CHAPTER 1

# Why PHP and MySQL?

✦ ✦ ✦ ✦

**In This Chapter**

✦ ✦ ✦ ✦

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.

## 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 fifth major rewrite, called PHP5 or just plain PHP.

PHP is a server-side scripting language, which can be embedded in HTML or used as a standalone binary (although the former use is much more common). Proprietary products in this niche are Microsoft's Active Server Pages, Macromedia's ColdFusion, and Sun's Java Server Pages. Some tech journalists used to call PHP "the open source ASP" because its functionality is similar to that of the Microsoft product — although this formulation was misleading, as PHP was developed before ASP. Over the past few years, however, PHP and server-side Java have gained momentum, while ASP has lost mindshare, so this comparison no longer seems appropriate.

We'll explore server-side scripting more thoroughly in Chapter 2, but for the moment you can think of it as a collection of super-HTML tags or small programs that run inside your Web pages — except on the server side, before they get sent to the browser. For example, you can use PHP to add common headers and footers to all the pages on a site or to store form-submitted data in a database.

Strictly speaking, PHP has little to do with layout, events, on the fly DOM manipulation, or really anything about what a Web page looks and sounds like. 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 HTML, because usually the result of PHP *is* HTML.

PHP is an official module of Apache HTTP Server, the market-leading free Web server that runs about 67 percent of the World Wide Web (according to the widely quoted Netcraft Web server survey). This means that the PHP scripting engine can be built into the Web server itself, leading to faster processing, more efficient memory allocation, and greatly simplified maintenance. Like Apache Server, PHP is fully cross-platform, meaning it runs native on several flavors of Unix, as well as on Windows and now on Mac OS X. All projects under the aegis of the Apache Software Foundation — including PHP — are open source software.

## 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 due to 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, Java, Perl, Python, and the like that have encouraged its use through stable, well-documented modules and extensions. MySQL has not failed to reward the loyalty of these users with the addition of both subselects and foreign keys as of the 4.1 series.

Databases in general are useful, arguably the most consistently useful family of software products — the “killer product” of 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 equals a database).

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

## The History of PHP

Rasmus Lerdorf — software engineer, Apache team member, and international man of mystery — is the creator and original driving force behind PHP. The first part of PHP was developed for his personal use in late 1994. This was a CGI wrapper that helped him keep track of people who looked at his personal site. The next year, he put together a package called the *Personal Home Page Tools* (a.k.a. the *PHP Construction Kit*) in response to demand from users who had stumbled into his work by chance or word of mouth. Version 2 was soon released under the title PHP/FI and included the *Form Interpreter*, a tool for parsing SQL queries.

By the middle of 1997, PHP was being used on approximately 50,000 sites worldwide. It was clearly becoming too big for any single person to handle, even someone as focused and energetic as Rasmus. A small core development team now runs the project on the open source “benevolent junta” model, with contributions from developers and users around the world. Zeev Suraski and Andi Gutmans, the two Israeli programmers who developed the PHP3 and PHP4 parsers, have also generalized and extended their work under the rubric of Zend.com (*Ze*ev, A*nd*i, *Zend*, get it?).

The fourth quarter of 1998 initiated a period of explosive growth for PHP, as all open source technologies enjoyed massive publicity. In October 1998, according to the best guess, just over 100,000 unique domains used PHP in some way. Just over a year later, PHP broke the one-million domain mark. When we wrote the first edition of this book in the first half of 2000, the number had increased to about two million domains. As we write this, approximately 15 million public Web servers (in the software sense, not the hardware sense) have PHP installed on them.

Public PHP deployments run the gamut from mass-market sites such as Excite Webmail and the Indianapolis 500 Web site, which serve up millions of pageviews per day, through "mass-niche" sites such as Sourceforge.net and Epinions.com, which tend to have higher functionality needs and hundreds of thousands of users, to e-commerce and brochureware sites such as The Bookstore at Harvard.com and Sade.com (Web home of the British singer), which must be visually attractive and easy to update. There are also PHP-enabled parts of sites, such as the forums on the Internet Movie Database (imdb.com); and a large installed base of nonpublic PHP deployments, such as LDAP directories (MCI WorldCom built one with over 100,000 entries) and trouble-ticket tracking systems.

In its newest incarnation, PHP5 strives to deliver something many users have been clamoring for over the past few years: much improved object-oriented programming (OOP) functionality. PHP has long nodded to the object programming model with functions that allow object programmers to pull out results and information in a way familiar to them. These efforts still fell short of the ideal for many programmers, however, and efforts to force PHP to build in fully object-oriented systems often yielded unintended results and hurt performance. PHP5's newly rebuilt object model brings PHP more in line with other object-oriented languages such as Java and C++, offering support for features such as overloading, interfaces, private member variables and methods, and other standard OOP constructions.

With the crash of the dot-com bubble, PHP is poised to be used on more sites than ever. Demand for Web-delivered functionality has decreased very little, and emerging technological standards continue to pop up all the time, but available funding for hardware, licenses, and especially headcount has drastically decreased. In the post-crash Web world, PHP's shallow learning curve, quick implementation of new functionality, and low cost of deployment are hard arguments to beat.

## The History of MySQL

Depending on how much detail you want, the history of MySQL can be traced as far back as 1979, when MySQL's creator, Monty Widenius, worked for a Swedish IT and data consulting firm, TcX. While at TcX, Monty authored UNIREG, a terminal interface builder that connected to raw ISAM data stores. In the intervening 15 years, UNIREG served its makers rather well through a series of translations and extensions to accommodate increasingly large data sets.

In 1994, when TcX began working on Web data applications, chinks in the UNIREG armor, primarily having to do with application overhead, began to appear. This sent Monty and his colleagues off to look for other tools. One they inspected rather closely was Hughes mSQL, a light and zippy database application developed by David Hughes. mSQL possessed the distinct advantages of being inexpensive and somewhat entrenched in the market, as well as featuring a fairly well-developed client API. The 1.0 series of mSQL release lacked indexing, however, a feature crucial to performance with large data stores. Although the 2.0 series of mSQL would see the addition of this feature, the particular implementation used was not compatible with UNIREG's B+-based features. At this point, MySQL, at least conceptually, was born.

Monty and TcX decided to start with the substantial work already done on UNIREG while developing a new API that was substantially similar to that used by mSQL, with the exception of the more effective UNIREG indexing scheme. By early 1995, TcX had a 1.0 version of this new product ready. They gave it the moniker MySQL and later that year released it under a combination open source and commercial licensing scheme that allowed continued development of the product while providing a revenue stream for MySQL AB, the company that evolved from TcX.

Over the past ten years, MySQL has truly developed into a world class product. MySQL now competes with even the most feature-rich commercial database applications such as Oracle and Informix. Additions in the 4.x series have included much-requested features such as transactions and foreign key support. All this has made MySQL the world's most used open source database.

## Reasons to Love PHP and MySQL

There are ever so many reasons to love PHP and MySQL. Let us count a few.

### Cost

PHP costs you nothing. Zip, zilch, nada, not one red cent. Nothing up front, nothing over the lifetime of the application, nothing when it's over. Did we mention that the Apache/PHP/MySQL combo runs great on cheap, low-end hardware that you couldn't even *think* about for IIS/ASP/SQL Server?

MySQL is a slightly different animal in its licensing terms. Before you groan at the concept of actually using commercial software, consider that although MySQL is open-source licensed for many uses, it is not and has never been primarily community-developed software. MySQL AB is a commercial entity with necessarily commercial interests. Unlike typical open source projects, where developers often have regular full-time (and paying) day jobs in addition to their freely given open source efforts, the MySQL developers derive their primary income from the project. There are still many circumstances in which MySQL can be used for free (basically anything nonredistributive, which covers most PHP-based projects), but if you make money developing solutions that use MySQL, consider buying a license or a support contract. It's still infinitely more reasonable than just about any software license you will ever pay for.

For purposes of comparison, Table 1-1 shows some current retail figures for similar products in the United States. All prices quoted are for a single-processor public Web server with the most common matching database and development tool; *$0* means a no-cost alternative is a common real-world choice.

**Table 1-1: Comparative Out-of-Pocket Costs**

| *Item* | *ASP/SQL Server* | *ColdFusion MX/SQL Server* | *JSP/Oracle* | *PHP/MySQL* |
|---|---|---|---|---|
| Development tool | $0–2499 | $599 | $0–~2000 | $0–249 |
| Server | $999 | $2298 | $0–~35,000 | $0 |
| RDBMS | $4999 | $4999 | $15,000 | $0–220 |

## Open source software: don't fear the cheaper

But as the bard so pithily observed, we are living in a material world — where we've internalized maxims such as, "You get what you pay for," "There's no such thing as a free lunch," and "Things that sound too good to be true usually are." You (or your boss) may, therefore, have some lingering doubts about the quality and viability of no-cost software. It probably doesn't help that until recently software that didn't cost money — formerly called *freeware*, *shareware*, or *free software* — was generally thought to fall into one of three categories:

✦ Programs filling small, uncommercial niches

✦ Programs performing grungy, low-level jobs

✦ Programs for people with bizarre socio-political issues

It's time to update some stereotypes once and for all. We are clearly in the middle of a sea change in the business of software. Much (if not most) major consumer software is distributed without cost today; e-mail clients, Web browsers, games, and even full-service office suites are all being given away as fast as their makers can whip up Web versions or set up FTP servers. Consumer software is increasingly seen as a loss-leader, the flower that attracts the pollinating honeybee — in other words, a way to sell more server hardware, operating systems, connectivity, advertising, optional widgets, or stock shares. The full retail price of a piece of software, therefore, is no longer a reliable gauge of its quality or the eccentricity-level of its user.

On the server side, open source products have come on even stronger. Not only do they compete with the best commercial stuff; in many cases there's a feeling that they far exceed the competition. Don't take our word for it! Ask IBM, any hardware manufacturer, NASA, Amazon.com, Rockpointe Broadcasting, Ernie Ball Corporation, the Queen of England, or the Mexican school system. If your boss still needs to be convinced, further ammunition is available at `www.opensource.org` and `www.fsf.org`.

## The PHP license

The freeness of open source and Free software is guaranteed by a gaggle of licensing schemes, most famously the *GPL* (*Gnu General Public License*) or *copyleft*. PHP used to be released under both the GPL and its own license, with each user free to choose between them. This has recently changed. The program as a whole is now released under its own extremely laissez-faire PHP license on the model of the BSD license, whereas Zend as a standalone product is released under the *Q Public License* (this clause applies *only* if you unbundle Zend from PHP and try to sell it).

You can read the fine print about the relevant licenses at these Web sites:

✦ `www.php.net/license/`

✦ `www.mysql.com/doc/en/GPL_license.html`

✦ `www.troll.no/qpl/annotated.html`

Most people get PHP or MySQL via download, but you may have paid for it as part of a Linux distribution, a technical book, or some other product. In that case, you may now be silently disputing our assertion that PHP costs nothing. Here's the twist: Although you can't require a fee for most open source software, you *can* charge for delivering that software in a more convenient format — such as by putting it on a disk and shipping the disk to the customer. You can also charge anything the market will bear for being willing to perform certain services or accept certain risks that the development team may not wish to undertake. For instance, you are allowed to charge money for guaranteeing that every copy of the software you distribute will be virus-free or of reasonable quality, taking on the risk of being sued if a bunch of customers get bad CD-ROMs that contain hard-drive-erasing viruses.

Usually, open source software users can freely choose the precisely optimal cost-benefit equation for each particular situation: no cost and no warranties, or expensive but well supported, or something in between. No organized attempt has been made yet to sell service and support for PHP (although presumably that will be one of the value-adds of Zend). MySQL AB does sell support as part some of its licensing packages for the MySQL product. Other open source products, such as Linux, have companies such as Red Hat standing by to answer your questions, but the commercialization process is still in the early stages for PHP.

## Ease of Use

PHP is easy to learn, compared to the other ways to achieve similar functionality. Unlike Java Server Pages or C-based CGI, PHP doesn't require you to gain a deep understanding of a major programming language before you can make a trivial database or remote-server call. Unlike Perl, which has been semijokingly called a "write-only language," PHP has a syntax that is quite easy to parse and human-friendly. And unlike ASP.NET, PHP is stable and ready to solve your problems today.

Many of the most useful specific functions (such as those for opening a connection to an Oracle database or fetching e-mail from an 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. In fact, it's entirely possible to use PHP just by modifying freely available scripts rather than starting from scratch — you'll still need to understand the basic principles, but you can avoid many frustrating and time-consuming minor mistakes.

We must mention one caveat: *Easy* means different things to different people, and for some Web developers it has come to connote a graphical, drag-and-drop, What You See Is What You Get development environment. To become truly proficient at PHP, you need to be comfortable editing HTML by hand. You can use WYSIWYG editors to design sites, format pages, and insert client-side features before you add PHP functionality to the source code. There are even ways, which we'll detail in Chapter 3, to add PHP functions to your favorite editing environment. It's not realistic, however, to think you can take full advantage of PHP's capabilities without ever looking at source code.

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 but probably simpler than JavaScript and definitely less conceptually complex than JSP or ASP.NET.

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. Again, the word *easy* is relative. However, MySQL's increasingly faithful adherence to the ANSI SQL-92 standard and a comprehensive suite of external client programs, coupled with graphical administration tools such as PHPMyAdmin and the new MySQL Control Center, will get even neophyte users up and running quickly compared to other databases. None of these will substitute for learning a little theory and employing good design practices, but that subject is for another chapter.

## HTML-embeddedness

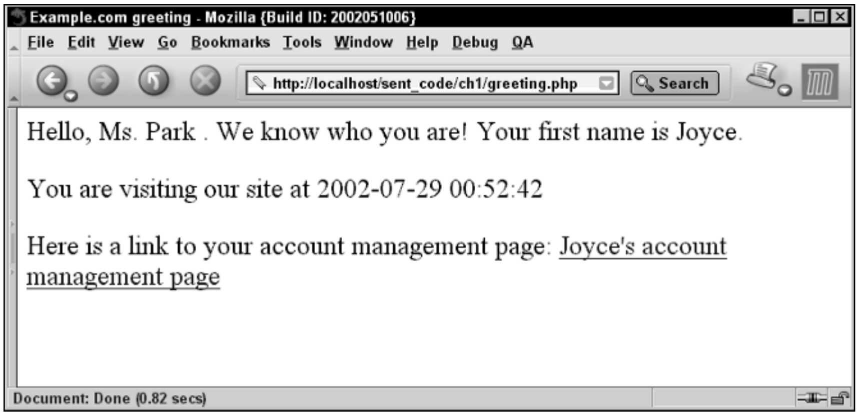
PHP is embedded within HTML. In other words, PHP pages are ordinary HTML pages that escape into PHP mode only when necessary. Here is an example:

```
<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.


Example.com greeting - Mozilla {Build ID: 2002051006}
File Edit View Go Bookmarks Tools Window Help Debug QA
http://localhost/sent_code/ch1/greeting.php
Search

Hello, Ms. Park . We know who you are! Your first name is Joyce.

You are visiting our site at 2002-07-29 00:52:42

Here is a link to your account management page: Joyce's account management page

Document: Done (0.82 secs)

**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, or right-click if you're using the AOL browser), 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 scripters.

✦ Every line of HTML does not need to be rewritten in a programming language.

✦ PHP can reduce labor costs and increase efficiency due to its shallow learning curve and ease of use.

Perhaps the sweetest thing of all about embedded scripting languages is that they don't need to be compiled into binary code before they can be tested or used — just write and run. PHP is interpreted (as are many newish computer languages), although the Zend Engine does some behind-the-scenes precompiling into an intermediate form for greater speed with complex scripts.

But what if you happen to want compilation? This can be desirable if you wish to distribute nonreversible binaries so others can use the code without being able to look at the source. The Zend team now offers a precompiler, *Zend Encoder*, which will deliver the code in a non-reversible intermediate representation, as well as substantially speed up large complex PHP scripts.

## Cross-platform compatibility

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

PHP is compatible with the three leading Web servers: Apache HTTP Server for Unix and Windows, Microsoft Internet Information Server, and Netscape Enterprise Server (a.k.a. iPlanet Server). It also works with several lesser-known servers, including Alex Belits' fhttpd, Microsoft's Personal Web Server, AOLServer, and Omnicentrix's Omniserver application server. Specific Web-server compatibility with MySQL is not required, since PHP will handle all the dirty work for you.

Table 1-2 shows a brief matrix of the possible OS/Web-server combinations.

**Table 1-2: Operating Systems and Web Servers for PHP**

| *Variables* | *UNIX* | *Windows* |
|---|---|---|
| Flavors | AIX, A/UX, BSDI, Digital UNIX/Tru64, FreeBSD, HP-UX, IRIX, Linux, Mac OS X, NetBSD, OpenBSD, SCO UnixWare, Solaris, SunOS, Ultrix, Xenix, and more | Windows 95/98/ME<br>Windows NT/2000/XP/2003 |
| Web servers | Apache, fhttpd, Netscape | IIS, PWS, Netscape, Apache, Omni |

Now that PHP runs on Macintosh, PHP is almost totally cross-platform. You can develop on almost any client OS using your favorite tools and then upload your PHP scripts to a server on almost any OS. We'll discuss the development process in more detail in Chapter 3.

## Not tag-based

PHP is a real programming language. ColdFusion, by contrast, is a bunch of predefined tags, like HTML. In PHP, you can define functions to your heart's content just by typing a name and a definition. In ColdFusion, you have to use tags developed by other people or go through the Custom Tag Extension development process.

As a witty PHP community member once said, "ColdFusion makes easy things easy, and medium-hard things impossible." And as every programmer will agree, once you experience the power of curly brackets and loops, you never go back to tags.

## Stability

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

- ✦ The server doesn't need to be rebooted 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. Although it is not the fastest nor the easiest to administer, once you get it set up, Apache HTTP Server seemingly never crashes. It also doesn't require server reboots every time a setting is changed (at least on the Unix side). PHP inherits this reliability; plus, its own implementation is solid yet lightweight. In a two-and-a-half-month head-to-head test conducted by the Network Computing labs in October 1999, Apache Server with PHP handily beat both IIS/Visual Studio and Netscape Enterprise Server/Java for stability of environment.

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 session 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. Applications written in PHP3 will function with little or no revision for PHP4 and 5. And because of the standards-based SQL support, MySQL 3.x databases are easily moved to more current versions (and most likely always will be).

## Speed

PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.

PHP5 is much faster for almost every use than CGI scripts. There is an unfortunate grain of truth to the joke that CGI stands for "Can't Go Instantly." Although many CGI scripts are written in C, one of the lowest-level and therefore speediest of the major programming languages, they are hindered by the fact that each request must spawn an entirely new process after being handed off from the `http` daemon. The time and resources necessary for this handoff and spawning are considerable, and there can be limits to the number of concurrent processes that can be running at any one time. Other CGI scripting languages such as Perl and Tcl can be quite slow. Most Web sites have moved away from use of CGI for performance and security reasons.

Although it takes a slight performance hit by being interpreted rather than compiled, this is far outweighed by the benefits PHP derives from its status as a Web server module. When compiled this way, PHP becomes part of the `http` daemon itself. Because there is no transfer to and from a separate application server (as there is with ColdFusion, for instance) requests can be filled with maximum efficiency.

Although no extensive formal benchmarks have compared the two, much anecdotal evidence and many small benchmarks suggest that PHP is at least as fast as ASP and readily outperforms ColdFusion or JSP in most applications.

## Open source licensing

We've already dealt with the cost advantages of open source software in the "Cost" section of this chapter. The other major consequence of these licenses is that the complete source code for the software must be included in any distribution.

In fact, the Unix version of PHP is released *only* as source code; so far, the development team has staunchly resisted countless pleas to distribute official binaries for any of the Unixes. At first, new users (particularly those also new to Unix) tend to feel that source code is about as useful as a third leg, and most vastly prefer a nice convenient `rpm`. But there are both pragmatic and idealistic reasons for including folders full of pesky `.c` and `.h` files.

The most immediate pragmatic advantage is that you can compile your PHP installation with only the stuff you really need for any given situation. This approach has performance and security advantages. For instance, you can put in hooks to the database(s) of your choice. You can recompile as often as you want: maybe when an Apache security release comes out, or when you wish to support a new database application. By compiling a custom application specifically suited to your system, or any given snapshot of your system, performance and stability are increased over their already respectable baseline.

What sets open source software apart from its competitors is not just price but control. Plenty of consumer software is now given away under various conditions. Careful scrutiny of the relevant licenses, however, will generally reveal limits as to how the software can be used. Maybe you can run it at home but not at the office. Perhaps you can load it on your laptop, but you're in violation if you use it for business purposes. Or, most commonly, you can use it for anything you want but forget about looking at the code — much less changing it. There are even community licenses that force you to donate your improvements to the codebase but charge you for use of the product at the end!

**Caution**

Don't even *think* about coming back with a riposte that involves violating a software license — we're covering our ears; we're *not* listening! Especially with the explosion in no-cost software, there's just no good reason to break the law. Besides, it's bad karma for software developers. What goes around, comes around, don't ya know?

For all their *openness*, the licenses for MySQL and PHP are quite different. You should not assume that you understand the MySQL terms simply because you have read the PHP license. They have many similarities to be sure but also some radically different provisions, especially when it comes to when you should pay.

Table 1-3 shows examples of the various source and fee positions in today's software marketplace.

**Table 1-3: Source/Fee Spectrum**

| *Fee Structure* | *Closed Source* | *Controlled Source* | *Open Source* |
|---|---|---|---|
| Fee for all uses | Macromedia ColdFusion | — | — |
| Fee for some uses | Corel WordPerfect | Sun Java | MySQL |
| No fee for any use | Microsoft IE | Sun StarOffice | GPLed software |

Genuinely open source software like PHP cannot seek to limit the purposes for which it is used, the people allowed to use it, or a host of other factors. The most critical of these rights is the one allowing users to make and distribute any modifications along with the original software. In the most extreme case, where one or more developers decide to release a separate, complete version of a piece of software, this practice is referred to as *code forking*.

If somewhere down the road you develop irreconcilable differences with the PHP development team, you can take every bit of code they've labored over for all these years and use it as the basis of your own product. You couldn't call it PHP, and you'd have to include stuff in your documentation that gave due credit to the authors — the rationale is that source code distributions make it next to impossible for any single person or group to hijack a program to the detriment of the community as a whole, because every user always has the power to take the source and walk.

Users new to the open source model should be aware that this right is also enjoyed by the developers. At any time, Rasmus, Zend, and company can choose to defect from the community and put all their future efforts into a commercial or competing product based on PHP. Of course, the codebase up to this point would still be available to anyone who wanted to pick up the baton, and for a product as large as PHP that could be a considerable number of volunteer developers.

This leads to one other oft-forgotten advantage of open source software: You can be pretty sure the software will be around in a few years, no matter what. In these days of products with the life spans of morning glories, it's hard to pick a tool with staying power. Fans of OS/2, Amiga, NeXT, Newton, Firefly, Netscape, BeOS, Napster, and a host of other once-hot technologies know the pain of abandonment when a company goes belly-up, decides to stop supporting a technology, or is sold to a buyer with a new agenda. The open source model reduces the chances of an ugly emergency port in a couple of years and thus makes long-term planning more realistic.

## 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 ODBC. In addition, PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time. PHP5 extends this support even further, offering a fully incorporated GD graphics library and revamped XML support with DOM and simpleXML.

Most things that PHP does not support are ultimately attributable to closed-source shops on the other end. For instance, Microsoft has not thus far been eager to cooperate with open source projects like PHP. Potential users who complain about lack of native Mac OS 9 or .NET support on the PHP mailing list are simply misinformed about where the fault lies.

## Fast feature development

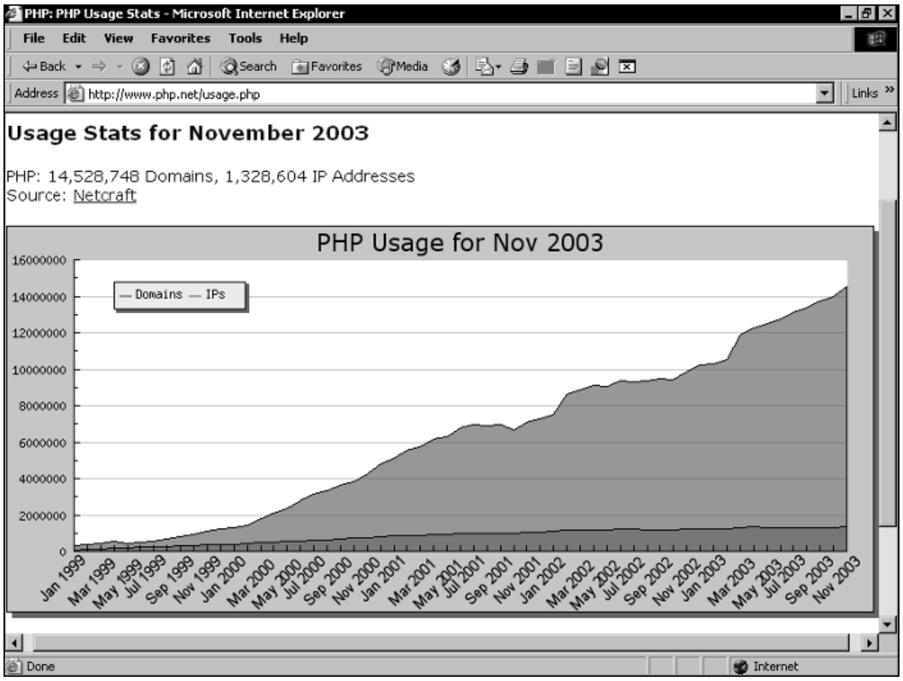
Users of proprietary Web development technologies can sometimes be frustrated by the glacial speed at which new features are added to the official product standard to support emerging technologies. With PHP, this is not a problem. All it takes is one developer, a C compiler, and a dream to add important 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 which may be later 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. Compare this release scheme to the .NET transition, which has left developers with almost a year in which Microsoft is not really improving IIS but has not yet released a prime-time version of .NET server.

It hasn't always been the case that MySQL added new features in a timely fashion. It would probably be fair to say that a significant chunk of PostgreSQL users are former MySQL users frustrated by the lack of transaction support, for example. However, the 4.0 and 4.1 versions have remedied this and other inequities. Transactions are in the software today, while subselects and foreign keys are experimental but coming along nicely.

## Popularity

PHP is fast becoming one of the most popular choices for so-called two-tier development (Web plus data). Figure 1-2 charts growth since 1999.


PHP: PHP Usage Stats - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.php.net/usage.php
Links
PHP Usage for Nov 2003
Domains
IPs
Done
Internet

Usage Stats for November 2003

PHP: 14,528,748 Domains, 1,328,604 IP Addresses
Source: Netcraft

Figure 1-2: Netcraft survey of PHP use

Although it's not evident from this graphic, the period October 1998 through October 1999 showed 800 percent growth in the number of domains. As Web sites become even more ubiquitous, and as more of them go beyond simple static HTML pages, PHP is expected to gain ground quickly in absolute numbers of users.

Although it's somewhat more difficult to get firm figures, it seems that PHP is also in a strong position relative to similar products. According to a 2002 Zend report, Microsoft Active Server Pages technology appears to be utilized on about 24 percent of Web servers, whereas ColdFusion is implemented on approximately 4 percent of surveyed domains. PHP is used on over 24 percent of all Web servers, as measured by a larger and more accurate sample, and is now said to be the most popular server-side scripting language on the Web.

Active Server Pages and ColdFusion used to be highly visible because they tended to be disproportionately selected by large e-commerce sites. However, the realities of the Web finally caught up with us — and it is the flashy e-commerce sites that were disproportionately thinned by the dot-bomb crash. It is now becoming clearer that most Web sites are informational rather than direct revenue centers and, therefore, do not repay high development expenses in an immediate way. PHP enjoys substantial advantages over its competitors in this development category, which has turned out to be the majority of the Internet.

## 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.

But 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, 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 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 Java Server Pages or ColdFusion are. 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 24/7/365 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 nonredistributive 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.

✦ ✦ ✦