Part I

Introducing PHP

IN THIS PART

Chapter 1

Why PHP and MySQL?

This first chapter is an introduction to PHP, MySQL, and the interaction of the two. In it, we'll try to address some of the most common questions about these tools, such as "What are they?" and "How do they compare to similar technologies?" Most of the chapter is taken up with an enumeration of the many, many reasons to choose PHP, MySQL, or the two in tandem. If you're a techie looking for some ammunition to lob at your PHB ("Pointy-Haired Boss," for those who don't know the Dilbert cartoons) or a manager asking yourself what is this P-whatever thing your geeks keep whining to get, this chapter will provide some preliminary answers.

IN THIS CHAPTER

Understanding PHP and MySQL

The benefits of using PHP and MySQL

What Is PHP?

PHP is the web development language written by and for web developers. PHP stands for *PHP: Hypertext Preprocessor.* The product was originally named *Personal Home Page Tools*, and many people still think that's what the acronym stands for, but as it expanded in scope, a new and more appropriate (albeit GNU-ishly recursive) name was selected by community vote. PHP is currently in its sixth major rewrite, called PHP6 or just plain PHP.

PHP is a server-side scripting language, usually used to create web applications in combination with a web server, such as Apache. PHP can also be used to create command-line scripts akin to Perl or shell scripts, but such use is much less common than PHP's use as a web language.

Strictly speaking, PHP has nothing to do with layout, events, on-the-fly Document Object Model (DOM) manipulation, or really anything about the look and feel of a web page. In fact, most of what PHP does is invisible to the end user. Someone looking at a PHP page will not necessarily be able to tell that it was not written purely in Hypertext Markup Language (HTML), because the result of PHP is usually HTML.

What Is MySQL?

MySQL (pronounced My Ess Q El) is an open source, SQL relational database management system (RDBMS) that is free for many uses (more detail on that later). Early in its history, MySQL occasionally faced opposition because of its lack of support for some core SQL constructs such as subselects and foreign keys. Ultimately, however, MySQL found a broad, enthusiastic user base for its liberal licensing terms, perky performance, and ease of use. Its acceptance was aided in part by the wide variety of other technologies such as PHP, Perl, Python, and the like that have encouraged its use through stable, well-documented modules and extensions.

Databases are generally useful, perhaps the most consistently useful family of software products (the "killer product") in modern computing. Like many competing products, both free and commercial, MySQL isn't a database until you give it some structure and form. You might think of this as the difference between a database and an RDBMS (that is, RDBMS plus user requirements equal a database).

There's lots more to say about MySQL, but then again, there's lots more space in which to say it.

Deciding on a Web Application Platform

There are many platforms upon which web applications can be built. This section compares PHP to a few other platforms and highlights some of PHP's and MySQL's strengths.

Cost

PHP is one of the "P's" in the popular LAMP stack. The LAMP stack refers to the popular combination of Linux, Apache, MySQL, and PHP/Perl/Python that runs many web sites and powers many web applications. Many of the components of the LAMP stack are free, and PHP is no exception. PHP is free, as in there is no cost to develop in and run programs made with PHP. Though MySQL's license and costs have changed, you can obtain the Community Server edition for free. MySQL offers several levels of support contracts for their database server. More information can be obtained at `www.mysql.com`. Both PHP and MySQL run on a variety of platforms, including many variants of Linux, Microsoft Windows, and others. Running on an operating system such as Linux gives the opportunity for a completely free web application platform, with no up-front costs.

Of course, when talking about software development and application platforms, the up-front cost of software licensing is only a portion of the total cost of ownership (TCO). Years of real-world experience with Linux, Apache, MySQL, and PHP in production environments has proved that the total cost of maintaining these platforms is lower, many times much lower, than maintaining an infrastructure with proprietary, non-free software.

Ease of Use

When compared to many other programming languages, PHP makes it easy to develop powerful web applications quickly (this is a blessing and a curse). Many of the most useful specific functions (such as those for opening a connection to an Oracle database or fetching e-mail from an Internet Message Access Protocol [IMAP] server) are predefined for you. A lot of complete scripts are waiting out there for you to look at as you're learning PHP.

Most advanced PHP users (including most of the development team members) are diehard hand-coders. They tend to share certain gut-level, subcultural assumptions — for instance, that hand-written code is beautiful and clean and maximally browser-compatible and therefore the only way to go — that they do not hesitate to express in vigorous terms. The PHP community offers help and trades tips mostly by e-mail, and if you want to participate, you have to be able to parse plain-text source code with facility. Some WYSIWYG users occasionally ask list members to diagnose their problems by looking at their web pages instead of their source code, but this rarely ends well.

That said, let us reiterate that PHP really is easy to learn and write, especially for those with a little bit of experience in a C-syntaxed programming language. It's just a little more involved than HTML. This small learning curve means that relatively inexperienced programmers can sometimes make mistakes that turn into large security issues. This is the curse of PHP. While this book has no specific chapter dedicated to security, I feel that security needs to be applied at every layer, during every phase of programming. Therefore dedicating a single chapter would not do justice to the importance of web application security.

If you have no relational database experience, or are coming from an environment such as Microsoft Access, MySQL's command-line interface and lack of implicit structure may at first seem a little daunting. MySQL has a few GUI (graphical user interface) tools to help work with databases. None of the GUI tools is a substitute for learning a little theory and employing good design practices, but that is a subject for another chapter.

HTML-embeddedness

PHP can be embedded within HTML. In other words, PHP pages are ordinary HTML pages that escape into PHP mode only when necessary. Here is an example:

```
<HEAD>
<TITLE>Example.com greeting</TITLE>
</HEAD>
<BODY>
<P>Hello,
<?php
// We have now escaped into PHP mode.
// Instead of static variables, the next three lines
// could easily be database calls or even cookies;
// or they could have been passed from a form.
$firstname = 'Joyce';
$lastname = 'Park';
```

```
$title = 'Ms.';
echo "$title $lastname";
// OK, we are going back to HTML now.
?>
.  We know who you are!  Your first name is <?php echo
$firstname; ?>.</P>

<P>You are visiting our site at <?php echo date('Y-m-d H:i:s');
?></P>

<P>Here is a link to your account management page:  <A
HREF="http://www.example.com/accounts/<?php echo
"$firstname$lastname"; ?>/"><?php echo $firstname; ?>'s account
management page</A></P>
</BODY>
</HTML>
```

When a client requests this page, the web server *preprocesses* it. This means it goes through the page from top to bottom, looking for sections of PHP, which it will try to resolve. For one thing, the parser will suck up all assigned variables (marked by dollar signs) and try to plug them into later PHP commands (in this case, the echo function). If everything goes smoothly, the preprocessor will eventually return a normal HTML page to the client's browser, as shown in Figure 1-1.

FIGURE 1-1

A result of preprocessed PHP

If you peek at the source code from the client browser (select Source or Page Source from the View menu, it will look like this:

```
<HEAD>
<TITLE>Example.com greeting</TITLE>
```

```
</HEAD>
<BODY>
<P>Hello,
Ms. Park
.  We know who you are!  Your first name is Joyce.</P>

<P>You are visiting our site at 2002-04-21 19:34:24</P>

<P>Here is a link to your account management page:  <A HREF="http://
www.example.com/accounts/JoycePark/">Joyce's account management page</
A></P>
</BODY>
</HTML>
```

This code is exactly the same as if you were to write the HTML by hand. So simple!

The HTML-embeddedness of PHP has many helpful consequences:

- PHP can quickly be added to code produced by WYSIWYG editors.
- PHP lends itself to a division of labor between designers and programmers.
- Every line of HTML does not need to be rewritten in a programming language.
- PHP can reduce labor costs and increase efficiency because of its shallow learning curve and ease of use.

Cross-platform compatibility

PHP and MySQL run native on every popular flavor of Linux/Unix (including Mac OS X) and Microsoft Windows. A huge percentage of the world's Hypertext Transfer Protocol (HTTP) servers run on one of these two classes of operating systems.

PHP is compatible with the leading web servers: Apache HTTP Server for Linux/Unix and Windows and Microsoft Internet Information Server. It also works with several lesser-known servers. Specific web server compatibility with MySQL is not required, since PHP will handle all the dirty work for you.

Stability

The word *stable* means two different things in this context:

- The server doesn't need to be rebooted or restarted often.
- The software doesn't change radically and incompatibly from release to release.

To our advantage, both of these connotations apply to both MySQL and PHP.

Apache Server is generally considered the most stable of major web servers, with a reputation for enviable uptime percentages. Most often, a server reboot isn't required for each setting change. PHP inherits this reliability; plus, its own implementation is solid yet lightweight.

PHP and MySQL are also both stable in the sense of feature stability. Their respective development teams have thus far enjoyed a clear vision of their project and refused to be distracted by every new fad and ill-thought-out user demand that comes along. Much of the effort goes into incremental performance improvements, communicating with more major databases, or adding better OOP support. In the case of MySQL, the addition of reasonable and expected new features has hit a rapid clip. For both PHP and MySQL, such improvements have rarely come at the expense of compatibility.

Many extensions

PHP makes it easy to communicate with other programs and protocols. The PHP development team seems committed to providing maximum flexibility to the largest number of users.

Database connectivity is especially strong, with native-driver support for about 15 of the most popular databases plus Open DataBase Connectivity (ODBC). In addition, PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. Earlier versions of PHP added support for Java and distributed object architectures (Component Object Model [COM] and Common Object Request Broker Architecture [CORBA]), making n-tier development a possibility for the first time, fully incorporated GD graphics library and revamped Extensible Markup Language (XML) support with DOM and simpleXML.

Fast feature development

Users of proprietary web development technologies can sometimes be frustrated by the glacial speed at which new features are added to the official product standard to support emerging technologies. With PHP, this is not a problem. All it takes is one developer, a C compiler, and a dream to add important new functionality. This is not to say that the PHP team will accept every random contribution into the official distribution without community buy-in, but independent developers can and do distribute their own extensions that may later be folded into the main PHP package in more or less unitary form. For instance, Dan Libby's elegant xmlrpc-epi extension was adopted as part of the PHP distribution in version 4.1, a few months after it was first released as an independent package.

PHP development is also constant and ongoing. Although there are clearly major inflection points, such as the transition between PHP4 and PHP5, these tend to be most important deep in the guts of the parser — people were actually working on major extensions throughout the transition period without critical problems. Furthermore, the PHP group subscribes to the open source philosophy of "release early, release often," which gives developers many opportunities to follow along with changes and report bugs.

Not proprietary

The history of the personal computer industry to date has largely been a chronicle of proprietary standards: attempts to establish them, clashes between them, their benefits and drawbacks for the consumer, and how they are eventually replaced with new standards.

In the past few years the Internet has demonstrated the great convenience of voluntary, standards-based, platform-independent compatibility. E-mail, for example, works so well because it enjoys a clear, firm standard to which every program on every platform must conform. New developments that break with the standard (for example, HTML-based e-mail stationery) are generally regarded as deviations, and their users find themselves having to bear the burdens of early adoption.

Furthermore, customers (especially the big-fish businesses with large systems) are fed up with spending vast sums to conform to a proprietary standard only to have the market uptake not turn out as promised. Much of the current momentum toward XML and web services is driven by years of customer disappointment with Java RMI (Remote Method Invocation), CORBA, COM, and even older proprietary methods and data formats.

Right now, software developers are in a period of experimentation and flux concerning proprietary versus open standards. Companies want to be sure that they can maintain profitability while adopting open standards. There have been some major legal conflicts related to proprietary standards, which are still being resolved. These could eventually result in mandated changes to the codebase itself or even affect the futures of the companies involved. In the face of all this uncertainty, a growing number of businesses are attracted to solutions that they know will not have these problems in the foreseeable future.

PHP is in a position of maximum flexibility because it is, so to speak, *antiproprietary*. It is not tied to any one server operating system, unlike Active Server Pages. It is not tied to any proprietary cross-platform standard or middleware, as is Java Server Pages or ColdFusion. It is not tied to any one browser or implementation of a programming language or database. PHP isn't even doctrinaire about working only with other open source software. This independent but cooperative pragmatism should help PHP ride out the stormy seas that seem to lie ahead.

Strong user communities

PHP is developed and supported in a collaborative fashion by a worldwide community of users. Some animals (such as the core developers) are more equal than others, but that's hard to argue with, because they put in the most work, had the best ideas, and have managed to maintain civil relationships with the greatest number of other users.

The main advantage for most new users is technical support without charge, without boundaries, and without the runaround. People on the mailing list are available 24/7/52 to answer your questions, help debug your code, and listen to your gripes. The support is human and real. PHP community members might tell you to read the manual, take your question over to the appropriate database mailing list, or just stop your whining — but they'll never tell you to wipe your C drive and then charge you for the privilege. Often, they'll look at your code and tell you what you're doing wrong or even help you design an application from the ground up.

As you become more comfortable with PHP, you may wish to contribute. Bug tracking, offering advice to others on the mailing lists, posting scripts to public repositories, editing documentation, and, of course, writing C code are all ways you can give back to the community.

MySQL, while open source licensed for non-redistributive uses, is somewhat less community driven in terms of its development. Nevertheless, it benefits from a growing community of users who are actively listened to by the development team. Rarely has a software project responded so vigorously to community demand, and the community of users can be extremely responsive to other users who need help. It's a point of pride with a lot of SQL gurus that they can write the complicated queries that get you the results you are looking for but had struggled with for days. In many cases, they'll help you for nothing more than the enduring, if small, fame that comes with the archived presence of their name on Google Groups. Try comparing *that* with $100 per incident support.

Summary

PHP and MySQL, individually or together, aren't the panacea for every web development problem, but they present a lot of advantages. PHP is built by web developers for web developers and supported by a large and enthusiastic community. MySQL is a powerful standards-compliant RDBMS that comes in at an extremely competitive price point, even more so if you qualify for free use. Both technologies are clear-cut cases of the community banding together to address its own needs.