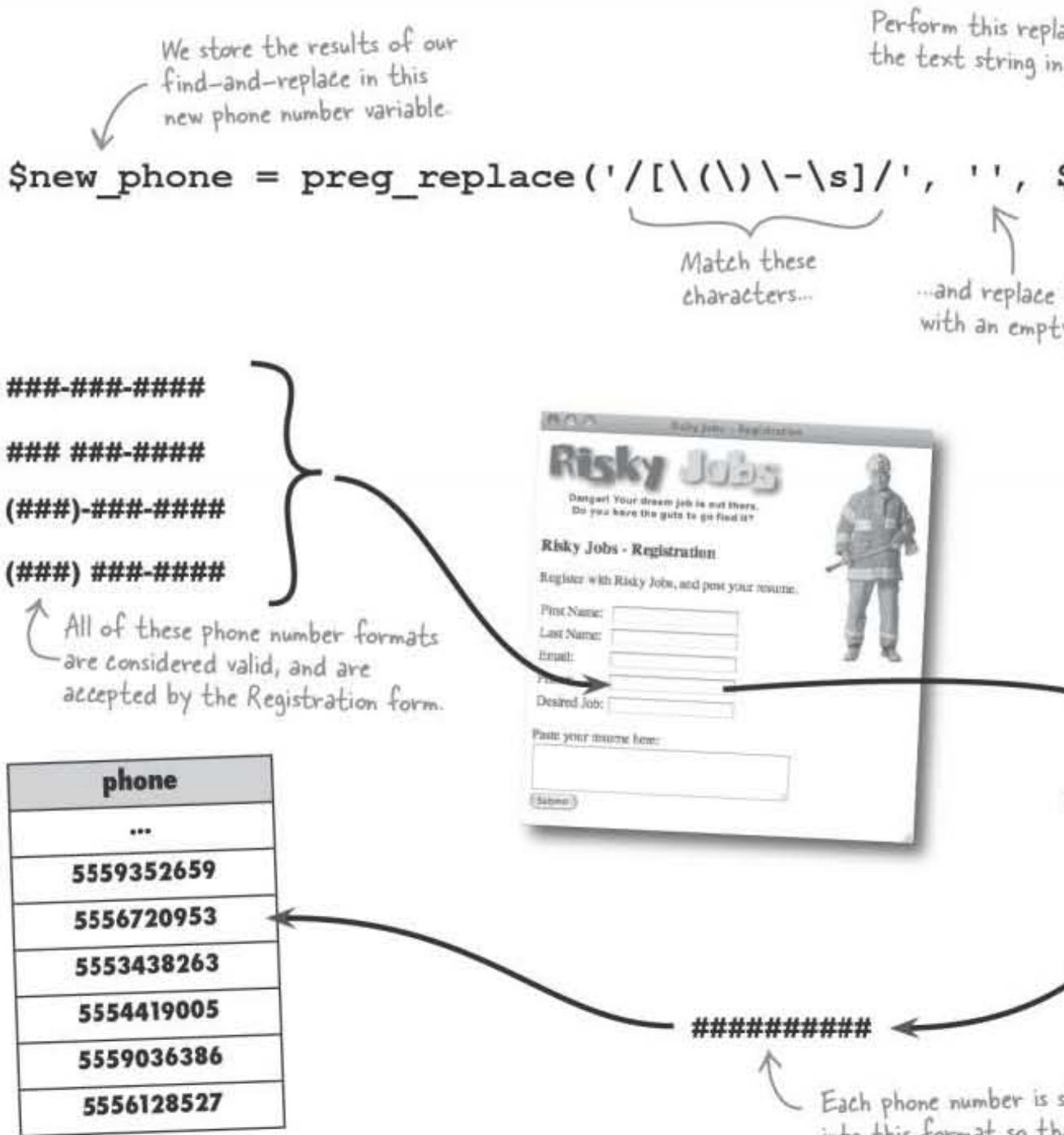
/ [\(\)\) \- \s] /

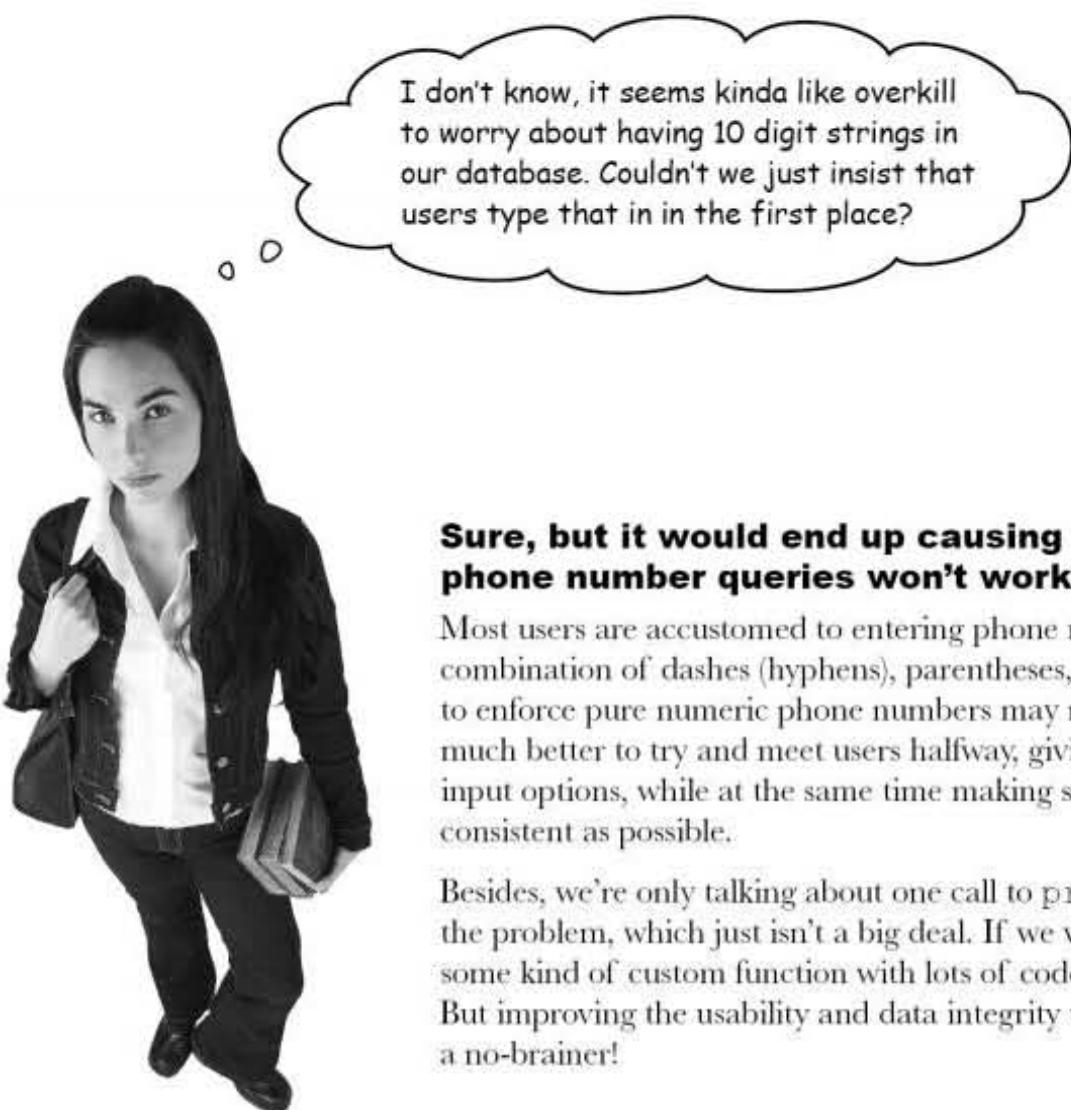
open parenthesis closing parenthesis dash space

stripping characters with preg_replace()

Get rid of the unwanted characters

Now that we have our pattern that finds those unwanted characters, we can apply it to phone numbers to clean them up before storing them in the database. But how? This is where the `preg_replace()` function really pays off. The twist here is that we don't want to replace the unwanted characters, we just want them gone. So we simply pass an empty string into `preg_replace()` as the replacement value. Here's an example that finds unwanted phone number characters and replaces them with empty strings, effectively getting rid of them:





I don't know, it seems kinda like overkill to worry about having 10 digit strings in our database. Couldn't we just insist that users type that in in the first place?

Sure, but it would end up causing problems later
phone number queries won't work as expected

Most users are accustomed to entering phone numbers with some combination of dashes (hyphens), parentheses, and spaces, so attempting to enforce pure numeric phone numbers may not work as expected. It's much better to try and meet users halfway, giving them reasonable input options, while at the same time making sure the data you store is consistent as possible.

Besides, we're only talking about one call to `preg_replace()` to fix the problem, which just isn't a big deal. If we were talking about some kind of custom function with lots of code, it might be a different story. But improving the usability and data integrity with a single line of code is a no-brainer!

test out registration.php



— Test Drive —

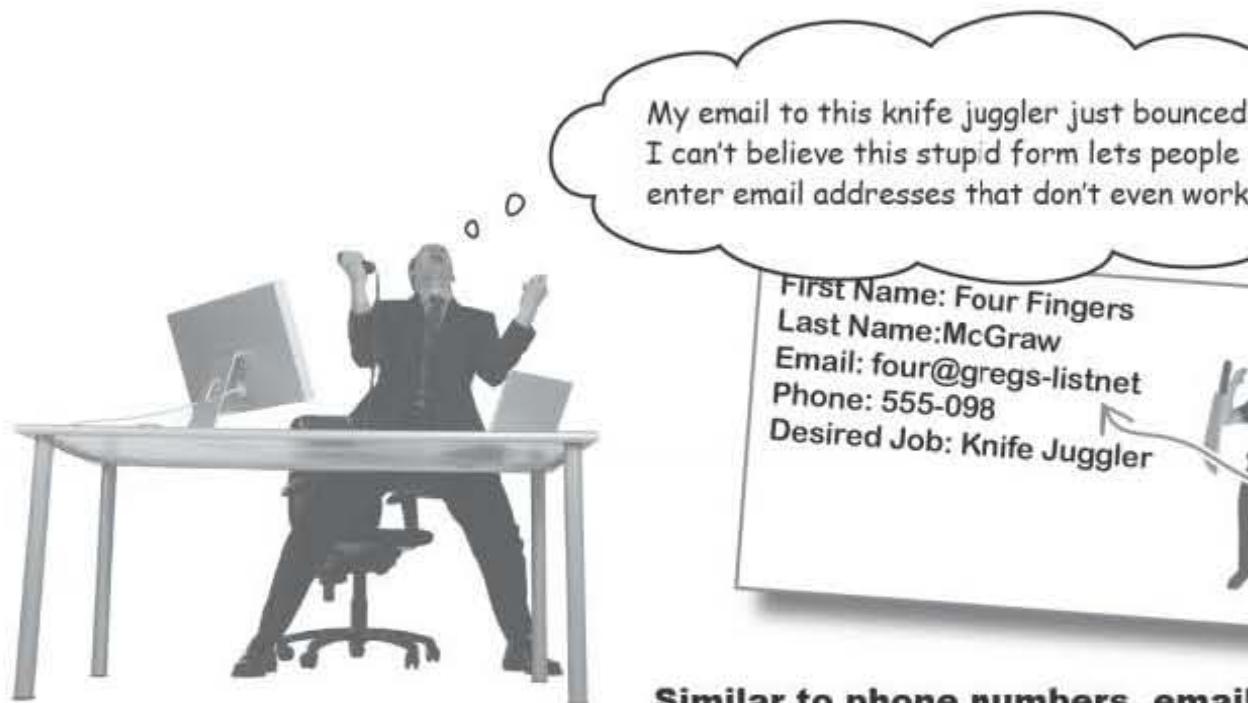
Clean up phone numbers in the Registration script.

Modify the registration.php script to clean up phone numbers by adding the following lines of code to the script, right after the line that thanks the user for registering with Risky Jobs:

```
$pattern = '/[\(\)\)-\s]/';
$replacement = '';
$new_phone = preg_replace($pattern, $replacement, $phone);
echo 'Your phone number has been registered as ' . $new_p
```

Upload the script to your web server, and then open it in a web browser. Fill out the form, making sure to enter a phone number with extra characters, such as (707) 7000. Submit the form and check out the results.

Try out a few other variations on the number, like these: 707.827.7000, (707)-827-7000. Notice how the regular expression and preg_replace() get rid of the extra characters.



Similar to phone numbers, email addresses have enough of a format to them to be validating for more than just numbers.

Just like with validating phone numbers earlier, we can determine the rules that valid email addresses must follow. We can formalize them as a regular expression and use them in our PHP script. So let's first take a look at what makes up an email address:

We know email addresses must contain these two characters:

LocalName@DomainPrefix.Domain

These will contain alphanumerics, where LocalName is at least one character and DomainPrefix is at least two characters.



See if you can come up with a regular expression that is flexible enough to match the email addresses to the right. Write it below:

.....

aviator.howard

cube_1c

a regex for email addresses

Matching email addresses can be tricky

It seems like it should be pretty simple to match email addresses, because at first glance, there don't appear to be as many restrictions on the characters you can use as there are with phone numbers.

For example, it doesn't seem like too big of a deal to match the *LocalName* portion of an email address (everything before the @ sign). Since that's just made up of alphanumeric characters, we should be able to use the following pattern:

`/^\w+ /`

Starts with...

...one or more alphanumeric characters.

This would allow any alphanumeric character in the local name, but unfortunately, it doesn't include characters that are also legal in email addresses.

Believe it or not, valid email addresses can contain any of these characters in the *LocalName* portion, although some of them can't be used to start an email address:

! \$ & * - = ^ ` | ~ # % ' + / ? _

If we want to allow users to register that have email addresses containing these characters, we really need a regex that looks something more like this:

`/^ [a-zA-Z0-9] [a-zA-Z0-9\._\-&!?=#]*`

All the can ap Local email a

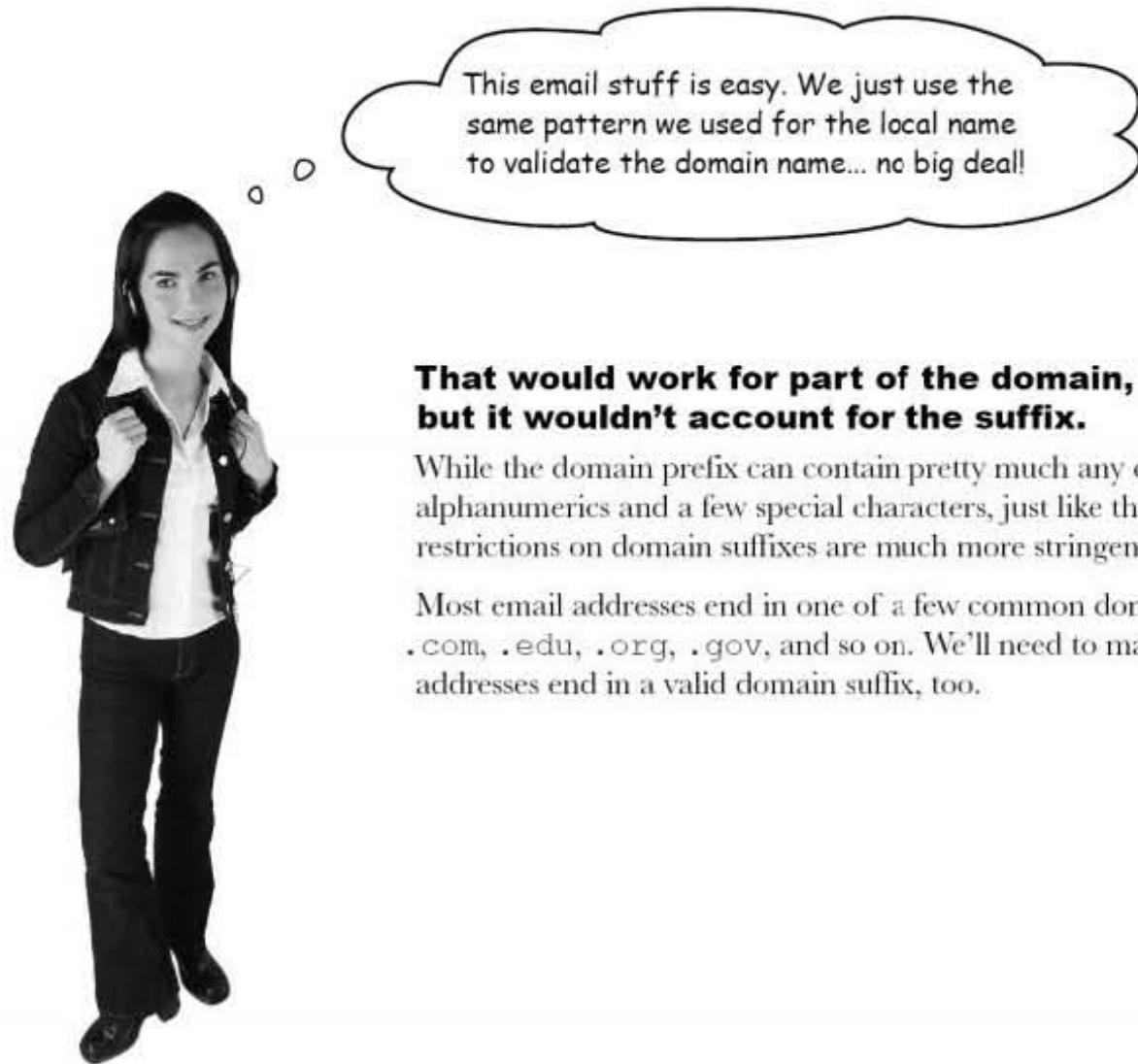
First character should be one of these.

The rest of the characters can be any of these...

...and we or more

This won't match every single valid *LocalName*, as we're still skipping some of the really obscure characters, but it's very practical to work with and should still match the email addresses of most of Risky Jobs users.

!\$&*-^`|~#%



**That would work for part of the domain, the
but it wouldn't account for the suffix.**

While the domain prefix can contain pretty much any combination of alphanumerics and a few special characters, just like the local name, restrictions on domain suffixes are much more stringent.

Most email addresses end in one of a few common domain suffixes: .com, .edu, .org, .gov, and so on. We'll need to make sure our email addresses end in a valid domain suffix, too.

there are no
Dumb Questions

Q: What if I want to allow every possible valid email address?

A: You can, and if it makes sense for your web site you certainly should. But sometimes it's best to take commonly accepted formats and not necessarily accept every possible variation. You need to decide what 99.9% of your users will have as their emails and be willing to not validate the remaining .1% simply for the sake of more streamlined code. Validation is really a trade-off between what's allowed and what is practical to accept.

If you do want to implement more robust email validation on your site, you can find some great open source (i.e., free) PHP code here: <http://code.google.com/p/php-email-address-validation/>

Q: Won't people get angry at me if I validate every email address I refuse to validate?

A: Possibly, but most people will not be angry. Most online email services have their own ways of preventing users from creating crazy, although valid, addresses like "i'm crazy"@gregs-list.com.

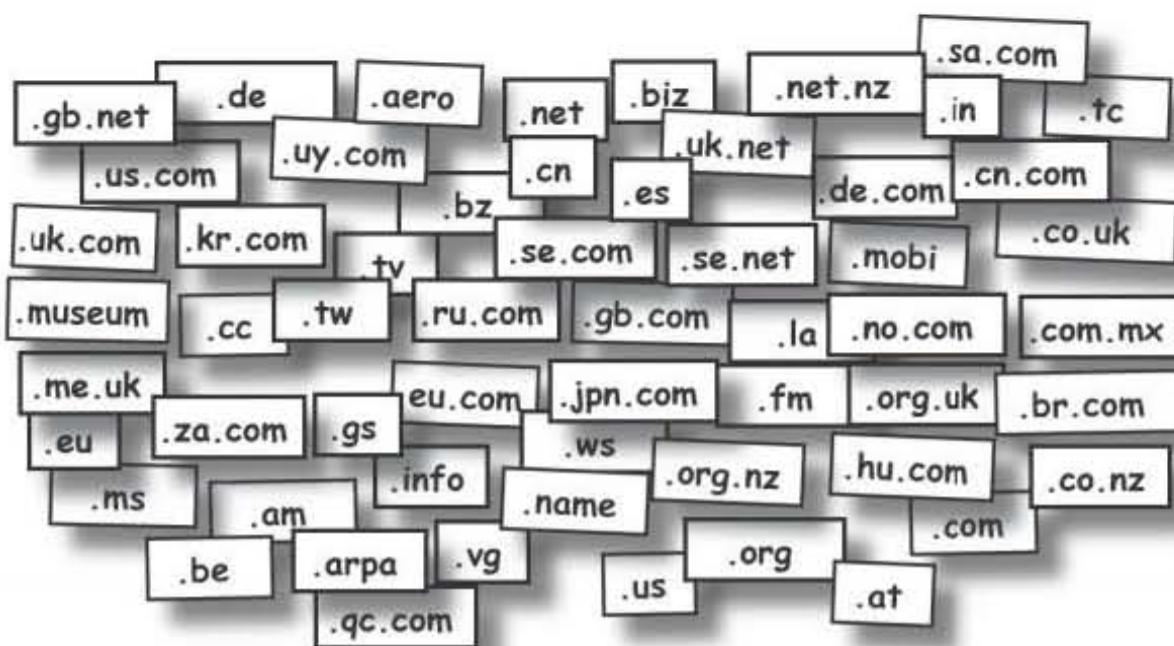
**Validation is often
between what's all
what is practical to accept**

matching domain suffixes

Domain suffixes are everywhere

In addition to super-common domain suffixes that you see quite frequently, like .com and .org, there are many, many other domain suffixes that are valid for use in email addresses. Other suffixes recognized as valid by the Domain Name System (DNS) that you may have seen before include .biz and .info. In addition, there's a list of suffixes that correspond to different countries, like .ca for Canada and .tj for Tajikistan.

Here is a list of just a few possible domain suffixes. This is not all of them.



Some of those domains are only two letters long. Some have 2 or 3 letters, and a period and then two or three letters. Some are even 4 and 5 letters long. So do we need to keep a list of them and see if there's a match?



We could do that, and it would work.

But there's an easier way. Instead of keeping track of all the possible domains and having to change our code if a new one is added, we can check the domain portion of the email address using the PHP function `checkdnsrr()`. This function connects to the Domain Name System, or DNS, and checks the validity of domains.

Use PHP to check the domain

PHP provides the `checkdnsrr()` function for checking whether a domain is valid. This method is even better than using regular expressions to match the pattern of an email address, because instead of just checking if a string of text could possibly be a valid email domain, it actually checks the DNS records and finds out if the domain is actually registered. So, for example, while a regular expression could tell you that `1asdjlkdflsalkjaf.com` is valid, `checkdnsrr()` can go one step further and tell you that, in fact, this domain is not registered, and that we should probably reject `sdfhfdskl@1asdjlkdflsalkjaf.com` if it's entered on our registration form.

The syntax for `checkdnsrr()` is quite simple:

Returns 1 if it's a real domain
or 0 if it's not

`checkdnsrr()` expects
containing a domain
everything after the

`checkdnsrr('headfirstlabs.com')`



If you're running PHP on a Windows server,
this command won't work for you.

Watch it!

Instead, you can use this code:

```
function win_checkdnsrr($domain, $recType='') {
    if (!empty($domain)) {
        if ($recType=='') $recType="MX";
        exec("nslookup -type=$recType $domain", $output);
        foreach($output as $line) {
            if (preg_match("/^$domain/", $line)) {
                return true;
            }
        }
        return false;
    }
    return false;
}
```

This is only
is Windows.
computer to
you're actually
Linux server

This executes
an external
running on the
check the domain

Just for fun,
the foreach:
Server: 68.87.64.14
oreilly.com
oreilly.com

email validation in five steps

Email validation: putting it all together

We now know how to validate both the *LocalName* portion of an email address using regular expressions, and the domain portion of an email address using `checkdnsrr()`. Let's look at the step-by-step of how we can put these two parts together to add full-fledged email address validation to Risky Jobs's registration form:

- 1 Use `preg_match()` to determine whether the *LocalName* portion of our email address contains a valid pattern of characters.

We can use the following regex to do so:

`/^ [a-zA-Z0-9] [a-zA-Z0-9\._\-\&!?=#]* @ /`

The email must start with an alphanumeric character, and then can contain any number of alphanumerics and certain special characters.

This time, we'll also add an at symbol (@), which separates our email address from the domain name.

- 2 If validation of the *LocalName* fails, echo an error to the user and reload the form.
- 3 If validation of the *LocalName* succeeds, pass the domain portion of the text string users submitted to `checkdnsrr()`.
- 4 If `checkdnsrr()` returns 0, then the domain is not registered, so we echo an error to the user and reload the form.
- 5 If `checkdnsrr()` returns 1, then the domain is registered, and we can be fairly confident that we've got a valid email address. We can proceed with validating the rest of the fields in the form.



Below is new PHP code to validate users' email addresses, but some pieces are missing. Fill in the blanks to get the code up and running.

```
if (!preg_match('.....', $email)) {  
    // $email is invalid because LocalName is bad  
    echo 'Your email address is invalid.<br />';  
    $output_form = 'yes';  
}  
  
else {  
    // Strip out everything but the domain from the email  
    $domain = preg_replace('.....',  
        // Now check if $domain is registered  
        if (.....) {  
            echo 'Your email address is invalid. <br />';  
            $output_form = 'yes';  
        }  
    }  
}
```

exercise solution

Below is new PHP code to validate users' email addresses, but some pieces are missing. Fill in the blanks to get the code up and running.

Our regex for matching the LocalName portion of an email address, ending in an at symbol.

```
if (!preg_match(' [a-zA-Z0-9][a-zA-Z0-9\.\-\&?=#]*@/ ', $email)) {
    // $email is invalid because LocalName is bad
    echo 'Your email address is invalid.<br />';
    $output_form = 'yes';
}
else {
    // Strip out everything but the domain from the email
    $domain = preg_replace('/^([a-zA-Z0-9][a-zA-Z0-9\.\-\&?=#]*@[a-zA-Z0-9\.\-\&?=#]*\.[a-zA-Z]{2,})$/i', '', $email);
    // Now check if $domain is registered
    if (!checkdnsrr($domain)) {
        echo 'Your email address is invalid. <br />';
        $output_form = 'yes';
    }
}
```

To strip out the LocalName after the at symbol, specify the empty string as the replacement string.

If you're on a Windows server, don't forget to include the code for `win_checkdnsrr()`, and then call it here.

Perform the registration check on the \$domain variable.

BULLET POINTS

- `preg_match()` locates matches for patterns in strings.
- `preg_replace()` changes matching strings.
- **Quantifiers** allow you to control how many times a character or set of characters can appear in a row.

- You can specify a set of characters in your pattern using a **character class**.
- In your pattern, `\d`, `\w`, and `\s` stands for digits, alphanumeric characters, and whitespace, respectively.
- `checkdnsrr()` checks the validity of domain names.



Test Drive

Add email validation to the Risky Jobs Registration script.

Use the code on the facing page to add email validation to the registration script. Then upload the script to your web server, and open it in a web browser. Submitting an invalid email address, and notice how the new regular expression rejects the form submission, and displays an error message to explain what happens.

That's it, I filled my quota of risky jobs. Nothing to do now but add up all the money I just made.

Risky Jobs - Registration

Danger! Your dream job is out there.
Do you have the guts to go find it?

Risky Jobs - Registration

Your email address is invalid.

Register with Risky Jobs, and post your resume.

First Name:	Ricky
Last Name:	Chumley
Email:	rickyc gregs-list.net
Phone:	555-546-8390
Desired Job:	Shark Bait

Paste your resume here:

I really love surfing, been doing it for years and years. And I have to tell you I'm quite good at spotting sharks and luring them to shore. I spent three years as a lifeguard here at Squeaky Beach.

Submit

With the help of validation in Risky Jobs Registration form, no longer a problem getting in with promising job candidates, filling job openings in record time.

CHAPTER 10

php & mysql toolbox



Your PHP & MySQL Toolbox

Looking for patterns in text can be very handy when it comes to validating data entered by the user into web forms. Here are some of the PHP techniques used to validate data with the help of regular expressions:

Regular expression

Rules that are used to match patterns of text in strings. PHP includes functions that allow you to use regular expressions to check a string for a certain pattern, as well as find-and-replace patterns of text within a string.

`preg_match()`

This PHP function checks a string of text to see if it matches a regular expression. The function returns true if there was a match, or false if not.

`preg_replace()`

Use this PHP function to replace a substring within a string based on a regular expression. The function does a find-and-replace using a regular expression for the find, and replacing with a string you provide it.

Character class

A set of rules for a single character within an expression. For example, `[A-Z]` matches the character A or D.

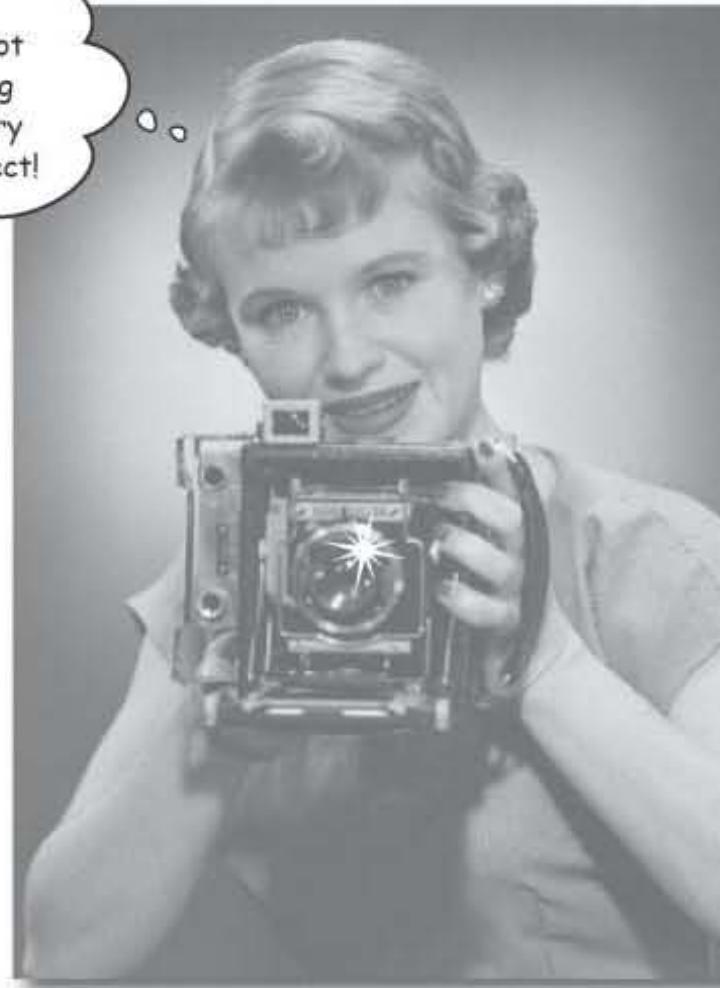
checkd

This PHP name to This is h email ad make su of the

11 Visualizing your data... and more!

Drawing dynamic graphics

Hold still. Wait, stop moving. Now look directly at me and smile. No, not you, your data. OK, let's try crossing your columns and tilting your primary key just a bit to the left. Ah, perfect!



Sure, we all know the power of a good query and a bunch of juicy results. But query results don't always speak for themselves. Sometimes it's helpful to cast data in a different light, a more visual light. PHP makes it possible to provide a graphical representation of database data: *pie charts*, *bar charts*, *Venn diagrams*, *Rorschach art*, you name it. Anything to help users get a grip on the data flowing through your application is game. But not all worthwhile graphics in PHP applications originate in your database. For example, did you know it's possible to thwart form-filling spam bots with dynamically generated images?

attack of the spambots!

Guitar Wars Reloaded: Rise of the Machines

The future is now. Robots have already been let loose in the virtual world and there isn't much to stop them other than some PHP coding vigilance. The robots are **spam bots**, which troll the Web for input forms that allow them to inject advertisements. These robots are chillingly efficient and care nothing about the intended usage of the forms they attack. Their one and only goal is to overrun your content with theirs in a bloodthirsty conquest of ad revenue for their masters. Sadly, the Guitar Wars high score application has fallen prey to the bots.



It's nothing personal; the
bots just want access to your
users' eyeballs for ad revenue.

Spam bots excel at **mindless repetition**, in this case filling out and submitting form after form of Guitar Wars high score data that really contains ads instead of scores.

Guitar Wars - Add Your High Score

Thanks for adding your new high score! It has been added to the high score list as soon as possible.

Name: www.classhates.com
Score: 999999999

CLASSHATES.COM

Reconnect and relive awful experiences with people you hated in high school

Guitar Wars - Add Your High Score

Thanks for adding your new high score! It has been added to the high score list as soon as possible.

Name: www.frowneycentral.com
Score: 999999999

Frowney Central

Every little opinion, guaranteed!

Guitar Wars - Add Your High Score

Thanks for adding your new high score! It has been added to the high score list as soon as possible.

Name: www.headlastlabs.com
Score: 999999999

Head Last Labs

Because sometimes it's just better not to think.

No input form is safe

Fortunately for Guitar Wars, the spam bot attacks remain invisible to the end user thanks to the human moderation feature that was added back in Chapter 6. However, the human moderator is now being completely overwhelmed by the huge volume of spam bot posts, making it tough to sift through and approve legitimate high scores. Human moderation is a great feature, but humans are at a disadvantage when facing an automated adversary that never gets tired.

Name	Date	Score	Action
www.classhates.com	2008-06-23 11:44:56	999999999	Remove / Approve
www.classhates.com	2008-06-23 11:45:15	999999999	Remove / Approve
www.classhates.com	2008-06-23 11:45:29	999999999	Remove / Approve
www.frowneycentral.com	2008-06-23 11:45:53	999999999	Remove / Approve
www.frowneycentral.com	2008-06-23 11:46:06	999999999	Remove / Approve
www.frowneycentral.com	2008-06-23 11:46:19	999999999	Remove / Approve
www.frowneycentral.com	2008-06-23 11:47:26	999999999	Remove / Approve
www.frowneycentral.com	2008-06-23 11:47:42	999999999	Remove / Approve
www.headlastlabs.com	2008-06-23 11:47:55	999999999	Remove / Approve
www.headlastlabs.com	2008-06-23 11:48:12	999999999	Remove / Approve
www.headlastlabs.com	2008-06-23 11:50:24	999999999	Remove / Approve
www.headlastlabs.com	2008-06-23 11:52:20	999999999	Remove / Approve
www.headlastlabs.com	2008-06-23 11:52:32	999999999	Remove / Approve

Human moderation of high score posts clearly isn't enough. We really need a way to prevent robots from being able to submit scores—head them off at the pass, so to speak. But this involves somehow discerning between an automated piece of software and a human with a real brain... a tricky problem, but one that can be solved.

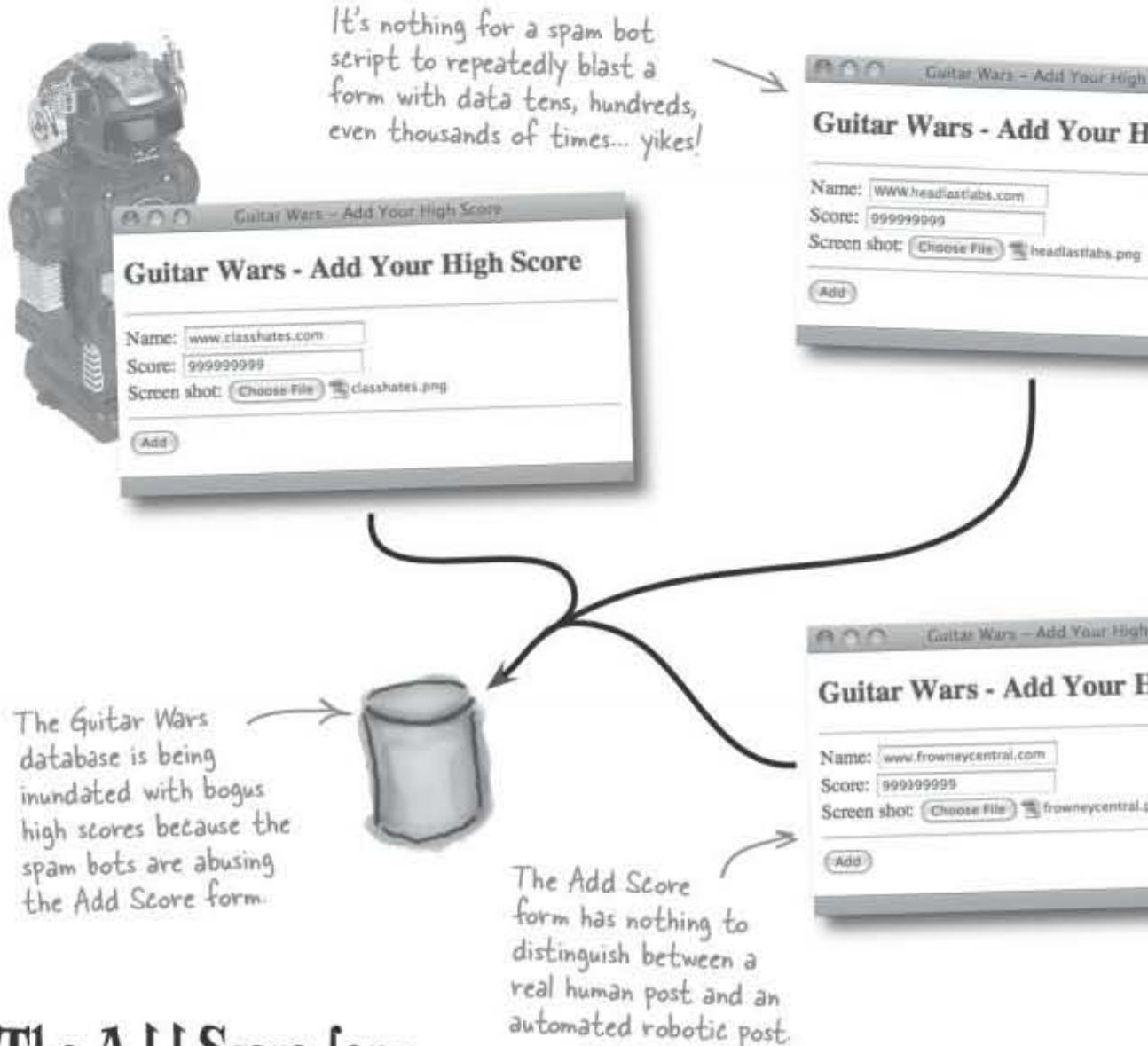
Write down three questions you would ask to distinguish between a real human and the artificial brain of a robot:

.....
.....
.....

guitar wars: for humans only

We need to separate man from machine

In order to figure out how to detect a real human on the other side of the Guitar Wars Add Score page, you have to first assess what exactly a spam bot is doing when filling out the form with bogus data.



The Add Score form needs a new field that requires human verification before allowing a high score to be submitted.

The problem with the Add Score form is that it does not distinguish between **automated** submissions, meaning that any reasonably clever bot can create a bot that repeatedly fills the form with ad data. Sure, it never makes it to the front page of the Guitar Wars site due to the moderation feature, but it in many ways renders moderation useless. A human moderator is left manually removing hundreds of posts.

The form needs a new verification field that must be entered in order for the score to submit. And the specific verification could be something that is easy for a real **human** but difficult for a bot to enter.



Following are some ideas for form fields that could potentially be used to prevent spam from submitting forms. Circle the form fields you think would simply and successfully human form submissions, making sure to annotate why.

Are you a robot? Yes No

What was Elvis' favorite food?

Retinal scan: Look into your web cam and click here

Enter the letters displayed: **kdyqmc**

What is the result of $7 + 5$?

What kind of animal is this?



Enter the letters displayed: **kdyqmc**

Thumbprint scan: Press your thumb down and click here

exercise solution



Exercise Solution

Too easy to guess – even with just a 50% success rate through guessing, you'd end up with a ton of bogus high score posts.

Following are some ideas for form fields that could potentially be used to prevent submitting forms. Circle the form fields you think would simply and successfully prevent human form submissions, making sure to annotate why.

Definitely difficult, potentially difficult as well. Not everyone likes peanut butter and jelly. This would also require lots of trivia questions.

Are you a robot? Yes No

What was Elvis' favorite food?

Not bad, assuming the pass-phrase letters appear as an image and not text, but potentially thwarted by bots smart enough to use optical character recognition (OCR).

Retinal scan: Look into your web cam and click here.

Enter the letters displayed: **kdyqmc**

What is the result of $7 + 5$?

Simple and effective, most bots aren't smart enough to calculate mathematical expressions. Just hope most humans are.

What kind of animal is this?

Remember Fang, the dog that was abducted by aliens earlier in the book?



Deceptively effective – bots struggle at interpreting the content of images. But it does require a database of images and answers.

Enter the letters displayed: **kdyqmc**

Not quite as much trouble as retinal scanning but still requires special hardware and software.

Thumbprint scan: Press your thumb down and click here.

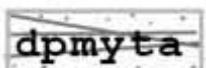
We can defeat automation with automation

A test to verify that the entity on the other side of a form is a real person is known as **CAPTCHA**, which stands for Completely Automated Public Turing Test to Tell Computers and Humans Apart. That's a long-winded way of referring to any "test" on a form that is ideally passable only by a human. Lots of interesting CAPTCHAs have been devised, but one of the most enduring involves generating a random pass-phrase that the user must enter. To help prevent craftier bots with optical character recognition (OCR) from beating the system, the pass-phrase letters are distorted or partially obscured with random lines and dots.

Since the letters in the pass-phrase are randomly generated, the phrase is different every time the form is displayed.

Enter the letters displayed:

A normal text field is used to allow the user to enter the CAPTCHA pass-phrase.



Random lines and dots distort the text just enough for optical character recognition to still allow humans to read it.

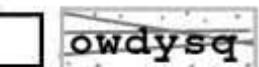
A CAPTCHA form field is just like any other form field except that its whole purpose is to prevent a form from being submitted unless the CAPTCHA test has been successfully completed. So unlike other form fields, which typically pass along data to the server upon submission, a CAPTCHA field is verified and used to control the submission process.

Since the spam bot can't make out the pass-phrase, all it can do is guess.



Fail!

Enter the letters displayed:

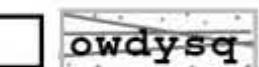


Identifying the pass-phrase successfully is easy for a real human.



Pass!

Enter the letters displayed:



It's very important for the CAPTCHA pass-phrase to be displayed on the form as an image and not just text; otherwise bots would have a much easier time figuring out the text.

A CAPTCHA
a program
protects
from auto
bots b
test of

no dumb questions: captcha edition*there are no*
Dumb Questions

Q: That image CAPTCHA with the dog is really cool. Could I use that instead of a pass-phrase CAPTCHA?

A: Absolutely. Just keep in mind that you'll need to maintain a database of images and descriptions of what they are, because one of the keys to any successful CAPTCHA is variety. A good CAPTCHA should have a deep enough repository of content that a form rarely displays the same test twice. That's the benefit of pass-phrase CAPTCHA: since the pass-phrase is generated from random letters, it's very unlikely the exact same test will appear twice to any given user, even with lots of repeated attempts.

Q: How does CAPTCHA impact the visually impaired? What if they can't pass a visual CAPTCHA test?

A: Visual CAPTCHAs aren't the best answer for users who are visually impaired. An ideal CAPTCHA solution might involve an audio alternative to visual CAPTCHAs. For example, there is an audio CAPTCHA where a series of numbers are read aloud, after which the user must enter them to pass the test. But the same problem exists, where crafty bots use voice recognition to defeat such CAPTCHAs, which is why some of them use highly distorted audio that sounds a little creepy. Audio CAPTCHAs are similar technically to image

CAPTCHAs in that they require a user to identify images and their respective answers. There are services like www.captcha.net. Such services are at the forefront of the latest in CAPTCHA technology, but they offer a service as seamlessly as a custom CAPTCHA would for your web application.

Q: But there are also people who are deaf or have difficulty with hearing impaired. What about them?

A: Ultimately, CAPTCHA is all about security. It's about thwarting spam bots against the risk of viruses and malware, and to viruses and anti-virus software, spam bots are the primary target. CAPTCHAs will likely continue to play a cat and mouse game, being created to defeat a certain CAPTCHA, and then another CAPTCHA, and on and on. Caught in this cycle, users may be left out due to the limited accessibility of CAPTCHAs. It's up to the individual web developer to weigh the attack against the potential loss of users. As a consolation, keep in mind that the majority of spammers aim for large targets with huge ad revenue, so they may not encounter a truly evil bot unless that bot is being a big enough target for high-profile sites.

OK, so a CAPTCHA pass-phrase has to be displayed as an image with random lines and dots. That's fine, but how in the world can that be created with PHP? PHP can only generate HTML code, right?

**PHP has graphics capabilities that can draw images you can then display using**

With the help of a graphics library called GD (Graphics Drawing API), PHP can dynamically generate images in popular formats such as GIF, JPEG, and PNG, and either return them to a web browser for display or save them to a local server. This capability of PHP is extremely important because it allows for the kind of being able to "draw" on a web page purely through PHP. For example, you could "draw" on a portion of a page by performing graphics operations on the image, and then displaying that image on the page using the familiar `imagejpeg()` function.

Generate the CAPTCHA pass-phrase text

Before we can even think about the graphical side of a pass-phrase CAPTCHA, we need to figure out how to generate the random pass-phrase itself, which begins as a sequence of text characters. A pass-phrase can be any number of characters, but somewhere in the range of six to eight characters is usually sufficient. We can use a constant for the pass-phrase length, which allows us to easily change the number of pass-phrase characters later if need be.

```
define('CAPTCHA_NUMCHARS', 6);
```

A CAPTCHA pass-phrase six characters long is probably sufficient to stop bots without annoying humans.

So how exactly do we go about generating a random string of text that is six characters long? This is where two built-in PHP functions enter the story: `rand()` and `chr()`. The `rand()` function returns a random number in the range specified by its two arguments, while `chr()` converts a numeric ASCII character code into an actual character. ASCII (American Standard Code for Information Interchange) is a standard character encoding that represents characters as numbers. We only need ASCII character codes in the range 97-122, which map to the lowercase letters a-z. If we generate a code in this range six times, we'll get a random six-character pass-phrase of lowercase letters.

```
// Generate the random pass-phrase
$pass_phrase = "";
for ($i = 0; $i < CAPTCHA_NUMCHARS; $i++) {
    $pass_phrase .= chr(rand(97, 122));
}
```

This code will eventually go into its own reusable script file, `captcha.php`.

Loop once for every character in the pass-phrase.

The pass-phrase is constructed one random character at a time.

`chr()`

This built-in function converts a number to its character equivalent. As an example, the ASCII code for the lowercase letter 'a' is 97, so calling `chr(97)` returns the single character 'a'.

rand()

This built-in function returns a random integer number, either within a specified range or between 0 and the built-in constant `RAND_MAX` (server dependent). To obtain a random number within a certain range, just pass the lower and upper limits of the range as two arguments to `rand()`.

The `rand()` function returns a random number within a certain range.

drawing a captcha

Visualizing the CAPTCHA image

With the random pass-phrase nailed down, we can move on to generating an image consisting of the pass-phrase text along with random lines and dots to help obscure the text from bots. But where to start? The first thing to do is decide what size the CAPTCHA image should be. Knowing that this image will be displayed on a form next to an input field, it makes sense to keep it fairly small. Let's go with 100×25, and let's put these values in constants so that the image size is set in one place, and therefore easy to change later if necessary.

```
define('CAPTCHA_WIDTH', 100);
define('CAPTCHA_HEIGHT', 25);
```

The size of the CAPTCHA image is stored in constants to make it easier to adjust the size later if desired.

Drawing the CAPTCHA image involves calling several functions in the GD library, all of which operate on an image in memory. In other words, you create an image in memory, then you draw on it, and then when you're all finished, you output it to the browser so that it can be displayed.

```
// Create the image
$img = imagecreatetruecolor(CAPTCHA_WIDTH, CAPTCHA_HEIGHT);

// Set a white background with black text and gray graphics
$bg_color = imagecolorallocate($img, 255, 255, 255); // white
$text_color = imagecolorallocate($img, 0, 0, 0); // black
$graphic_color = imagecolorallocate($img, 64, 64, 64); // dark gray

// Fill the background
imagefilledrectangle($img, 0, 0, CAPTCHA_WIDTH, CAPTCHA_HEIGHT, $bg_color);

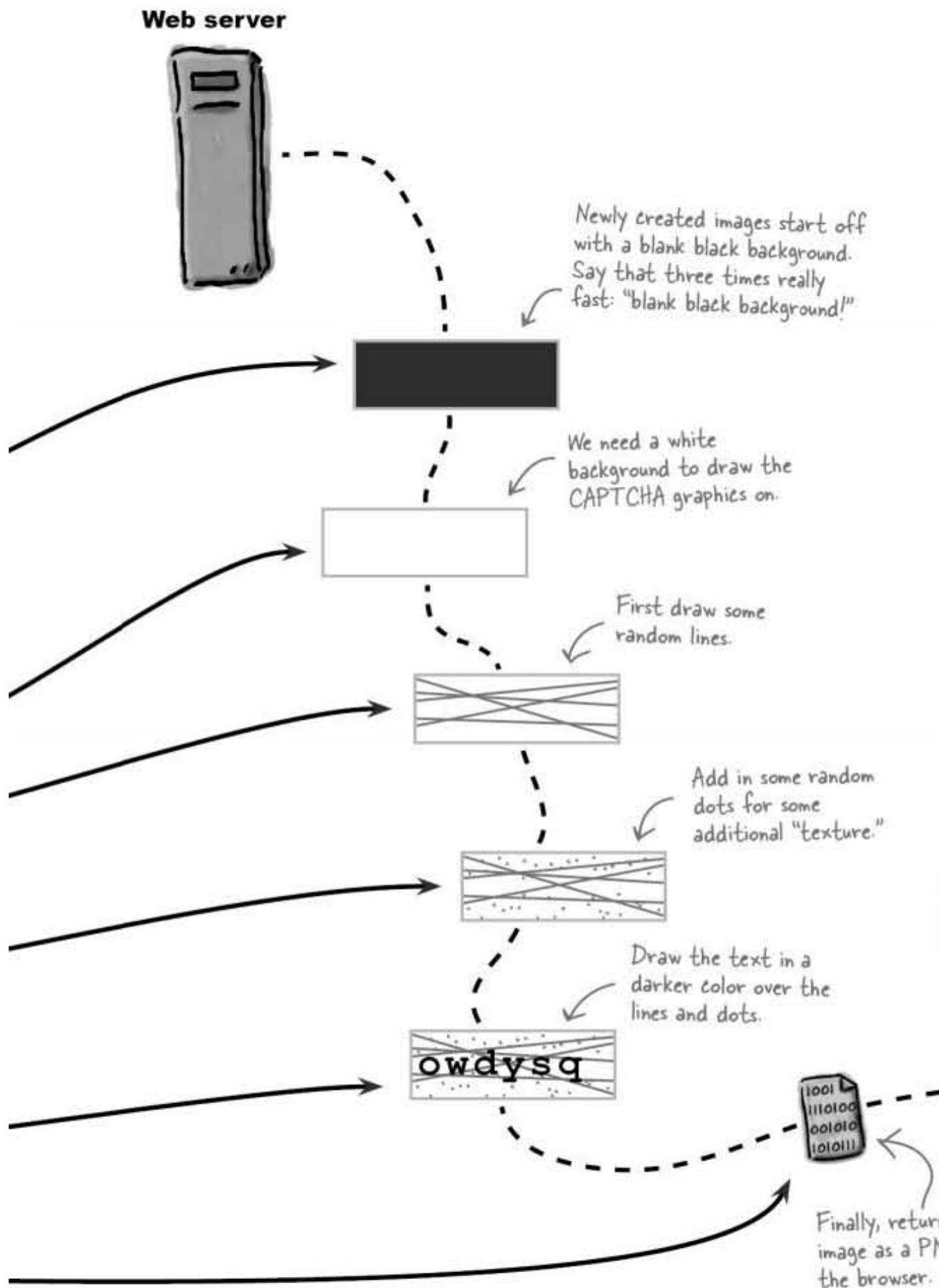
// Draw some random lines
for ($i = 0; $i < 5; $i++) {
    imageline($img, rand() % CAPTCHA_HEIGHT, CAPTCHA_WIDTH,
              rand() % CAPTCHA_HEIGHT, $graphic_color);
}

// Sprinkle in some random dots
for ($i = 0; $i < 50; $i++) {
    imagesetpixel($img, rand() % CAPTCHA_WIDTH,
                 rand() % CAPTCHA_HEIGHT, $graphic_color);
}

// Draw the pass-phrase string
imagettftext($img, 18, 0, 5, CAPTCHA_HEIGHT - 5, $text_color,
             'Courier New Bold.ttf', $pass_phrase);

// Output the image as a PNG using a header
header("Content-type: image/png");
imagepng($img);
```

Drawing
image is
required
library



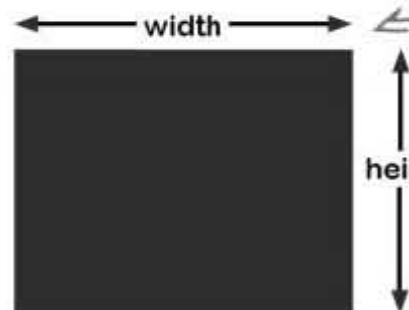
gd graphics functions

Inside the GD graphics functions

The magic behind CAPTCHA image creation is made possible by the GD graphics library, which you've already learned offers functions for dynamically drawing graphics to an image using PHP code. Let's examine some of these functions in more detail as they relate to generating a CAPTCHA image.

`imagecreatetruecolor()`

This function creates a blank image in memory ready to be drawn to with other GD functions. The two arguments to `imagecreatetruecolor()` are the width and height of the image. The image starts out solid black, so you'll typically want to fill it with a background color, such as white, before drawing anything. You can do this by calling the `imagefilledrectangle()` function. The return value of `imagecreatetruecolor()` is an **image identifier**, which is required as the first argument of most GD functions to identify the image being drawn.

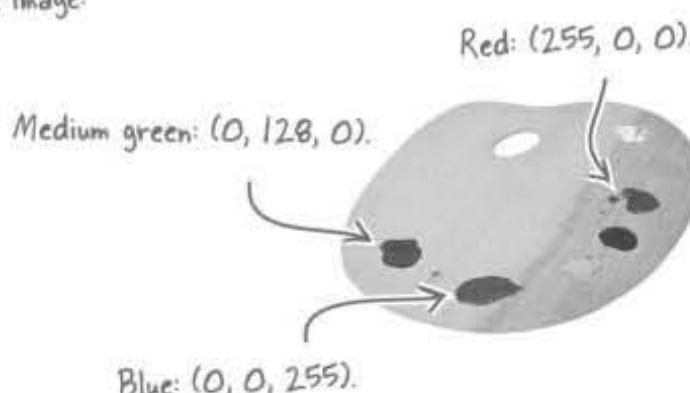


The width of the new image, in pixels.

The function returns an image identifier that is required by other drawing functions to actually draw on the image.

`$img = imagecreatetruecolor(CAPTCHA_WIDTH, CAPTCHA_HEIGHT);`

This code creates an image that is 100x25 in size thanks to our constants.



The return value is a color identifier that you can use in other drawing functions to control the color being used, such as the color of CAPTCHA text.

`$text_color = imagecolorallocate($img, 0, 0, 0);`

The identifier of the image the color will be used with.

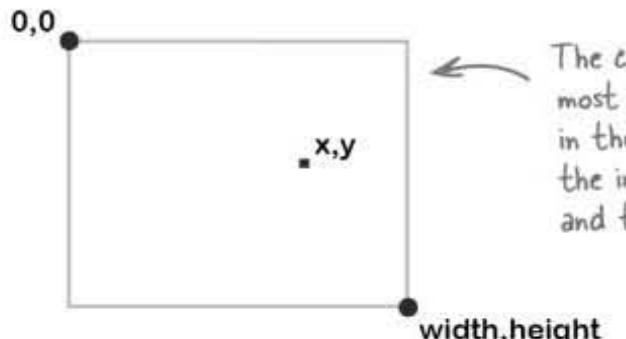
The red component in this

`imagecolorallocate()`

Use this function to allocate a color for use with other drawing functions. The first argument is the image resource identifier, followed by three arguments representing the three components of the RGB (Red, Green, Blue) color value. Each of these values is a value between 0 and 255. The return value is a **color identifier**, which can be used to specify a color in other drawing functions, often as the last argument.

imagesetpixel()

This function draws a single pixel at a specified coordinate within the image. Coordinates start at 0,0 in the upper left corner of the image, and increase to the right and down. Like most GD functions, the pixel is drawn using the color that is passed as the last argument to the function.



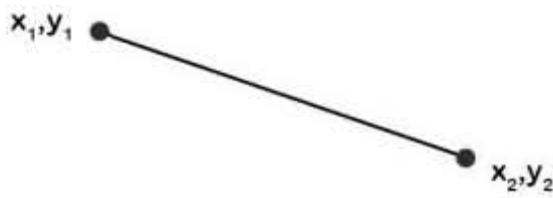
`imagesetpixel($img, rand() % CAPTCHA_WIDTH, rand() % CAPTCHA_HEIGHT)`

The image (identifier) the pixel is being drawn on.

The XY coordinate of the pixel, relative to the upper-left corner of the image, which in this case ends up being a random location within the CAPTCHA image.

imageline()

Call this function to draw a line between two coordinates. The coordinates are specified relative to the upper-left corner of the image, and the line is drawn in the color passed as the last argument.



The XY coordinate of the start of the line, in this case along the left edge of the CAPTCHA image.

`imageline($img, 0, rand() % CAPTCHA_HEIGHT, CAPTCHA_WIDTH, rand() % CAPTCHA_HEIGHT, $gr)`

The XY end point of the line, which here lies on the right edge of the CAPTCHA image.

imagerectangle()

Draw a rectangle starting at one point (x_1, y_1) and ending at another point (x_2, y_2), in a certain specified color. The two points and the color are provided as the second through sixth arguments of the function, following the image identifier argument.



The imagerectangle() function takes the exact same arguments as imagefilledrectangle().

imagefilledrectangle()

Similar to imagerectangle(), this function draws a rectangle whose interior is filled with a specified color.

`imagefilledrectangle($img, 0, 0, CAPTCHA_WIDTH, CAPTCHA_HEIGHT, $gr)`

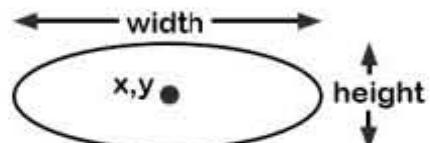
The XY coordinates of the rectangle's corners.

gd graphics functions: part deux

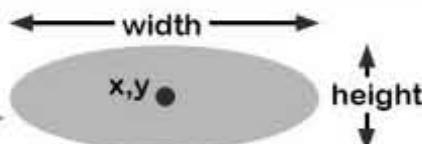
The GD graphics functions continued...

`imageellipse()`

For drawing circles and ellipses, this function accepts a center point and a width and height. A perfect circle is just an ellipse with an equal width and height. The color of the ellipse/circle is passed as the last argument to the function.



Ellipses aren't used in the CAPTCHA image but they're still quite handy!



Both `imageellipse()` and `imagefilledellipse()` take the same arguments.

The width and height of the ellipse – set these the same to draw a perfect circle.

`imagefilled()`

Need a filled ellipse instead? `imagefilledellip()` is the same as `imageelli`, but the specified color is used to fill it.

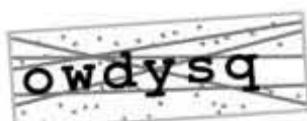
→ `imagefilledellipse($img, 0, 0, 320, 240, $color)`

The XY coords of the center of the ellipsis.

`imagepng()`

When you're all finished drawing to an image, you can output it directly to the client web browser or to a file on the server by calling this function. Either way, the end result is an image that can be used with the HTML `` tag for display on a web page. If you elect to generate a PNG image directly to memory (i.e., no filename), then you must also call the `header()` function to have it delivered to the browser via a header.

An image can be output directly to the browser or to an image file on the server.



The image identifier that you've been using in other draw functions.

→ `imagepng($img);`

The function returns true or false depending on whether the image was successfully created.

You can pass a filename as an optional second argument – without it, the function generates an image in memory that can be passed back to the browser in a header.

Cleaning up after your images is a good idea to keep the server from wasting resources after you're finished with them.



imagedestroy()

It takes system resources to work using the GD library, and this function of cleaning up when you're finished with an image. Just call it after you image with `imagepng()` to clean

Similar to `imagepng()`, this function returns true upon success, or false otherwise.

`imagedestroy($img);`

The i
you w

Always try to pair up a call to this function with each image that you create so that all images are destroyed.

Always free up images in memory with `imagedestroy()` once you've output them.

imagestring()

This function draws a string of text using PHP's built-in font in the color specified. In addition to the image resource identifier, you pass the function the size of the font as a number (1–5), along with the coordinate of the upper left corner of the string, the string itself, and then the color.

The font size of the string, in the range 1 to 5.

`imagestring($img, 3, 75, 75, 'Sample text', $color);`

The XY coordinate of the upper-left corner of the string.

x,y • Sample text

The built-in f
for basic text
limited in ter

This is the string of text to be drawn.

The co
the te

Text drawn with `imagestringup()` is rotated 90 degrees counter-clockwise so that it appears vertically.



imagestringup()

Similar to `imagestring()`, this function draws a string of text using the built-in font, but it draws the text vertically, as if it were rotated 90 degrees counterclockwise. This function is called with the same arguments as `imagestring()`.

the `imagettftext()` function

Drawing text with a font

The `imagestring()` function is easy to use for drawing but it's fairly limited in terms of the control you have over the appearance of the text. To really get a specific look, you need to use a TrueType font of your own. The CAPTCHA pass-phrase image is a good example of this need, since the characters must be drawn fairly large and ideally in a bold font. To get such a custom look, you need the help of one last GD graphics function, which draws text using a TrueType font that you provide on the server.

`imagettftext()`

For drawing truly customized text, place a TrueType font file on your web server and then call this function. Not only do you get to use any font of your choosing, but you also get more flexibility in the size of the font and even the angle at which the text is drawn. Unlike `imagestring()`, the coordinate passed to this function specifies the "basepoint" of the first character in the text, which is roughly the lower-left corner of the first character.

This function does require you to place a TrueType font file on your server, and then specify this file as the last argument. TrueType font files typically have a file extension of `.ttf`.

```
imagettftext($img, 18, 0, 5, CAPTCHA_HEIGHT - 5, $text_color,
    'Courier New Bold.ttf', $pass_phrase);
```

The XY coordinate of the lower left corner of the text

The size of the font, usually specified in "points."

The angle of the font, specified in counter-clockwise degrees (0 is normal text).

The actual text being drawn

You must place the TrueType font file on your web server so that the GD graphics library can find it.



Geek Bits

If you'd like to take a stab at creating your very own TrueType font to further customize your CAPTCHA, check out www.fontstruct.com. It's an online font-building community including a web-based tool for creating custom fonts.

Highly customized text
requires a TrueType font
file on the server.

x,y, Sample text

Unlike `imagestring()`, the coordinate used with `imagettftext()` is the lower-left corner of the text.

The XY coordinate of the lower left corner of the text

You must place the TrueType font file on your web server so that the GD graphics library can find it.

Use the `imagettftext()` function to draw highly customized text with your own TrueType font.

imagecolorallocate

Match each piece of PHP graphics drawing code to the graphical image that it generates. Assume the image (\$img) and colors (\$black_color, \$white_color, and \$gray_color) have already been created.

`imagecolorallocate($img, 0, 0, 0); imagecolorallocate($img, 255, 255, 255);`

`imagefilledrectangle($img, 10, 10, 90, 90, $gray_color);
imagefilledellipse($img, 50, 50, 60, 60, $white_color);
imagefilledrectangle($img, 40, 40, 60, 60, $black_color);`

`imageline($img, 15, 15, 50, 50, $black_color);
imageline($img, 15, 85, 50, 50, $black_color);
imageline($img, 50, 50, 85, 50, $black_color);
imagefilledellipse($img, 15, 15, 20, 20, $gray_color);
imagefilledellipse($img, 15, 85, 20, 20, $gray_color);
imagefilledellipse($img, 50, 50, 20, 20, $gray_color);
imagefilledellipse($img, 85, 50, 20, 20, $gray_color);`

`imagefilledrectangle($img, 10, 10, 90, 60, $gray_color);
imagesetpixel($img, 30, 25, $black_color);
imagesetpixel($img, 70, 25, $black_color);
imageline($img, 35, 45, 65, 45, $black_color);
imagefilledrectangle($img, 45, 50, 55, 90, $gray_color);`

`imageellipse($img, 45, 45, 70, 70, $black_color);
imagefilledellipse($img, 75, 75, 30, 30, $gray_color);
imagesetpixel($img, 10, 10, $black_color);
imagesetpixel($img, 80, 15, $black_color);
imagesetpixel($img, 20, 15, $black_color);
imagesetpixel($img, 90, 60, $black_color);
imagesetpixel($img, 20, 80, $black_color);
imagesetpixel($img, 45, 90, $black_color);`

`imagefilledrectangle($img, 25, 35, 75, 90, $black_color);
imageline($img, 10, 50, 50, 10, $black_color);
imageline($img, 50, 10, 90, 50, $black_color);
imagefilledrectangle($img, 45, 65, 55, 90, $white_color);
imageline($img, 0, 90, 100, 90, $black_color);`

who draws what solution

Match each piece of PHP graphics drawing code to the graphical image that it generates. Assume the image (\$img) and colors (\$black_color, \$white_color, and \$gray_color) have already been created.

```
imagefilledrectangle($img, 10, 10, 90, 90, $gray_color);
imagefilledellipse($img, 50, 50, 60, 60, $white_color);
imagefilledrectangle($img, 40, 40, 60, 60, $black_color);
```

```
imageline($img, 15, 15, 50, 50, $black_color);
imageline($img, 15, 85, 50, 50, $black_color);
imageline($img, 50, 50, 85, 50, $black_color);
imagefilledellipse($img, 15, 15, 20, 20, $gray_color);
imagefilledellipse($img, 15, 85, 20, 20, $gray_color);
imagefilledellipse($img, 50, 50, 20, 20, $gray_color);
imagefilledellipse($img, 85, 50, 20, 20, $gray_color);
```

```
imagefilledrectangle($img, 10, 10, 90, 60, $gray_color);
imagesetpixel($img, 30, 25, $black_color);
imagesetpixel($img, 70, 25, $black_color);
imageline($img, 35, 45, 65, 45, $black_color);
imagefilledrectangle($img, 45, 50, 55, 90, $gray_color);
```

```
imageellipse($img, 45, 45, 70, 70, $black_color);
imagefilledellipse($img, 75, 75, 30, 30, $gray_color);
imagesetpixel($img, 10, 10, $black_color);
imagesetpixel($img, 80, 15, $black_color);
imagesetpixel($img, 20, 15, $black_color);
imagesetpixel($img, 90, 60, $black_color);
imagesetpixel($img, 20, 80, $black_color);
imagesetpixel($img, 45, 90, $black_color);
```

```
imagefilledrectangle($img, 25, 35, 75, 90, $black_color);
imageline($img, 10, 50, 50, 10, $black_color);
imageline($img, 50, 10, 90, 50, $black_color);
imagefilledrectangle($img, 45, 65, 55, 90, $white_color);
imageline($img, 0, 90, 100, 90, $black_color);
```

Generate a random CAPTCHA image

Putting all the CAPTCHA code together results in the brand-new captcha.php script, which takes care of generating a random pass-phrase and then returning a PNG image to the browser.

```
<?php
session_start();

// Set some important CAPTCHA constants
define('CAPTCHA_NUMCHARS', 6); // number of characters in pass-phrase
define('CAPTCHA_WIDTH', 100); // width of image
define('CAPTCHA_HEIGHT', 25); // height of image

// Generate the random pass-phrase
$pass_phrase = "";
for ($i = 0; $i < CAPTCHA_NUMCHARS; $i++) {
    $pass_phrase .= chr(rand(97, 122));
}

// Store the encrypted pass-phrase in a session variable
$_SESSION['pass_phrase'] = sha1($pass_phrase);

// Create the image
$img = imagecreatetruecolor(CAPTCHA_WIDTH, CAPTCHA_HEIGHT);

// Set a white background with black text and gray graphics
$bg_color = imagecolorallocate($img, 255, 255, 255); // white
$text_color = imagecolorallocate($img, 0, 0, 0); // black
$graphic_color = imagecolorallocate($img, 64, 64, 64); // dark gray

// Fill the background
imagefilledrectangle($img, 0, 0, CAPTCHA_WIDTH, CAPTCHA_HEIGHT, $bg_color);

// Draw some random lines
for ($i = 0; $i < 5; $i++) {
    imageline($img, 0, rand() % CAPTCHA_HEIGHT, CAPTCHA_WIDTH, rand() % CAPTCHA_HEIGHT);
}

// Sprinkle in some random dots
for ($i = 0; $i < 50; $i++) {
    imagesetpixel($img, rand() % CAPTCHA_WIDTH, rand() % CAPTCHA_HEIGHT, $graphic_color);
}

// Draw the pass-phrase string
imagefttext($img, 18, 0, 5, CAPTCHA_HEIGHT - 5, $text_color, "Courier New Bold");

// Output the image as a PNG using a header
header("Content-type: image/png");
imagepng($img);

// Clean up
imagedestroy($img);
?>
```

The code starts by defining constants for the number of characters in the pass-phrase, the width and height of the image. It then generates a random pass-phrase and stores its SHA-1 hash in a session variable. The image is created using the `imagecreatetruecolor` function. The background is set to white. The pass-phrase is drawn in black font. Some random lines and dots are drawn in dark gray. Finally, the image is output as a PNG file using the `header` and `imagepng` functions. The image is destroyed after it is output.

Finish up by destroying the image

Some version of graphics library path to the image is through a header.

Create a random number of characters in the CAPTCHA image height of the image

Although you could store the pass-phrase in the database to just stick it in a session variable, we have to store it so the application can access it.

The PNG image is actually delivered to the browser through a header.

try out `captcha.php`

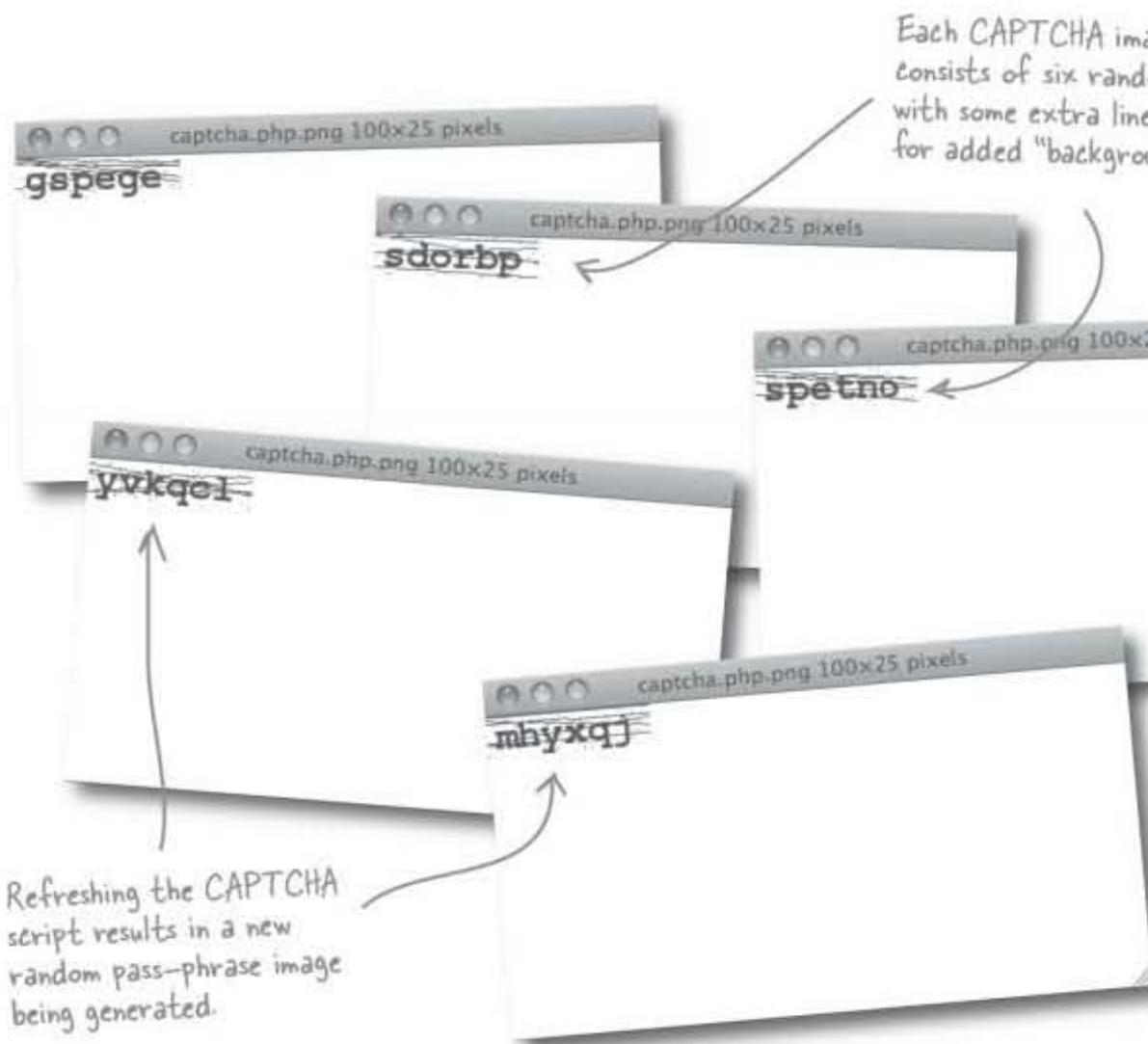


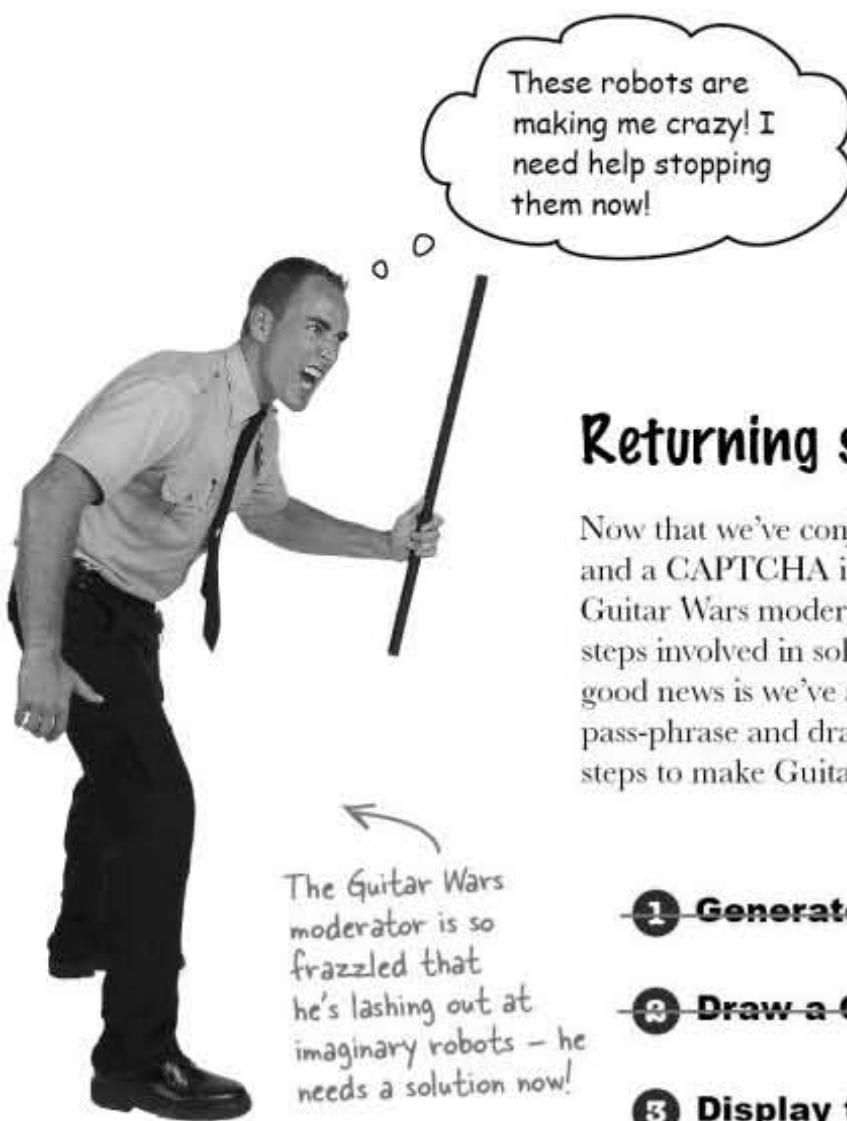
Test Drive

Create the CAPTCHA script and try it out.

Create a new text file named `captcha.php`, and enter the code for the CAPTCHA script from the previous page (or download the script from the Head First Labs website at www.headfirstlabs.com/books/hfphp).

Upload the script to your web server, and then open it in a web browser. You'll immediately see the CAPTCHA image with the random pass-phrase in the browser window. To generate a new random pass-phrase, refresh the browser.





Returning sanity to Guitar Wars

Now that we've conjured up your inner PHP artist with a pass-phrase and a CAPTCHA image, it's time to use the CAPTCHA to stop the Guitar Wars moderator from the spam bot assault. The four steps involved in solving the bot problem with a pass-phrase and CAPTCHA are good news: we've already knocked out two of them: generating the pass-phrase and drawing the CAPTCHA image. Let's finish the last two steps to make Guitar Wars officially bot-free!

- 1 Already done!** **Generate a random pass-phrase.**
- 2 Draw a CAPTCHA image using the GD library.**
- 3 Display the CAPTCHA image on the Add Score form and prompt the user to enter the pass-phrase.**
- 4 Verify the pass-phrase against the CAPTCHA image.**



Complete Step 3 of the Guitar Wars Add Score CAPTCHA by writing the verification text input form field that prompts the user to enter the CAPTCHA pass-phrase. Make sure to give it a label, and follow it with an `` tag that displays the CAPTCHA image generated by the `captcha.php` script.

.....
.....
.....

exercise solution

Complete Step 3 of the Guitar Wars Add Score CAPTCHA by writing the HTML verification text input form field that prompts the user to enter the CAPTCHA. Make sure to give it a label, and follow it with an `` tag that displays the image generated by the `captcha.php` script.

A `<label>` tag is used to label the new verification text field.

`<label for="verify">Verification: </label>`

`<input type="text" id="verify" name="verify" value="Enter the pass-phrase." />`

The "source" of the image is the name of the PHP script that dynamically generates the CAPTCHA image. This works because the `captcha.php` script returns an image directly to the browser via `imagepng()` and a header.

Guitar Wars - Add Your High Score

Name:

Score:

Screen shot: no file selected

Verification: Enter the pass-phrase.

Add

Finished! One more step to go.

- 5 Display the CAPTCHA image on the Guitar Wars Add Score form and prompt the user to enter the pass-phrase.

Add CAPTCHA to the Add Score script

On the client side of the equation, the addscore.php script contains the new Verification text field with the CAPTCHA image beside it. The most important change, however, is the new if statement in the Add Score script (Step 4) that checks to make sure the user-entered pass-phrase matches the CAPTCHA pass-phrase.

```

<?php
    session_start();
?>

<html>
<head>
    <title>Guitar Wars - Add Your High Score</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
    <h2>Guitar Wars - Add Your High Score</h2>

    <?php
        require_once('appvars.php');
        require_once('connectvars.php');

        if (isset($_POST['submit'])) {
            // Connect to the database
            $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

            // Grab the score data from the POST
            $name = mysqli_real_escape_string($dbc, trim($_POST['name']));
            $score = mysqli_real_escape_string($dbc, trim($_POST['score']));
            $Screenshot = mysqli_real_escape_string($dbc, trim($_FILES['Screenshot']['name']));
            $Screenshot_type = $_FILES['Screenshot']['type'];
            $Screenshot_size = $_FILES['Screenshot']['size'];

            // Check the CAPTCHA pass-phrase for verification
            $user_pass_phrase = sha1($_POST['verify']);
            if ($SESSION['pass_phrase'] == $user_pass_phrase) {
                ...
            } else {
                echo '<p class="error">Please enter the verification pass-phrase exactly as shown above!</p>';
            }
        }
    ?>

    <hr />
    <form enctype="multipart/form-data" method="post" action="<?php echo $SERVER[<br/>
        <input type="hidden" name="MAX_FILE_SIZE" value="<?php echo GW_MAXFILESIZE; ?><br/>
        <label for="name">Name: </label>
        <input type="text" id="name" name="name" value="<?php if (!empty($name)) echo $name; ?>"><br/>
        <label for="score">Score: </label>
        <input type="text" id="score" name="score" value="<?php if (!empty($score)) echo $score; ?>"><br/>
        <label for="Screenshot">Screenshot: </label>
        <input type="file" id="Screenshot" name="Screenshot" /><br/>
        <label for="verify">Verification: </label>
        <input type="text" id="verify" name="verify" value="Enter the pass-phrase." />
        <br/>
        <hr />
        <input type="submit" value="Add" name="submit" />
    </form>
</body>
</html>

```

This is where the CAPTCHA was "wired" to the Add Score script in Step 3, resulting in the image being displayed on the page.

test drive addscore.php with captcha functionality



Test Drive

Modify the Add Score script to support CAPTCHA.

Change the `addscore.php` script so that it has a new Verification form field, as using the `captcha.php` script to display a CAPTCHA image. Also add the code to check and make sure the user entered the correct pass-phrase before adding a score.

Upload both scripts to your web server, and then open `addscore.php` in a web browser. Try to add a new score without entering a CAPTCHA pass-phrase. Now again after entering the pass-phrase displayed in the CAPTCHA image.

Guitar Wars - Add Your High Score

Please enter the verification pass-phrase exactly as shown.

Name:	<input type="text" value="www.headlastiabs.com"/>
Score:	<input type="text" value="99999999"/>
Screen shot:	<input type="button" value="Choose File"/> no file selected
Verification:	<input type="text" value="azjyam"/>

A constantly changing CAPTCHA pass-phrase makes it tough for automated bots to spam the Guitar Wars form.

Our human moderator finally gets some peace thanks to a little automation of our own!

Guitar Wars - Add Your High Score

Please enter the verification pass-phrase exactly as shown.

Name:	<input type="text" value="www.frowneycentral.com"/>
Score:	<input type="text" value="99999999"/>
Screen shot:	<input type="button" value="Choose File"/> no file selected
Verification:	<input type="text" value="feebdy"/>

Guitar Wars - Add Your High Score

Please enter the verification pass-phrase exactly as shown.

Name:	<input type="text" value="www.classhates.com"/>
Score:	<input type="text" value="99999999"/>
Screen shot:	<input type="button" value="Choose File"/> no file selected
Verification:	<input type="text" value=""/>

there are no Dumb Questions

Q: Can I use the GD functions to create images in formats other than PNG?

A: Yes. The `imagegif()` and `imagejpeg()` functions work very similarly to `imagepng()` but create GIF and JPEG images, respectively.

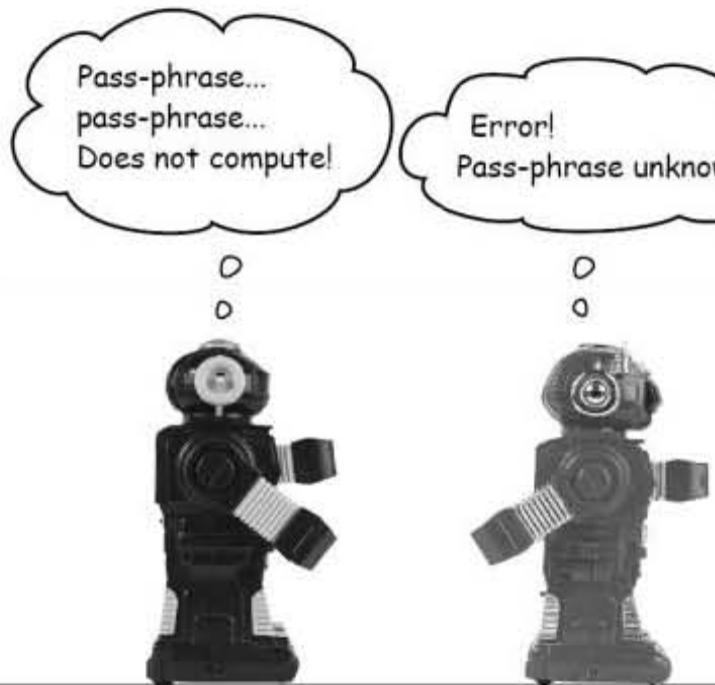
Q: Can the image creation functions create images with transparency?

A: Yes! There is a function called `imagecolortransparent()` that sets a color as a transparent color within an image. This must be a color that you've already created using the `imagecolorallocate()` function. After setting the color as transparent, anything drawn with that color in the image will be considered transparent. To generate the image with transparency, just call `imagegif()` or `imagepng()`; you can't use `imagejpeg()` because JPEG images don't support transparency.

Q: When using `imagepng()` to output a PNG image directly to the client browser, where is the .png file for the image stored, and what is its name?

A: There is no .png file for the image, and the reason is because the image isn't stored in a file. Instead, the `imagepng()` function generates a binary PNG image in memory on the server and then delivers it directly to the browser via a header. Since the image data is created and sent directly to the browser, there's no need to store it in an image file.

Q: Is that why I'm able to put the name of the CAPTCHA script directly in the `src` attribute of an `` tag?



BULLET POINTS

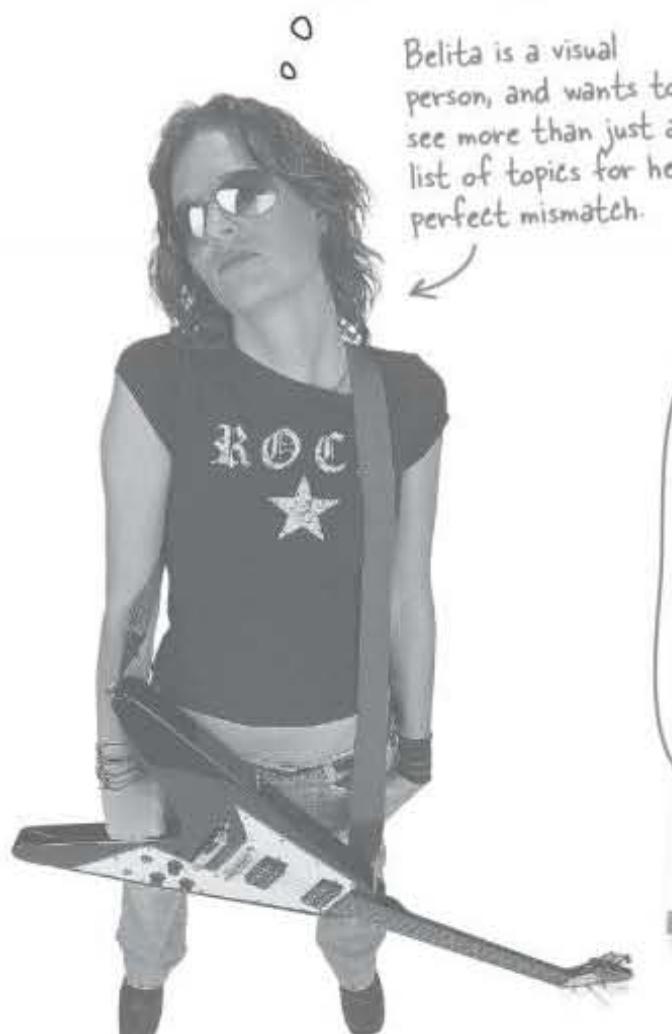
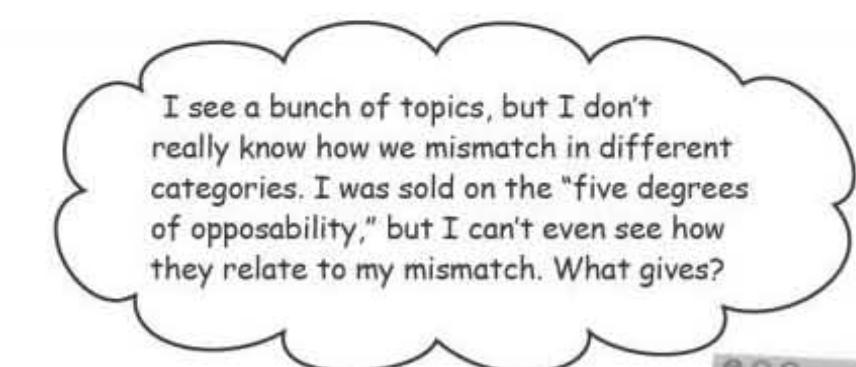
- All web forms are at risk of attack from spam bots, but all spam bots are at risk from clever PHP programmers who use techniques such as CAPTCHA to thwart them.
- GD is a standard PHP graphics library that allows you to dynamically create images and then draw all kinds of different graphics and text on them.

- The `createtrcolor()` function is used to create a blank image.
- To output a PNG image to the browser, call the `imagepng()` function.
- When you're finished working with an image, call the `imagedestroy()` to clear it from memory.

visualizing mismatch's data

Five degrees of opposability

Since Mismatch is a community of registered (human!) users, spam bots haven't been a problem. However, users want a little more out of the mismatch feature of the site, primarily the "five degrees of opposability" that they've been hearing about. Mismatch users want more than just a list of topics for their ideal mismatch—they want some visual context of how those topics break down for each major category of "mismatchiness."



Mismatch - My Mismatch

Home ♥ View Profile ♥ Edit Profile ♥ Questionnaire ♥ My Mismatch

Jason Filmington
Hollywood, CA

You are mismatched on the following 18 topics:

- Tattoos
- Cowboy boots
- Long hair
- Reality TV
- Horror movies
- Easy listening music
- The opera
- Sushi
- Spicy food
- Peanut butter & banana sandwiches
- Martini
- Bill Gates
- Hugh Hefner
- Yoga
- Weightlifting
- Cube puzzles
- Karaoke
- Hiking

[View Jason's profile.](#)

The list of mismatch topics is interesting and users really want visual perspective on their mismatch with each other.

Copyright ©2008 Mismatch Enterprises, Inc.

Charting mismatchiness

If you recall, Mismatch includes a categorized questionnaire where users select Love or Hate for a variety of topics. These responses are what determine the topics for an ideal mismatch. When presenting a user's ideal mismatch, the My Mismatch script displays a list of mismatched topics that it builds as an array from the Mismatch database. But users now want more than a list of topics... they want a visual categorized breakdown of their "mismatchiness," perhaps in the form of a bar graph?

Topic	
Space	Love (1)Hate
Spicy food	Hate (1)Love
Peanut butter & banana sandwiches	Love (1)Hate
Martinis	(1)Love (1)Hate
People	
Street urchin	(1)Love (1)Hate
Bill Gates	(1)Love (1)Hate
Hugh Hefner	(1)Love (1)Hate
Hugh Hefner	(1)Love (1)Hate
Martinis	(1)Love (1)Hate
Activities	
Yoga	(1)Love (1)Hate
Weightlifting	(1)Love (1)Hate
Cube puzzles	(1)Love (1)Hate
Karaoke	(1)Love (1)Hate
Hiking	(1)Love (1)Hate



- Tattoos
- Cowboy boots
- Long hair
- Reality TV
- Horror movies
- Easy listening music
- The opera
- Sushi
- Spicy food
- Peanut butter & banana sandwiches
- Martinis
- Bill Gates
- Hugh Hefner
- Yoga
- Weightlifting
- Cube puzzles
- Karaoke
- Hiking

We somehow need to turn this list of topics into a bar graph of categories.



Exercise

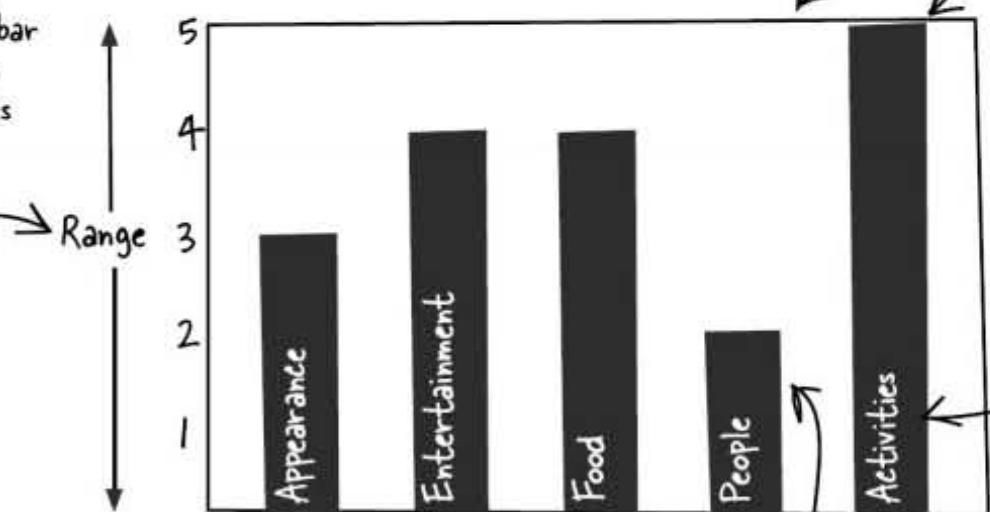
Draw a bar graph for visually shows the topics from Belita and Jason. A bar graph is a chart that shows the bar graph mean

storing graph data in arrays
**Exercise
SOLUTION**

Draw a bar graph for the Mismatch data that visually shows the “five degrees of mismatch” for Belita and Jason. Annotate what the information in the bar graph means.

Although there are lots of different ways you could visualize the mismatch data, a bar graph isn’t a bad option since the categories have an equal number of topics each.

The range of a bar graph establishes the possible values for each bar.



Each bar represents the number of mismatches in a given category of topics.

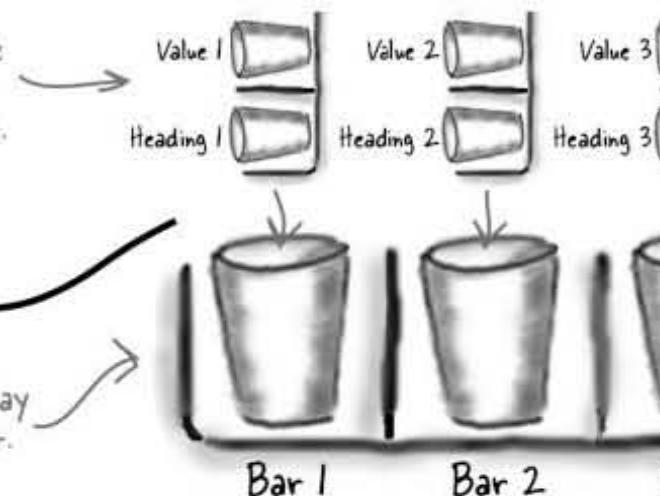
The height of each bar reflects the value for that category.

Storing bar graph data

When it comes down to it, the data behind a bar graph is perhaps even more important than the graphics. Knowing that a bar graph is really just a collection of headings and values, we can view the data for a bar graph as a two-dimensional array where the main array stores the bars, while each sub-array stores the heading/value pair for each bar.

```
$graph_data = array(
    array("Heading 1", $value1),
    array("Heading 2", $value2),
    array("Heading 3", $value3),
    ...
);
```

Each item in the main array corresponds to a single bar.



there are no
Dumb Questions

Q: Does a bar graph have to be fed with a two-dimensional array of data?

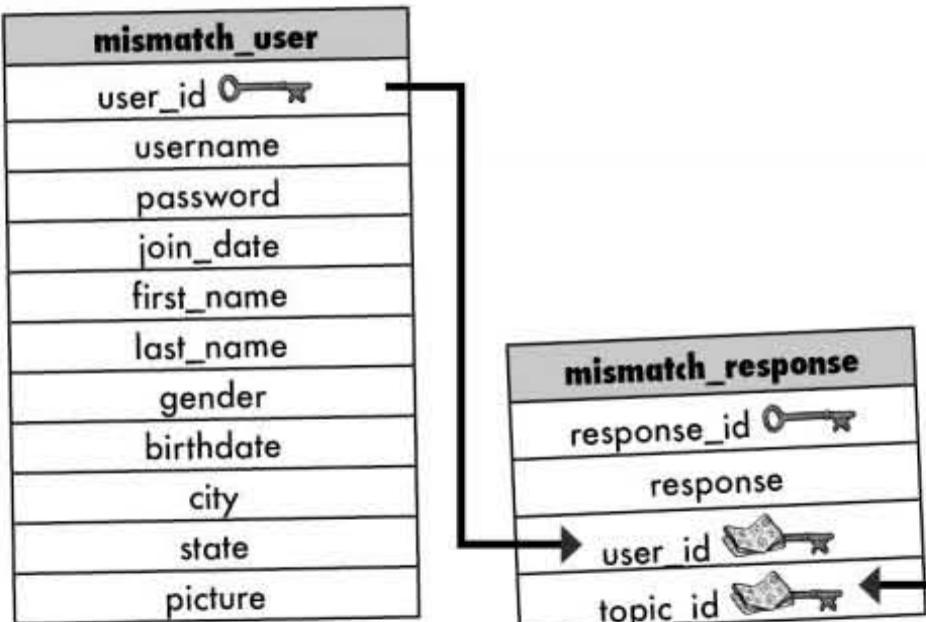
A: No, not at all. But keep in mind that each bar in a bar graph typically involves two pieces of information: a heading and a value. And each bar graph consists of multiple bars, so a two-dimensional

array is a logical and efficient way to store a bar graph. As the old saying goes, "If you really don't understand the problem, it is how to best store the data that is injected." One particular solution that works out perfectly is to use a two-dimensional array. Of course, the challenge still remains: this two-dimensional array of Mismatch objects is to isolate what data in the database fac-



The database schema for the Mismatch application is shown below. Circle all of the data that factor into the dynamic generation of the "five degrees of opposition".

making sure to annotate how they are used to create the graph.



mismatch	
category	
name	

mismatch	
topic_id	PK
name	

exercise solution

The database schema for the Mismatch application is shown below. Circle all of the data that factor into the dynamic generation of the "five degrees of opposition". Making sure to annotate how they are used to create the graph.

The `user_id` column is used to query for questionnaire responses so that the best mismatch for a user can be determined.

The `category_id` column is used to associate a category with a response. This leads to a mismatched response. The name of the category is used to provide headings for the responses.

mismatch user	
<code>user_id</code>	key
username	
password	
join_date	
first_name	
last_name	
gender	
birthdate	
city	
state	
picture	

The `response_id` column is used to match up responses for two users to determine if they are set to opposite values – a mismatch!

mismatch response	
<code>response_id</code>	key
response	
<code>user_id</code>	key
<code>topic_id</code>	key

mismatch	
<code>category_id</code>	key

mismatch	
<code>topic_id</code>	key

The `topic_id` column serves as the go-between for categories and responses, which is what allows you to figure out the category of each mismatched response.

The bar chart requires a mismatched category value for each category.



Bar Graph Exposed

This week's interview:
Reading between the lines with the
master of charts

Head First: So you're the guy people call when they need a visual representation of some data. Is that right?

Bar Graph: Oh yeah. I'm adept at all facets of data visualization, especially the rectangular variety.

Head First: So your drawing capabilities are limited mostly to rectangles?

Bar Graph: I'd say "limited" is a strong word in this case. It's one of those deals where simpler is better—people just seem to relate to the bars, maybe because they're used to seeing things measured that way. You know, like the little meter on mobile phones that tells you how good your signal is. "Can you hear me now?" I love that.

Head First: Right. But I've also seen some pretty effective graphs that are round. Makes me think comforting thoughts... like apple pie, know what I mean?

Bar Graph: I know where you're headed, and I'm fully aware of Pie Chart. Look, it's two different ways of thinking about the same thing. Pie Chart sees the world in curves; I see it a bit straighter, that's all.

Head First: But don't people inherently relate better to pie than a bunch of bars?

Bar Graph: No, they don't. At least people who aren't hungry don't. You see, Pie Chart is really good at revealing parts of a whole, where the data being represented adds up to something that matters, like 100%, 32 teams, or 50 states. There are 50 states, right?

Head First: Yes. Well, assuming you count Washington, D.C. as a "capital district" and places like Puerto Rico and Guam as "territories." But anyway, I see what you're saying about Pie Chart being more about revealing parts of a whole, but don't you do the same thing?

Bar Graph: Yes, but keep in mind that I'm much more flexible than Pie Chart. You can add as many bars to me as you want and I have no problem at all showing them.

The more you add to Pie Chart, the more you have to get. At some point the parts start to lose their meaning because of the whole. All that matter is that the bars all have values that can show the whole picture.

Head First: What does that mean?

Bar Graph: Well, it's difficult for a chart to show wildly different values, unless you consider the context. For example, just think about the difference between the height of a person and the height of a building. Both are within the same range. For example, I could use me to show the price of gasoline over a period, in which case all the values would be within a reasonably constrained range, like the price of gas.

Head First: You sure about that?

Bar Graph: I know, the price of gasoline has a wildly varying value, but not really what I deal with.

Head First: So you've seen some crazy things?

Bar Graph: You wouldn't believe it. I once had a guy who built a web application that tracked how many miles he dragged his motorcycle. He blogged about it constantly, and called it "travels." Pretty crazy but people loved it.

Head First: So is that where you get your ideas for your charts?

Bar Graph: Yeah, I guess so. Anyways, I can take a page and provide some eye appeal that would otherwise be a little dull and hard to look at. It's a good day.

Head First: Glad to hear it. Hey, I'm glad you're here, sharing your thoughts, and I hope you'll come back again.

Bar Graph: It was my pleasure. And I hope you'll be seeing me around.

building an array for categories

From one array to another

When we last left off at Mismatch, we had a list of topics that corresponded to mismatches between two users. More specifically, we really have an **array** of topics. Problem is, the bar graph we're trying to draw isn't about topics per se—it's about the **categories** that are associated with the topics. So we're one level removed from the data we actually need. It appears that some additional SQL querying is in order. Not only do we need the array of mismatched topics, but we also need a similar, parallel array of mismatched categories.

A multiple join connects the category table to the response table so that the category name can be extracted.

```
$query = "SELECT mr.response_id, mr.topic_id, mr.
          mt.name AS topic_name, mc.name AS category_name
      "FROM mismatch_response AS mr "
      "INNER JOIN mismatch_topic AS mt USING (topic_id)
      "INNER JOIN mismatch_category AS mc USING (category_id)
      "WHERE mr.user_id = '" . $_SESSION['user_id'] . "'
```

The additional join in this query causes the category name corresponding to each response topic to be tacked on to the result data, ultimately making its way into the `$user_responses` array. But remember, we need only the **mismatched** categories, not all of the categories. We need to build another array containing just the mismatched categories for the responses.



topic_name	category_name
Tattoos	Appearance
Gold chains	Appearance
Body piercings	Appearance
Cowboy boots	Appearance
Long hair	Appearance
Reality TV	Entertainment
Professional wrestling	Entertainment
Horror movies	Entertainment
Easy listening music	Entertainment
The opera	Entertainment
Sushi	Food
Spam	Food
Spicy food	Food
Peanut butter & banana sandwiches	Food
Martinis	Food
Howard Stern	People
Bill Gates	People
Barbara Streisand	People
Hugh Hefner	People

`$user_responses`

Back in Chapter 8, the `$user_responses` two-dimensional array was created and filled with result data corresponding to the current user's responses.

The new "column" of mismatch result data holds the category name of every response.

We need to extract only the category names of mismatched responses into an array.

But we're still falling short of an array of mismatched categories. We need to revisit the code that handles mismatched topics...

An array
of categories
to the
mismatches



Test Drive

Try out the new query to grab mismatched topics and categories.

Using a MySQL tool, issue the following query to SELECT mismatched topics and categories for a specific user. Make sure to specify a user ID for a user who not only exists in the system but who also has filled out the Mismatch questionnaire form:

```
SELECT mr.response_id, mr.topic_id, mr.response,
       mt.name AS topic_name, mc.name AS category_name
  FROM mismatch_response AS mr
 INNER JOIN mismatch_topic AS mt USING (topic_id)
 INNER JOIN mismatch_category AS mc USING (category_id)
 WHERE mr.user_id = 3;
```

The user ID must be for a valid user who has answered the Mismatch questionnaire.

File Edit Window Help Oppose

```
mysql> SELECT mr.response_id, mr.topic_id, mr.response,
       mt.name AS topic_name, mc.name AS category_name
  FROM mismatch_response AS mr
 INNER JOIN mismatch_topic AS mt USING (topic_id)
 INNER JOIN mismatch_category AS mc USING (category_id)
 WHERE mr.user_id = 3;
```

response_id	topic_id	response	topic_name
26	1	1	Tattoos
27	2	2	Gold chains
28	3	1	Body piercings
29	4	2	Cowboy boots
30	5	1	Long hair
31	6	2	Reality TV
32	7	1	Professional wrestling
33	8	1	Horror movies
34	9	2	Easy listening music

Notice that the results of this query match up with the `fuser_responses` array on the facing page, which is what we want.

It's out before

building an array for topics

Build an array of mismatched topics

We now have a query that performs a multiple join to grab the **category** of each response in addition to the topic, which is then extracted into the `$user_responses` array. Remember that another similar query also grabs data for each other user in the database so that mismatch comparisons can be made. So `$user_responses` holds the response data for the user logged in to Mismatch, while `$mismatch_responses` holds one of the other users in the system. This allows us to loop through all of the users and update `$mismatch_responses` for each mismatch user comparison.

We're already using these two arrays to score mismatches and build an array of mismatched topics. We can now add a new line of code to also construct an array of mismatched categories—**this array contains the category of each mismatched topic between two users**.

This is the same code from the earlier version of Mismatch, except now it builds an array of mismatched categories in addition to the array of topics.

```
$categories = array();
for ($i = 0; $i < count($user_responses); $i++) {
    if ($user_responses[$i]['response'] + $mismatch_responses[$i]['response'] > 0) {
        $score += 1;
        array_push($topics, $user_responses[$i]['topic_name']);
        array_push($categories, $user_responses[$i]['category_name']);
    }
}
```

This code results in an array containing only the mismatched categories.

When all is said and done, the `$categories` array contains one category for each and every mismatch.

An array away with e

there are no
Dumb Questions

Q: I'm a little confused. What's the difference between a MySQL result set and a PHP array?

A: One big difference is access. A result set is only made available a row at a time, while an array can hold multiple "rows" of data thanks to multiple dimensions. Extracting a result set into a two-dimensional array lets us move through rows of data efficiently without having to constantly go back to the database server to fetch and re-fetch rows. This doesn't work with huge sets of data, as you'd be creating huge arrays, but in the case of Mismatch responses, the arrays are never larger than the total number of topics in the system.

Q: Don't we still need to count how many times each category was mismatched in order to generate this array?

A: Yes. An array of mismatched categories is not enough. Remember that the idea behind the Mismatch API is that each category in the array represents a mismatched category, and the value represents how many times the category was mismatched. This means that we need to come up with a count of how many times each category was mismatched. But it's probably worth taking a general plan of attack...

Formulating a bar graphing plan

With an array of mismatched categories and a bunch of big ideas about how to use it to generate a bar graph image for the My Mismatch page, we're still missing a plan. As it turns out, there are only three steps required to dynamically generate the bar graph, and we've already knocked out one of them.

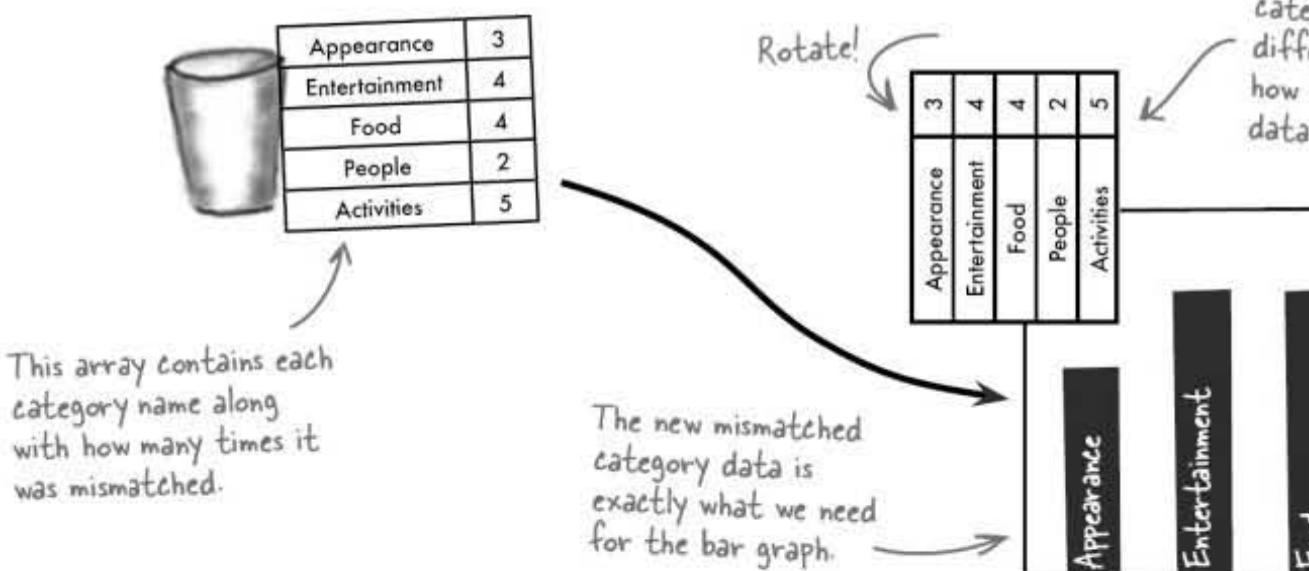
- 1 **Query the Mismatch database for mismatched category names.**
- 2 **Calculate the mismatch totals for each category.**
- 3 **Draw the bar graph using the categorized mismatch totals.**

This step provides us with the list of mismatched categories.

The next step needs a list.

With the categories in hand, we can get to the fun part of drawing the bar graph with PHP.

To complete Step 2 of our plan, we somehow need to take the array of mismatched categories and turn it into a set of category totals, meaning a count of how many times each category appears in the mismatched category array. If you recall, this is precisely the kind of data required to drive a bar graph, where the categories are the headings and the count for each category is the value of each bar. A two-dimensional array can be used to combine categories and totals into a single data structure.



Once this array of category totals is built, we'll be ready to move on to Step 3 and actually use some GD functions to crank out the bar graph visuals.

some array math

Crunching categories

The challenge now is to get the array of categories totaled up and put into a two-dimensional array of headings and values. We have an array of mismatched categories stored in the `$categories` array. We need to build a new array called `$category_totals` that contains one entry for each category, along with the number of mismatches for each.

**We need to
go from this...**

Appearance
Appearance
Appearance
Entertainment
Entertainment
Entertainment
Entertainment
Food
Food
Food
Food
People
People
Activities

`$categories`

The total number of occurrences of a category in the `$categories` array appears as a total in the `$category_totals` array.

...to this

Appearance
Entertainment
Food
People
Activities

`$category_t`

How would you go about totaling the categories in the `$categories` array? Hint: it's a two-dimensional `$category_total`.

Doing the category math

Moving from a one-dimensional array of mismatched categories to a two-dimensional array of category totals is a bit trickier than you might think at first glance. For this reason, it's helpful to work through the solution in **pseudocode** before actually cranking out any PHP. Pseudocode frees you from syntactical details and allows you to focus on the core ideas involved in a particular coding solution.

Create a new two-dimensional array to store the category totals, make sure to initialize the first element with the first mismatched category and a count of 1.

Loop through the array of mismatched categories. For each category:

Is the last element in the category totals array a different category than the current mismatched category?

◆ **Yes! This is a new category so add it to the category totals array and initialize its count to 0.**

◆ **No. This is another instance of the current category so increment the count of the last element in the category totals array by 1.**

The product of this code is a two-dimensional array of category totals where the main array holds a single category, while each sub-array contains the category name and its value.



Translate the pseudocode to finish the real PHP code that builds a two-dimensional array of category totals. Mismatch category data called \$category_totals.

```
$category_totals = array(array($mismatch_categories[0],  
    foreach ($mismatch_categories as $category) {
```

```
.....  
.....  
.....  
.....  
.....  
})
```

exercise solution



Translate the pseudocode to finish the real PHP code that builds a two-dimensional Mismatch category data called \$category_totals.

Arrays are zero-based, so the last element is always counted.

```
$category_totals = array(array($mismatch_categories[0],  
foreach ($mismatch_categories as $category) {  
    if ($category_totals[count($category_totals) - 1][0] != $category) {  
        array_push($category_totals, array($category, 1));  
    }  
    else {  
        $category_totals[count($category_totals) - 1][1]++;  
    }  
}
```

This is a new element being added to the array as a new subarray under the category with the count of 1.

The increment is applied to the element in the array which is the last one.

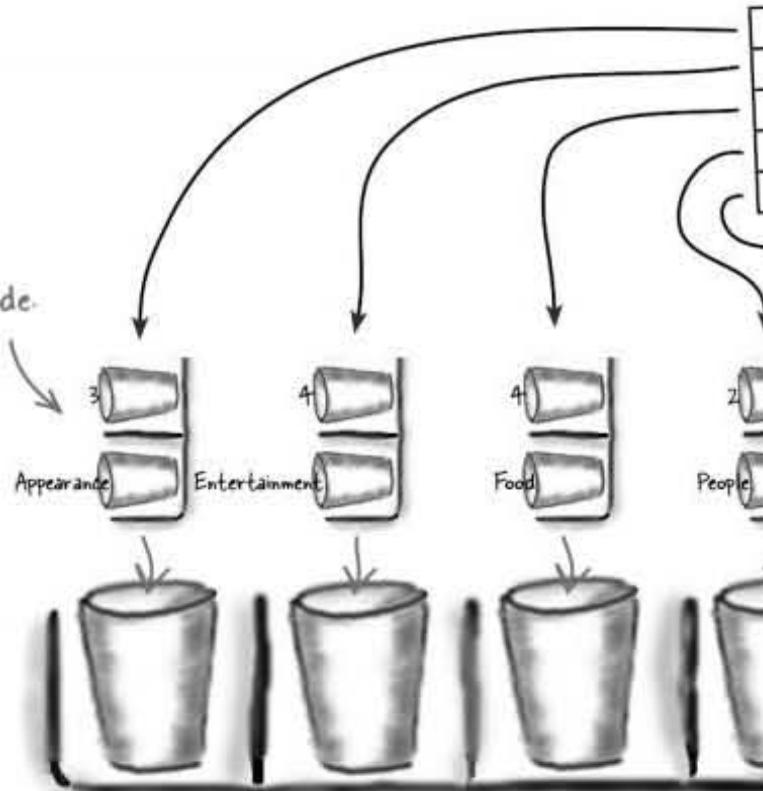
\$category_totals

The \$category_totals variable now holds precisely the data needed to generate a bar graph for the mismatched categories.

This is the end result of this code.

We can now safely scratch this step off the list, leaving us only with the drawing of the bar graph.

- ③ Calculate the mismatch totals for each category.



there are no Dumb Questions

Q: What happens to the category total code if the categories in the \$mismatch_categories array aren't in order?

A: Big problems. The code is entirely dependent upon the categories in the \$mismatch_categories array being in order. This is revealed in how the code assumes that any change in the category is the start of a new category, which works as long as categories are grouped together. Fortunately, the query in the Questionnaire script that originally selects topics for insertion into the mismatch_response table is smart enough to order the responses by category.

```
SELECT topic_id FROM mismatch_topic ORDER BY category_id, topic_id
```

This query is the one that first grabs topics from the database and then inserts them as empty responses for a given user. This ensures that user responses are stored in the database ordered by category, which allows the category total code to work properly.

Q: But isn't it risky writing code that is dependent upon the order of data stored in a database table?

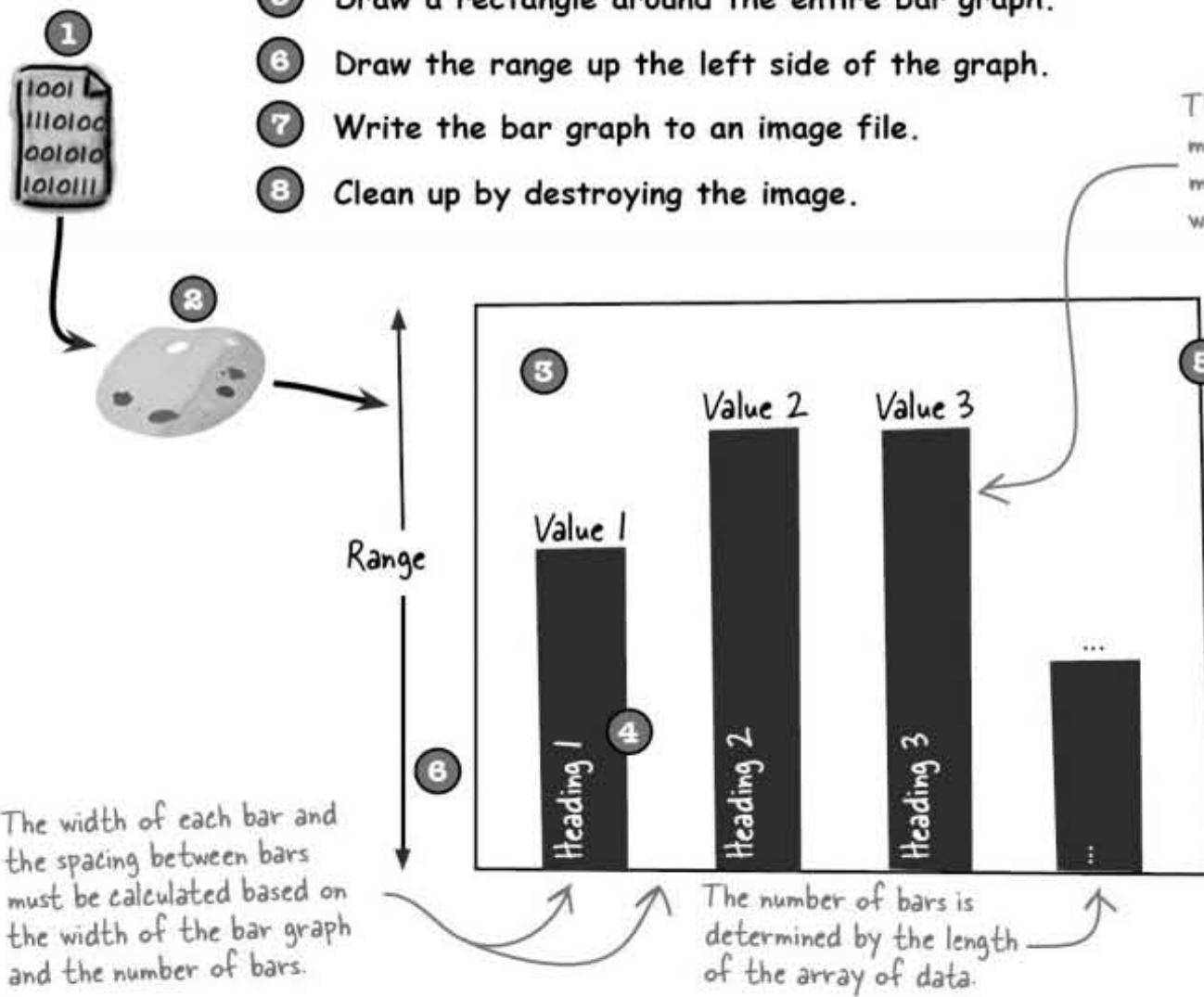
A: Yes and no. Remember that this database is entirely controlled by script code that you write, so the order of the data really only changes if you write script code that changes it. Even so, an argument could certainly be made for ordering the join query in the My Mismatch script by category to make absolutely sure that the mismatched category list is in order.

drawing a bar graph

Bar graphing basics

With a shiny new two-dimensional array of mismatched category data burning a hole in your pocket, it's time to get down to the business of drawing a bar graph. But rather than focus on the specifics of drawing the Mismatch bar graph, why not take a more generic approach? If you design and create an all-purpose bar graph function, it's possible to use it for Mismatch **and** have it at your disposal for any future bar graphing needs. In other words, it's **reusable**. This new function must perform a series of steps to successfully render a bar graph from a two-dimensional array of data.

- 1 Create the image.
- 2 Create the colors you'll be using to draw the graphics and shapes.
- 3 Fill the background with a background color.
- 4 Draw the bars and headings.
- 5 Draw a rectangle around the entire bar graph.
- 6 Draw the range up the left side of the graph.
- 7 Write the bar graph to an image file.
- 8 Clean up by destroying the image.





PHP Magnets

The My Mismatch script contains a new `draw_bar_graph()` function that takes a graph given a width, height, a two-dimensional array of graph data, a maximum value, and a filename for the resulting PNG image. Use the magnets to add the missing GD code.

```

function draw_bar_graph($width, $height, $data, $max_value, $filename) {
    // Create the empty graph image
    $img = ..... ($width, $height);

    // Set a white background with black text and gray graphics
    $bg_color = ..... ($img, 255, 255, 255); // white
    $text_color = ..... ($img, 255, 255, 255); // white
    $bar_color = ..... ($img, 0, 0, 0); // black
    $border_color = ..... ($img, 192, 192, 192); // light gray

    // Fill the background
    ..... ($img, 0, 0, $width, $height, $bg_color);

    // Draw the bars
    $bar_width = $width / ((count($data) * 2) + 1);
    for ($i = 0; $i < count($data); $i++) {
        ..... ($img, ($i * $bar_width * 2) + $bar_width, $height
                - ($i * $bar_width * 2) + ($bar_width * 2), $height - ((($height / $max_value) * $data[$i]) * $bar_width));
        ..... ($img, 5, ($i * $bar_width * 2) + ($bar_width), $height - 5);
        ..... ($img, $text_color);
    }

    // Draw a rectangle around the whole thing
    ..... ($img, 0, 0, $width - 1, $height - 1, $border_color);

    // Draw the range up the left side of the graph
    for ($i = 1; $i <= $max_value; $i++) {
        ..... ($img, 5, 0, $height - ($i * ($height / $max_value)), $i, $bar_color);
    }

    // Write the graph image to a file
    ..... ($img, $filename, 5);
    ..... ($img);
}

```

`imagecreatetruecolor`

`imagestringup`

`imagerectangle`

`imagecolorallocate`

`imagepng`

`imagestring`

php magnets solution

PHP Magnets Solution

The My Mismatch script contains a new `draw_bar_graph()` function that takes a graph given a width, height, a two-dimensional array of graph data, a maximum value, and a filename for the resulting PNG image. Use the magnets to add the missing GD code.

```

function draw_bar_graph($width, $height, $data, $max_value, $filename) {
    // Create the empty graph image
    $img = imagecreatetruecolor($width, $height); First create a blank image for drawing.

    // Set a white background with black text and gray graphics
    $bg_color = imagecolorallocate($img, 255, 255, 255); // white
    $text_color = imagecolorallocate($img, 255, 255, 255); // white
    $bar_color = imagecolorallocate($img, 0, 0, 0); // black
    $border_color = imagecolorallocate($img, 192, 192, 192); // light gray

    // Fill the background
    imagefilledrectangle($img, 0, 0, $width, $height, $bg_color); Clear the background to get it ready for the bar graph graphics.

    // Draw the bars
    $bar_width = $width / ((count($data) * 2) + 1);
    for ($i = 0; $i < count($data); $i++) {
        imagefilledrectangle($img, ($i * $bar_width * 2) + $bar_width, $height - ($i * $bar_width * 2) + ($bar_width * 2), $height - ((($height / $max_value) * $data[$i]) * $bar_width)); Draw a bar as a filled rectangle.
        imagestringup($img, 5, ($i * $bar_width * 2) + ($bar_width), $height - ($i * $bar_width * 2) + ($bar_width * 2), $text_color); Draw the heading for the bar as a string of text oriented vertically.
    }

    // Draw a rectangle around the whole thing
    imagerectangle($img, 0, 0, $width - 1, $height - 1, $border_color); Draw a rectangle around the entire bar graph.

    // Draw the range up the left side of the graph
    for ($i = 1; $i <= $max_value; $i++) {
        imagestring($img, 5, 0, $height - ($i * ($height / $max_value)), $i, $bar_width); Draw the range up the left side of the graph as normal horizontal strings of text.
    }

    // Write the graph image to a file
    imagepng($img, $filename, 5); Write the image to a PNG file with the specified filename and a compression level of 5 (medium).
    imagedestroy($img); Destroy the image in memory to clean up.
}

```

The folder on the server where the file is to be written must be writeable in order for this function to work.

there are no Dumb Questions

Q: Why does the `draw_bar_graph()` function write the bar graph image to a file instead of just returning it directly to the browser?

A: Because the function isn't contained within its own script that can return an image through a header to the browser. Remember, the only way to return a dynamically generated image directly to the browser is for a script to use a header, meaning that the entire purpose of the script has to be the generation of the image.

Q: So then why isn't the `draw_bar_graph()` function placed in its own script so that it can return the bar graph image directly to the browser using a header?

A: While it is a good idea to place the function in its own script for the purposes of making it more reusable, there is still a problem when it comes to returning an image via a header. The problem has to do with how you reuse code. When code is included in a script using `include`, `include_once`, `require`, or `require_once`, the code is dropped into the script as if it had

originally existed there. This works great for anything that manipulates the browser, like the output of a script, which can be problematic. It's not that you can't send a header from a script that actually does so in earlier examples. That's to be extremely careful, and in some cases, that headers haven't already been sent. For example, can't return an image to the browser if the output HTML code containing mismatched data that dynamically generates and returns a header conflict.

Q: OK, so can I just reference the `captcha.php` script from `Guitar` fine without an `include`, right?

A: Yes, it did, and it referenced the `captcha.php` script directly from the `src` attribute of an `` tag. The problem is that we have a lot of data that needs to be included in the `src` attribute, and this would be very cumbersome for a POST.

Draw and display the bar graph image

The `draw_bar_graph()` function makes it possible to dynamically generate a bar graph image, provided you give it the proper information. In the case of the Mismatch bar graph, this involves sending along a suitable width and height (480 and 240 pixels), a two-dimensional array of mismatched category data (an array of arrays that works on the My Mismatch page (480×240), the maximum range value (5 as the maximum range value (maximum number of mismatch topics per category)), and a suitable upload path and filename for the resulting bar graph image. After calling the function, the image is generated and suitable for display using an HTML `` tag.

```
echo '<h4>Mismatched category breakdown:</h4>';
```

```
draw_bar_graph(480, 240, $category_totals, 5, MM_UPLOADPATH . 'mymismatchgraph.png');
```

With the help of a reusable function, it's possible to go from database data to a bar graph stored in an image file.

The same path and image filename are specified in the `src` attribute of the `` tag.

Appearance	3
Entertainment	4
Food	4
People	2
Activities	5



test out mismatch, now with bar graphs



Test Drive

Create the My Mismatch script and try it out.

Create a new text file named `mymismatch.php`, and enter the code for the My Mismatch script (or download the script from the Head First Labs site at www.headfirstbooks/hfphp). Also add a new menu item for My Mismatch to the navmenu.

Upload the scripts to your web server, and then open the main Mismatch page (index.php) in a web browser. Log in if you aren't already logged in, and then click 'My Mismatch' in the main navigation menu. Congratulations, this is your ideal mismatch!

Mismatch - My Mismatch

Home • View Profile • Edit Profile • Questionnaire • My Mismatch • Log Out

Jason Filmington
Hollywood, CA

You are mismatched on the following 18 topics:

Tattoos	Cowboy boots	Long hair	Reality TV
Horror movies	Easy listening music	The opera	Sushi
Spicy food	Peanut butter & banana sandwiches	Martinis	Bill Gates
Hugh Hefner	Yoga	Weightlifting	Cube puzzles
Karaoke	Hiking		

Mismatched category breakdown:

Category	Count
Appearance	3
Entertainment	4
Food	4
People	2
Activities	5

[View Jason's profile.](#)

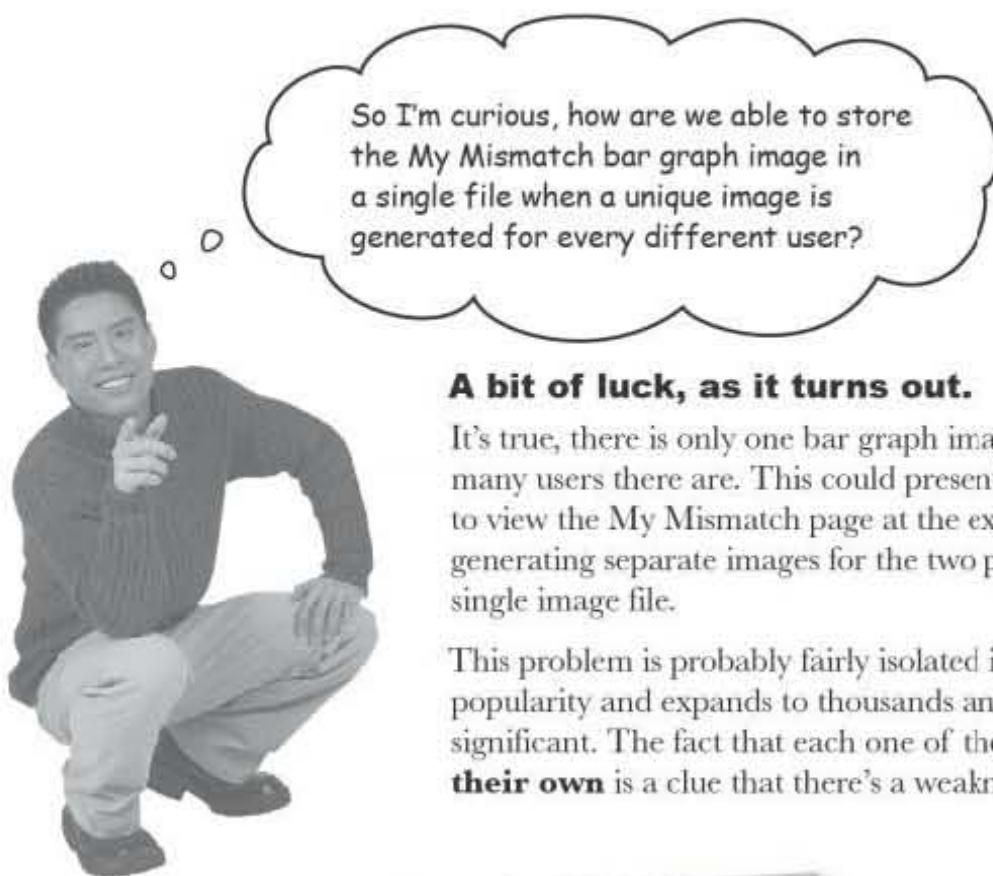
Copyright ©2006 Mismatch Enterprises, Inc.

Now that's what I'm talking about! The visual category breakdown is all I needed to see to know Jason's the one.

The topics have been reformatted into a table to make room for the bar graph.

The bar graph fits neatly on the My Mismatch page along with the list of mismatched topics.

- ⑤ Draw the bar graph using the categorized mismatch totals.



A bit of luck, as it turns out.

It's true, there is only one bar graph image at any given time, no matter how many users there are. This could present a problem if two people tried to view the My Mismatch page at the exact same moment. One way would be to generate separate images for the two people and then try to merge them into a single image file.

This problem is probably fairly isolated in reality, but as Myspace grew in popularity and expanded to thousands and thousands of users, this becomes significant. The fact that each one of the users thinks of themselves as **their own** is a clue that there's a weakness in the single image file.

Here we clearly have three different bar graph images in view, but we know only one image file is used to store them.



A single continuous stream for everyone

Mismatch

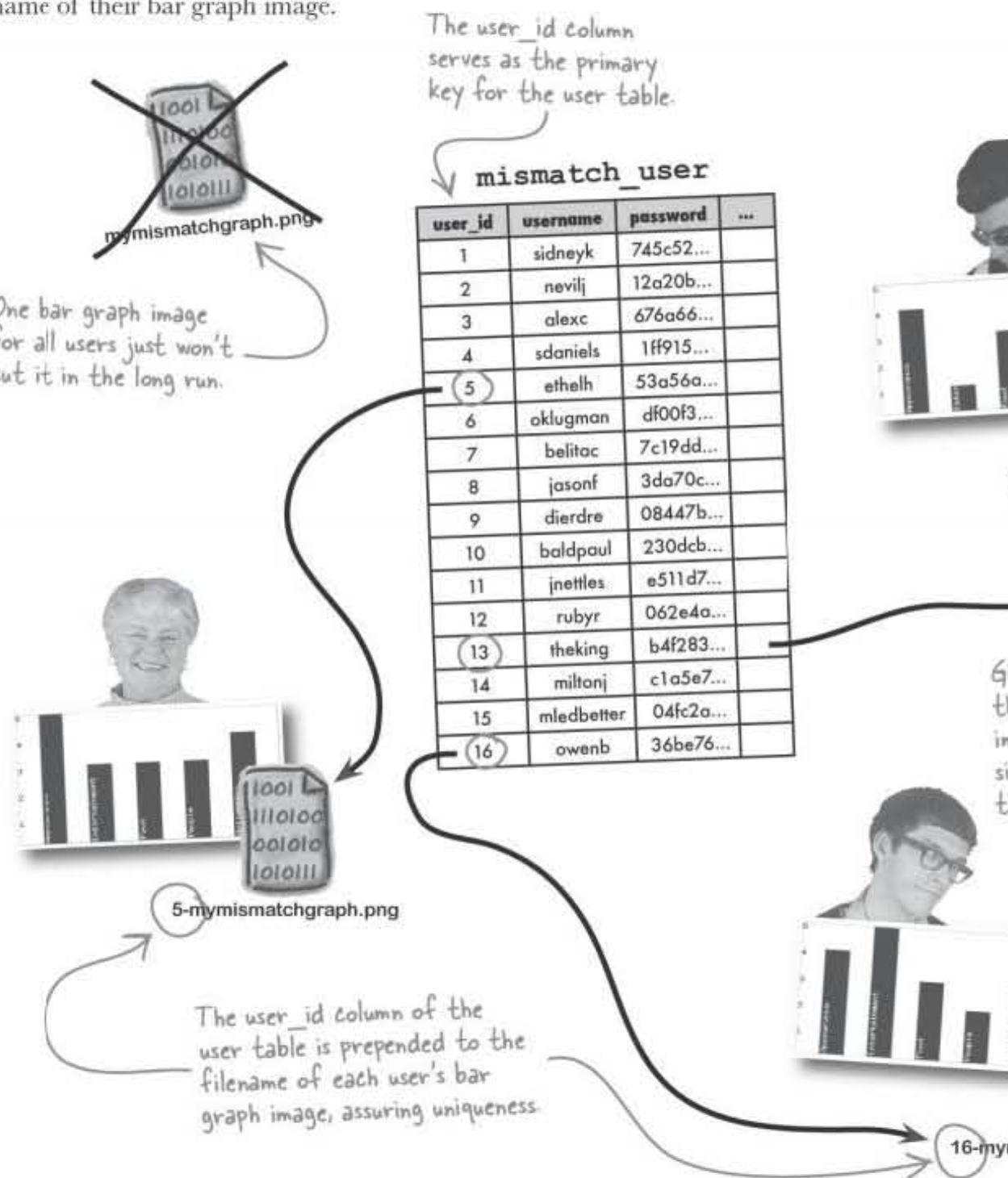


How would you change the code to ensure that users see the same bar graph in

a bar graph for every user

Individual bar graph images for all

The solution to the shared bar graph image problem lies in generating multiple images, one for every user, in fact. But we still need to ensure that each of these images is tied to exactly one user and no more. That's where a familiar database design element comes into play... the primary key! The primary key for the `mismatch_user` table, `user_id`, uniquely identifies each user, and therefore provides an excellent way to uniquely name each bar graph image and also associate it with a user. All we have to do is prepend the users' IDs to the filename of their bar graph image.



there are no
Dumb Questions

Q: Is there any advantage to outputting dynamically generated images as PNG images versus GIFs or JPEGs?

A: No, none beyond the reasons you would choose one image format over another for static images. For example, GIFs and PNGs are better for vector-type graphics, whereas JPEGs are better for photorealistic graphics. In the case of Mismatch, we're dealing with vector graphics, so either PNG or GIF would work fine. PNG happens to be a more modern image standard, which is why it was used, but GIF would've worked too. To output a GD image to a GIF and JPEG, respectively, call the `imagegif()` and `imagejpeg()` functions.

Q: How do I know what compression level to use when outputting PNG images to a file?

A: The compression level settings for the `imagepng()` function enter the picture when outputting a PNG image to a file, and they range from 0 (no compression) to 9 (maximum compression). There are no hard rules about how much compression to use when, so you may want to experiment with different settings. Mismatch uses 5 as the compression level for the bar graphs, which appears to be a decent tradeoff between quality and efficiency.



Below is the Mismatch code that dynamically generates a bar graph image it on the page. Rewrite the code so that it generates a unique image for each user by using `$_SESSION['user_id']` to build a unique image filename for each user.

```
echo '<h4>Mismatched category breakdown:</h4>';
draw_bar_graph(480, 240, $category_totals, 5, MM_UPLOADPATH . 'mymismatchgraph.png');
echo '' . "\n";
```



exercise solution

Below is the Mismatch code that dynamically generates a bar graph image on the page. Rewrite the code so that it generates a unique image for `$_SESSION['user_id']` to build a unique image filename for each user.

```
echo '<h4>Mismatched category breakdown:</h4>';
draw_bar_graph(480, 240, $category_totals, 5, MM_UPLOADPATH . 'mymismatchgraph.png');
echo '<img src="" . MM_UPLOADPATH . "mymismatchgraph.png" alt="Mismatch category graph" /><br />';
```

The standard file upload path is still used to ensure that the image is stored in the desired place on the server.

Make sure this folder on the server is writeable so that the image file can be written.

The unique image file is in the form X-mymismatchgraph.png where X is the ID of the user.

```
echo '<h4>Mismatched category breakdown:</h4>';
draw_bar_graph(480, 240, $category_totals, 5,
MM_UPLOADPATH . $_SESSION['user_id'] . '-mymismatchgraph.png');
echo '<img src="" . MM_UPLOADPATH . $_SESSION['user_id'] . "-mymismatchgraph.png" alt="Mismatch category graph" /><br />';
```

We can use the user ID that we already have stored away in a session variable.

The same image is generated when setting the source URL of the tag to the URL of the bar graph image in the browser.



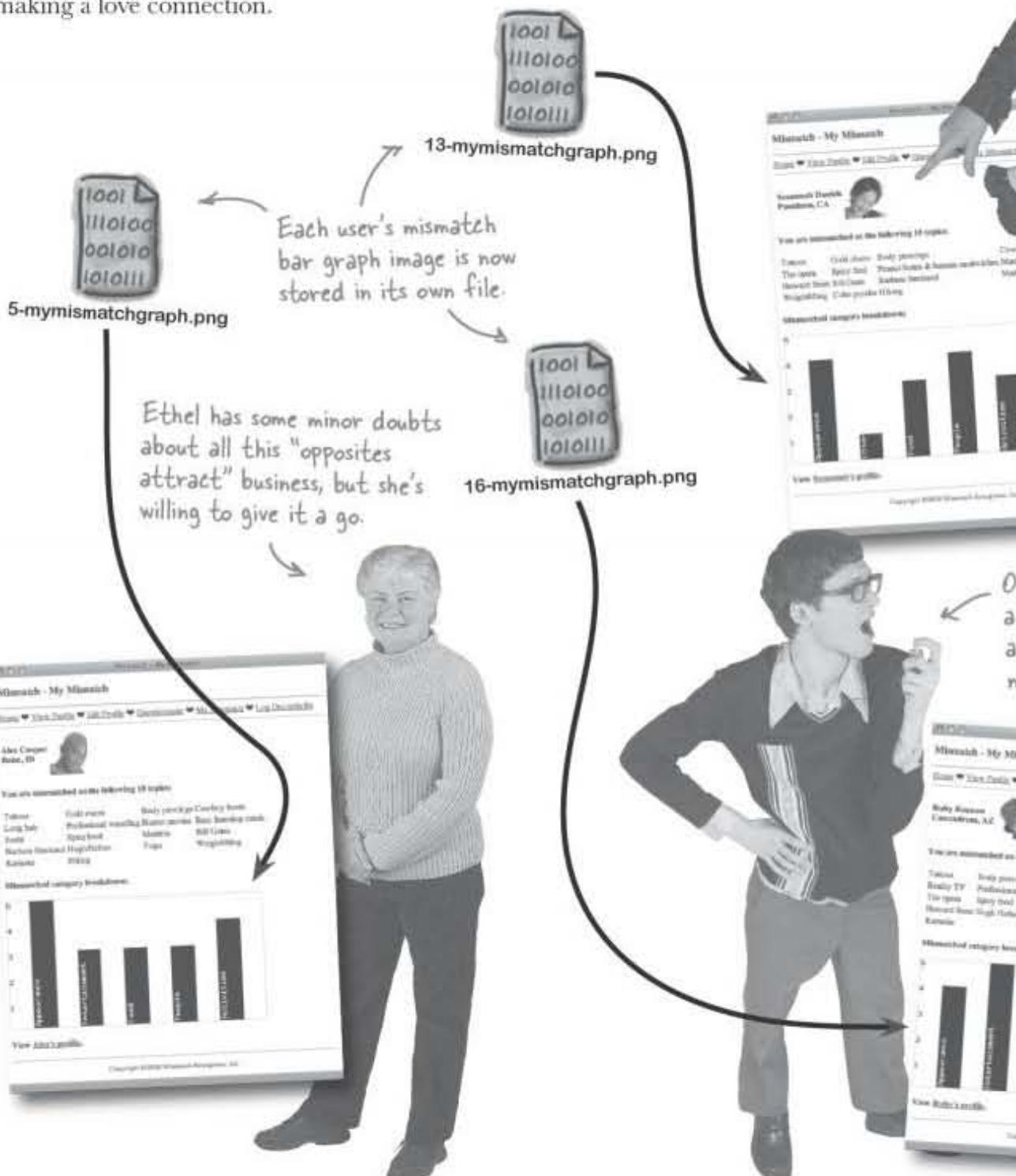
Test Drive

Change the My Mismatch script to generate unique bar graphs

Modify the My Mismatch script so that it generates a unique bar graph for each user. Upload the `mymismatch.php` script to your web server, and then open it in a web browser. The page won't look any different, but you can view source on it to see that the bar graph image now has a unique filename.

Mismatch users are digging the bar graphs

With the shared bar graph image problem solved, you've helped eliminate a potential long-term performance bottleneck as more and more users join Mismatch and take advantage of the "five degrees of opposability" graph. Each user now generates their own unique bar graph image when viewing their ideal mismatch. Fortunately, this fix took place behind the scenes, unbeknownst to users, who are really taking advantage of the mismatch data in the hopes of making a love connection.



CHAPTER 11

php & mysql toolbox

Your PHP & MySQL Toolbox

Dynamic graphics open up all kinds of interesting possibilities in terms of building PHP scripts that generate custom images on the fly. Let's recap what makes it all possible.

CAPTCHA

A program that protects a web site from automated spam bots by using a test of some sort. For example, a CAPTCHA test might involve discerning letters within a distorted pass-phrase, identifying the content of an image, or analyzing an equation to perform a simple mathematical computation.

`imageline()`,
`imagerectangle()`, ...

The GD graphics library offers lots of functions for drawing graphics primitives, such as lines, rectangles, ellipses, and even individual pixels. Each function operates on an existing image that has already been created with `imagecreatetruecolor()`.

GD library

A set of PHP functions that are used to draw graphics onto an image. The GD library allows you to dynamically create and draw on images, and then either return them directly to the browser or write them to image files on the server.

imagepng()

When you're finished drawing to an image using GD graphics functions, this function outputs the image so that it can be displayed. You can choose to output the image directly to the web browser or to an image file on the server.

image
This fu
GD gr
used t
drawin
create
outpu
callin
imag

ima
ima
ima

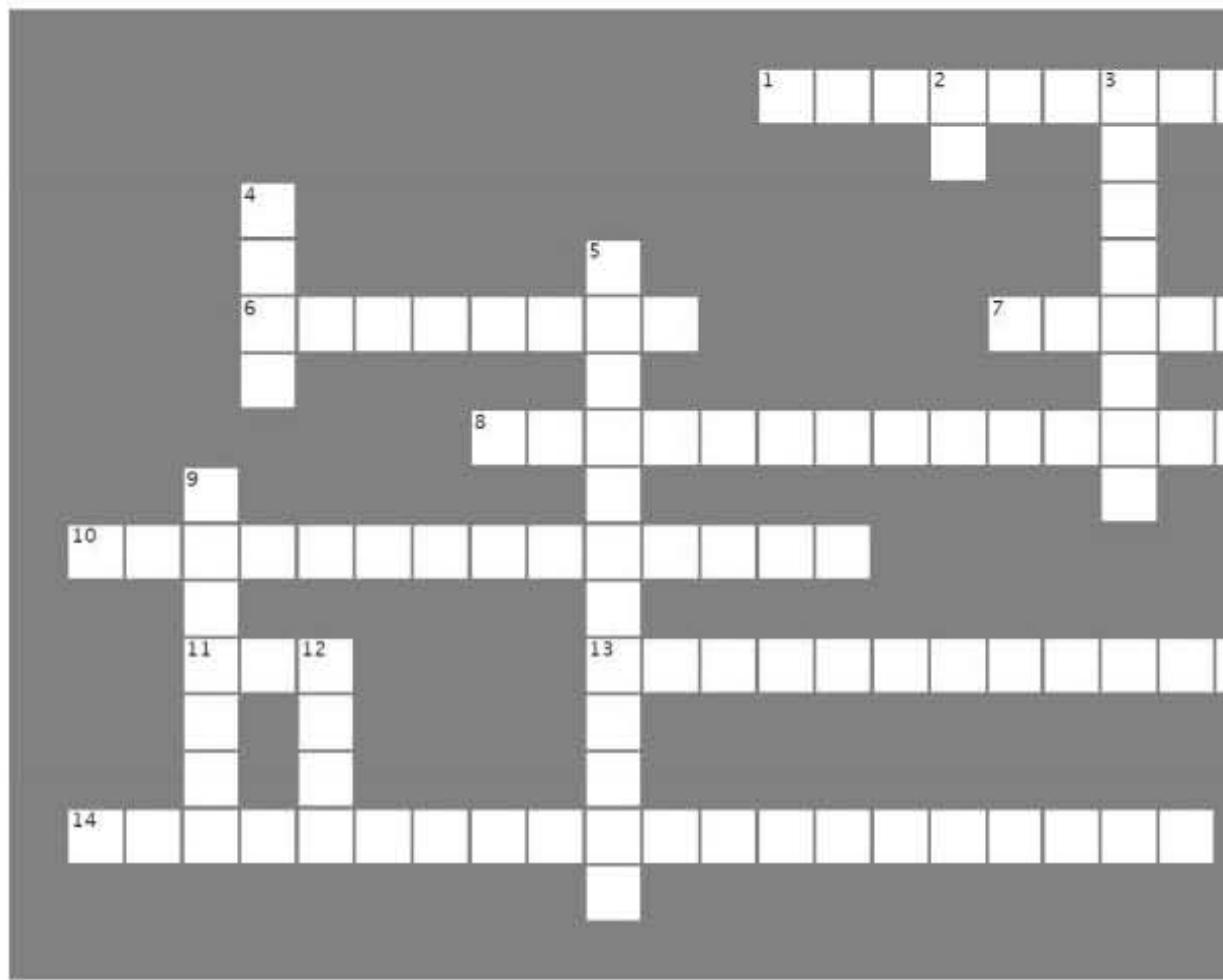
The
allow
with
a Tru
choos

ima
Aft
out
a go
res
call



PHP&MySQLcross

When you could actually use a robot, they're nowhere to be found. Oh well, your analog brain is up to the challenge of solving this little puzzle.



Across

- This PHP graphics function draws a line.
- The visual used to show how mismatched users compare on a categorized basis.
- To generate custom bar graph images for each user in Mismatch, this piece of information is used as part of the image filename.
- Mismatch uses this kind of array to store bar graph data.
- Give it two points and this graphics function will draw a rectangle.
- If you want to draw text in a certain font, call the `image...text()` function.
- Always clean up after working with an image in PHP by calling this function.
- Call this graphics function to create a new image.

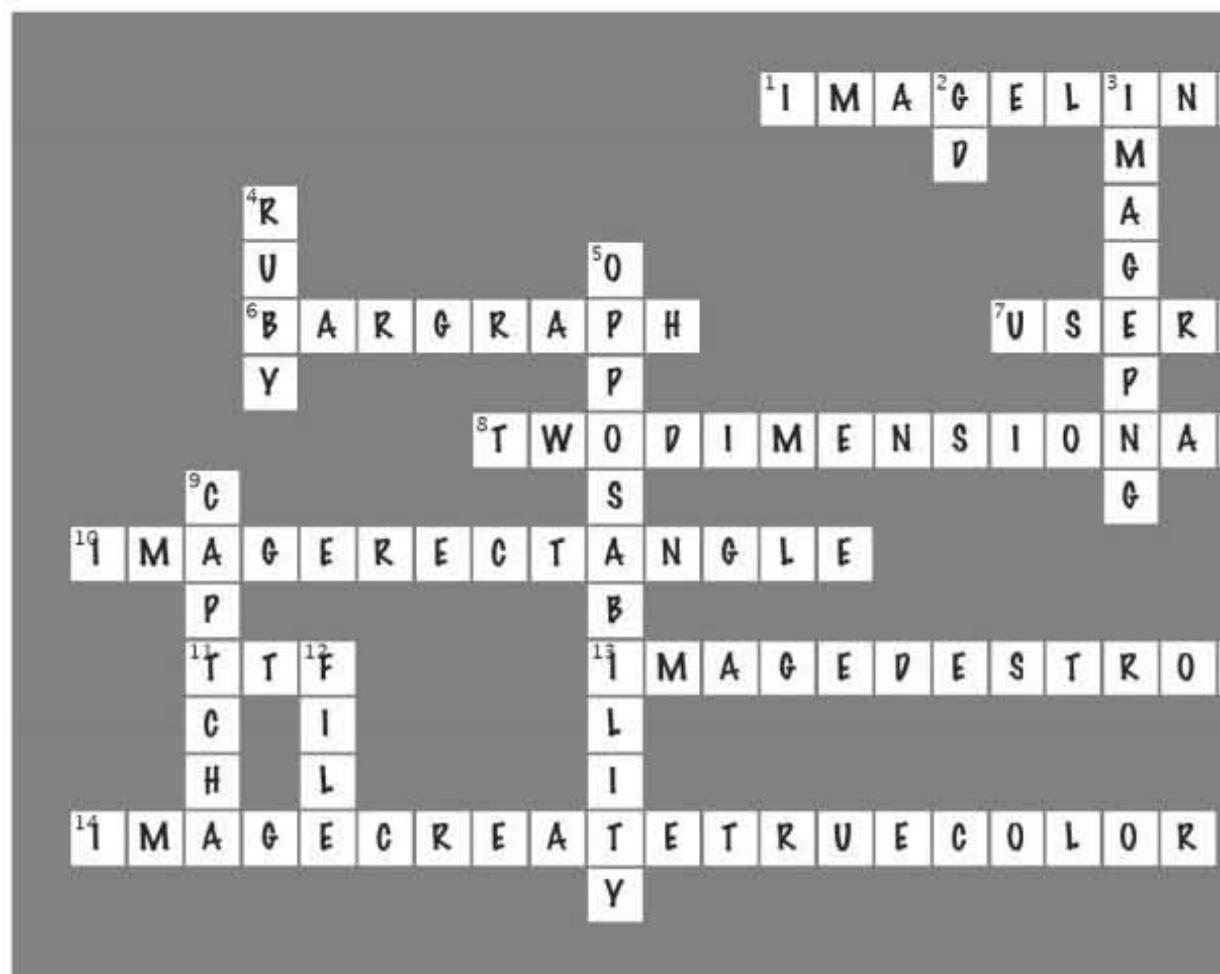
Down

- The name of PHP's graphics library.
- Call this function to output an image as
- Owen's ideal mismatch.
- Mismatch uses a bar graph to compare "five degrees of".
- A test used to distinguish between people and spam bots.
- When PHP outputs an image, the image goes directly to the client browser or stored in

php&mysqlcross solution



PHP&MySQL cross Solution



12 syndication and web services

Interfacing to the world



It's a big world out there, and one that your web application can't afford to ignore. Perhaps more importantly, you'd rather the world not ignore your web application. One excellent way to tune the world in to your web application is to make its data available for syndication, which means users can subscribe to your site's content instead of having to visit your web site directly to find new info. Not only that, your application can interface to other applications through web services and take advantage of other people's data to provide a richer experience.

spreading the word beyond owen's web site

Owen needs to get the word out about Fang

One of the big problems facing any web site is keeping people coming back. It's one thing to snare a visitor, but quite another to get them to come back again. Even sites with the most engaging content can fall off a person's radar simply because it's hard to remember to go visit a web site regularly. Knowing this, Owen wants to offer an alternative means of viewing alien abduction reports—he wants to "push" the reports to people, as opposed to them having to visit his site on a regular basis.

Aliens Abducted Me - Report an Abduction

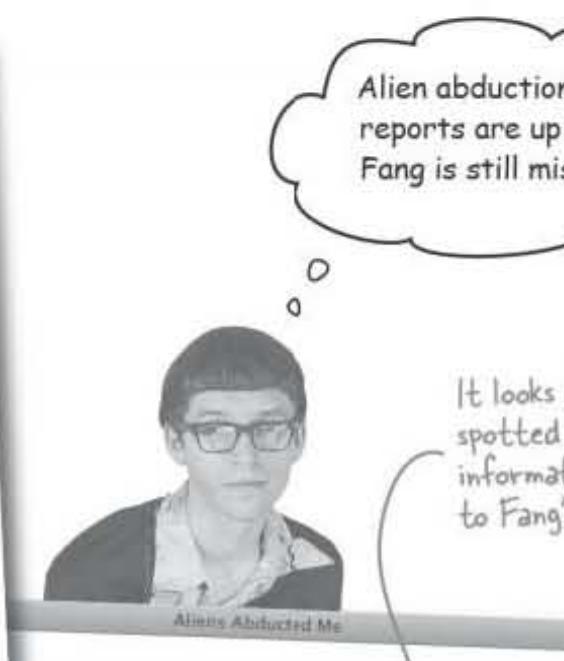
Share your story of alien abduction:

First name: Belta
Last name: Chevy
What is your email address? belitac@rockin.net
When did it happen? 2008-06-21
How long were you gone? almost a week
How many did you see? 27
Describe them: Clumsy little buggers, had no rhythm.
What did they do to you? Tried to get me to play bad music.
Have you seen my dog Fang? Yes No

Anything else you want to add?
 Looking forward to playing some Guitar Wars

The Report an Abduction form is working great, but Owen thinks the site needs more exposure.

Since you last saw Owen, he's created a main page for viewing user-submitted alien abduction reports.



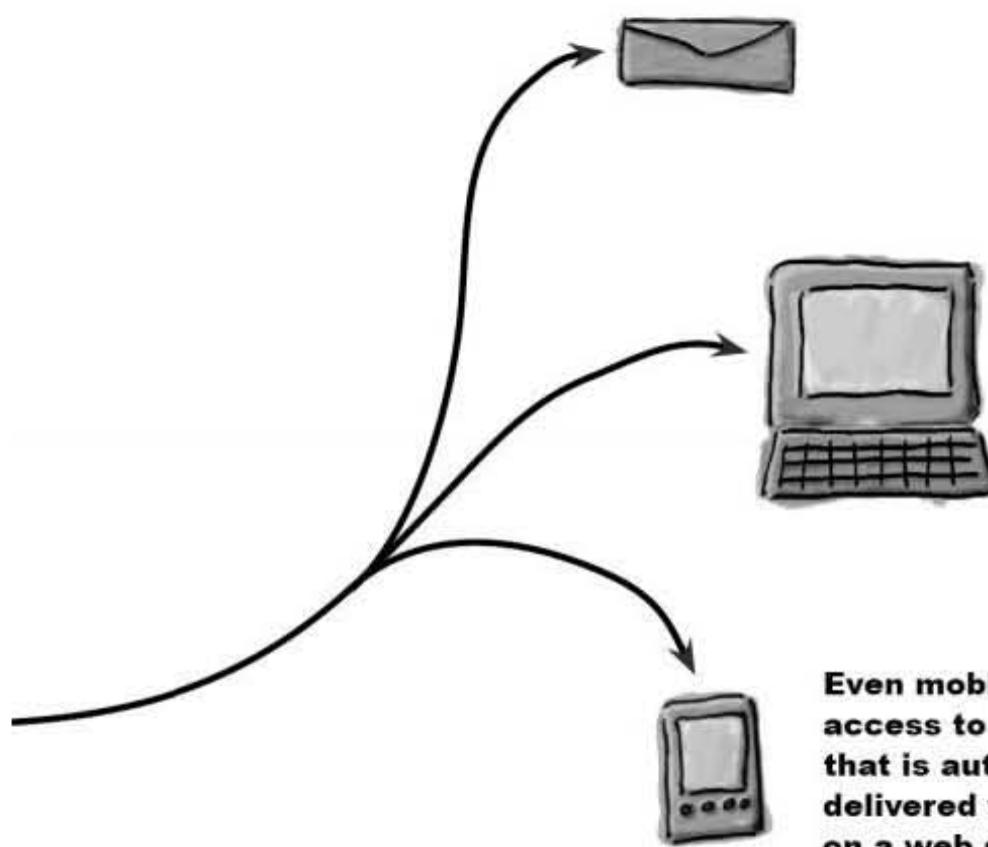
- 2008-06-21 : Belta Chevy
Abducted for: Alien description:
almost a week Clumsy little buggers, had no rhythm.
- 2008-05-11 : Sally Jones
Abducted for: Alien description:
1 day green with six tentacles
- 2000-07-12 : Alf Nader
Abducted for: Alien description:
one week It was a big non-recyclable shiny disc full of what appeared to be mutated labor union members.
- 1991-09-14 : Don Quayle
Abducted for: Alien description:
37 seconds They looked like donkeys made out of metal with some kind of jet packs attached to their backs.
- 1969-01-21 : Rick Nixon
Abducted for: Alien description:
nearly 4 years They were pasty and pandering, and not very forgiving.

Owen hopes that more exposure for the site will increase his odds of finding Fang.

Push alien abduction data to the people

By pushing alien abduction content to users, Owen effectively creates a virtual team of people who can help him monitor abduction reports. With more people on the case, the odds of identifying more Fang sightings and hopefully homing in on Fang's location increase.

Some email clients support “push” content, allowing you to receive web site updates the same way you receive email messages.



Many regular web browsers also let you browse “push” content that quickly reveals the latest news posted to a web site.

Even mobile devices provide access to “push” content that is automatically delivered when something on a web site changes.

Owen's virtual team of alien abduction content viewers will hopefully increase the chances of him finding Fang.

use RSS to syndicate your site

RSS pushes web content to the people

The idea behind posting HTML content to the Web is that it will be viewed by people who visit a web site. But what if we want users to receive our web content without having to ask for it? This is possible with RSS, a data format that allows users to find out about web content without actually having to visit a site.

RSS is kinda like the web equivalent of a digital video recorder

(DVR). DVRs allow you to “subscribe” to certain television shows, automatically recording every episode as it airs. Why flip channels looking for your favorite show when you can just let the shows come to you by virtue of the DVR? While RSS doesn’t actually record anything, it is similar to a DVR in that it brings web content to you instead of you having to go in search of it.

By creating an RSS feed for his alien abduction data, Owen wants to notify users when new reports are posted. This will help ensure that people stay interested, resulting in more people combing through the data. The cool thing is that the **same database** can drive both the web page and the RSS feed.

Aliens Abducted Me

Welcome, have you had an encounter with extraterrestrials? Were you abducted? Here you see my abductions! Report it here!

Most recent reported abductions:

- 2008-06-21 : Bella Chevy
Abducted her! Alien description:
about a week - Clumsy little buggers, had no rhythm.
- 2008-05-11 : Sally Jones
Abducted her! Alien description:
1 day - green with six tentacles
- 2000-07-12 : Alf Nader
Abducted her! Alien description:
one week - It was a big non-recyclable shiny disc full of what appeared to be mutated labor union officials.
- 1991-09-14 : Don Quayle
Abducted her! Alien description:
37 seconds - They looked like Starwars made out of metal with some kind of jet pack attached to them
- 1969-01-21 : Rick Nixon
Abducted her! Alien description:
nearly 4 years - They were party and pandering, and not very frigging

Aliens Abducted Me - Newsfeed

- Bella Chevy - Clumsy little buggers, had no rhythm...
They try to get me to play their music. [Read more...](#) [12:00 AM]
- Sally Jones - green with six tentacles...
We just talked and played with a dog. [Read more...](#) [11:00 AM]
- Alf Nader - It was a big non-recyclable shiny disc full of what appeared to be mutated labor union officials...
Starwars made out of the big stuff everlasted and was not recycling. [Read more...](#) [11:00 AM]
- Don Quayle - They looked like donkeys made out of metal...
I was eating dinner reading a book and suddenly when "Porkies" the team of light from the sky... [Read more...](#) [11:00 AM]
- Rick Nixon - They were party and pandering, and not very frigging...
I planned to go to the moon, then they invaded me. [Read more...](#) [11:00 AM]

Even though the web page is dynamically generated from database data, you have to revisit it to see if new data has been posted.

Here, the newsfeed is built into the browser is b

RSS offers a view on web data that is delivered to users automatically as new content is made available. An RSS view on a particular set of data is called an **RSS feed**, or **newsfeed**. Users subscribe to the feed and receive new content as it is posted to the web site—no need to visit the site and keep tabs.

To view an RSS feed, all a person needs is an RSS **newsreader**. Most popular web browsers and email clients can subscribe to RSS feeds. You just provide the newsreader with the URL of the feed, and it does all the rest.

RSS is really XML

RSS is like HTML in that it is a plain text **markup language** that uses tags and attributes to describe content. RSS is based on XML, which is a general markup language that can be used to describe any kind of data. XML's power comes from its flexibility—it doesn't define any specific tags or attributes; it just sets the rules for how tags and attributes are created and used. It's up to specific languages such as HTML and RSS to establish the details regarding what tags and attributes can be used, and how.

In order to be proficient with RSS, you must first understand the ground rules of XML. These rules apply to all XML-based languages, including RSS and the modern version of HTML known as XHTML. These rules are simple but important—your XML (RSS) code won't work if you violate them! Here goes:

Tags that contain content must appear as matching pairs.

Incorrect! The empty tag needs a space and a forward slash before the >.

Empty tags that have no content must be coded with a space and a forward slash at the end before the closing brace.

Incorrect! To be enclosed in quotes.

All attribute values must be enclosed in double quotes.

Unlike PHP, which allows you to use double or single quotes in most situations, XML is rigid in only allowing double quotes for attribute values.

XML is a markup language used to describe any kind of data.

no dumb questions about rss

there are no
Dumb Questions

Q: Why is RSS so much better than someone just coming to my web site?

A: If people regularly visited your web site to seek out the latest content, then RSS wouldn't be any better than simply displaying content on your web site. But most people forget about web sites, even ones they like. So RSS provides an effective means of taking your web content directly to people, as opposed to requiring them to seek it out.

Q: What does RSS stand for?

A: Nowadays RSS stands for Really Simple Syndication. Throughout its storied history there have been several different versions, but the latest incarnation of RSS (version 2.0) stands for Really Simple Syndication, which is all you need to worry about.

Q: So what does RSS consist of?

A: RSS is a data format. So just as HTML is a data format that allows you to describe web content for viewing in a web browser, RSS is a data format that describes web content that is accessible as a news feed. Similar to HTML, the RSS data format is pure text, and consists of tags and attributes that are used to describe the content in a newsfeed.

Q: Where do I get an RSS reader?

A: Most web browsers have a built-in RSS reader. Some email clients even include RSS readers, in which case RSS news items appear as email messages in a special news feed folder. There are also stand-alone RSS readers available.



Below is RSS code for an Aliens Abducted Me news feed. Annotate the highlighted tags and explain what you think each tag is doing.

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
  <channel>
    <title>Aliens Abducted Me - Newsfeed</title>
    <link>http://aliensabductedme.com/</link>
    <description>Alien abduction reports from around the world courtesy
      abducted dog Fang.</description>
    <language>en-us</language>

    <item>
      <title>Belita Chevy - Clumsy little buggers, had no rh...</title>
      <link>http://www.aliensabductedme.com/index.php?abduction_id=7</link>
      <pubDate>Sat, 21 Jun 2008 00:00:00 EST</pubDate>
      <description>Tried to get me to play bad music.</cescription>
    </item>

    <item>
      <title>Sally Jones - green with six tentacles...</title>
      <link>http://www.aliensabductedme.com/index.php?abduction_id=8</link>
      <pubDate>Sun, 11 May 2008 00:00:00 EST</pubDate>
      <description>We just talked and played with a dog</description>
    </item>

  ...
  </channel>
</rss>
```

annotated rss code



Exercise Solution

```

<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
  <channel>
    <title>Aliens Abducted Me - Newsfeed</title>
    <link>http://aliensabductedme.com/</link>
    <description>Alien abduction reports from around the world courtesy
      abducted dog Fang.</description>
    <language>en-us</language>
    <item>
      <title>Belita Chevy - Clumsy little buggers, had no rh...</title>
      <link>http://www.aliensabductedme.com/index.php?abduction_id=7</link>
      <pubDate>Sat, 21 Jun 2008 00:00:00 EST</pubDate>
      <description>Tried to get me to play bad music.</description>
    </item>
    <item>
      <title>Sally Jones - green with six tentacles...</title>
      <link>http://www.aliensabductedme.com/index.php?abduction_id=8</link>
      <pubDate>Sun, 11 May 2008 00:00:00 EST</pubDate>
      <description>We just talked and played with a dog</description>
    </item>
  ...
</channel>
</rss>

```

Below is RSS code for an Aliens Abducted Me news feed. Annotate the highlights to explain what you think each tag is doing.

This line of code isn't a tag - it's an XML "directive" that identifies this document as containing XML code.

This `<title>` tag applies to the channel as a whole.

The link for a channel points to the web site associated with the news feed.

Every channel needs a description to explain the kind of news it contains.

Newsfeeds can be created in different languages - this tag establishes the language of a channel.

The name of the alien abductee and the alien description data are combined to serve as the title of each news item.

This RSS document only contains a single channel, which is perfectly fine if you don't need to break up news items into different categories.

The date specified in the `<pubDate>` tag adheres to the RFC-822 date/time format, which is a standard for representing a detailed date and time as text.

Every XML tag must have a start-tag and an end-tag - this end-tag closes the RSS document.



Yes, sort of. But you don't typically create XML by hand, and it often doesn't get stored in files.

It's true that XML can and often does get stored in files. But with we're talking about dynamic data that is constantly changing, so make sense to store it in files—it would quickly get outdated and have to continually rewrite the file. Instead, we want XML code generated on-the-fly from a database, which is how the HTML of the main Aliens Abducted Me page already works. So we want PHP to dynamically generate RSS (XML) code and return it directly to an RSS newsreader upon request.

rss newsreaders explained

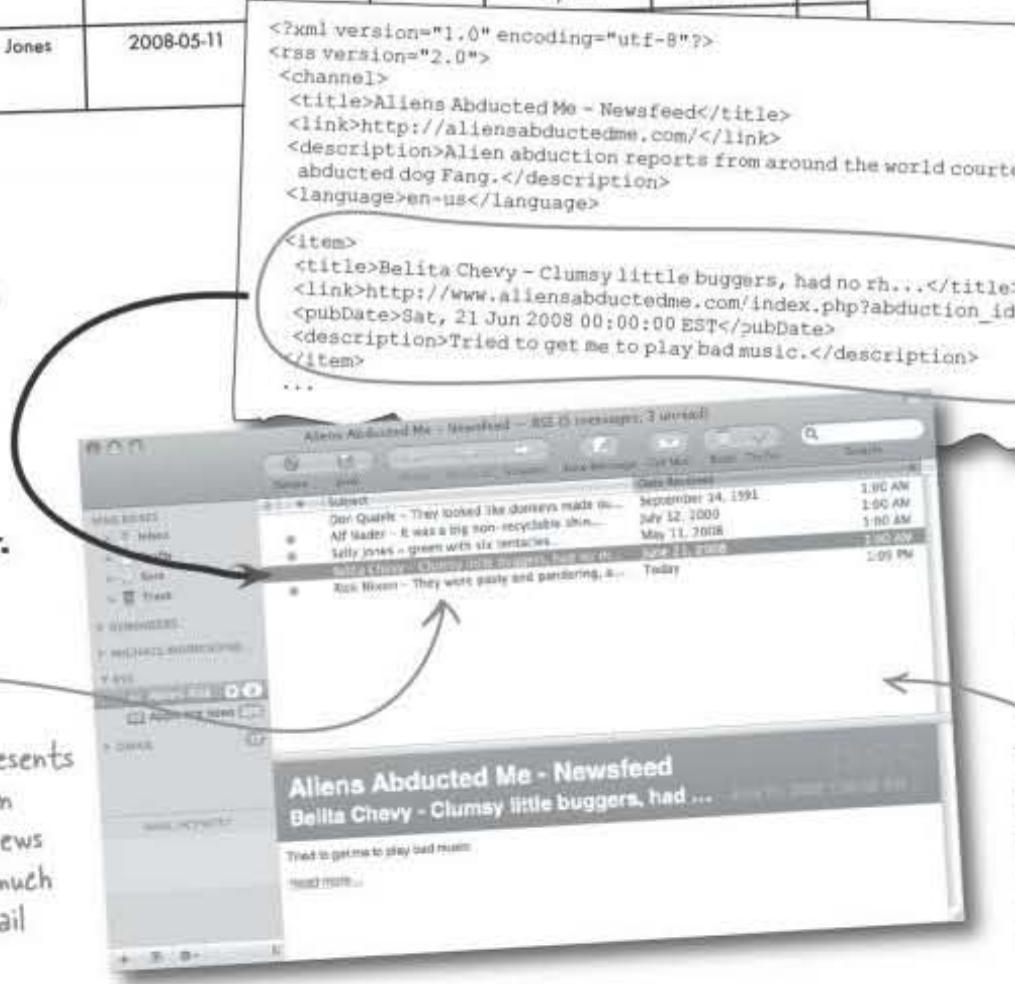
From database to newsreader

In order to provide a newsfeed for alien abduction data, Owen needs to dynamically generate RSS code from his MySQL database. This RSS code forms a complete RSS **document** that is ready for consumption by RSS newsreaders. So PHP is used to format raw alien abduction data into the RSS format, which is then capable of being processed by newsreaders and made available to users. The really cool part of this process is that once the newsfeed is made available in RSS form, everything else is automatic—it's up to newsreaders to show updated news items as they appear.

aliens_abduction								
abduction_id	first_name	last_name	when_it_happened	how_long	how_many	alien_description	what_they_did	...
1	Alf	Nader	200-07-12	one week	at least 12	It was a big non-recyclable shiny disc full of...	Swooped down from the sky and...	-
2	Don	Quayle	1991-09-14	37 seconds	dunno	They looked like donkeys made out of metal...	I was sitting there eating a baked...	-
3	Rick	Nixon	1969-01-21	nearly 4 years	just one	They were pasty and pandering, and not very...	Impacted me, of course, then they probed...	-
4	Belita	Chevy	2008-06-21	almost a week	27	Clumsy little buggers, had no rhythm.	Tried to get me to play bad music.	-
5	Sally	Jones	2008-05-11	<?xml version="1.0" encoding="utf-8"?><rss version="2.0"><channel><title>Aliens Abducted Me - Newsfeed</title>				

The newsreader knows how to interpret individual news items in XML code and show them to the user.

Each newsreader presents news items in its own unique way – here news items are shown in much the same way as email messages.



The logo features the words "WHO DOES WHAT?" in a stylized, bubbly font. A question mark is integrated into the letter "O". There are small stars and a decorative swirl above the text.

Creating RSS feeds is all about understanding the RSS language, which means getting to know the tags that are used to describe news items. Match each RSS tag to its description.

<rss>

This tag has nothing to do with RSS. But it sure sounds like a cool name for a piece of news data!

<channel>

The publication date is an important piece of information for any news item, and this tag is used to specify it.

<cronkite>

This tag represents a single channel in an RSS feed, a container for descriptive data and individual news items.

<title>

Represents an individual news item, or story, which is described by child elements.

<language>

This tag always contains a URL that serves as the link to the channel or news item.

<link>

Encloses an entire RSS feed—all other tags must appear within this tag.

<description>

This tag stores the title of a channel or news item, and is typically used within the <channel> and <item> tags.

<pubDate>

Used to provide a brief description of a channel or news item, appearing within either the <channel> and <item> tags.

<item>

This tag applies to a channel, and specifies the language used by the channel, such as en-us (U.S. English).

who does what solution

WHO DOES WHAT?

SOLUT

Every RSS feed consists of at least one channel, which is basically a group of related news items.

Creating RSS feeds is all about understanding the RSS language, which means getting to know the tags that are used to describe news items. Match each RSS tag to its description.

The `<rss>` tag is the "root" tag for an RSS document – all other tags must appear inside of it.

This tag has nothing to do with RSS. But it sure sounds like a cool name for a piece of news data!

The publication date is an important piece of information for any news item, and this tag is used to specify it.

This tag represents a single channel in an RSS feed, a container for descriptive data and individual news items.

Represents an individual news item, or story, which is described by child elements.

This tag always contains a URL that serves as the link for a channel or news item.

Encloses an entire RSS feed—all other tags must appear inside of this tag.

This tag stores the title of a channel or news item, and is typically used within the `<channel>` and `<item>` tags.

Used to provide a brief description of a channel or news item, appearing within either the `<channel>` and `<item>` tags.

This tag applies to a channel, and specifies the language used by the channel, such as `en-us` (U.S. English).

The `<title>`, `<link>`, `<pubDate>`, and `<description>` tags are used within `<item>` to describe a news item.

`<rss>`
`<channel>`
`<cronkite>`

`<title>`
 This tag is only used in channels.
`<language>`

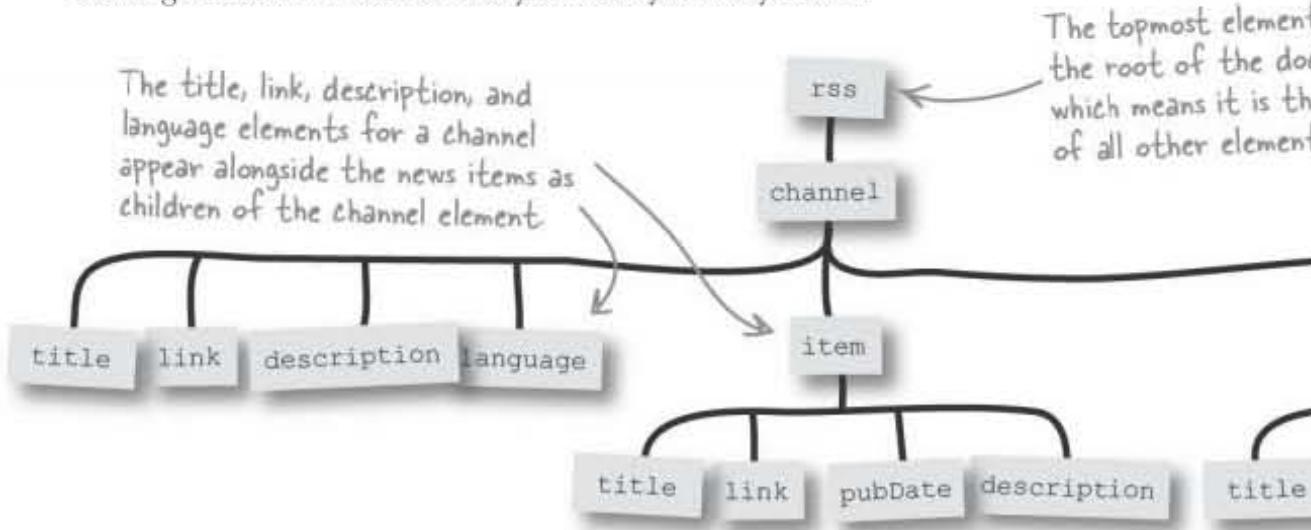
`<link>`
`<description>`
 This tag only applies to news items.
`<pubDate>`

`<item>`
 The `<title>`, `<link>`, `<pubDate>`, and `<description>` tags are used within `<item>` to describe a news item.

RSS

Visualizing XML

You already learned that XML code consists of tags, which are also sometimes referred to as **elements**, that form **parent-child relationships** within the context of a complete XML document. It is very helpful to be able to visualize this parent-child relationship as you work with XML code. As an example, the RSS document on the facing page can be visualized as a hierarchy of elements, kind of like a family tree for newsfeed data, with parent elements at the top fanning out to child elements as you work your way down.



Below is a brand-new alien abduction report that has been added to the alien database. Write the XML code for an RSS <item> tag for this abduction to adhere to the RSS format for newsfeeds.

aliens_abduction

abduction_id	first_name	last_name	when_it_happened	how_long	how_many	alien_description
14	Shill	Watner	2008-07-05	2 hours	don't know	There was a bright light in the sky...

exercise solution

Below is a brand-new alien abduction report that has been added to the alien database. Write the XML code for an RSS `<item>` tag for this abduction report to adhere to the RSS format for newsfeeds.

aliens_abduction

abduction_id	first_name	last_name	when_it_happened	how_long	how_many	alien_description	what
...							
14	Shill	Watner	2008-07-05	2 hours	don't know	There was a bright light in the sky...	The me ga

`<item>` The `<item>` tag encloses the news item.

`<title>Shill Watner - There was a bright light in the sky...</title>`

`<link>http://www.aliensabductedme.com/index.php?abduction_id=14</link>`

`<pubDate>Sat, 05 Jul 2008 00:00:00 EST</pubDate>`

`<description>They beamed me toward a gas station...</description>`

`</item>`

The `<title>` and `<desc` the detail

The must with can't <PU

there are no Dumb Questions

Q: Is XML case-sensitive?

A: Yes, the XML language is case-sensitive, so it matters whether text is uppercase or lowercase when specifying XML tags and attributes. A good example is the RSS `<pubDate>` tag, which must appear in mixed case with the capital D. Most XML tags are either all lowercase or mixed-case.

Q: What about whitespace? How does it fit into XML?

A: First of all, whitespace in XML consists of carriage returns (\r), newlines (\n), tabs (\t), and spaces (' '). The majority of whitespace in most XML documents is purely for aesthetic formatting purposes, such as indenting child tags. This "insignificant" whitespace is typically ignored by applications that process XML data, such as RSS news readers. However, whitespace that appears inside of a tag is considered "significant," and is usually rendered exactly as it appears. This is what allows things like poems that have meaningful spacing to be accurately represented in XML.

Q: Can an RSS feed contain images?

A: Yes. Just keep in mind that not every RSS reader is able to display images. Also, in RSS 2.0 you can add images to a channel, not individual news items. You can add an image to a channel using the `<image>` tag, which is a child of the `<channel>` tag. Here's an example:

```
<image>
<url>http://www.aliensabductedme.com/images/logo.png</url>
<title>My dog Fang</title>
<link>http://www.aliensabductedme.com/</link>
</image>
```

It is technically possible to include an image in an RSS 2.0 feed; the trick is to use the HTML `` tag instead of the `<image>` tag of the item. While this is possible, it requires some work to use the HTML tag using XML entities, and in many cases it goes against the premise of an RSS item being pure text data.



RSS Revealed

This week's interview:

What makes a newsman tick

Head First: So I hear that when people are looking for news on the Web, they turn to you. Is that true?

RSS: I suppose it depends on what you consider “news.” I’m mainly about packaging up information into a format that is readily accessible to newsreaders. Now whether that content is really news or not... that’s something I can’t control. That’s for people to decide.

Head First: Ah, so by “newsreaders,” you mean individual people, right?

RSS: No, I mean software tools that understand what I am and how I represent data. For example, a lot of email programs support me, which means that you can subscribe to a newsfeed and receive updates almost like receiving email messages.

Head First: Interesting. So then how are you different than email?

RSS: Oh, I’m a lot different than email. For one thing, email messages are sent from one person to another, and are usually part of a two-way dialog. So you can respond to an email message, get a response back, etc. I only communicate one way, from a web site to an individual.

Head First: How does that make it a one-way communication?

RSS: Well, when a person elects to receive a newsfeed by subscribing to it in their newsreader software, they’re basically saying they want to know about new content that is posted on a given web site. When new content actually gets posted, I make sure it gets represented in such a way that the news reader software knows about it and shows it to the person. But they aren’t given an opportunity to reply to a news item, which is why it’s a one-way communication from a web site to an individual.

Head First: I see. So what are you exactly?

RSS: I’m really just a data format, an agreed-upon way to store content so that it can be recognized and consumed by news readers. Use me to store data, and newsreaders will be able to access it as a newsfeed.

Head First: OK, so how are you different from XML?

RSS: Well, we’re both text data formats, but I’m ultimately based on XML, which means I have elements and attributes in describing data. Even though I’m designed specifically to be processed by browsers, I’m designed to be processed by newsreaders. You could say that we’re both built on the same data.

Head First: But I’ve seen where some newsreaders display newsfeeds. How does that work?

RSS: Good question. As it turns out, newsreaders can include built-in newsreaders, so they can do both in one. But when you view a newsfeed in your browser, you’re looking at something completely different than an HTML web page.

Head First: But most newsfeeds I see are just plain pages, correct?

RSS: That’s right. So I work hand-in-hand with my XML brother to provide better access to web content. For example, if you use me to learn about new content on a web site, you can click on a link to go visit a web site directly. Then if you want to find out more about, say, the latest news item, you can click on its title to go to the actual page. That’s why each newsitem has a link to its full story.

Head First: So you’re sort of a plugin for XML.

RSS: Yeah, kinda like that. But remember, I’m not limited to you, you don’t have to come to me. I’m also used by people who really like about me—I keep them updated on my favorite web sites to keep tabs on new content.

Head First: I see. That is indeed a good way of clarifying your role on the Web.

RSS: Hey, glad to do it. Stay class,

generate rss with php

Dynamically generate an RSS feed

Understanding the RSS data format is all fine and good, but Owen still needs a newsfeed to take alien abduction reports to the people. It's time to break out PHP and dynamically generate a newsfeed full of alien abduction data that has been plucked from Owen's MySQL database. Fortunately, this can be accomplished by following a series of steps:

The re
isn't s
it is a

1 Set the content type of the document to XML.

We have to set the content type of the RSS document to XML by using a header.

```
<?php header('Content-Type: text/xml'); ?>
```

2 Generate the XML directive to indicate that this is an XML document.

```
<?php echo '<?xml version="1.0" encoding="utf-8"?>'; ?>
```

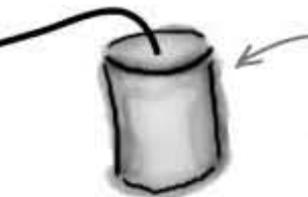
3 Generate the static RSS code that doesn't come from the database such as the <rss> tag and the channel information.

```
<rss version="2.0">
  <channel>
    <title>...
    <link>...
    <description>...
    <language>...
```

This code isn't affected by the database - it's always the same for this newsfeed.

4 Query the aliens_abduction database for alien abduction data

abduction_id	first_name	last_name
when_it_happened	alien_description	
	what_they_did	

**5 Loop through the data generating RSS code for each news item**

```
<item>
  <title>...
  <link>...
  <pubDate>...
  <description>...
</item>
```

This code contains data extracted from the database, and therefore must be carefully generated.

6 Generate the static RSS code required to finish up the document including closing </channel> and </rss> tags.

```
</channel>
</rss>
```



PHP & MySQL Magnets

Owen's Aliens Abducted Me RSS newsfeed script (`newsfeed.php`) is missing some magnets. Carefully choose the appropriate magnets to finish the code and dynamically generate an RSS feed.

```

1 <?php header('Content-Type: text/xml'); ?>
2 <?php echo '<?xml version="1.0" encoding="utf-8"?>'; ?>
<rss version="2.0">

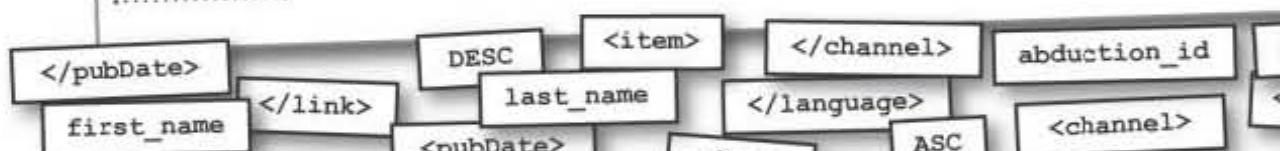
.....
<title>Aliens Abducted Me - Newsfeed</title>
      http://aliensabductedme.com/ .....
3 <description>Alien abduction reports from around the world courtesy of Owen
      and his abducted dog Fang.</description>
      en-us.....
.....
<?php
    require_once('connectvars.php');

    // Connect to the database
    $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

    // Retrieve the alien sighting data from MySQL
    $query = "SELECT abduction_id, first_name, last_name,
              DATE_FORMAT(when_it_happened, '%a, %d %b %Y %T') AS when_it_happened_rfc,
              alien_description, what_they_did
            FROM aliens_abduction
            ORDER BY when_it_happened .....";
    $data = mysqli_query($dbc, $query);

    // Loop through the array of alien sighting data, formatting it as RSS
    while ($row = mysqli_fetch_array($data)) {
        // Display each row as an RSS item
        echo '.....';
        echo ' <title>' . $row['first_name'] . ' ' . $row['last_name'] . ' - ' .
              substr($row['alien_description'], 0, 32) . '...</title>';
        echo ' <link>http://www.aliensabductedme.com/index.php?abduction_id=' .
              $row['.....'] . '</link>';
        echo ' .....' . $row['when_it_happened_rfc'] . ' ' . date('T') . ' ';
        echo ' <description>' . $row['what_they_did'] . '</description>';
        echo '</item>';
    }
?>
</channel>
.....

```



php & mysql & xml magnets solution



& XML!

PHP & MySQL Magnets Solution

Owen's Aliens Abducted Me RSS newsfeed script (`newsfeed.php`) is missing some magnets. Carefully choose the appropriate magnets to finish the code and dynamically generate an RSS feed.

```

1 <?php header('Content-Type: text/xml'); ?> ←
2 <?php echo '<?xml version="1.0" encoding="utf-8"?>'; ?>
<rss version="2.0">

    <channel> . . .
        <title>Aliens Abducted Me - Newsfeed</title>
        <link> http://aliensabductedme.com/ .. </link>
3        <description>Alien abduction reports from around the world courtesy of Owen
and his abducted dog Fang.</description>
        <language> en-us ... </language>

<?php
require_once('connectvars.php');

// Connect to the database
$dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

// Retrieve the alien sighting data from MySQL
$query = "SELECT abduction_id, first_name, last_name,
DATE_FORMAT(when_it_happened, '%a, %d %b %Y %T') AS when_it_happened_rfc,
alien_description, what_they_did
FROM aliens_abduction";

4        "ORDER BY when_it_happened . DESC . ";

$data = mysqli_query($dbc, $query);

// Loop through the array of alien sighting data, formatting it as RSS
while ($row = mysqli_fetch_array($data)) {
    // Display each row as an RSS item
    echo ' . <item> . ';
    echo ' <title>' . $row['first_name'] . ' ' . $row['last_name'] . ' - ' .
substr($row['alien_description'], 0, 32) . '...</title>';
    echo ' <link>http://www.aliensabductedme.com/index.php?abduction_id=' .
$row[' . abduction_id . '] . '</link>';
    echo ' <pubDate> ' . $row['when_it_happened_rfc'] . ' ' . date('T') . ' ';
    echo ' <description>' . $row['what_they_did'] . '</description>';
    echo '</item>';
}
?>

    </channel>
</rss> .

```

first_name

last_name

</channel>

ASC

<



— Test Drive —

Add the RSS Newsfeed script to Aliens Abducted Me.

Create a new text file named newsfeed.php, and enter the code for Owen's Newsfeed script from the Magnets exercise a few pages back (or download it from the Head First Labs site at www.headfirstlabs.com/books/hfpi).

Upload the script to your web server, and then open it in a newsreader. Most web browsers support RSS feeds, and some email clients allow you to view newsfeeds, so you can try those if you don't have a stand-alone newsreader application. The Newsfeed script should contain alien abductions pulled straight from the Aliens Abducted Me database.

Aliens Abducted Me - Newsfeed

Meinhold Ressner - They were in a ship the size of ... Aug 10, 1:00 AM
Carried me to the top of a mountain and dropped me off. [Read more...](#)

Mickey Mikens - Huge heads, skinny arms and legs... Jul 11, 1:00 AM
Read my mind. [Read more...](#)

Shill Watner - There was a bright light in the ... Jul 8, 1:00 AM
They beamed me toward a gas station in the desert. [Read more...](#)

Bellita Chevy - Clumsy little buggers, had no rh... Jul 21, 1:00 AM
Tried to get me to play bad music. [Read more...](#)

Sally Jones - green with six tentacles... May 11, 1:00 AM
We just talked and played with a dog. [Read more...](#)

...er - It was a big non-recyclable shin... Jul 12, '98, 1:00 AM
...om the sky and snatched me up with no warning. [Read more...](#)

They looked like donkeys made ou... Sep 14, '91, 1:00 AM
...ing a baked potatoe when "Zwoosh!", this beam of light took me away. [Read more...](#)

They were pasty and pandering, a... Jun 21, '00, 1:00 AM
...course, then they probed me. [Read more...](#)

Source: Aliens Abducted Me
Actions: Update Now, Mail Link To This, Subscribe in My, Add Bookmark

A thought bubble originates from the bottom of the woman's head, pointing towards the newsfeed interface. The text inside the bubble reads: "The newsfeed looks great, but how do site visitors find out about it?" An arrow points from the right side of the newsfeed interface towards the "Actions" sidebar.



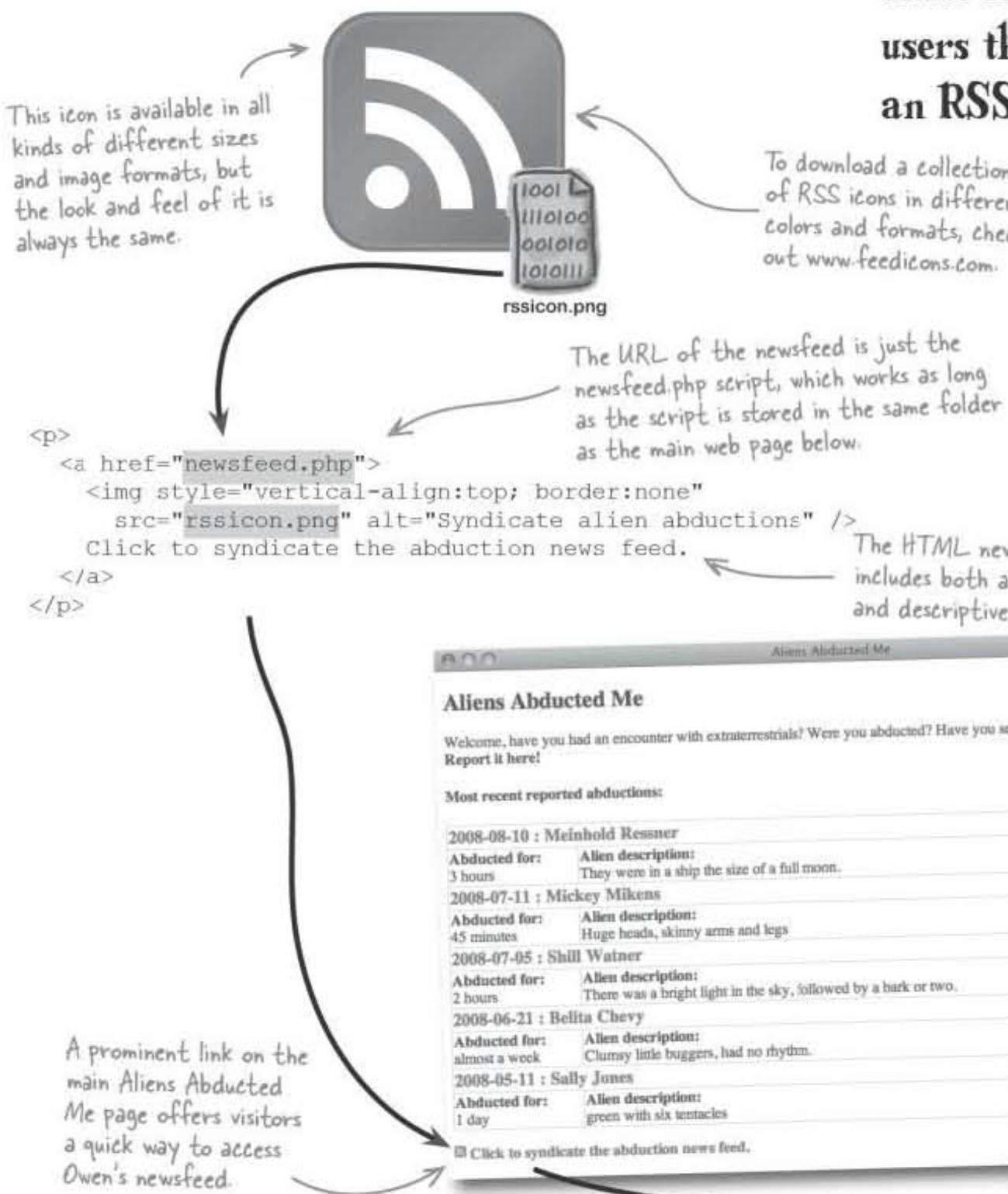
Just provide a link to it from the

Don't forget that newsfeed.php is nothing more than an ordinary PHP script—the only difference between it and most of the other scripts throughout the book is that it generates an RSS feed, which is an XML-based XML document. But you still access it just like any other PHP script—just specify the name of the script in the URL. This is a great way to share this URL with people who don't have a newsreader, with very little effort by **providing a syndication feed**. In fact, just add a link to the newsfeed.php script on Owen's server.

providing an rss link

Link to the RSS feed

It's important to provide a prominent link to the newsfeed for a web site because a lot of users will appreciate that you offer such a service. To help aid users in quickly finding an RSS feed for a given site, there is a standard icon you can use to visually call out the feed. We can use this icon to build a newsfeed link at the bottom of Owen's home page (`index.php`).





Test DRIVE

Add the newsfeed link to the Aliens Abducted Me home page

Modify the index.php script for Aliens Abducted Me to display the news feed near the bottom of the page. Also download the rssicon.png image code from this chapter from the Head First Labs site at www.headfirstlabs.com/books/hfphp.

Upload the index.php script and rssicon.php image to your web server, open the script in a web browser. Click the new link to view the RSS news feed.

With all these abductions going on, I'm always on the lookout for aliens.

Thanks to RSS, new alien abduction reports are "pushed" to subscribers without them having to visit the Aliens Abducted Me web site directly.

Aliens Abducted Me - Newsfeed — RSS (8 messages; 7 unread)

Date Received	Subject
August 10, 2008	Mainhold Ressner - They were in a ship the size...
Today	Rick Nixon - They were pasty and pandering, a...
July 11, 2008	Mickey Mikens - Huge heads, skinny arms and...
July 5, 2008	Shill Watner - There was a bright light in the ...
June 21, 2008	Belta Chevy - Clumsy little buggers, had no rh...
May 11, 2008	Sally Jones - green with six tentacles...
July 12, 2000	Alf Nader - It was a big non-recyclable shin...
September 14, 1991	Don Quayle - They looked like donkeys made ou...

Aliens Abducted Me - Newsfeed
Mainhold Ressner - They were in a ship th...

Carried me to the top of a mountain and dropped me off.
[Read more...](#)

I wo...
sam...
You...

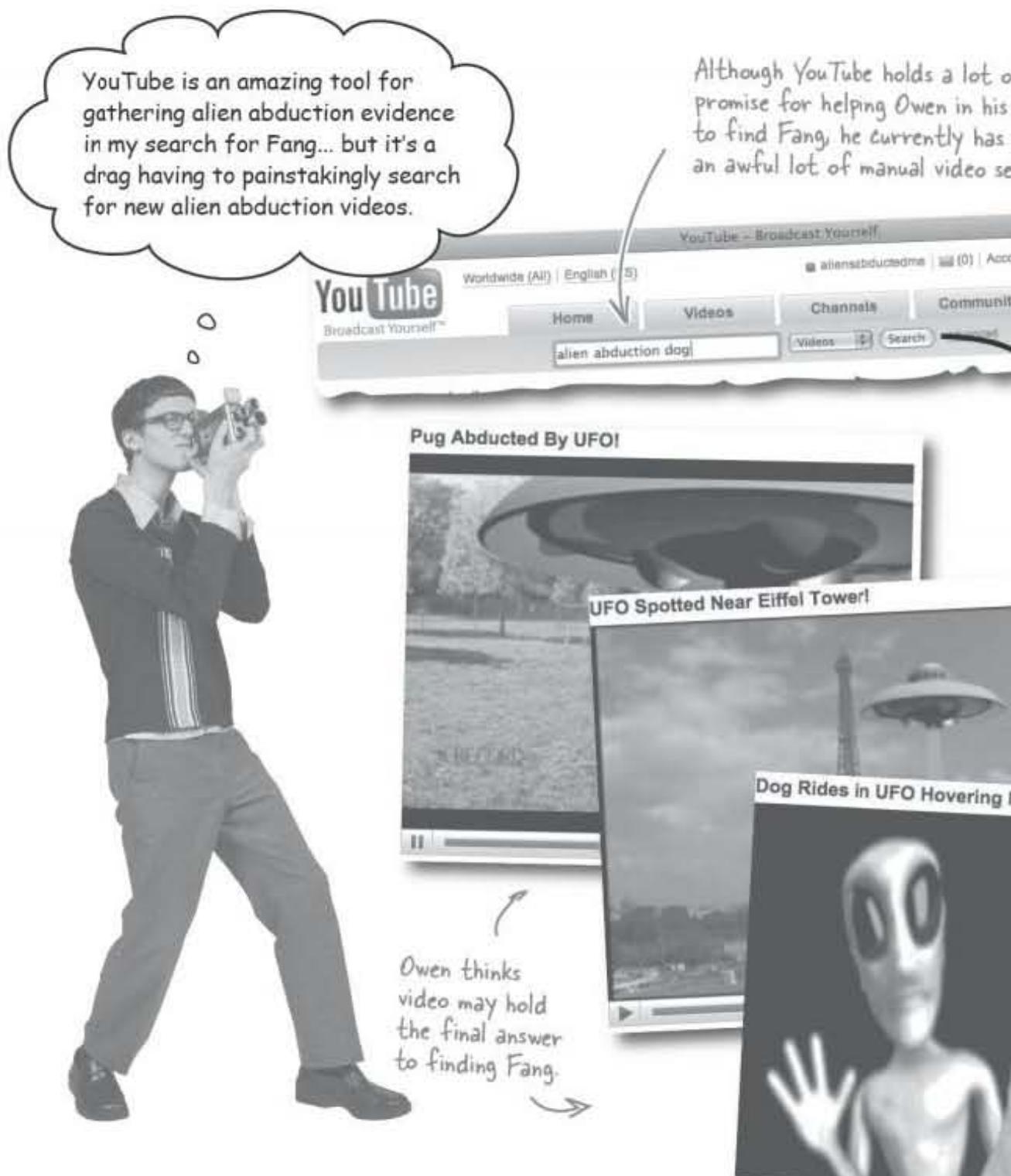
Chloe, an avid Aliens Abducted Me newsfeed reader, thinks she might have seen Fang in a YouTube video.



add youtube content to owen's site

video million A picture is worth a thousand words

After a newsfeed subscriber alerted Owen to a YouTube video with a dog in it that resembles Fang, Owen realized that he's going to have to use additional technology to expand his search for Fang. But how? If Owen could incorporate YouTube videos into Aliens Abducted Me, his users could all be on the lookout for Fang. Not only that, but he really needs to come up with a way to avoid constantly doing manual video searches on YouTube.





Try this!

- 1 Visit Owen's YouTube videos at www.youtube.com/user/aliensabductedm
- 2 Watch a few of the alien abduction videos that Owen has found.
Do you think the dog in the videos is Fang?



Wouldn't it be dreamy if I could see
videos directly on Aliens Abducted M
rather than having to search on YouT
If only there was a way I could just g
web page and have the search already
for me. But that's nothing but a drea

pulling content is different than pushing content

Pulling web content from others

The idea behind an RSS newsfeed is that it **pushes your content** to others so that they don't have to constantly visit your web site for new content. This is a great way to make it more convenient for people to keep tabs on your site, as Owen has found out. But there's another side to the web syndication coin, and it involves **pulling content from another site** to place on your site. So you become the consumer, and someone else acts as the content provider. In Owen's case of showing YouTube videos on his site, YouTube becomes the provider.

You
provi

Aliens Abducted Me is the consumer of videos.

The screenshot shows a web page titled "Aliens Abducted Me". The main content area displays a table of recent abduction reports:

Date	Abductee	Alien Description	Fang Spotted
2008-08-10	Meinhold Ressner	They were in a ship the size of a full moon.	No
2008-07-11	Mickey Mikens	Huge heads, skinny arms and legs	Yes
2008-07-05	Shill Watner	There was a bright light in the sky, followed by a bark or two.	Yes
2008-06-21	Belta Chevy	Clumsy little buggers, had no rhythm.	No
2008-05-11	Sally Jones	green with six tentacles	Yes

Below the table, there is a link to "Click to syndicate the abduction news feed." To the right of the table, there is a "Video thumbnail" section with the text "images go here!" and a small image of a dog.

The design of the Aliens Abducted Me home page will need to change slightly to make room for the video search results.

It's important to understand that Owen doesn't just want to embed a specific YouTube video or a link to a video. That's easy enough to accomplish by simply cutting and pasting HTML code from YouTube. He wants to actually **perform a search** on YouTube videos and display the results of that search. So Aliens Abducted Me needs to perform a real-time query on YouTube data, and then dynamically display the results. This allows Owen and his legion of helpful Fang searchers to keep up-to-the-minute tabs on alien abduction videos that have been posted to YouTube.

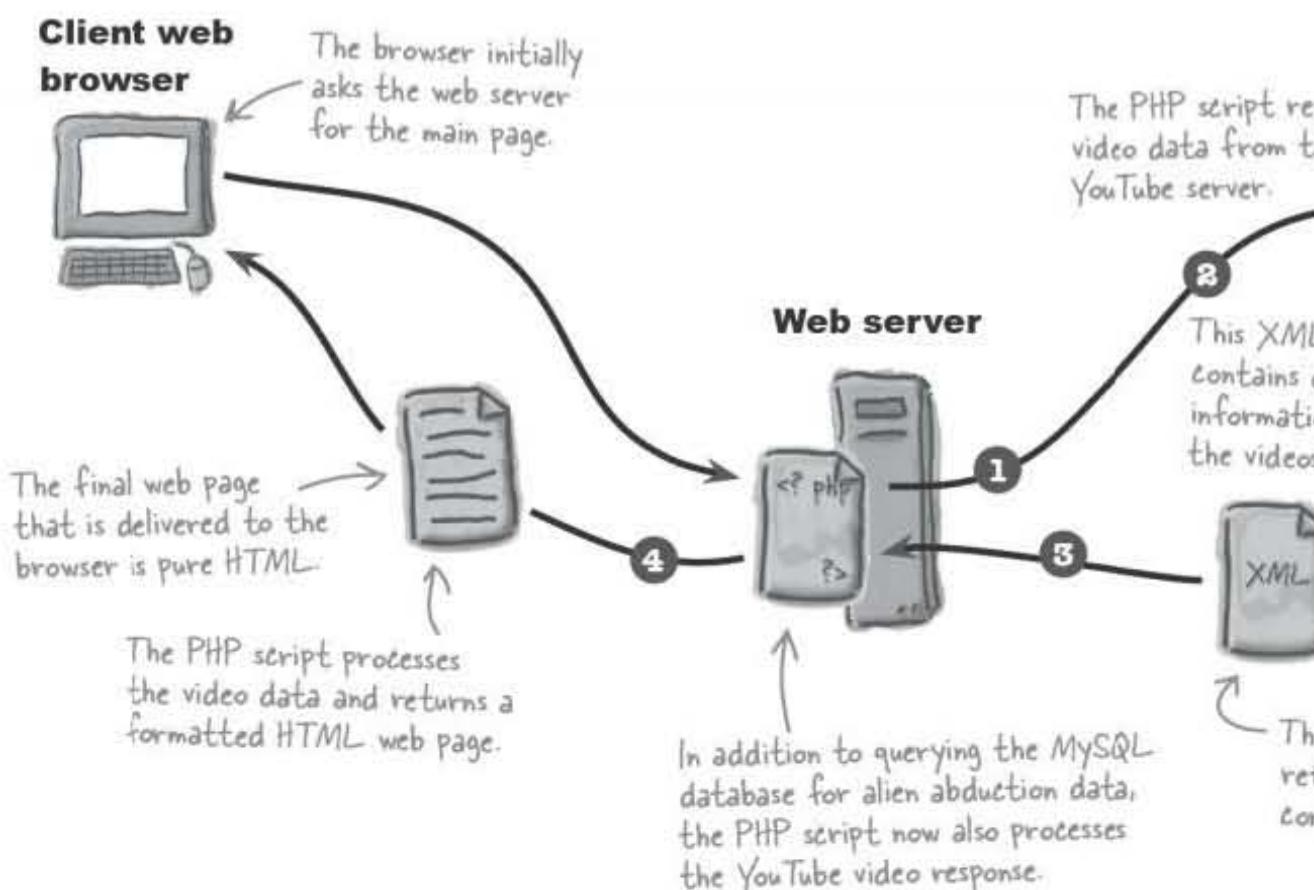
Video
result
abduc
retur
fed in

Syndicating YouTube videos

In order to source videos from YouTube, we must learn exactly how YouTube makes videos available for syndication. YouTube offers videos for syndication through a **request/response communication process** where you make a request for certain videos and then receive information about those videos in a response from YouTube's servers. You are responsible for both issuing a request in the format expected by YouTube and handling the response, which includes sifting through response data to get at the specific video data you need (video title, thumbnail image, link, etc.).

Following are the steps required to pull videos from YouTube and display them:

- 1 Build a request for YouTube videos. This request is often in the form of a URL.
- 2 Issue the video request to YouTube.
- 3 Receive YouTube's response data containing information about the video.
- 4 Process the response data and format it as HTML code.



introducing REST requests

Make a YouTube video request

Pulling videos from YouTube and incorporating them into your own web pages begins with a request. YouTube expects videos to be queried through the use of a **REST request**, which is a custom URL that leads to specific resources, such as YouTube video data. You construct a URL identifying the videos you want, and then YouTube returns information about them via an XML document.

The details of the URL for a YouTube request are determined by what videos you want to access. For example, you can request the favorite videos of a particular user. In Owen's case, the best approach is probably to perform a keyword search on all YouTube videos. The URL required for each of these types of video REST requests varies slightly, but the base of the URL always starts like this:

`http://gdata.youtube.com/feeds/api/`

This base URL is used for all YouTube REST requests.



Request videos by user

Requesting the favorite videos for a particular YouTube user involves adding onto the base URL, and also providing the user's name on YouTube.

The user name of user provides access to the user's favorite video.

`http://gdata.youtube.com/feeds/api/users/username/favorites`

To request the favorite videos for the user elmerpriestley, use the following URL:

`http://gdata.youtube.com/feeds/api/users/elmerpriestley/favorites`



Request videos with a keyword search

A more powerful and often more useful YouTube video request is to carry out a keyword search that is independent of users. You can use more than one keyword as long as you separate them by forward slashes at the end of the URL.

Multiples in a video them will

`http://gdata.youtube.com/feeds/api/videos/-/keyword1/keyword2/...`

The URL starts the same as requesting by user but here you use "videos" instead of "users".

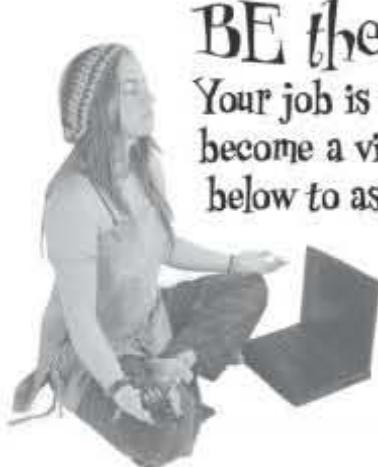
Don't forget the slashes and the hyphen!

To request the favorite videos for the keywords "elvis" and "impersonator," use the following URL:

`http://gdata.youtube.com/feeds/api/videos/-/elvis/impersonator`

The keywords are case-insensitive, so "elvis", "Elvis", and "eLvis" all give you the same result.

Here the "elvis" and "impersonator" used to se



BE the YouTube REST Request

Your job is to get inside the mind of YouTube and become a video REST request. Use the magnets below to assemble video REST requests for the following YouTube videos, and then try them out in your web browser.

All videos that match the keyword “Roswell”:

.....

All videos that match the keywords “alien” and “abduction”:

.....

All videos tagged as favorites for the user headfirstmork:

.....

All videos that match the keywords “ufo”, “sighting”, and “dog”:

.....

All videos tagged as favorites for the user aliensabductedme:



be the youtube REST request solution



BE the YouTube REST Request Solution

Your job is to get inside the mind of YouTube and become a video REST request. Use the magnets below to assemble video REST requests for the following YouTube videos, and then try them out in your web browser.

You may have used some of the magnets more than once.

The same base YouTube URL is used for all of the REST requests.

All videos that match the keyword “Roswell”:

`http://gdata.youtube.com/feeds/api/` `videos` `/` `-` `/` `Roswell`

The single `-` appears last

Each of the `/` appear at the are separated

All videos that match the keywords “alien” and “abduction”:

`http://gdata.youtube.com/feeds/api/` `videos` `/` `-` `/` `alien` `/` `abduc`

All videos tagged as favorites for the user `headfirstmork`:

`http://gdata.youtube.com/feeds/api/` `users` `/` `headfirstmork` `/` `favor`

The URL for a user's favorites requires the word "users" here instead of "videos".

All videos that match the keywords “ufo”, “sighting”, and “dog”:

`http://gdata.youtube.com/feeds/api/` `videos` `/` `-` `/` `ufo` `/` `sightin`

All videos tagged as favorites for the user `aliensabductedme`:

`http://gdata.youtube.com/feeds/api/` `users` `/` `aliensabductedme` `/` `fa`

Area 51

This magnet wasn't used... it's a conspiracy!

This is the name of the user whose favorite videos you want to

there are no
Dumb Questions

Q: How is REST different than, say, a GET request?

A: It's not. Any time you've used a **GET** request, such as simply requesting a web page, you're using **REST**. You can think of a normal web page as a **REST** resource in that it can be accessed via a URL, and **GET** is the **REST** "action" used to access the resource. Where **REST** gets more interesting is when it is used to build queries, such as YouTube video requests. In this case you're still dealing with **REST** requests but they are querying a database for data instead of simply requesting a static web page.

Q: Does the order of arguments matter when performing a YouTube keyword search?

A: Yes. The first keywords are given a higher precedence than later keywords, so make sure to list them in order of decreasing importance.

Q: When there are multiple matches for a video search, how does YouTube determine what videos to return?

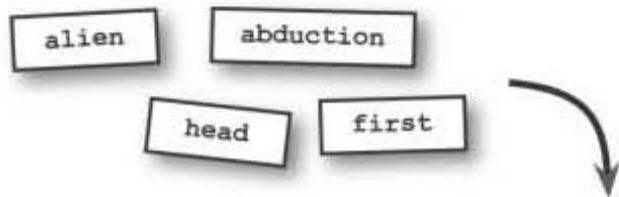
A: YouTube keyword video requests return videos based on search relevance, meaning that you will get the videos that best match the keywords, regardless of when the videos were posted to YouTube.

building a REST request

I'm ready to see some video result

Owen is ready to build a REST request

Since Owen's goal is to scour YouTube for alien abduction videos that might have Fang in them, a keyword search makes the most sense as the type of REST request to submit to YouTube. There are lots of different keyword combinations that could be used to search for possible Fang videos, but one in particular will help home in on videos related specifically to Fang:



<http://gdata.youtube.com/feeds/api/videos/-/alien/abduction/head/first>

While you probably wouldn't reference the title of a book series when carrying out a normal YouTube video search, it just so happens to be a good idea in this particular case. Let's just say it's a coincidence that a lot of alien abduction videos have been made by Head First fans! With a REST request URL in hand, Owen can scratch off Step 1 of the YouTube video syndication process.

The last two help to make the alien ab related to O

The first step is knocked out thanks to the YouTube request URL

- 1 Build a request for YouTube
- 2 Issue the video request to YouTube
- 3 Receive YouTube's response containing information about the video
- 4 Process the response data as HTML code.



Test DRIVE

Try out Owen's YouTube request URL.

Enter Owen's YouTube request URL in a web browser:

<http://gdata.youtube.com/feeds/api/videos/-/alien/abductedme>

What does the browser show? Try viewing the source of the page to take a look at the actual code returned by YouTube.

Video Title	Author	Timestamp	Description	Link
UFO Spotted Crashing Party at Graceland...	aliensabductedme	Today, 12:54 AM	In an unusual conspiratorial twist of fate, a UFO was spotted crashing a raucous party at Graceland, the home of the undisputed King of Rock 'n Roll.	Read more...
Aliens Turn Face of Sphinx Into a Dog!	aliensabductedme	Today, 12:45 AM	Watch as a UFO laser chisels the face of the Sphinx into a Pug.	Read more...
Dog Rides in UFO Hovering Near San...	aliensabductedme	Yesterday, 10:08 PM	This shocking video shows a pug taking a ride in a UFO as it flies around near the Golden Gate Bridge.	Read more...
UFO Spotted Near Eiffel Tower!	aliensabductedme	Yesterday, 10:00 PM	Check out this video of a UFO zapping cats in Paris.	Read more...
Pug Abducted By UFO!	aliensabductedme	Yesterday, 9:51 PM	Help! My dog was abducted by aliens.	Read more...

The web browser views the XML data returned by the YouTube response as a newsfeed, except in this case each item is actually a video.

making REST requests in a php script

Requesting videos from YouTube by typing a URL into a web browser is neat and all, but what does that have to do with PHP? Why can't we access the video results from a script?

The SimpleXML extension to the `simplexml_load_file()` function was added to PHP in version 5. So prior to version 5, you didn't have built-in support for this.

We can, we just need a PHP function that allows us to submit a REST request and receive a response.

The built-in PHP function `simplexml_load_file()` is designed to handle REST requests that result in XML responses, such as YouTube's. It takes an XML document object, which we can then use to drill down into the XML and extract whatever specific information is needed. So how do we make Owen's YouTube video request? Check out this code, which defines a constant to hold a YouTube URL, and then issues a REST request using the `simplexml_load_file()` function:

```
define('YOUTUBE_URL', 'http://gdata.youtube.com/feeds/api/videos/-/alien');
$xml = simplexml_load_file(YOUTUBE_URL);
```

Although not strictly necessary, it's generally a good idea to store static URLs in constants so that you know where to change them if the need ever arises.

- ➊ Build a request for YouTube videos.
- ➋ Issue the video request to YouTube.
- ➌ Receive YouTube's response data containing information about the videos.
- ➍ Process the response data and format it as HTML code.

These two steps



Don't sweat it!
You know what a PHP object is, especially in

A PHP object

that allows data to be packaged together in a single construct. All you need to know is that it's much easier to process data by using objects. You'll learn more about this in just a bit.

```





























































































































































































```

Awesome... an even bigger
What on earth do we do
messy XML data? There
PHP script can make ser...

**Oh, but there is! The XML code returned by YouTube
isn't really as messy as it looks... you just have to
know where to look.**

the request returns xml

YouTube speaks XML

The video response from YouTube isn't exactly a DVD packaged up in a shiny box and delivered to your front door. No, it's an XML document containing detailed information about the videos you requested, not the videos themselves.

```

<?xml version='1.0' encoding='UTF-8'?>
<feed xmlns='http://www.w3.org/2005/Atom'
      xmlns:openSearch='http://a9.com/-/spec/opensearchrss/1.0/'
      xmlns:gml='http://www.opengis.net/gml'
      xmlns:georss='http://www.georss.org/georss'
      xmlns:media='http://search.yahoo.com/mrss/'
      xmlns:batch='http://schemas.google.com/gdata/batch'
      xmlns:yt='http://gdata.youtube.com/schemas/2007'
      xmlns:gd='http://schemas.google.com/g/2005'>
  <id>http://gdata.youtube.com/feeds/api/users/aliensabductedme/favorites</id>
  <updated>2008-07-25T03:22:37.001Z</updated>
  <category scheme='http://schemas.google.com/g/2005#kind'
            term='http://gdata.youtube.com/schemas/2007#video'/>
  <title type='text'>Favorites of aliensabductedme</title>
  ...
  <entry>
    <id>http://gdata.youtube.com/feeds/api/videos/_6UiBqf0vtA</id>
    <published>2006-06-20T07:49:05.000-07:00</published>
    ...
    <media:group>
      <media:title type='plain'>UFO Sighting in Yosemite Park near Area 51</media:title>
      <media:description type='plain'>I went on a trip to Yosemite Park in 2002. Yosemite  
close to the border between California and Nevada, and close to Area 51...</media:description>
      <media:keywords>51, alien, aliens, area, ca, california, nevada, sighting, sightin  
ufo</media:keywords>
      <yt:duration seconds='50' />
      <media:category label='Travel & Events'  
          scheme='http://gdata.youtube.com/schemas/2007/categories.cat'>Travel</media:category>
      <media:content url='http://www.youtube.com/v/_6UiBqf0vtA' type='application/x-  
medium='video' isDefault='true' expression='full' duration='50' yt:format='5' />
      <media:content url='rtsp://rtsp2.youtube.com/ChOLENy73wIaEQnQvvSnbiK1_xMYDSANB  
type='video/3gpp' medium='video' expression='full' duration='50' yt:format='1' />
      <media:content url='rtsp://rtsp2.youtube.com/ChOLENy73wIaEQnQvvSnbiK1_xYESARF  
type='video/3gpp' medium='video' expression='full' duration='50' yt:format='6' />
      <media:player url='http://www.youtube.com/watch?v=_6UiBqf0vtA' />
      <media:thumbnail url='http://img.youtube.com/vi/_6UiBqf0vtA/2.jpg' height='97'  
tine='00:00:25' />
      <media:thumbnail url='http://img.youtube.com/vi/_6UiBqf0vtA/1.jpg' height='97'  
tine='00:00:12.500' />
      <media:thumbnail url='http://img.youtube.com/vi/_6UiBqf0vtA/3.jpg' height='97'  
tine='00:00:37.500' />
      <media:thumbnail url='http://img.youtube.com/vi/_6UiBqf0vtA/0.jpg' height='240'  
tine='00:00:25' />
    </media:group>
    <yt:statistics viewCount='2478159' favoriteCount='1897' />
    <gd:rating min='1' max='5' numRaters='1602' average='4.17' />
    <gd:comments>
      <gd:feedLink href='http://gdata.youtube.com/feeds/api/videos/_6UiBqf0vtA/comme  
countHint='4426' />
    </gd:comments>
  </entry>
  <entry>
    <id>http://gdata.youtube.com/feeds/api/videos/XpNd-Dg6_zQ</id>
    <published>2006-11-19T16:44:43.000-08:00</published>
    ...
  </entry>
</feed>
```

This `<entry>` tag starts another video within the XML response data.



Sharpen your pencil

Study the highlighted XML code for the YouTube response page and answer the following questions. You might learn more about YouTube's video XML format than you thought.

1. What is the title of the video?
 2. What are three keywords associated with the video?
 3. How long is the video, in seconds?
 4. To what YouTube video category does the video belong?
 5. How many times has the video been viewed?
 6. What average rating have users given the video?

sharpen your pencil solution

Sharpen your pencil Solution

Study the highlighted XML code for the YouTube response and answer the following questions. You might just know more about YouTube's video XML format than you thought at first!

```
<media:title type='plain'>UFO Sighting in Yosemite Park near Area 51</media:title>
```

1. What is the title of the video? ...UFO Sighting in Yosemite Park near Area 51

```
<media:keywords>51 alien, aliens area, ca, california, nevada, sight  
ufo</media:keywords>
```

2. What are three keywords associated with the video? ...51, aliens, n

3. How long is the video, in seconds? ...50.....

XML encodes some characters using special codes, such as &, which represents an ampersand (&).

```
<media:category label='Travel & Events'  
scheme='http://gdata.youtube.com/schemas/2007/categories.cat'>Travel</media:category>
```

4. To what YouTube video category does the video belong? ...Travel...

```
<yt:statistics viewCount='2478159' favoriteCount='1897' />
```

5. How many times has the video been viewed? ...2478159....

Wow
view

```
<gd:rating min='1' max='5' numRaters='1602' average='4.17' />
```

6. What average rating have users given the video? ...4.17....

average



Hmm, I'm a little confused with those XML tags that have two names separated by a colon. Is that somehow a way to organize tags? And what about the weird & code in the video category?

syndic

The unusual XML code uses namespace which help organize tags and encode special characters.

When you see an XML tag that has two names separated by a colon, like `<media:title type='plain'>`, you're looking at a **namespace**, which is a way of organizing tags into a logical group. The purpose of namespaces is to keep different XML vocabularies from clashing when multiple XML vocabularies are used in the same document. As an example, consider the following two XML documents:

```
<title type='text'>Favorites of aliensabotage</title>
```

```
<media:title type='plain'>UFO Sighting in Yosemite Park near Area 51</title>
```

Namespaces are named groups of XML tags, while entities are used to encode special characters within XML documents.

It may seem odd that a Yahoo! namespace appears in YouTube XML code – it just means that YouTube relies partly on an XML data format created by Yahoo!

Without the media namespace in the second `<title>` tag, it would be impossible to tell the two tags apart if they appeared in the same XML document. Just as a namespace can serve as a surname for tags—it helps keep an XML vocabulary from clashing with other vocabularies. It's like giving “first names” from clashing by hanging a “last name” on related tags. In fact, the XML response code uses several different namespaces, which means it can mix and match different XML languages at once—namespaces allow us to do that.

To ensure uniqueness, an XML namespace is always associated with a specific URL. For example, the media namespace used in YouTube XML data is associated with the URL `http://search.yahoo.com/mrss/`. You can see this in the XML code for the `<feed>` tag like this:

```
xmlns:media='http://search.yahoo.com/mrss/'
```

The other strange thing in the YouTube XML code is ɪmp;#x27; and ɪmp;#x27;. These are XML entities for representing the ampersand character (&). This is an XML way of referencing a special character, such as &, <, or >, all of which have a special meaning within XML code. Following are the five predefined XML entities that you will likely encounter as you delve deeper into XML code:

`&` = &

`<` = <

`>` = >

`"` = "

anatomy of a youtube xml response

Deconstruct a YouTube XML response

Once you get to know the structure of a YouTube response, extracting the video data you need is pretty straightforward. In addition to understanding what tags and attributes store what data, it's also important to understand how the tags relate to one another. If you recall from earlier in the chapter when analyzing an RSS feed, an XML document can be viewed as a hierarchy of elements. The same is true for the XML data returned in a YouTube video response.

```

<entry>
  <id>http://gdata.youtube.com/feeds/api/videos/_6Uibqf0vtA</id>
  <published>2006-06-20T07:49:05.000-07:00</published>
  ...
  <media:group>
    <media:title type='plain'>UFO Sighting in Yosemite Park near Area 51</media:title>
    <media:description type='plain'>I went on a trip to Yosemite Park in 2002. Yosemite is close to the border between California and Nevada, and close to Area 51...</media:description>
    <media:keywords>51, alien, aliens, area, ca, california, nevada, sighting, sight, ufo</media:keywords>
    <yt:duration seconds='50'/>
    <media:category label='Travel & Events' scheme='http://gdata.youtube.com/schemas/2007/categories.cat'>Travel</media:category>
    <media:content url='http://www.youtube.com/v/_6Uibqf0vtA' type='application/x-shockwave-flash' medium='video' isDefault='true' expression='full' duration='50' yt:format='5' />
    <media:content url='rtsp://rtsp2.youtube.com/ChoLENY73WIaEQnQvvSnbik1_xMYDS' type='video/3gpp' medium='video' expression='full' duration='50' yt:format='5' />
    <media:content url='rtsp://rtsp2.youtube.com/ChoLENY73WIaEQnQvvSnbik1_xYES' type='video/3gpp' medium='video' expression='full' duration='50' yt:format='5' />
    <media:player url='http://www.youtube.com/watch?v=_6Uibqf0vtA' />
    <media:thumbnail url='http://img.youtube.com/vi/_6Uibqf0vtA/2.jpg' height='90' width='160' time='00:00:25' />
    <media:thumbnail url='http://img.youtube.com/vi/_6Uibqf0vtA/1.jpg' height='90' width='160' time='00:00:12.500' />
    <media:thumbnail url='http://img.youtube.com/vi/_6Uibqf0vtA/3.jpg' height='90' width='160' time='00:00:37.500' />
    <media:thumbnail url='http://img.youtube.com/vi/_6Uibqf0vtA/0.jpg' height='240' width='320' time='00:00:25' />
  </media:group>
  <yt:statistics viewCount='2478159' favoriteCount='1897' />
  <gd:rating min='1' max='5' numRaters='1602' average='4.17' />
  <gd:comments>
    <gd:feedLink href='http://gdata.youtube.com/feeds/api/videos/_6Uibqf0vtA/comments' countHint='4426' />
  </gd:comments>
</entry>
```

The "gd" namespace stands for Google Data, and includes tags defined by Google for representing various kinds of data - YouTube is part of Google.

The average user rating of the video.

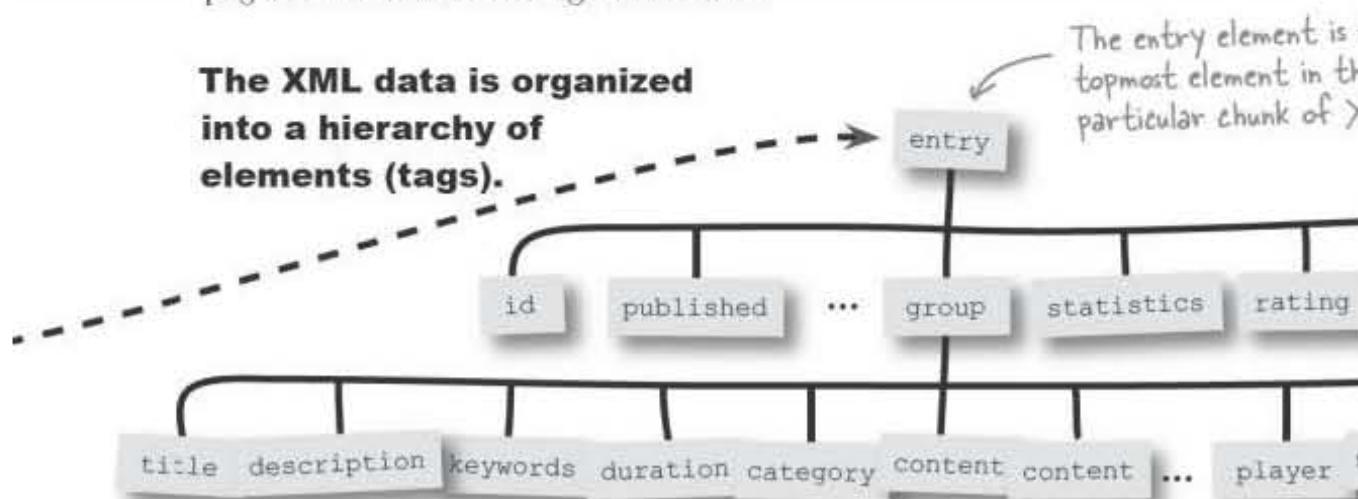
A thumbnail for the video.

The title of the video.

One important clue toward understanding the video data buried in this XML code is the different namespaces being used. The media namespace accompanies most of the tags specifically related to video data, while the `yt` namespace is used solely with the `<statistics>` tag. Finally, comments are enclosed within the `<comments>` tag, which falls under the `gd` namespace. These namespaces will matter a great deal when you begin writing PHP code to find specific tags and their data.

Visualize the XML video data

Earlier in the chapter when working with RSS code, it was revealed that an XML document can be visualized as a hierarchy of elements (tags) that have a parent-child relationship. This relationship becomes increasingly important as you begin to process XML code and access data stored within it. In fact, it can be an invaluable skill to be able to look at an XML document and immediately visualize the relationship between the elements. Just remember that any element enclosed within another element is a child, and the enclosing element is its parent. Working through the XML code for the YouTube video on the facing page results in the following visualization.



The significance of this hierarchy of elements is that you can navigate from any element to another by tracing its path from the top of the hierarchy. So, for example, if you wanted to obtain the title of the video, you could trace its path like this:

Navigating to an element in an XML document involves following the path from parent to child.

there are no
Dumb Questions

Q: Why do I even need to worry about namespaces?

A: Because XML code generated by others often involves namespaces, which affects how you access XML elements programmatically. As you're about to find out, the namespace associated with an element directly affects how you find the element when writing PHP code that processes XML data. So the namespace must be factored into code that is attempting to grab the data for a given element.

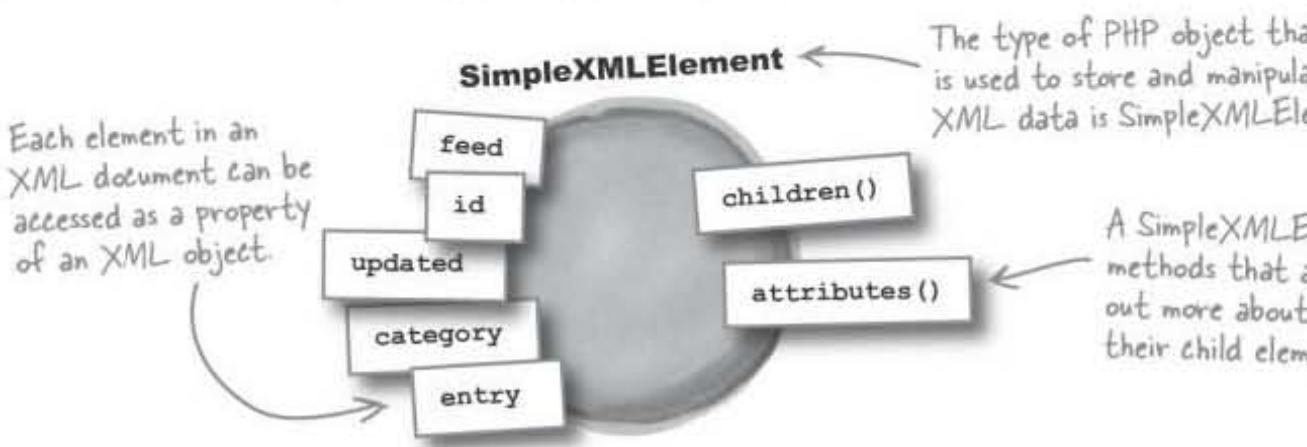
Q: How do I know if a

A: Although it's possible for namespaces not to explicitly appear in the code, they will always appear in the namespace right there in the XML code. For example, in <media:title> instead of <title>, the "media:" part to the left of the colon is always the namespace prefix.

all about php objects

Access XML data with objects

There are lots of different ways to work with XML data with PHP, and one of the best involves objects. An **object** is a special PHP data type that combines data and functions into a single construct. But what does that have to do with XML? The entire hierarchy of elements in an XML document is contained within a single variable, an object. You can then use the object to drill down into the data and access individual elements. Objects also have **methods**, which are functions that are tied to an object, and let us further manipulate the object's data. For an object that contains XML data, methods let us access the collection of child elements for an element, as well as its attributes.



You've already seen how to create this XML object for Owen's alien abduction YouTube keyword search:

```
define('YOUTUBE_URL', 'http://gdata.youtube.com/feeds/api/videos/-/alien');
$xml = simplexml_load_file(YOUTUBE_URL);
```

This code results in a variable named \$xml that contains all of the XML YouTube video data packaged into a PHP object. To access the data you use object **properties**, which are individual pieces of data stored within an object. Each property corresponds to an XML element. Take a look at the following example, which accesses all of the entry elements in the document:

```
$entries = $xml->entry;
```

The `->` operator lets you access a property within an object.

By specifying the name of the element (`entry`), you can grab all of the elements that are in the XML data.

This code accesses all the entry elements in the XML data using a property. Since there are multiple entry elements in the data, the \$entries variable stores an array of objects that you can use to access individual video entries. And since we're now dealing with an array, each video `<entry>` tag can be accessed by indexing the array. For example, the first `<entry>` tag in the document is the first item in the array, the second tag is the second item, etc.

Object
PHP
combin
function

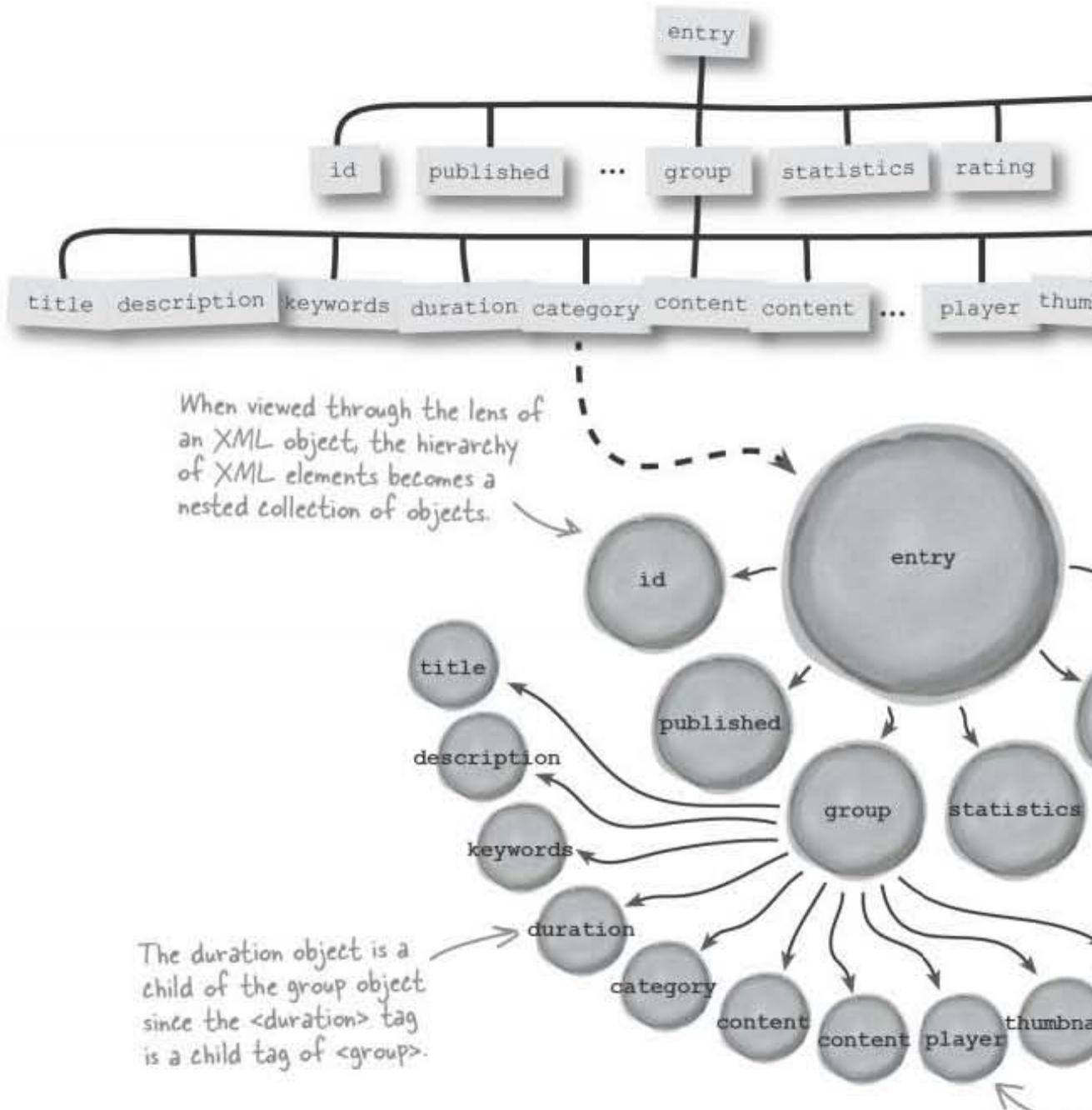
This function
object of type
containing all
the YouTube

All of the vid
stored in the



From XML elements to PHP objects

When it comes to XML data and PHP objects, you're really dealing with a **collection** of objects. Remember that stuff about visualizing an XML document as a hierarchy of elements? Well, that same hierarchy is realized as a **collection of objects** in PHP. Take a look:



This element hierarchy/object collection stuff forms the basis of understanding how to dig through XML data in PHP. With the relationship between individual pieces of XML data in mind, it becomes possible to write code that navigates through the data. Then we can isolate the content stored in a particular tag or attribute down deep in an XML document.

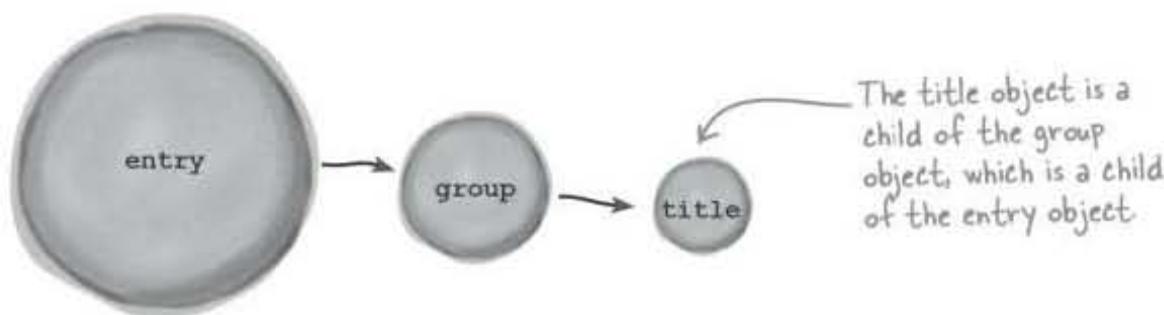
accessing object data

Drill into XML data with objects

Getting back to Owen, our goal is to pull out a few pieces of information for videos that are returned as part of the XML YouTube response. We know how to retrieve the XML data into a PHP object using the `simplexml_load_file()` function, but most of the interesting data is found down deeper in this data. How do we navigate through the collection of objects? The answer is the `->` operator, which is used to reference a property or method of an object. In the case of an XML object, the `->` operator accesses each child object. So this code displays the title of a video entry stored in a variable named `$entry`:

`echo $entry->group->title;` Here the `->` operator is used to drill down through nested child objects to access the title object.

This code relies heavily on the relationship between the `title`, `group`, and `entry` objects, which form a parent-child relationship from one to the next.

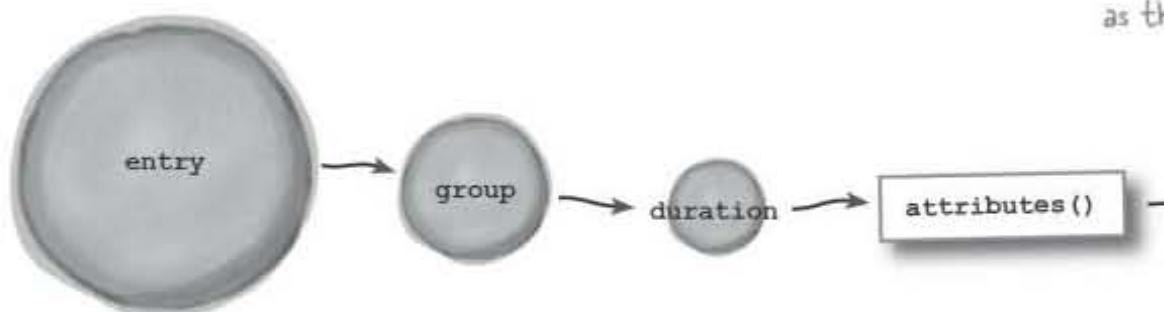


The `->` operator references a child object from a parent object. So `title` is a child of `group`, which is a child of `entry`. Remember that the `->` operator can be used to access both properties and methods. One method that comes in particularly handy is the `attributes()` method, which is able to pluck out the value of an XML attribute for a given element.

`$attrs = $entry->group->duration->attributes();
echo $attrs['seconds'];`

The `attributes()` method obtains an array of attributes for an object (element).

This code drills down to the `duration` element and then grabs all of its attributes and stores them in the `$attrs` variable, which is an array of all the attributes. The value of the `seconds` attribute is then retrieved from the array.



Not without a namespace!

There's a small problem with the code on the facing page that accesses XML data using objects, and it has to do with namespaces. If you recall, namespaces act as surnames for tags by organizing them into meaningful collections. So in a YouTube response, the `<duration>` tag is actually coded as `<yt:duration>`, and the title for a video is coded as `<media:title>`, not `<title>`. When an element is associated with a namespace, you can't just reference it by tag name in PHP code. Instead, you have to first isolate it by namespace by calling the `children()` method on the parent object.

```
$media = $sentry->children('http://search.yahoo.com/mrss/');
```

This code retrieves all the child objects of the video entry whose namespace is `http://search.yahoo.com/mrss/`. But that's the URL for a namespace, not the namespace itself. This URL is located in the `<feed>` tag at the start of the XML document. This is where you'll find all the namespaces being used.

```
<feed xmlns='http://www.w3.org/2005/Atom'  
      xmlns:openSearch='http://a9.com/-/spec/opensearchrss/1.0/'  
      xmlns:gml='http://www.opengis.net/gml'  
      xmlns:georss='http://www.georss.org/georss'  
      xmlns:media='http://search.yahoo.com/mrss/'  
      xmlns:batch='http://schemas.google.com/gdata/batch'  
      xmlns:yt='http://gdata.youtube.com/schemas/2007'  
      xmlns:gd='http://schemas.google.com/g/2005'>
```

This code reveals how each namespace is associated with a URL. More specifically, it shows how the `media` and `yt` namespaces are specified for use in the document. This is all you need to find tags related to these two namespaces.

Once you've isolated the child elements for a particular namespace by calling the `children()` method on the parent element, you can then resume accessing child objects with the `->` operator. For example, this code obtains the video title from the `<media:group>` tag:

```
$title = $media->group->title;
```



Sharpen your pencil

Using the namespace information and PHP code above, write the PHP code that gets the duration (in seconds) of a video.

```
$yt = $media->children('.....')  
$attrs = .....;  
echo $attrs['.....'];
```

no dumb questions on objects

Sharpen your pencil

Solution

Using the namespace information and PHP code above, here's some PHP code that gets the duration (in seconds) of a video.

```
$yt = $media->children('http://gdata.youtube.com/schemas/2007...');
$attrs = ...$yt->duration->attributes();
echo $attrs['seconds'];
```

The name of the attribute is
used as the key for accessing
the attribute array.

Grab all of the attributes
for the <yt:duration> tag.

there are no Dumb Questions

Q: How is an object different than an array? Don't arrays also store collections of data?

A: Yes. Arrays and objects are actually a lot alike. But one huge difference is that objects can have executable code attached to them in the form of methods. Methods are pretty much the same as functions except that they are tied to an object, and are usually designed to work specifically with the data stored in an object. Arrays are purely about storing a set of related data, and have no notion of methods. Additionally, array elements are accessed by specifying the index or key of an element inside square brackets ([]), while object properties and methods are accessed by name using the `->` operator.

Q: What exactly is an object? Is it like a normal variable?

A: Yes. An object is exactly like any other variable in PHP; it's just that it is able to store more complex data. So instead of just storing a string of text or a number, an object is able to store a combination of strings, numbers, etc. The idea is that by combining related data together with functions that act on them, the overall design and coding of applications becomes more logical.

Q: So how do objects help in processing XML data?

A: Objects help in regard to XML data processing because they are able to model the element hierarchy of an XML document in nested child objects. The benefit to this approach is that you can navigate through child objects using the `->` operator and access whatever data you need.

Q: I thought the `->` operator was for properties. How does it allow me to access objects?

A: The reason is that when dealing with objects, objects are actually stored as properties. So when you want to use the `->` operator to access a child object, you're really accessing a property. The SimpleXMLElement class makes this possible.

Q: Hang on, what's the SimpleXMLElement class?

A: Every object in PHP has a specific type, but the term "object" is really a generic term. So when you're creating an object of a specific type, you're actually accomplishing a specific task. In the case of SimpleXMLElement, and it is automatically created when you call the `simplexml_load_file()` function. This function creates an object of type SimpleXMLElement.

Q: What do I need to know about SimpleXMLElement?

A: Surprisingly, not a whole lot. The main thing to remember is that it exposes the elements in an XML document as properties, so that these properties lead to child objects. These child objects are instances of the SimpleXMLElement class. You can then use the SimpleXMLElement object also has methods for navigating through the XML document, such as `getElementsByName()` and `getAttributes()`.

Fang sightings are on the rise

While Owen has been busy brushing up on XML and figuring out how to communicate with YouTube, Fang has been busy. Numerous video sightings have turned up with the little guy apparently serving as a tour guide for his alien abductors. Owen is ready to finish up the YouTube script, get some videos showing on the Aliens Abducted Me home page, and find his lost dog.

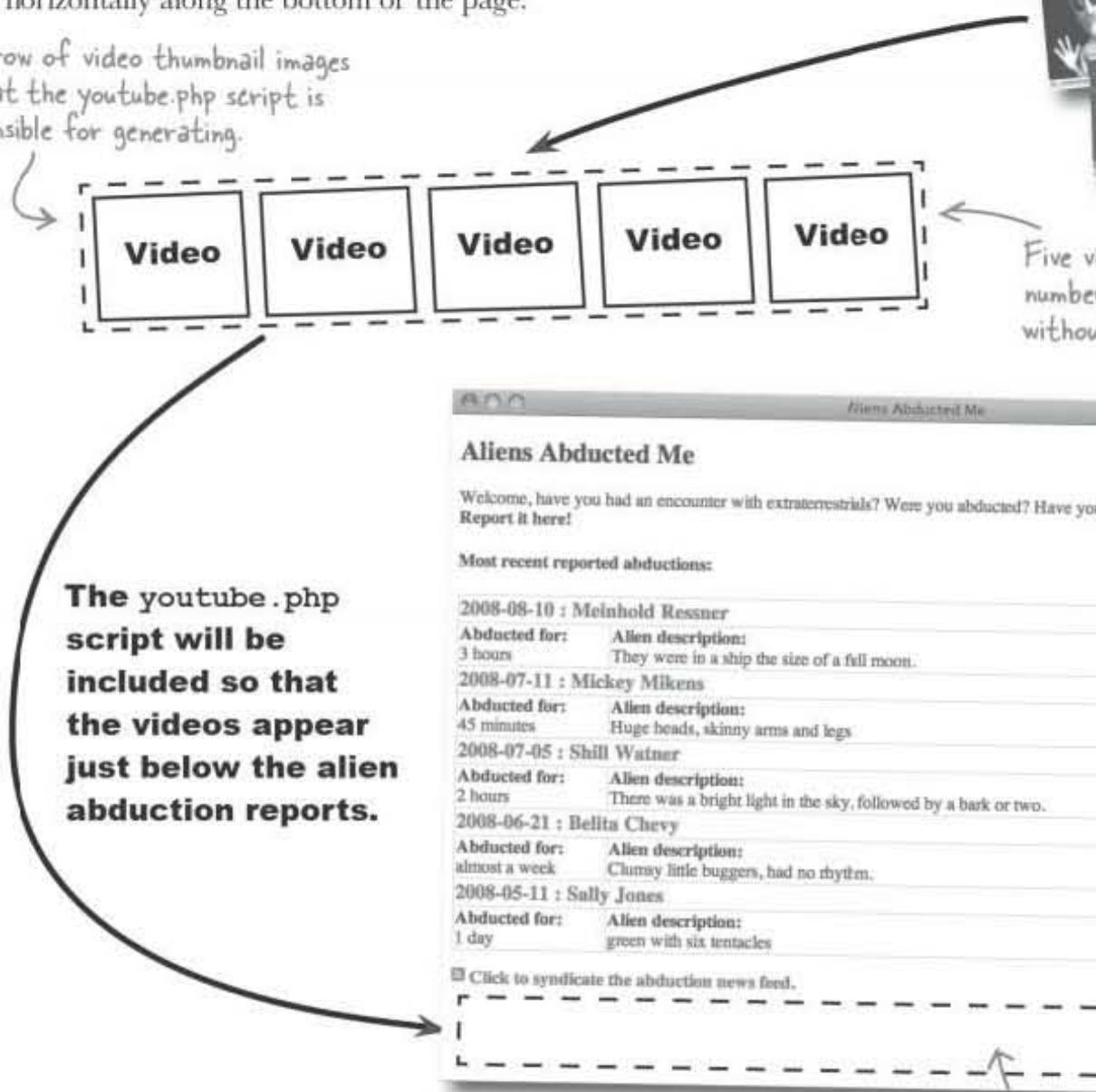


arranging the youtube videos

Lay out videos for viewing

The idea behind the youtube.php script is that it will be included in the main index.php script for Aliens Abducted Me. This means that the youtube.php script needs to take care of submitting a video request, processing the XML response, and formatting the individual videos so that they are displayed via HTML in such a way that they can coexist with the alien abduction reports that are already on the main page. A good way to accomplish this is to arrange the videos horizontally along the bottom of the page.

This row of video thumbnail images
is what the youtube.php script is
responsible for generating.



Arranging the videos horizontally on the main page keeps them from detracting too much from the alien abduction reports. Also, we're talking about arranging the video thumbnail images, not the videos themselves, so users will have to click a thumbnail to visit YouTube and see the actual video. It would eat up too much screen real estate to attempt to show multiple videos large enough to be embedded directly on the Aliens Abducted Me page.

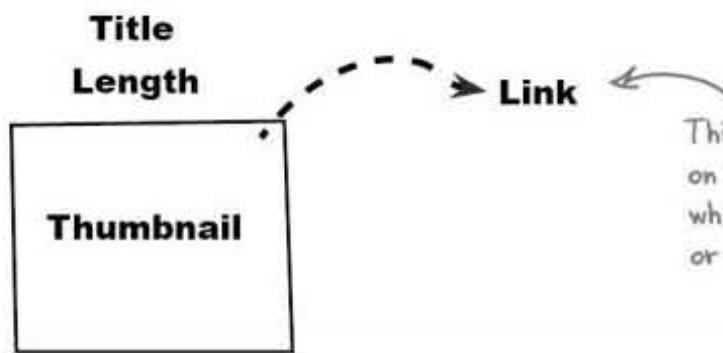
Format video data for display

Although a video thumbnail image is certainly one of the most important pieces of information when assessing whether or not a video is worth watching, it isn't the only data useful for Owen's YouTube script. For example, the title of a video could hold some important information about the nature of the video—like whether it might include a dog. The length of the video could also be helpful. And of course, we need the URL of the video link to YouTube so that the user can click on a video thumbnail to actually view a video. So the following information is what we need to extract from the XML data in the YouTube response:

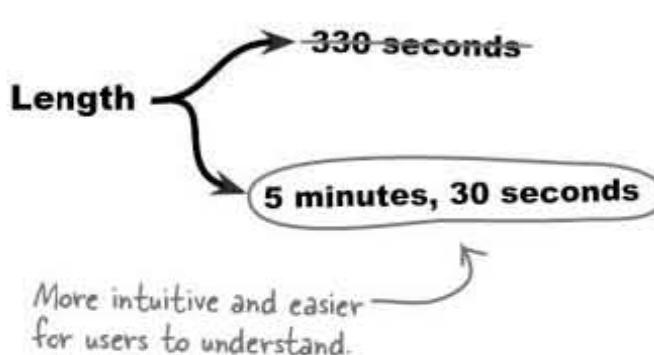
Title	Length	Thumbnail	Link
--------------	---------------	------------------	-------------

Several
data are
order to
videos c

This data forms the basis for the HTML code that displays a horizontal row of videos. In fact, each video in the row ends up looking like this:



In the YouTube response data, the length of a video is specified in the seconds attribute of the `<yt:duration>` tag. Unfortunately, most people don't think in terms of total seconds because we're accustomed to times being specified in minutes and seconds. For example, it isn't immediately obvious that 330 seconds is a five-and-a-half-minute video—you have to do the math for the value to make sense as a length of time. Knowing this, it's a good idea to go ahead and do the math for users when displaying the length of a video, converting seconds into minutes and seconds.



Geek Bits —

It isn't necessary to factor in hours for length calculation because YouTube allows videos longer than 10 minutes.

complete the youtube.php script



The youtube.php script uses PHP code to grab the top five matches for YouTube video search. It then displays thumbnail images for those videos with links to the actual videos on YouTube. Fill the missing code for the script. YouTube XML video response data on the facing page as a guide.

```
<?php

define('YOUTUBE_URL', 'http://gdata.youtube.com/feeds/api/videos/-/alien/abdu
define('NUM_VIDEOS', 5);

// Read the XML data into an object
$xml = .....(YOUTUBE_URL);

$num_videos_found = count(.....);
if ($num_videos_found > 0) {
    echo '<table><tr>';
    for ($i = 0; $i < min($num_videos_found, NUM_VIDEOS); $i++) {
        // Get the title
        $entry = $xml->entry[$i];
        $media = $entry->children('http://search.yahoo.com/mrss/');
        $title = $media->group->....;
        // Get the duration in minutes and seconds, and then format it
        $yt = $media->children('http://gdata.youtube.com/schemas/2007');
        $attrs = $yt->duration->attributes();
        $length_min = floor($attrs['.....'] / 60);
        $length_sec = $attrs['.....'] % 60;
        $length_formatted = $length_min . (($length_min != 1) ? ' minutes, ':' minute, ');
        $length_sec . (($length_sec != 1) ? ' seconds':' second');

        // Get the video URL
        $attrs = $media->group->player->.....();
        $video_url = $attrs['url'];
    }
}
```

```

    // Get the thumbnail image URL
    $attrs = $media-> ..... ->thumbnail[0]->attributes();
    $thumbnail_url = $attrs['url'];

    // Display the results for this entry
    echo '<td style="vertical-align:bottom; text-align:center" width="'
    . "%><a href="" . $video_url . "">' . ..... .
    $length_formatted . '</span><br /><img src="" . "'
    . '">' . ..... .
}

echo '</tr></table>';
}
else {
    echo '<p>Sorry, no videos were found.</p>';
}
?>

```

...

<entry>

<id>http://gdata.youtube.com/feeds/api/videos/_6Uibqf0vtA**</id>**

<published>2006-06-20T07:49:05.000-07:00**</published>**

...

<media:group>

<media:title type='plain'>UFO Sighting in Yosemite Park near Area 51</media:title>

<media:description type='plain'>I went on a trip to Yosemite Park in 2002. Yosemite is close to the border between California and Nevada, and close to Area 51...</media:description>

<media:keywords>51, alien, aliens, area, ca, california, nevada, sighting, sighting, ufo</media:keywords>

<yt:duration seconds='50'>

<media:category label='Travel & Events' scheme='http://gdata.youtube.com/schemas/2007/categories.cat'>Travel</media:category>

<media:content url='http://www.youtube.com/v/_6Uibqf0vtA' type='application/x-mpegURL' medium='video' isDefault='true' expression='full' duration='50' yt:format='mp4'>

<media:content url='rtsp://rtsp2.youtube.com/ChOLENy73w1aEQnQvvSnb1K1_xMYD...' type='video/3gpp' medium='video' expression='full' duration='50' yt:format='3gp'>

<media:content url='rtsp://rtsp2.youtube.com/ChOLENy73w1aEQnQvvSnb1K1_xMYD...' type='video/3gpp' medium='video' expression='full' duration='50' yt:format='3gp'>

<media:player url='http://www.youtube.com/watch?v=_6Uibqf0vtA'>

<media:thumbnail url='http://img.youtube.com/vi/_6Uibqf0vtA/2.jpg' height='360' time='00:00:25'/>

<media:thumbnail url='http://img.youtube.com/vi/_6Uibqf0vtA/1.jpg' height='360' time='00:00:12.500'/>

<media:thumbnail url='http://img.youtube.com/vi/_6Uibqf0vtA/3.jpg' height='360' time='00:00:37.500'/>

<media:thumbnail url='http://img.youtube.com/vi/_6Uibqf0vtA/0.jpg' height='360' time='00:00:25'/>

</media:group>

<yt:statistics viewCount='2478159' favoriteCount='1897' />

<gd:rating min='1' max='5' numRaters='1602' average='4.17' />

<gd:comments>

<gd:feedLink href='http://gdata.youtube.com/feeds/api/videos/_6Uibqf0vtA/countHint='4426' />

</gd:comments>

</entry>

<entry>

the completed youtube.php



The youtube.php script uses PHP code to grab the top five matches from YouTube video search. It then displays thumbnail images for those videos with links to the actual videos on YouTube. Fill in the missing code for the sample YouTube XML video response data on the facing page as a guide.

```

<?php
define('YOUTUBE_URL', 'http://gdata.youtube.com/feeds/api/videos/-/alien/abdu...');

define('NUM_VIDEOS', 5);           ← The number of videos to be displayed is stored as a constant.

// Read the XML data into an object
$xml = simplexml_load_file(YOUTUBE_URL);           ← The simplexml_load_file() function is used to request the XML data from YouTube.

$num_videos_found = count($xml->entry);           ← Check to see how many videos were actually returned by YouTube by counting the number of <entry> tags.

if ($num_videos_found > 0) {
    echo '<table><tr>';

    for ($i = 0; $i < min($num_videos_found, NUM_VIDEOS); $i++) {           ← Loop through video data entry at a time.

        // Get the title
        $entry = $xml->entry[$i];
        $media = $entry->children('http://search.yahoo.com/mrss/');

        $title = $media->group->title;           ← Extract the title of the video entry, which is stored in the <media:title> tag.

        // Get the duration in minutes and seconds, and then format it
        $yt = $media->children('http://gdata.youtube.com/schemas/2007');
        $attrs = $yt->duration->attributes();
        $length_min = floor($attrs['seconds'] / 60);
        $length_sec = $attrs['seconds'] % 60;           ← Grab the duration of the video in seconds from the <yt:duration> tag, and then convert it to minutes.

        $length_formatted = $length_min . (($length_min != 1) ? ' minutes, ':' minute, ');
        $length_sec . (($length_sec != 1) ? ' seconds':' second');

        // Get the video URL
        $attrs = $media->group->player->attributes();
        $video_url = $attrs['url'];           ← Grab the video link (URL) from the attribute of the <media:player> tag.
    }
}

```

```
// Get the thumbnail image URL
$attrs = $media->group->thumbnail[0]->attributes();
$thumbnail_url = $attrs['url']; // Extract the first thumbnail URL from the url attribute of the <mediathumbnail> tag.

// Display the results for this entry
echo '<td style="vertical-align:bottom; text-align:center" width="100px">' . $length_formatted . '</td>' . $title . '<br /><span style="font-size:1.5em; color:#0000ff; cursor:pointer;">' . $video_url . '</span>' . '<br /><img alt="Thumbnail image" src=' . $thumbnail_url . '" />' . '</td>' . '</tr></table>';

else {
    echo '<p>Sorry, no videos were found.</p>';
}


```

- 1 Build a request
 - 2 Issue the request
 - 3 Receive Your Response containing JSON
 - 4 Process the response as HTML or XML

```
<entry>
<id>http://gdata.youtube.com/feeds/api/videos/_6Uibqf0vtA</id>
<published>2006-06-20T07:49:05.000-07:00</published>
...
<media:group>
<media:title type='plain'>UFO Sighting in Yosemite Park near Area 51</media:title>
<media:description type='plain'>I went on a trip to Yosemite Park in 2002. Yosemite is close to the border between California and Nevada, and close to Area 51...</media:description>
<media:keywords>51, alien, aliens, area, ca, california, nevada, sighting, ufo</media:keywords>
<yt:duration seconds='50'>
<media:category label='Travel & Events'
  scheme='http://gdata.youtube.com/schemas/2007/categories.cat'>Travel</media:category>
<media:content url='http://www.youtube.com/v/_6Uibqf0vtA' type='application/x-mpegURL' medium='video' isDefault='true' expression='full' duration='50' yt:format='1'>
<media:content url='rtsp://rtsp2.youtube.com/ChOLENy73wIaEQnQvvSnbik1_xMYD' type='video/3gpp' medium='video' expression='full' duration='50' yt:format='1'>
<media:content url='rtsp://rtsp2.youtube.com/ChOLENy73wIaEQnQvvSnbik1_xMYD' type='video/3gpp' medium='video' expression='full' duration='50' yt:format='1'>
<media:player url='http://www.youtube.com/watch?v=_6Uibqf0vtA'>
<media:thumbnail url='http://img.youtube.com/vi/_6Uibqf0vtA/2.jpg' height='360' time='00:00:25'>
<media:thumbnail url='http://img.youtube.com/vi/_6Uibqf0vtA/1.jpg' height='360' time='00:00:12.500'>
<media:thumbnail url='http://img.youtube.com/vi/_6Uibqf0vtA/3.jpg' height='360' time='00:00:37.500'>
<media:thumbnail url='http://img.youtube.com/vi/_6Uibqf0vtA/0.jpg' height='360' time='00:00:25'>
</media:group>
<yt:statistics viewCount='2478159' favoriteCount='1897' />
<gd:rating min='1' max='5' numRaters='1602' average='4.17' />
<gd:comments>
  <gd:feedLink href='http://gdata.youtube.com/feeds/api/videos/_6Uibqf0vtA/countHint=4426' />
</gd:comments>
</entry>
<entry>
```

aliensabductedme—now with youtube videos!



Test DRIVE

Add the YouTube script to Aliens Abducted Me.

Create a new text file named `youtube.php`, and enter the code for Owen's YouTube previous two pages (or download the script from the Head First Labs site at www.headfirstlabs.com/books/hfphp). You still need to plug the script into the `index.php` script to videos loose on the main Aliens Abducted Me page. Here are the two lines of PHP code:

```
echo '<h4>Most recent abduction videos:</h4>';
require_once('youtube.php');
```

Upload the scripts to your web server, and then open `index.php` in a web browser. The bottom of the page should show a dynamically generated row of YouTube video links that are related to alien abductions.

Including the `youtube.php` script in the main page is all it takes to add the row of alien abduction videos.

YouTube videos have helped Owen narrow Fang's location.

Aliens Abducted Me

Welcome, have you had an encounter with extraterrestrials? Were you abducted? Have you seen my abducted dog, Fang? Report it here!

Most recent reported abductions:

2008-08-10 : Meinhold Ressner	Alien description:	Fang sp
Abducted for: 3 hours	they were in a ship the size of a full moon	no
2008-07-11 : Mickey Mikens	Alien description:	Fang sp
Abducted for: 45 minutes...and counting	huge heads, skinny arms and legs	yes
2008-07-05 : Shill Watner	Alien description:	Fang sp
Abducted for: 2 hours	there was a bright light in the sky, followed by a bark or two	yes
2008-06-21 : Belita Chevy	Alien description:	Fang sp
Abducted for: almost a week	clumsy little buggers, had no rhythm	no
2008-05-11 : Sally Jones	Alien description:	Fang sp
Abducted for: 1 day	green with six tentacles	yes

Click to syndicate the abduction news feed.

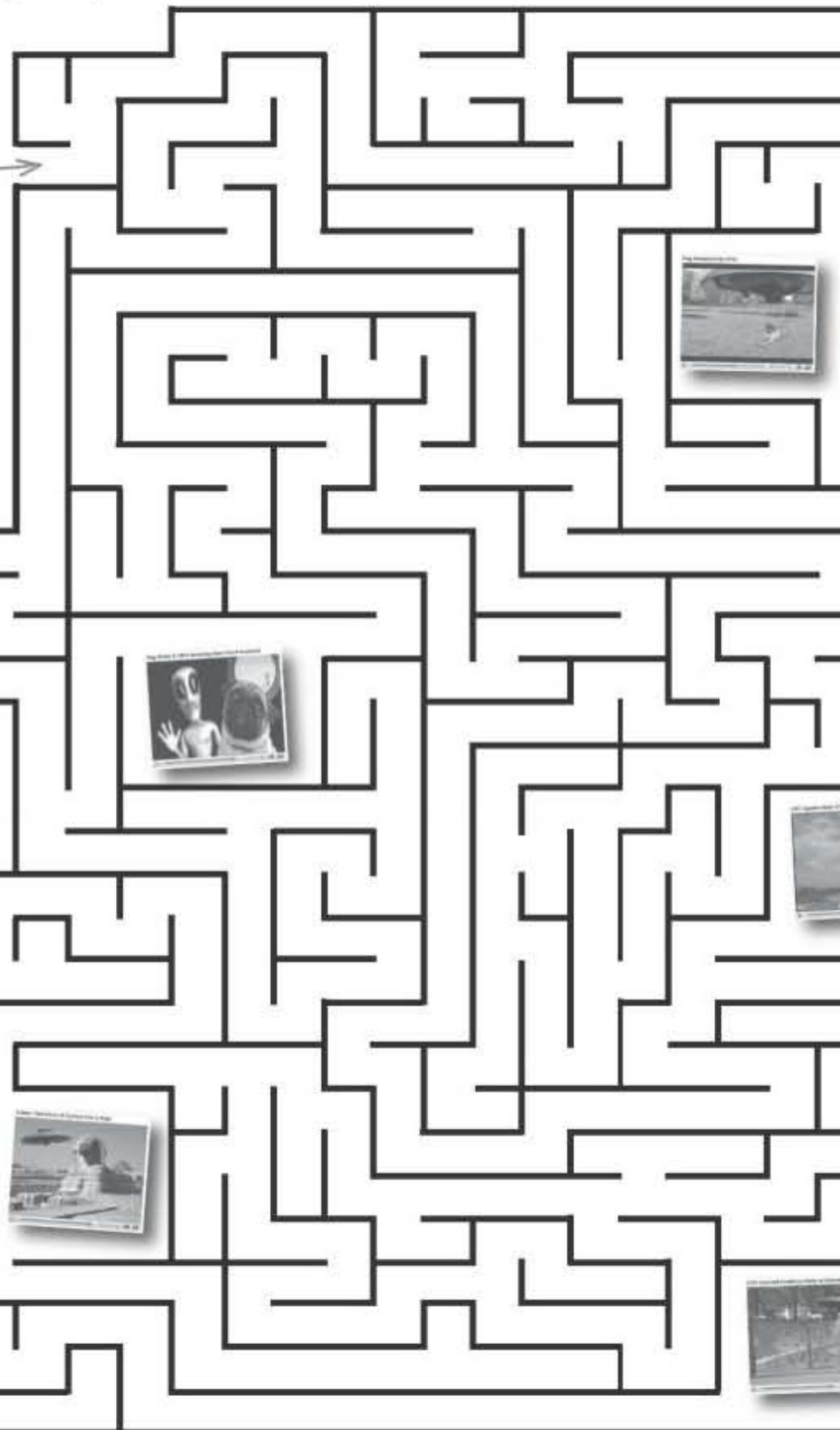
Most recent abduction videos:

UFO Spotted Crashing Party at Graceland! 8 minutes, 10 seconds	Aliens Turn Face of Sphinx Into a Dog! 6 minutes, 10 seconds	Dog Rides in UFO Hovering Near San Francisco! 0 minutes, 11 seconds	UFO Spotted Near Eiffel Tower! 0 minutes, 13 seconds	Pug Abdu... UFO 0 minutes, 11 seconds
--	--	---	--	---------------------------------------

Sharpen your pencil

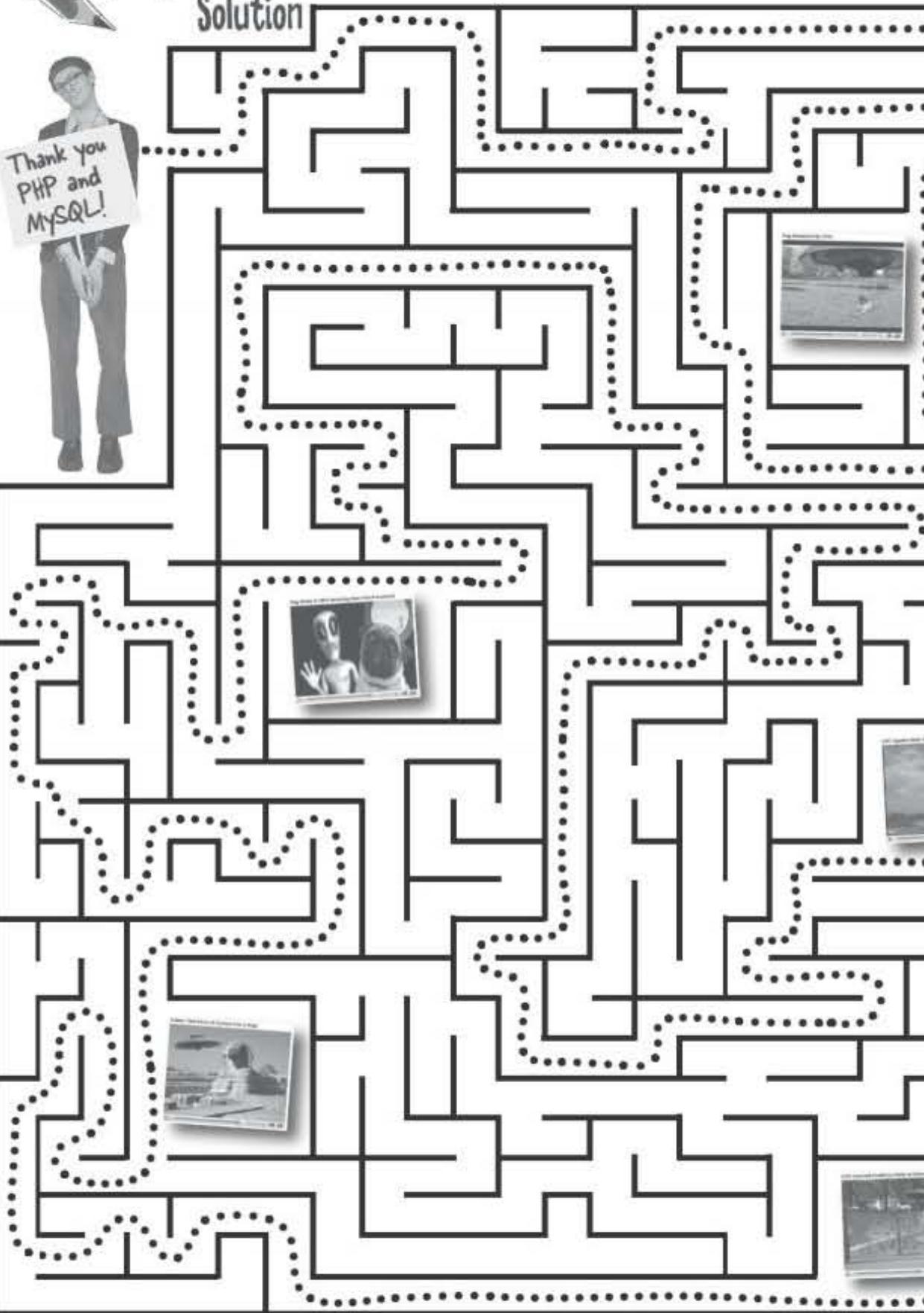


Start here!



you found fang!

Sharpen your pencil Solution





Your PHP & MySQL Toolbox

With Fang now accounted for, it's possible to reflect a little on what all it took to track him down. As it turns out, PHP and MySQL required some help from a few other technologies.

XML

A generic markup language used to provide a predictable structure to data. There are many different markup languages built from XML, such as XHTML and RSS. The idea is that you create a set of tags specific to whatever data you're storing as XML.

`simplexml_load_file()`

This built-in PHP function loads an XML file from a URL, and then makes the resulting XML data accessible through an object.

SimpleXMLElement

A built-in PHP object that is used to access XML data. This object is returned by the `simplexml_load_file()` function, and contains the entire document hierarchy of an XML document.

REST

A means of access on the web purely URLs. REST allows powerful data requests by creating a URL often referred to as requests.

RSS

An XML-based language used to store syndicated content, such as news stories. RSS allows web sites to make their data available to other applications and web sites for syndication, and allows you to take advantage of data made available by other sites.

Namespace

A way of organizing XML tags into a sort of like how it organizes your files. It's a named group. A namespace is always associated with a prefix which ensures uniqueness from other namespaces.

The End.



appendix i: leftovers



The Top Ten Topics *(we didn't cover)*



Even after all that, there's a bit more. There are just a few more things we think you need to know. We wouldn't feel right about ignoring them, even though they only need a brief mention. So before you put the book down, take a read through these short but important PHP and MySQL tidbits. Besides, once you're done here, all that's left are a couple short appendices... and the index... and maybe some ads... and then you're really done. We promise!

retrofit your php code

#1. Retrofit this book for PHP4 and mysql functions

With the exception of XML functions in Chapter 12, most of the code in this book will run on PHP 4 servers with only a little modification. We've used the mysqli family of functions in this book, which are only available in PHP 4.1 and later, and since the library has to be manually installed, some servers won't support mysqli.

Mysqli functions are generally faster, but this really only begins to matter when your database becomes huge. Small or average databases won't be perceptibly slower with the older mysql functions. This section is designed to tell you how to retrofit your mysqli functions to work as mysql functions with older versions of PHP.

If you see:

```
$dbc = mysqli_connect(localhost, 'mork', 'frommork');

mysqli_select_db($dbc, 'alien_database');
```

you'll use:

```
$dbc = mysql_connect(localhost, 'mork', 'frommork');

mysql_select_db('alien_database', $dbc);
```

The database connection
is not the first argument
as it is with mysqli_select_db

In general, you just remove the *i* from mysqli, making it mysql, and then swap the order of the arguments so that the database connection variable (\$dbc in this example) appears last.

But it gets a little trickier when the mysqli_connect() function sidesteps mysqli_select_db() and uses a database name. There's nothing quite like that in the mysql family of functions. For the single mysqli_connect() function that uses a database name, you'll need two mysql functions.

If you see:

```
$dbc = mysqli_connect(localhost, 'mork', 'frommork', 'alien_database');
```

you'll need to use two commands:

```
$dbc = mysql_connect(localhost, 'mork', 'frommork');

mysql_select_db('alien_database', $dbc);
```

This connection variable is
also known as a database
connection "link."

Here the database
selected as part
of the connection
that isn't possible
step with mysql

With mysql functions, it always
takes two function calls to establish
a connection with a specific database.

Here's how mysql and mysqli functions match up.

Close MySQL connection	<code>mysqli_close(conn)</code>	<code>mysqli_close(conn)</code>
Open a connection to a MySQL server	<code>mysql_connect(host, username, password)</code> You must use <code>mysql_select_db()</code> to select a database.	<code>mysqli_connect(host, password, database)</code> You don't need <code>mysqli_select_db()</code> to select a database.
Return the text of the error message from previous MySQL operation	<code>mysql_error(conn)</code>	<code>mysqli_error(conn)</code>
Escape a string	<code>mysql_escape_string(string, conn)</code> The order of arguments is opposite, expects the string, then the connection (link).	<code>mysqli_escape_string(string, link)</code> Expects the connection (link) string.
Fetch a result row as an associative array, a numeric array, or both	<code>mysql_fetch_row(result)</code>	<code>mysqli_fetch_row(result)</code>
Get number of rows in result	<code>mysql_num_rows(result)</code>	<code>mysqli_num_rows(result)</code>
Execute a MySQL query	<code>mysql_query(conn, query)</code>	<code>mysqli_query(conn, query)</code>
Escape special characters in a string	<code>mysql_real_escape_string(string, conn)</code> The order of arguments is opposite, expects the string, then the connection (link).	<code>mysqli_real_escape_string(string, link)</code> Expects the connection (link) string.
Select a MySQL database	<code>mysql_select_db(dbname, conn)</code> The order of arguments is opposite, expects the string, then the connection (link).	<code>mysqli_select_db(conn, dbname)</code> Expects the connection (link) string.

setting mysql user permissions

#2. User permissions in MySQL

Suppose you've created a web application that only allows visitors to SELECT data from your database. You want to let users run queries on your data using a specific database, and MySQL gives you power over your data.

But consider this: the login and password you use in your mysqli connection string would, if connected via the MySQL terminal or GUI, allow the user to INSERT, UPDATE, and DELETE data.

If your application doesn't need to do those things, there's no reason why the user/password you give it needs to be able to. With MySQL, you can limit the access to your database. You can tell MySQL to let the user to SELECT. Or SELECT and INSERT. Or any combination you need.

And what's even more impressive, you can control access to specific tables. For example, if your application only needs to interact with a table called `alien_info` and doesn't need access to the `cyborg_info` table, you can limit the user to just those two tables.

First, you may want to create an entirely new user/password to be used for your application. You can do this via the MySQL terminal:

```
File Edit Window Help Aliens!
mysql> CREATE USER alienguy IDENTIFIED BY 'aliensRsc4ry';
Query OK, 0 rows affected (0.07 sec)
```

Then you can use the MySQL GRANT command to control what `alienguy` can do to your database. If he only needed to SELECT and INSERT data into your database, this would work:

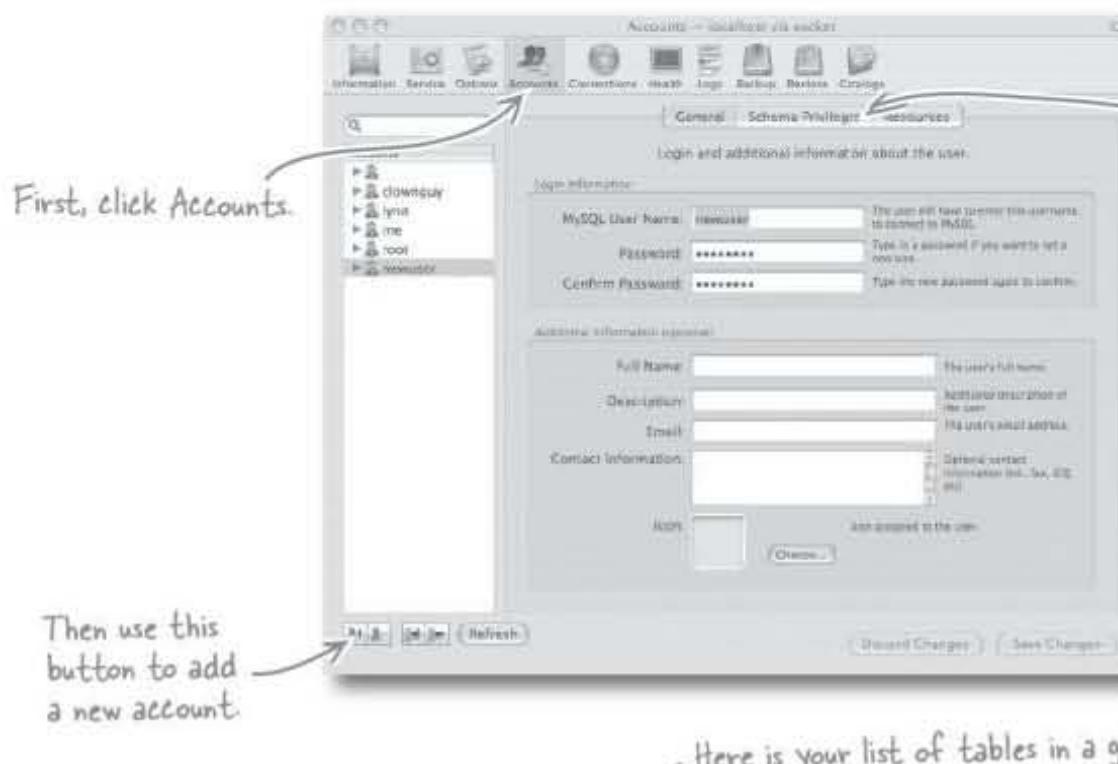
```
File Edit Window Help TheyLive
mysql> USE alien_database;
Database changed
mysql> GRANT SELECT, INSERT ON alien_info TO alienguy;
Query OK, 0 rows affected (0.03 sec)
```

If you don't like using the MySQL terminal to create users and set privileges, you can download and install a handy program called MySQLAdministrator. Get it here: <http://dev.mysql.com/downloads/gui-tools/5.0.html>.

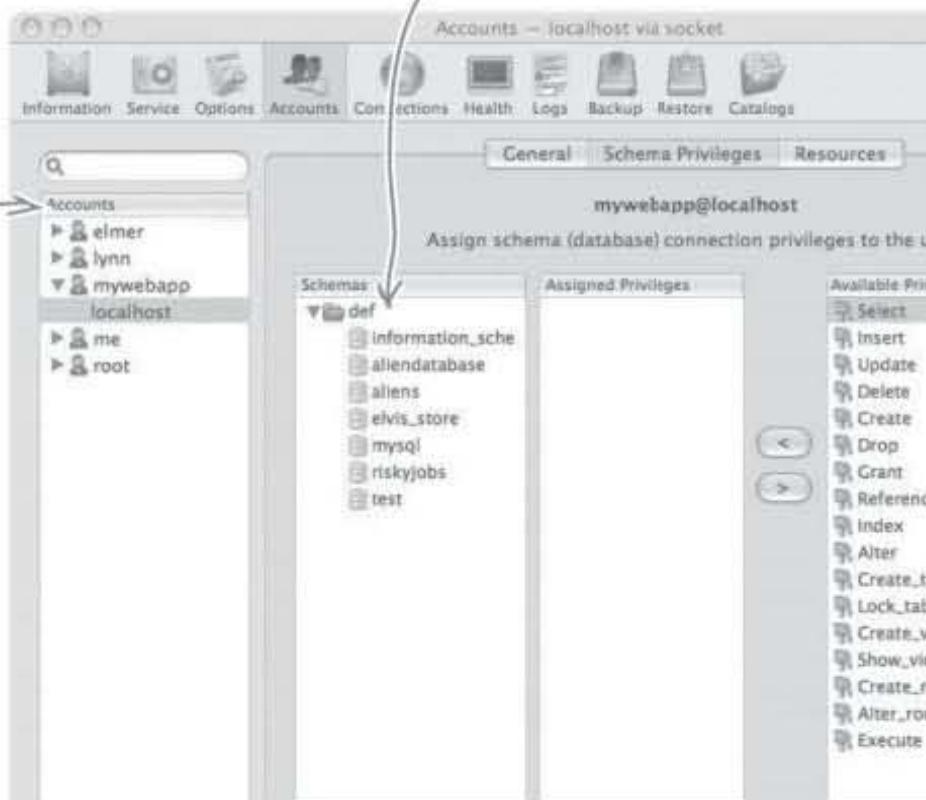
You can set very specific user privileges, even control what they can do to a specific column. To learn more, check out Head First MySQL.

The MySQL Administrator lets you control your user accounts and what each user account can do in your database. It even allows you to specify which kind of queries that user can perform on each table in your database. To control the access every user has to each table and each query, open the MySQL Administrator application, and click on the **Accounts** tab.

Here's the interface and an overview of how to control what each user can do. First, create a new user account.



This is your user list. You can create new users to specifically set the control each has for a given application. Select the one you want to modify here.



mysql error reporting

#3. Error reporting for MySQL

In many of our code examples, you'll see lines like this:

```
mysqli_connect('localhost', 'mork', 'fromork') or die ('Could not connect');
```

When this command fails, the words "Could not connect." are displayed on the web page. It tells us something went wrong, but it doesn't give us information beyond that.

Fortunately, PHP offers a function, **mysql_error()**, that will give us a clue as to exactly what went wrong. Consider this code where we are trying to connect to a MySQL server that doesn't exist:

```
<?php
    mysqli_connect('badhostname', 'mork', 'fromork') or die (mysql_error());
?>
Unknown MySQL server host 'badhostname' (1) ← Here's the error message you'll see.
```

This will return clear information as to what actually went wrong when the `mysqli_connect()` function fails. You can also use `mysqli_error()` with other MySQLi functions:

```
<?php
$dbc = mysqli_connect('localhost', 'mork', 'fromork');
mysqli_select_db($dbc, 'alien_database'); ← We try to connect to a database that doesn't exist
echo mysqli_error($dbc) . '<br />';
mysqli_select_db($dbc, 'alien_database');
mysqli_query($dbc, "SELECT * FROM alien_info");
echo mysqli_error($dbc);
?>
```

Here's the output:



Here are some other error messages you might see:

Table 'test.no_such_table' doesn't exist

Can't create table

Can't create database 'yourdatabase'; database exists

Can't drop database 'yourdatabase'; database doesn't exist

There are dozens more, and it would be a waste of paper to list them here. Browse on over to the MySQL documentation to get more information:

<http://dev.mysql.com/doc/refman/5.0/en/error-messages-server.html>

If you're retrofitting your mysql functions, as mentioned in #1, you can use `mysql_error()` instead of `mysqli_error()`.

#4. Exception handling PHP errors

Exception handling allows you to change the normal flow of your code and execute a special block of code when a specific exception occurs. PHP 5 and 6 offer exception handling. Here's a brief introduction.

Let's say you want to withdraw \$200 bucks from an ATM.

But maybe you're required to have a minimum balance of \$1000, and this withdrawal will put you under \$1000. That isn't allowed.

Transaction failed!

Here's how this scenario might play out in PHP code with the help of exception handling to catch the failure.



```
<?php
    function checkBalance($balance) {
        if($balance < 1000) {
            throw new Exception("Balance is less than $1000.");
        }
        return true;
    }
    try {
        checkBalance(999); ← We check our
                           balance here.
        echo 'Balance is above $1000.';
    }
    catch(Exception $e) {
        echo 'Error: ' . $e->getMessage(); ← If our exception occurs,
                                         execute the code in this block.
    }
?>
```

When the code runs, you'll see this:

Error: Balance less than \$1000.

Here's the feedback sent out if our balance is less than 1000.

The "try" block is used to test our value without ending the code flow.

We check our balance here.

If our exception occurs, execute the code in this block. In this case, we echo

exception handling in php

#4. Exception handling PHP errors (cont.)

Exception handling consists of three blocks of code:

1. **Try** - This block is where you check to see if your value is what you expect.

If it is, everything is great, and your code continues on its way. If not, an exception occurred. In programmerese, an exception is “thrown.”

And when something is thrown, there needs to be something to catch it. If no exception, the “catch” block code is executed. If not, the code will continue.

2. **Throw** - The “throw” command sends the “catch” block an error message. Each “throw” has at least one “catch.”

```
<?php
    function checkBalance($balance) {
        if($balance < 1000) {
            throw new Exception("Balance is less than $1000");
        }
        return true;
    }

    try {
        checkBalance(999);
        echo 'Balance is above $1000.';
    } catch(Exception $e) {
        echo 'Error: ' . $e->getMessage();
    }
?>
```

3. **Catch** - An **object** is created with the exception information. More information on objects, on the facing page.

#5. Object-oriented PHP

Object-oriented languages use a very different programming model than their procedural counterparts. You've been using PHP procedurally, but it also has an object-oriented side. Instead of a chronological step-by-step set of instructions, particular structures become objects. **Objects include not only a definition of your data, but also all the operations that can be performed on it.** When you use object-oriented PHP, you create and work with **objects**.

Before we discuss why you might want to use OO PHP, let's write some:

1 Write your class.

```
class Song
{
    var $title;           ← These are instance
    var $lyrics;          ← variables
    function Song($title, $length) { ← This sets the title
        $this->title = $title;      and lyrics of a song
        $this->lyrics = $lyrics;   when we create one.
    }
    function sing() { ← This is a method that uses the
        echo 'This is called ' . $this->title . '.<br />';
        echo 'One, two, three...' . $this->lyrics;
    }
}
```

2 Create a new object.

```
$shoes_song = new Song('Blue Suede Shoes', 'Well it\'s one for the money');
$shoes_song->sing();
```

Our new song has the value "Blue Suede Shoes" for its name.

Here's where we call the sing() method for our object.

3 Your song can sing itself!

When you run this code, you get this:



This is called Blue Suede Shoes.
One, two, three...Well it's one for the
money...

But if you
code with
why use C

There are

object-oriented php

#5. Object-oriented PHP (cont.)

Instead of a chronological step-by-step set of instructions, your data structures become objects. **Objects include not only a definition of your data, but also all the operations that can be performed on it.** In our Song example, we set the title and lyrics of the song inside the class, and we create the `sing()` method inside the class. If we needed to add more functionality to our Song object, we'd add new methods and variables to our Song class. For example, if we wanted the songwriter for each song to be associated with each song object, we could add that as a variable in our class.

The power of OO really shines as an application grows. Suppose we decided to use the Song class as part of a karaoke application with hundreds or even thousands of individual song objects, all with their own unique titles, lyrics, and songwriters. Now let's say someone wants to choose from only songs that were written by Elvis. All we'd have to do is look at the songwriter instance variable of each object.

And to actually feed the lyrics to the karaoke application? We could just call the `sing()` method on each song object when it is being performed. Even though we're calling the exact same method on each object, it is accessing data unique to each of the objects.

So two big advantages of using Object Oriented PHP are:

Objects can be easily reused. They are designed to be independent of the code where they are used and can be reused as needed.

The code is easier to understand and maintain. If a data type needs to change, the change occurs only in the object, nowhere else in the code.

A big disadvantage is that, in general, OO code can be longer and take more time to write. If you simply need to display the lyrics from one song, then writing a small procedural program might be your best bet. But if you think you might want to build that online karaoke app, consider diving further into object-oriented PHP.

#6. Securing your PHP application

There are some simple steps you can follow to protect your PHP scripts from those nefarious hackers that are crouched over their keyboards waiting for you to slip up.

- 1** Remove `phpinfo()` references. When you first start building PHP applications on new web servers, you'll probably create a script that contains the `phpinfo()` function, so you can see what version of PHP you are using and if it has MySQL support, along with a list of other installed libraries. It's fine to check with `phpinfo()`, but you should remove that function after you've taken a look. If you don't, any hacker out there who discovers a new PHP vulnerability will be able to see if your site is susceptible to it.
- 2** If you aren't using a web hosting service and have access to the `php.ini` file, there are a few changes you can make to it to further secure your PHP applications. Ironically, the location of your `php.ini` file can be found by using `phpinfo()`:

The screenshot shows a web browser window with the title 'phpinfo()' at the top. The main content area displays a table of PHP configuration settings. One specific row, 'Configuration File (php.ini) Path', is highlighted with a red arrow pointing to its value, which is '/user/local/etc/php5/php.ini'. The table includes rows for System, Build Date, Configure Command, Server API, Virtual Directory Support, PHP API, PHP Extension, Zend Extension, Debug Build, Thread Safety, Zend Memory Manager, IPv6 Support, Registered PHP Streams, Registered Stream Socket Transports, and Registered Stream Filters.

System	FreeBSD pro24.abac.com 5.5-RELEASE-p2 FreeBSD 5.5-RELEASE-p2 #0: Fri Jun 16 11:29:40 PDT 2006 root@ftp1.abac.com:/usr/obj/usr/src/sys/PRO i386
Build Date	Apr 3 2007 13:21:53
Configure Command	'./configure' '--enable-versioning' '--with-layout=GNU' '--disable-all' '--enable-libxml' '--with-libxml-dir=/usr/local' '--enable-reflection' '--program-prefix=' '--enable-fastcgi' '--with-regex=php' '--with-zend-vm=CALL' '--disable-ipv6' '--prefix=/usr/local'
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/user/local/etc/php5/php.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	disabled
Registered PHP Streams	php, file, data, http, ftp, compress,zlib
Registered Stream Socket Transports	tcp, udp, unix, udg
Registered Stream Filters	string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, zlib.*

php security measures

#6. Securing your PHP application (cont.)

Here are some specific settings you should consider changing in the `php.ini` file. Open the file in a text editor, make the changes, save them, and then restart your web server.

```
safe_mode = On
```

When you turn on `safe_mode`, no PHP scripts can be called by another script with a different owner on the same web server. Obviously, if you need to allow scripts from other owners to call yours, you can't use this setting.

```
open_basedir = directory[ : ... ]
```

This restricts the scripts and files that PHP will be able to execute or access to this directory and subdirectories beneath it.

```
expose_php = Off
```

With this set to `On`, every web browser that visits your site will be sent header information that reveals information about your PHP server. Turning it off hides that information and makes your server a little less exposed.

```
display_errors = Off
```

Once you've developed your application and are running it on your live web server, you don't need to see all those error messages. Hopefully, you've already addressed errors, but sometimes things slip through the cracks. To hide the error messages from site visitors, set this to `Off`.

```
log_errors = On
```

This sends your errors to an error log. When you want to check your application for errors, this is a good place to begin. With `display_errors` set to `Off` and `log_errors` set to `On`, you'll be able to see problems, but your site's visitors won't.

```
error_log = filename
```

You'll have to check with your particular web server software to locate this file. This is where your errors will be written when `log_errors` is set to `On`.

#7. Protect your app from cross-site scripting

You may have heard of cross-site scripting sometimes referred to as XSS. Cross-site scripting is an attack against a web app where script code is passed to your form processing script and hijacks your output. It's a big security problem in PHP web apps. Let's take a look at precisely what it is and how to defend against it.

Cross-site scripting usually takes advantage of sites that display user-submitted data. Any data you get from your users and display could potentially be corrupt and cause visitors to your site to be vulnerable to a hacker.

Using an XSS attack, a hacker can do any number of things. One of the worse is to redirect your results page to a page on a site under their control that might ask the user for further information. Your user might not notice that he's no longer on your site, and since he trusts your site, he might willingly submit sensitive information directly on the attackers server.

Here's how it might happen on the Guitar Wars site:

Ethel, instead of submitting her name in the Name field on the form, types in some JavaScript code. In the example, she's using the `window.location` function to redirect the browser to her own site. And since she controls her own site, she can show the visitor anything she wants, including a site that looks just like Guitar Wars. She could do something even more nefarious with sites that expect people to submit more important information than high scores, such as financial information.

There are other, even more insidious things that she could do, including stealing cookies or presenting the user with a screen that appeared to be a login screen. As soon as the user logs in, she has his username and password and can pretend to be him back on the original site.

So how do you avoid cross-site scripting attacks on your web applications?

Guitar Wars - Add Your High Score

Name: Ethel Heckel

Score: 10000000, ethellscore2.gif

Screen shot: Choose File ethellscore2.gif

Add

If Ethel can't cheat,
she'll redirect the scores
page to her own site
with cross-site scripting.



preventing cross-site scripting

#7. Protect your app from cross-site scripting (cont.)

Fortunately, if you are validating your data, you are already on the road to protecting your application. You've already learned how to do just that in *Guitar Wars*. Here are three guidelines that will keep your applications safe:

Validate everything

Any data that you receive, such as form input, needs to be validated so that hacker code is detected before it can harm your application. If you assume the data is bad until you prove that it's not through validation, you'll be much safer.

Built-in PHP functions can help

Use built-in PHP functions such as `strip_tags()` to help you sanitize external data. `strip_tags()` is a great function that removes any html tags from a string. So if you use `strip_tags()` on Ethel's `$_POST['name']`, you'll end up with this:

```
window.location='http://ethelrulz.com'
```

While this is still not a name, it won't actually redirect the browser because the important JavaScript tags have been removed.

Data is guilty until proven innocent

Start with the most restrictive validation you can, and then only ease up if you have to. For example, if you begin by accepting only numbers in a phone number field, then start allowing dashes or parentheses, you'll be much safer than if you allowed any alphanumeric characters in the first place. Or in the case of *Guitar Wars*, if we don't allow anything except letters in the name field, we'll never even get the less than sign (<) that opens Ethel's evil JavaScript code. Regular expressions (Chapter 10) can go a long way toward making sure only the exact data you want is allowed.

#8. Operator precedence

Consider this line of code.

```
$marbles = 4 / 2 - 1; ← It will be 1.
```

The value stored by \$marbles could be either 1 or 4. We can't tell from the code, but we can certain rules of **precedence**. By precedence, we mean the **order** in which they are executed. Operators in PHP are carried out in a certain order. In the example above, the division will take before the subtraction does, so \$marbles will equal 1.

Depending on what we need our code to output, we could have written it two different ways

```
$marbles = (4 / 2) - 1;  
$marbles = 4 / (2 - 1);
```

In the first expression, we divide 4 by 2 and then subtract 1. In the second case, we subtract 1 first, then divide 4 by the resulting 1. Using parentheses allow you to precisely control the order of operations. But knowing the precedence of operators in PHP can help you figure out what's going on in a complex expression. And, trust us, it will help you debug your code when you've forgotten to use parentheses.

Before we get to the PHP operator precedence list, here's another reason why you should use parentheses when writing code. Consider this:

```
$marbles = 4 - 3 - 2; ← It will be -1.
```

No precedence rules apply here. The result could be either 3 or -1. This is pretty confusing when writing code. Instead, it's better to code with parentheses, like in these two lines:

```
$marbles = 4 - (3 - 2);  
$marbles = (4 - 3) - 2;
```

Now the list, from highest precedence (evaluated first) to lowest (evaluated last).

Operator	Operator Type
<code>++ --</code>	increment/decrement
<code>* / %</code>	arithmetic
<code>+ - .</code>	arithmetic and string
<code>< <= > >= <></code>	comparison
<code>== != === !==</code>	comparison
<code>&&</code>	logical
<code> </code>	logical
<code>= += -= *= /= .= %= &= = ^=</code>	assignment
<code><<= >>=</code>	
<code>and</code>	logical
<code>xor</code>	logical
<code>or</code>	logical

php 5 versus php 6

#9. What's the difference between PHP 5 and PHP 6

As of the writing of this book, PHP 5 is the latest production version of PHP. But PHP 6 is being worked on and is available for developers here: <http://snaps.php.net/>.

The differences between PHP 4 and 5 are much greater than between 5 and 6. In many ways, 6 offers a refinement of the object-oriented features introduced in 5. Other changes include more support for XML and Unicode.

More Unicode support

Suppose your application needed to output text in Greek.



Consider the kinds of things you sometimes have to do with strings, such as needing to know the length of them or sorting them. It's straightforward in English, but when you are working with characters in other languages, string operations become more complicated.

Unicode is a set of characters and technologies to encode them. In Unicode, the Greek character that looks like a triangle has a specific numeric value assigned to it, along with other characters in other languages. Unicode is a standard, which means it receives wide support from major technology providers. In Unicode, every character has a unique number, no matter what language, program, or platform is used. Before the advent of PHP 5, PHP had no real support for Unicode. PHP 6 has enhanced support for Unicode strings in its functions and functions built specifically for creating and decoding Unicode.

#9. What's the difference between PHP 5 and PHP 6 (OO refinements, XML support, and other changes)

PHP 5 offers an object-oriented programming model but still allows for the mingling of procedural style. PHP 6 moves farther into the object-oriented realm. One of the biggest changes here is that dynamic functions will no longer be permitted to be called with static syntax. There are any number of small, but important, changes to the way PHP handles its OO code that make it more consistent with other OO languages such as C++ and Java.

A few other changes are:

- Both XML Reader and XML Writer will be extensions in PHP 6, making it easier to work with XML files.
- The `register_globals`, `magic_quotes`, and `safe_mode` options in the `php.ini` file will no longer be available.
- The `ereg` extension, which provided another way to build regular expressions, is removed. Fortunately, the same `preg_match()` code covered in this book will be the main way to build regular expressions in PHP 6.
- A 64-bit integer type will be added.
- Multi-dimensional arrays will be able to use `foreach`.
- Version 6 of PHP is, more than anything, a version that cleans up and refines the language.

None of
book us
so you
about a
working

popular php apps

#10. Reusing other people's PHP

It's not always necessary to write your own PHP code from scratch. Sometimes it's best to reuse someone else's. The following are several popular and highly successful PHP-based software packages that you should consider using if you have a need and would prefer not reinventing the PHP wheel. Oh, and they're all free!

Drupal

One of the most impressive PHP projects to date, Drupal is a powerful content management system that can be used to build just about any kind of content-driven web site. NASA, The Onion, the Electronic Frontier Foundation, and Popular Science all use Drupal for their web sites. It's flexible enough to build just about anything that is heavy on content. Check it out at <http://drupal.org/>.

phpBB

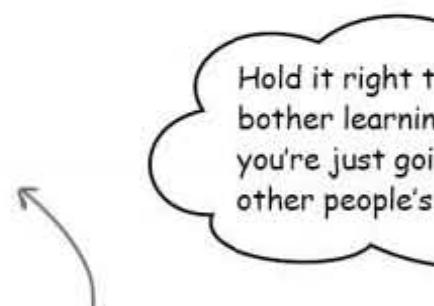
A category killer in the realm of online message boards (forums), phpBB is easy-does-it when it comes to building your own forum. It is extremely flexible and hard to beat at the one thing it does so well—managing threaded discussions. Find out more at <http://www.phpbb.com/>.

Coppermine Gallery

If image hosting is what you have in mind, Coppermine Gallery is the PHP application to check out. In an era of Flickr, Photobucket, Shutterfly, and Snapfish, hosting your own photo library sounds downright quaint. But with control comes power, and if you want complete control over your photos, take a look at Coppermine Gallery at <http://coppermine-gallery.net/>.

WordPress

One of the heavy hitters in the blogosphere, WordPress is PHP-based blogging software that lets you build and maintain a blog with minimal hassle. There's lots of competition out there, so you might want to do some exploring, but you could do worse than to pick WordPress if you're launching a blog. Download it at <http://wordpress.org/>.



Hold it right t
bother learning
you're just goi
other people's

Another really nice
PHP-based content
management system is
Joomla!, which you can
learn about at
<http://www.joomla.org/>

**Because reusing c
as simple as it sou
requires PHP skills.**

Many PHP software packa
customization, and that of
PHP development skills. N
may elect to only reuse a s
someone else's code, or no
way, by having PHP know
and options are always a g

appendix ii: set up a development environment

A place to play



You need a place to practice your newfound PHP and MySQL skills without making your data vulnerable on the web. It's always a good idea to have a safe place to develop your PHP application before unleashing it on the world (wide web). This appendix contains instructions for installing a web server, MySQL, and PHP to give you a safe place to work and practice.

installing php & mysql locally

Create a PHP development environment

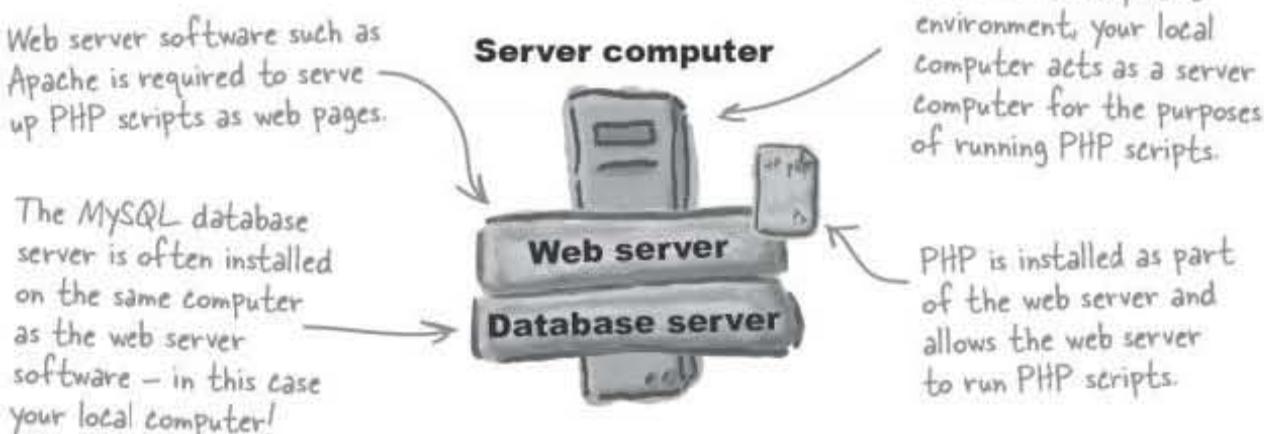
Before you can put your finished application on the web, you need to develop it. And it's never a good idea to develop your web application on the Web where everyone can see it. You can **install software locally that lets you build and test your application before you put it online.**

There are three pieces of software you'll need on your local computer to build and test PHP applications:

1. A web server
2. PHP
3. A MySQL database server

PHP isn't a server; it's a set of rules that your web server understands that allow it to interpret PHP code. Both the web server and the MySQL server are executable programs that run on a computer.

Keep in mind that we're talking about setting up your **local computer** as a web server for PHP development. You'll ultimately still need an **online web server** to upload your finished application to so that other people can access and use it.



Find out what you have

Before trying to install any of the pieces of the PHP development puzzle, your best bet is to first evaluate what you already have installed. Let's take a look at the three pieces and how you can tell what's already on your system.

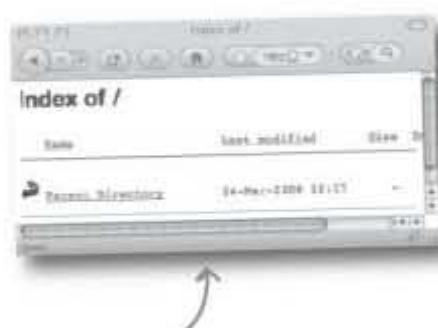
The platform of your local computer makes a big difference when it comes to what's already installed. For example, Mac OS X has a web server installed by default, while most Windows computers do not.

NOTE: This covers Windows Vista, Windows 2003/2008, Windows 7, For Mac, it OS X 10.3.

set up a dev

Do you have a web server?

You probably already have a web server if you are using a newer PC or Mac. To find out quickly on either system, open a browser window and type `http://localhost` in the address bar. If you get an introductory page, that means your web browser is alive and well on your local machine.



If you have a Mac or Windows machine with the Apache web server installed, you might see something like this.



If you have a Windows machine with IIS, you see something like this:

Do you have PHP? Which version?

If you have a web server, you can check to see if you have PHP installed very easily, as well as which version you have. Create a new script named `info.php` and type this in it:

```
<?php phpinfo(); ?>
```

Save this file to the directory your web server uses. On Windows it's typically:

C: \inetpub\wwwroot\

On the Mac, it's usually something like:

/Users/yourname/sites/

If you try to open this file in your browser by typing `http://localhost/info.php`, you'll see something like this if you have PHP installed:

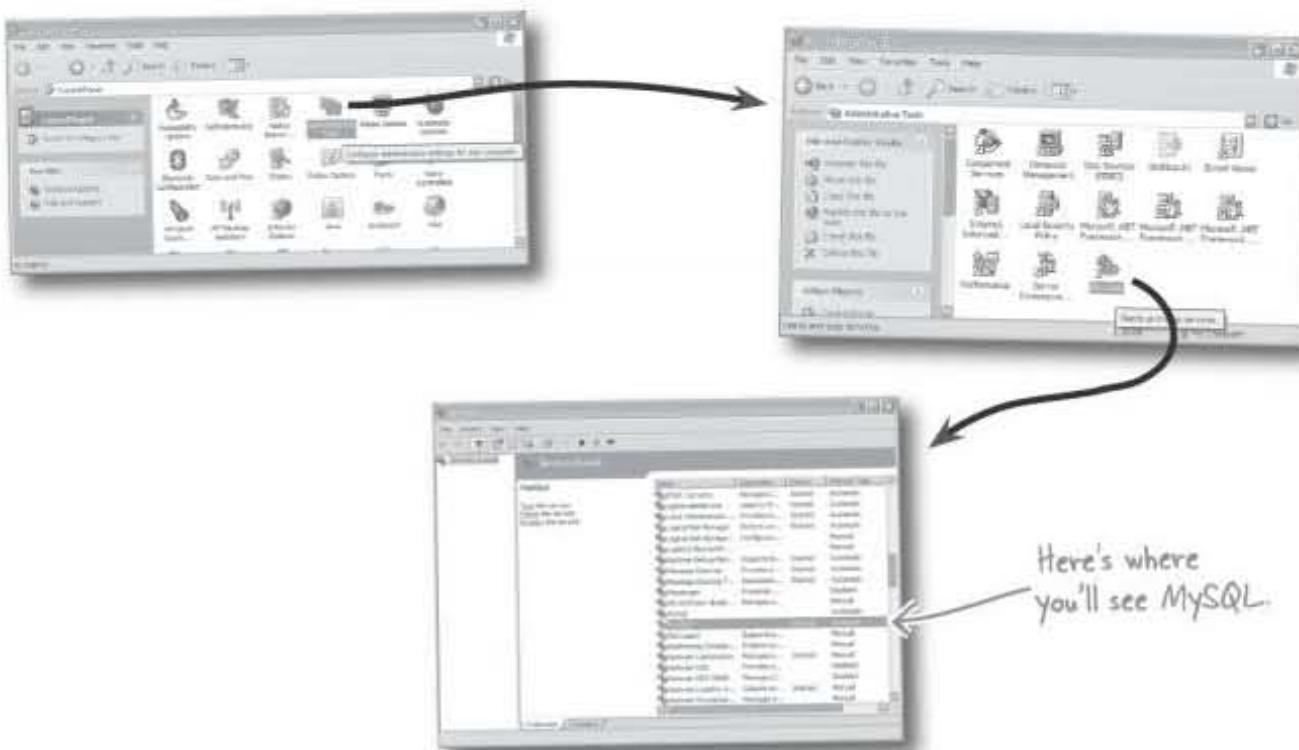
Here's the version of
PHP you have installed



checking your mysql version

Do you have MySQL? Which version?

On Windows, you can tell by opening the **Control Panel --> Administrative Tools --> Services**



To determine if you have MySQL on the Mac, open your terminal and type:

```
cd /user/local/mysql
```

If the command works, you have MySQL installed. To check the version, type:

```
mysql
```

If this command succeeds, it means MySQL is installed.

Here's the version of MySQL you have installed.

The terminal window displays the following text:

```
File Edit Window Help iHeartPHP
$ cd /usr/local/mysql
$ mysql
Welcome to the MySQL monitor. Commands end with ; or \
Your MySQL connection id is 3
Server version: 5.0.51b MySQL Community Server (GPL)
Type 'help;' or '\?' for help. Type '\c' to clear the buffer.
mysql>
```

A callout bubble with the text "Here's the version of MySQL you have installed." points to the line "Server version: 5.0.51b MySQL Community Server (GPL)". Another callout bubble with the text "If this command succeeds, it means MySQL is installed." points to the first line of the command input "\$ cd /usr/local/mysql".

Start with the Web Server

Depending on the version of Windows you have, you can download Microsoft's Internet Information Server (IIS), or the open source Apache web server. If you need a server on the Mac, you should probably go with Apache since it's already installed.

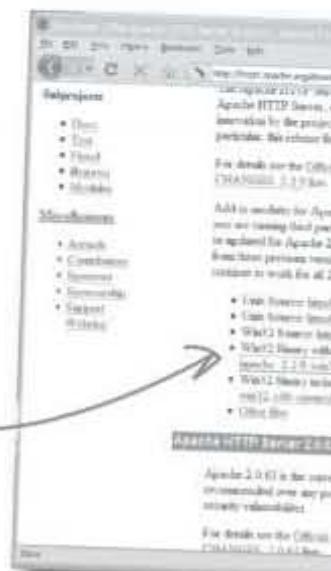
Here's a brief overview of installing Apache on Windows:

Head over to <http://httpd.apache.org/download.cgi>

If you're using Windows, we suggest you download the `apache_2.2.9-win32-x86-no_ssl-r2.msi` file. This will automatically install Apache for you after you download and double click it.

*Grab this version
and double click
on it after you've
downloaded it.*

Next you'll see the Installation Wizard. Most of the instructions are straightforward, and you can accept the default choices.



Choose the domain your computer is on. If you don't have one, you can enter `localhost`.



Your best bet is to choose the typical installation option.



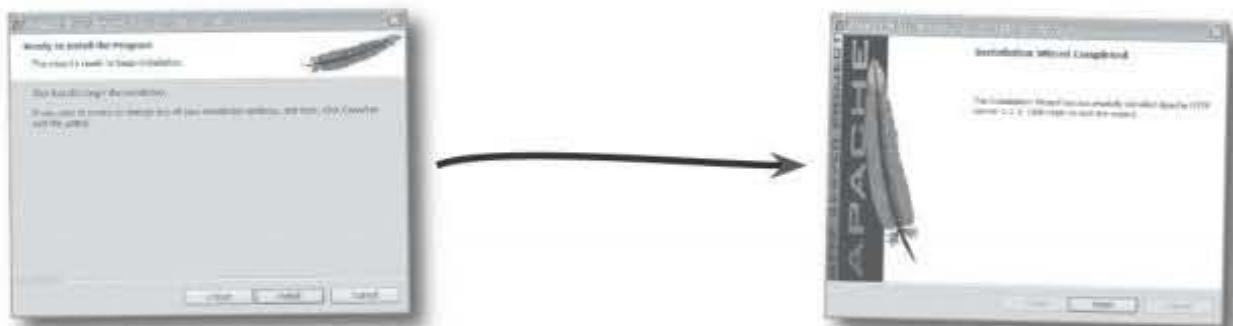
You can use the default installation.



installing php

Apache installation... concluded

You're nearly finished. Click **Install** and wait a minute or so for the installation to complete. That's it!

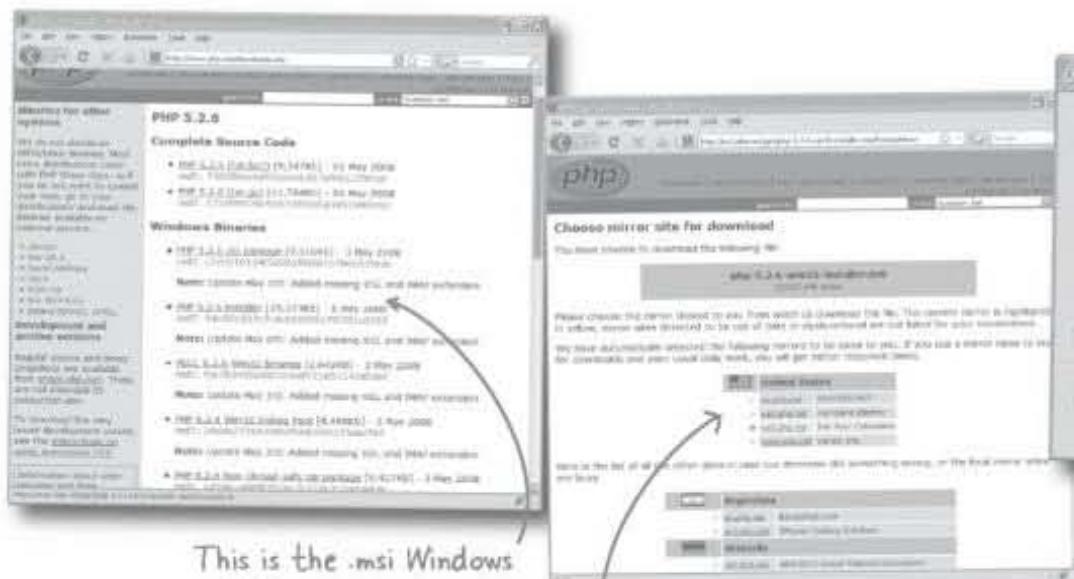


Your web server is set to start automatically when you start up your computer. But you can control it using the **Services** panel by stopping and starting it in the the **Control Panel --> Administrative Tools --> Services** dialogue where it will now show up.

PHP installation

Go to <http://www.php.net/downloads.php>.

Just as with Apache, if you're using Windows, we suggest you download the Windows installer version, `php-5.2.6-win32-installer.msi`. This will automatically install PHP for you after you download and double click it.



This is the .msi Windows installer version.

After you click the file, click one of the locations and download it.

After you've double clicked the button to...

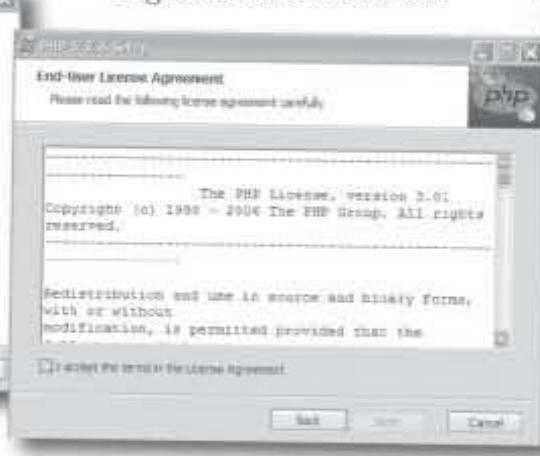
set up a dev

PHP installation steps

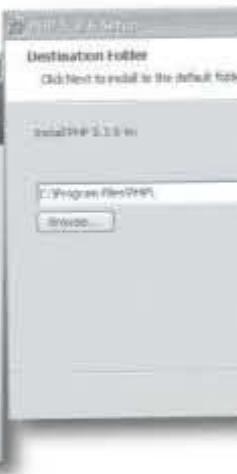
It starts with a basic Setup.



Accept the License Agreement to continue.



Selecting the folder is usual



Careful on this screen. If you're using Apache, select the right version. If you're using IIS, you will probably select the IISAPI module. Check with your particular software to determine exactly what you need.



This next screen is also tricky. You need to scroll down to **Extensions** and choose **MySQLi**. This will enable the built in PHP mysqli functions that we use throughout.



Scroll down below "Extensions" and click on the "Entire feature".

installing mysql on windows

PHP installation steps... concluded

That's it. Click on **Install**, then **Done** to close the installer.

Now try looking at your localhost/info. browser and see what happens.



Installing MySQL

Instructions and Troubleshooting

You still need MySQL, so let's work through the downloading and installing of MySQL. The official name for the free version of the MySQL RDBMS server these days is **MySQL Community Server**.

The following is a list of steps for installing MySQL on Windows and Mac OS X. This is **not** meant to replace the excellent instructions found on the MySQL web site, and **we strongly encourage you to go there and read them!** For much more detailed directions, as well as a troubleshooting guide, go here:

<http://dev.mysql.com/doc/refman/6.0/en/windows-installation.html>

You'll also like the MySQL Query Browser we talked about. There, you can type your queries and see the results inside the software interface, rather than in a console window.

Steps to Install MySQL on Windows

1

Go to:

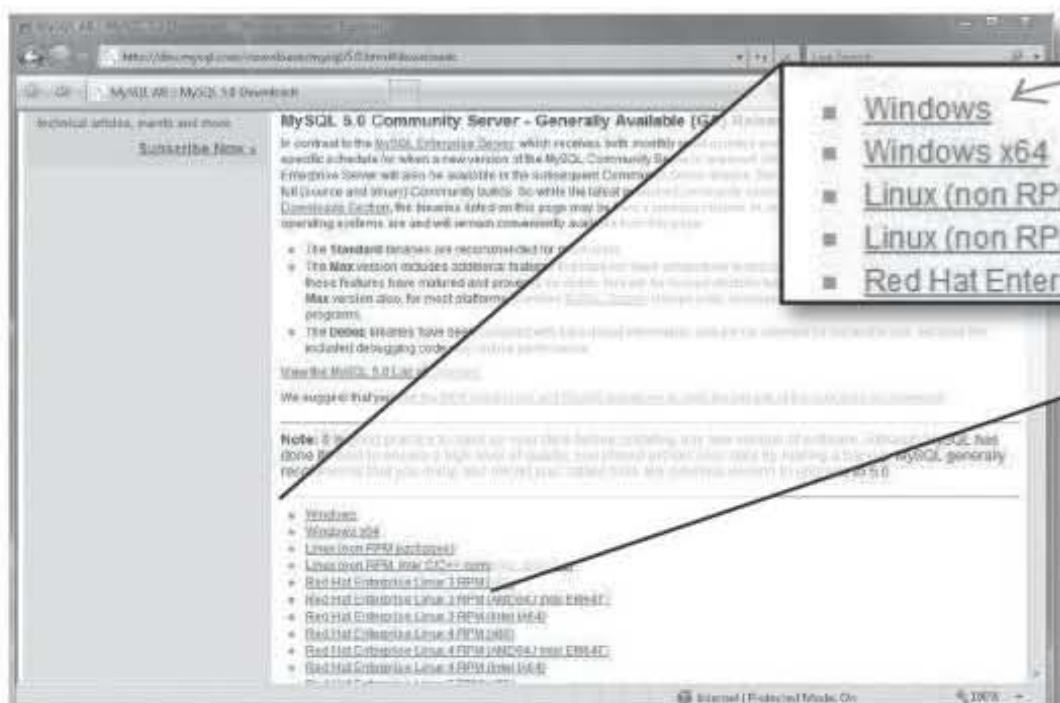
<http://dev.mysql.com/downloads/mysql/6.0.html>

and click on the **MySQL Community Server** download button.



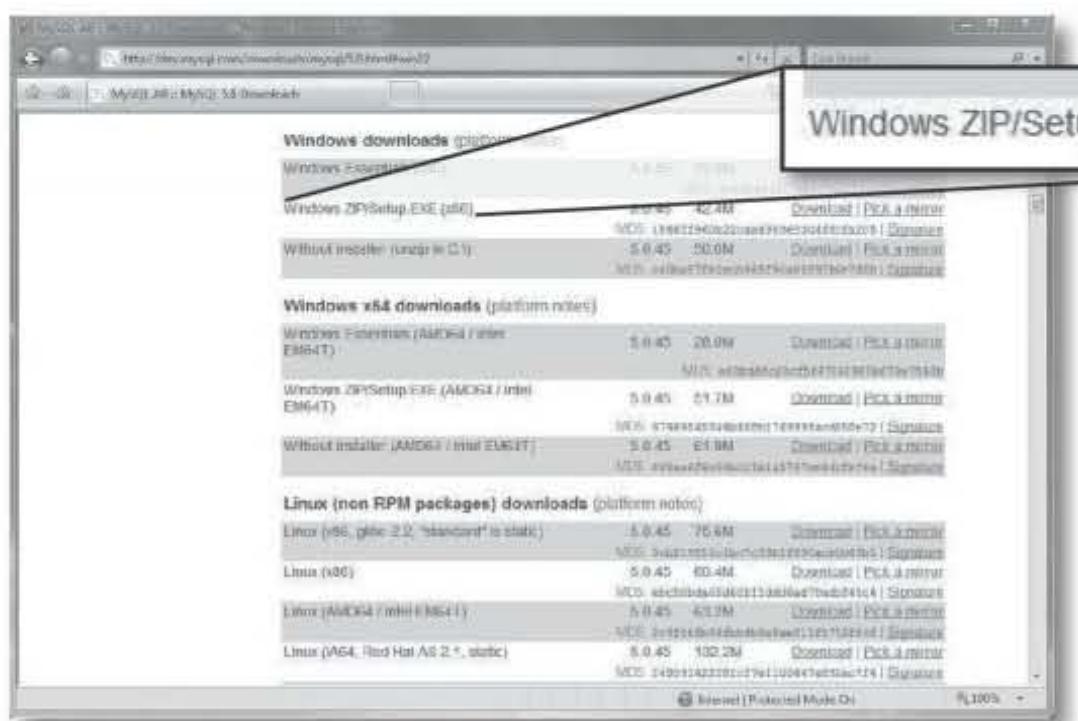
2

Choose **Windows** from the list.



*installing mysql on windows (continued)***Download your installer**

- 3** Under **Windows downloads**, we recommend that you choose the Windows ZIP/Setup.EXE option because it includes an installer that greatly simplifies the installation. Click on **Pick a Mirror**.



- 4** You'll see a list of locations that have a copy you can download; choose the one closest to you.
- 5** When the file has finished downloading, double-click to launch it. At this point, you will be walked through the installation with the **Setup Wizard**. Click the **Next** button.



When you've double-clicked the file and the Setup Wizard appears, click the Next button.

Pick a destination folder

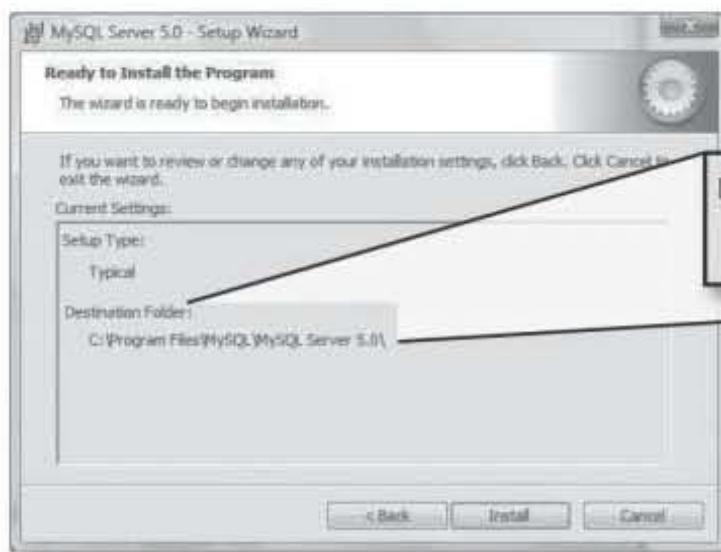
6

You'll be asked to choose *Typical*, *Complete*, or *Custom*. For our purposes in this book, choose *Typical*.

You can change the location on your computer where MySQL will be installed, but we recommend that you stay with the default location:

`C:\Program Files\MySQL\MySQL Server 6.0`

Click the **Next** button.



Click "Install" and you're done!

7

You'll see the *Ready to Install* dialog with the *Destination Folder* listed. If you're happy with the destination directory, click **Install**. Otherwise, go **Back**, **Change** the directory, and return here.

Click **Install**.

installing mysql on mac os x

Enabling PHP on Mac OS X

PHP is included on Macs with OS X version 10.5+ (Leopard), but it's not enabled by default. You have to access the main Apache configuration file and comment out a line of code in order to get PHP going. This file is called `http.conf`, and is a hidden file located down inside the Apache install folder.

You're looking for the following line of code, which has a pound symbol (#) in front of it to comment it out:

```
#LoadModule php5_module libexec/apache2/libphp5.so
```

You need to remove the pound symbol and restart the server to enable PHP. The `http.conf` document is owned by "root," which means you'll have to enter your password to change it. You'll probably also want to tweak the `php.ini` file so that Apache uses it. For more detailed information about how to carry out these steps and enable PHP, visit http://foundationphp.com/tutorials/php_leopard.php.

Steps to Install MySQL on Mac OS X

If you are running Mac OS X Server, a version of MySQL should already be installed.

Before you begin, check to see if you already have a version installed. Go to **Applications/Server/MySQL Manager** to access it.

- 1 Go to:

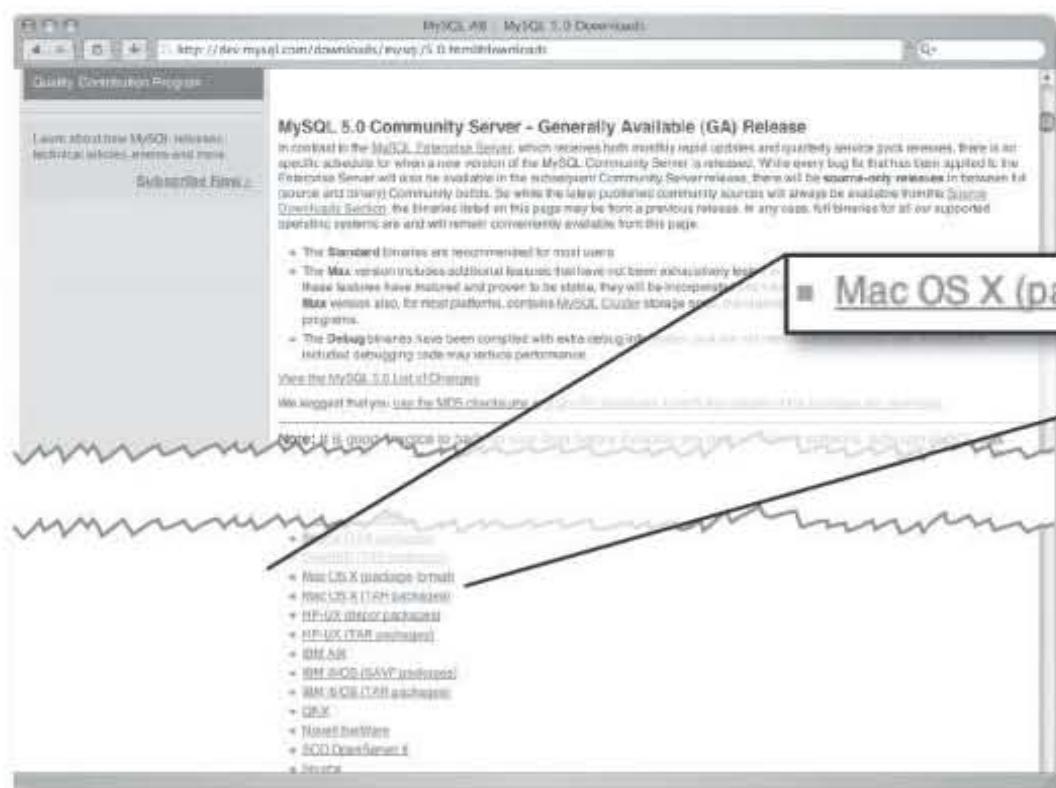
<http://dev.mysql.com/downloads/mysql/6.0.html>

and click on the MySQL Community Server **Download** button.



2

Choose **Mac OS X (package format)** from the list.

**3**

Choose the appropriate package for your Mac OS X version.
Click on **Pick a Mirror**.

4

You'll see a list of locations that have a copy you can download; choose the one closest to you.

5

When the file has finished downloading, double-click to launch it. You can now open a Terminal window on your Mac and type:

```
shell> cd /usr/local/mysql
shell> sudo ./bin/mysqld_safe
```

(Enter your password, if necessary)

(Press Control-Z)

```
shell> bg
```

(Press **Control-D** or enter **exit** to exit the shell)

If you're using a GUI tool such as phpMyAdmin, check its documentation for how to access it once MySQL is successfully installed.

making your site live

Moving from production to a live site

You've spent days or weeks working on your site, and you feel it's ready to go live. To move your PHP and MySQL site from your local computer to the web requires a little planning and a few specific techniques.

First, you need to make sure that the place your site is going has the same versions of PHP and MySQL you expect. If not, you may need to make your code to match what is available. Most of the code in this book is portable, but you may need to retrofit your PHP code back to the mysql functions, as opposed to the mysqli functions we use in this book. If that's the problem, check out #1 of The Top Ten Topics (we didn't cover) for more information.

If the software on your live site is compatible, then moving your site over is simple. Here are the steps:

1. Upload the PHP files from your production server to the web directory on your live server. Keep the file structure intact, and make sure you don't lose any folders you might have created to contain your included files.
2. Do a database dump (which we'll show you in a moment) to get the MySQL statements you need to create your tables and the INSERT statements you need to move your data from the table on the production server to the live server.
3. Log in to your live database where you can run the CREATE and INSERT MySQL statements to move your data from your local site to the live site.
4. Modify any database connection code in your PHP files to point at the live database server. If you don't change this, your live code will try to connect to your production site and won't be able to connect.

Change those `mysqli_connect()` statements to point at your MySQL server associated with your live site, along with the correct username and password to get you connected.

Dump your data (and your tables)

You've FTP'ed your PHP files to the live server, but your data is still not on the live site's MySQL server. When your table is full of data, the idea of moving it to another MySQL server can be daunting. Fortunately, bundled with MySQL is the **MySQLdump** program, which gives you an easy way to recreate the CREATE TABLE statement that can recreate your table and all the INSERT statements with the data in your table. You simply need to use the MySQLdump program. To make a copy of your data that you can move to another MySQL server, type this in your terminal:

```

File Edit Window Help DumpYourData
$ mysqldump
Usage: mysqldump [OPTIONS] database [tables]
OR      mysqldump [OPTIONS] --databases [OPTIONS] DB1 [DB2 DB3]
OR      mysqldump [OPTIONS] --all-databases [OPTIONS]
For more options, use mysqldump --help

$mysqldump riskyjobs jobs > riskyjobstable.sql

```

This sends the CREATE TABLE statement for the jobs table to a text file we just created named `riskyjobstable.sql`. If you leave off the `>riskyjobstable.sql` part, then the CREATE TABLE and INSERT statements will simply scroll by you on the screen in your terminal. Try it to see what we mean. It's not very useful, but you'll see all your data fly by, nicely formatted in INSERT statements.

Once you've sent all that data to your new file using the greater than sign, you can grab that file and use the contents as MySQL queries at your hosting site to move your tables and your data.

Prepare to use your dumped data

Get ready to move your data by running a CREATE DATABASE statement on your live MySQL statement. Then run a USE DATABASE on your new database. Now you are ready to move your data from your production server to your live server.

putting mysql data on the live server

Move dumped data to the live server

You've created a file called `riskyjobstable.sql` that contains MySQL statements that create your table and insert data into it. The file `riskyjobstable.sql` probably looks a bit like this:

```

These are all comments, you can ignore them.          -- MySQL dump 10.11
-- Host: localhost      Database: riskyjobs
-- -----
-- Server version      5.0.51b
-- 
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT
*;
-- 
-- Table structure for table `jobs`
-- 
DROP TABLE IF EXISTS `jobs`;
CREATE TABLE `jobs` (
  `job_id` int(11) NOT NULL auto_increment,
  `title` varchar(200) default NULL,
  `description` blob,
  `city` varchar(30) default NULL,
  `state` char(2) default NULL,
  `zip` char(5) default NULL,
  `co_id` int(11) default NULL,
  PRIMARY KEY  (`job_id`)
) ENGINE=MyISAM AUTO_INCREMENT=14 DEFAULT CHARSET=utf8;
-- 
-- Dumping data for table `jobs`
-- 
LOCK TABLES `riskyjobs` WRITE;
/*!40000 ALTER TABLE `riskyjobs` DISABLE KEYS */;
INSERT INTO `riskyjobs` VALUES (8,'Custard Walker','We need people willing to test the theory that you can walk on custard.\r\n\r\nWe\'re going to fill a swimming pool with custard, and you\'ll walk on it. \r\n\r\nCustard and other kinds of starchy fluids are known as non-Newtonian fluids. They become solid under high pressure (your feet while you walk) while remaining in their liquid form otherwise.\r\n\r\nTowel provided, own bathing suit, a must.\r\n\r\nNote: if you stand on for too long on the custard\'s surface, you will slowly sink. We are not liable for any custard sinkages;');

The mysqldump always writes a DROP statement to start with a clean slate before doing a CREATE and INSERT.
Mysqldump makes a single INSERT statement that inserts every row in the table.

```

If you know there isn't a table named "jobs" where you are creating this one, you can ignore this command.

Here's the CREATE TABLE statement.

You can ignore this statement and copy/paste starting at INSERT statement.

Take the entire text of the .sql file and paste it into your MySQL or the query window of your MySQL graphical client (like phpMyAdmin).

This performs the queries in the file. In the case of the example on this page, the dump contains a CREATE TABLE statement and an INSERT statement. Along the way, the query tells your MySQL server to drop any existing table and to LOCK (or keep anyone from) table while you INSERT the new data.

Connect to the live server

You've moved your PHP files to your live site. You've taken your table structures as CREATE TABLE statements and your data as a massive INSERT statement from the mysqldump and executed them on your live web server, so your data has been moved.

There's a small step left. The PHP code you FTP'ed to your live web site isn't connecting to your live MySQL server.

You need to change the connection string in your mysqli_connect() function to point to your live MySQL server. Anywhere in your PHP code where you call the mysqli_connect() function, you'll need to change it.

```
$dbc = mysqli_connect('localhost', 'myusername', 'mypassword');
or die('Error connecting to MySQL server.');
```

This will be the name or IP address of your live site. It will only be "localhost" if your MySQL server is on the same machine as your PHP pages.

And these will be the password that connects to your live MySQL server.

That's it!

- You've copied your FTP files to your web server,
- you've dumped your tables and data into a .sql file,
- you've run the queries in the .sql file on your live MySQL server,
- and you've changed your PHP file to call your live MySQL server database.

Your site should now be live!

appendix iii: extend your php

Get even more



I know I have everything
any run-of-the-mill,
heartbreakingly beautiful,
fiendishly clever femme fatale
needs, but it's not enough.

Yes, you can program with PHP and MySQL and create great web applications. But you know there must be more to it. And there is. This short appendix will show you how to install the mysqli extension and GD graphics library extension. Then we'll mention a few more extensions to PHP you might want to get. Because sometimes it's okay to want more.

installing new php modules

Extending your PHP

This book discusses installing both the mysqli and GD modules on Windows. In this section, we'll show you how to see what modules you have, how to get GD or mysqli if you are missing them, and how to install them in Windows. Unfortunately, installing these modules on a Mac or Linux system is kinda tricky. More on that at the end of this appendix.

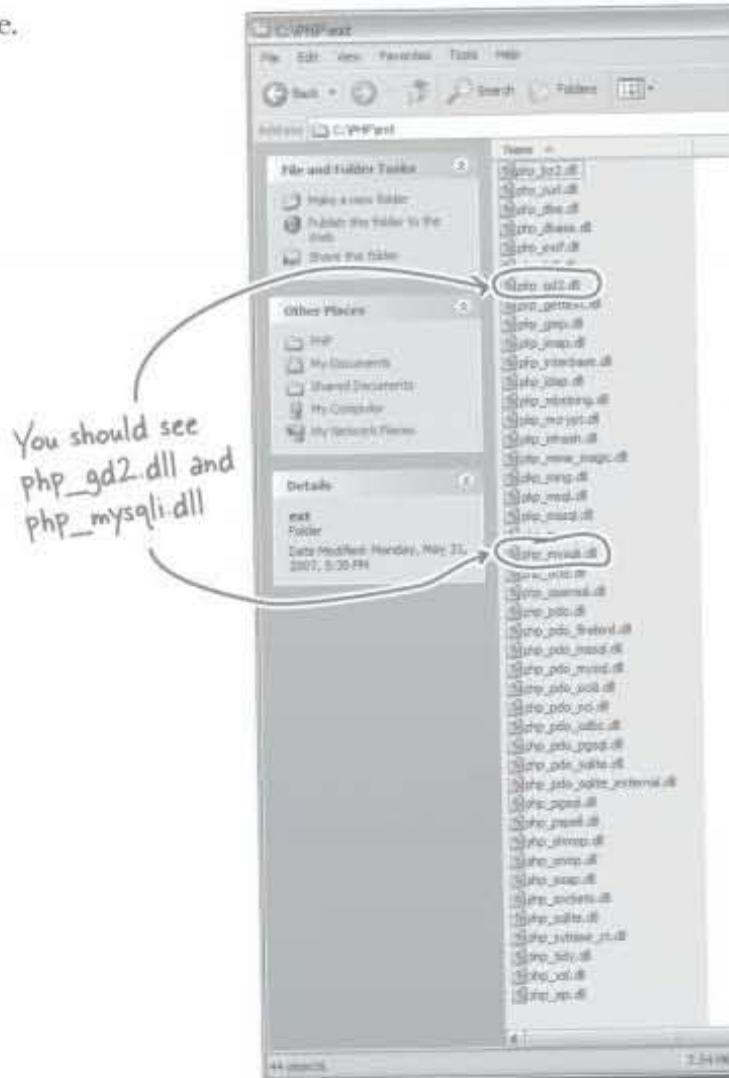
NOTE: This appendix covers Windows 2000, XP, Vista, Windows Server 2003/2008, or other 32-bit Windows operating system.

If you're using Windows, you're in luck

You probably already have both the mysqli and GD modules on your computer. And even if you don't, adding them is relatively easy. We'll show you how to check to see what you have, if you're missing one of them, how to get it, and how to activate one or both modules.

It starts with checking to see what you have.

- 1 First, figure out if GD or mysqli is on your system. To do that, begin by navigating to the directory where the PHP extensions are installed. They are typically in the C:/PHP/ext directory, although the path may be different on your machine. Open the ext directory and look for `php_gd2.dll` and `php_mysqli.dll`. In general, these are installed with PHP 5 and later, and simply need to be activated. If you have them, great, move on to step 3. If not, go to step 2.



2

If you're missing either `php_mysqli.dll` or `php_gd2.dll`, you'll have to get it. Chances are you already have both DLLs on your machine, but if you don't, you can find `php_gd2.dll` at: <http://www.libgd.org/Downloads>. Download it and copy it to the folder ext under your PHP install. In our examples, it's located at `C:/PHP/ext`.

You can get the `mysqli` extension from MySQL.com. First, browse to <http://www.mysql.com>. Click on **Downloads** (along the top) --> **Connectors** (it's in the left menu) --> **MySQL native driver for PHP** --> **Download** `php_mysqli.dll` for PHP 5.2.1 (Windows) (Make sure this is your version).



3

By now you should have `php_mysqli.dll` and `php_gd2.dll` copied to your `/PHP/ext` folder. We need to tell our `php.ini` file to use these DLLs. To do that, browse to the directory it's in, and open the file in a text editor.

Sometimes your PHP install ends up in the Program Files\PHP directory. Find your `php.ini` file and open it for the next step.



installing new php modules (continued)

- 4** Dig through your `php.ini` file and locate the lines:

```
extension=php_gd2.dll
```

and

```
extension=php_mysqli.dll
```

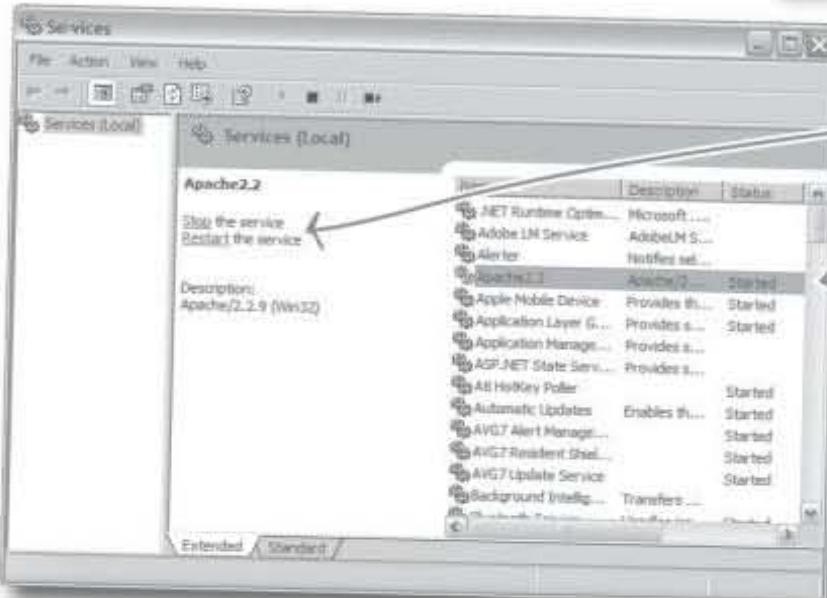
If either of these have semicolons (;) or pound signs (#) in front of them, that means they are commented out. Remove them and save your file.

Delete the semicolons from in front of these two lines if they have them. Then save your file.

```
Windows Extensions
Note that ODBC support is built in, so you can connect to a MySQL database using the PHP MySQL extension.
Note that many DLL files are located in the ext directory and its subfolders.
Be sure to appropriately set the extension paths in your php.ini file.

;extension=php_bz2.dll
;extension=php_curl.dll
;extension=php_dba.dll
;extension=php_dbase.dll
;extension=php_exif.dll
;extension=php_fdf.dll
;extension=php_gd2.dll
;extension=php_gettext.dll
;extension=php_gmp.dll
;extension=php_ifx.dll
;extension=php_imap.dll
;extension=php_interbase.dll
extension=php_ldap.dll
;extension=php_mbstring.dll
;extension=php_mcrypt.dll
;extension=php_mhash.dll
;extension=php_mime_magic.dll
;extension=php_ming.dll
;extension=php_msqli.dll
;extension=php_mssql.dll
extension=php_mysql.dll
extension=php_mysql.dll
extension=php_ocis.dll
extension=php_openssl.dll
extension=php_pdo.dll
extension=php_pdo_firebird.dll
extension=php_pdo_mssql.dll
extension=php_pdo_mysql.dll
extension=php_pdo_oci.dll
extension=php_pdo_ocis.dll
extension=php_pdo_odbc.dll
extension=php_pdo_pgsql.dll
extension=php_pdo_sqlite.dll
extension=php_pgsql.dll
extension=php_psspell.dll
extension=php_shmop.dll
extension=php_snmp.dll
extension=php_soap.dll
extension=php_sockets.dll
extension=php_sqlite.dll
extension=php_sybase_ct.dll
extension=php_tidy.dll
extension=php_xmlrpc.dll
extension=php_xsl.dll
extension=php_zip.dll
```

- 5** The last step is to restart your Apache web server so that the changes you made to your `php.ini` file will take effect. To do this, go to your Windows Control Panel, double-click on **Administrative Tools**, then click **Services**. You should see this:



Select Apache and...

Click the **Apache** service on **Restart** from the left. The next time you use GD or mysqli functions, they will work correctly.

And on the Mac...

Unfortunately, it's quite a bit more difficult. Adding modules on the Mac means recompiling the PHP source code and passing in arguments to add in the modules you want. There are simply too many possible combinations of Mac operating systems and PHP versions to include in this short appendix. There is a terrific guide that may help you install the GD module located here:

<http://macoshelp.blogspot.com/2008/02/adding-gd-library-for-mac-os-x.html>

It will only work if you have the right OS X version (Leopard), and the right PHP version (5). If you don't, or the instructions don't work for you, you may want to dig through the comments on that site and on the original GD website, <http://www.libgd.org/>, for more detailed and specific installation instructions for your flavor of OS X and PHP.

For help in adding mysqli to your Mac version of PHP, which also means recompiling PHP, we recommend the instructions here:

<http://dev.mysql.com/downloads/connector/php-mysqlnd/>

Keep in mind
installing the
only applies if
web server or
development:
using a Mac
is being upload
other server,