Beginning PHP 5 and MySQL E-Commerce: From Novice to Professional

CRISTIAN DARIE AND MIHAI BUCICA

Apress®

Beginning PHP 5 and MySQL E-Commerce: From Novice to Professional
Copyright © 2005 by Cristian Darie and Mihai Bucica

ISBN (pbk): 1-59059-392-8

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Contributing Authors: Emilian Balanescu, Radu Calauz, Lucian Gheorghe, Karli Watson
Lead Editor: Jason Gilmore
Technical Reviewers: Mark Evans, Radu Calauz
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, John Franklin, Jason Gilmore, Chris Mills, Dominic Shakeshaft, Jim Sumser
Project Manager: Tracy Brown Collins
Copy Edit Manager: Nicole LeClerc
Copy Editor: Julie McNamee
Production Manager: Kari Brooks-Copony
Production Editor: Janet Vail
Compositor and Artist: Diana Van Winkle
Proofreader: Patrick Vincent
Indexer: John Collin
Interior Designer: Diana Van Winkle
Cover Designer: Kurt Krames
Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013, and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders@springer-ny.com`, or visit `http://www.springer-ny.com`. Outside the United States: fax +49 6221 345229, e-mail `orders@springer.de`, or visit `http://www.springer.de`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Downloads section.

CHAPTER 2

Laying Out the Foundations

Now that you've convinced the client that you can create a cool web site to complement his or her activity, it's time to stop celebrating and start thinking about how to put into practice all the promises you've made. As usual, when you lay down on paper the technical requirements you must meet, everything starts to seem a bit more complicated than initially anticipated.

To ensure this project's success, you need to come up with a smart way to implement what you have signed the contract for. You want to develop the project smoothly and quickly, but the ultimate goal is to make sure the client is satisfied with your work. Consequently, you should aim to provide your site's increasing number of visitors with a positive web experience by creating a pleasant, functional, and responsive web site.

The requirements are high, but this is normal for an e-commerce site today. To maximize the chances of success, we'll analyze and anticipate as many of the technical requirements as possible, and implement solutions in a way that supports changes and additions with minimal effort.

This chapter lays down the foundations for the future TShirtShop web site. We will talk about the technologies and tools you'll use, and even more importantly, how you'll use them. Your goals for this chapter are to

- Analyze the project from a technical point of view
- Analyze and choose an architecture for your application
- Decide which technologies, programming languages, and tools to use
- Discuss naming and coding conventions
- Create the basic structure of the web site and set up the database

Designing for Growth

The word *design* in the context of a web application can mean many things. Its most popular usage probably refers to the visual and user interface design of a web site.

This aspect is crucial because, let's face it, the visitor is often more impressed with how a site looks and how easy it is to use than about which technologies and techniques are used behind the scenes, or what operating system the web server is running. If the site is slow, hard to use, or easy to forget, it just doesn't matter what rocket science was used to create it.

Unfortunately, this truth makes many inexperienced programmers underestimate the importance of the way the invisible part of the site is implemented—the code, the database, and so on. The visual part of a site gets visitors interested to begin with, but its functionality makes them come back. A web site can sometimes be implemented very quickly based on certain initial requirements, but if not properly architected, it can become difficult, if not impossible, to change.

For any project of any size, some preparation must be done before starting to code. Still, no matter how much preparation and design work is done, the unexpected does happen and hidden catches, new requirements, and changing rules always seem to work against deadlines. Even without these unexpected factors, site designers are often asked to change or add new functionality many times after the project is finished and deployed. This will also be the case for TShirtShop, which will be implemented in three separate stages, as discussed in Chapter 1.

You will learn how to create the web site so that the site (or you) will not fall apart when functionality is extended or updates are made. Because this is a programming book, instead of focusing on how to design the user interface or on marketing techniques, we'll pay close attention to designing the code that makes them work.

The phrase, *designing the code*, can have different meanings; for example, we'll need to have a short talk about naming conventions. Still, the most important aspect that we need to take a look at is the application architecture. The architecture refers to the way you split the code for a simple piece of functionality (for example, the product search feature) into smaller components. Although it might be easier to implement that functionality as quickly and as simply as possible, in a single component, you gain great long-term advantages by creating more components that work together to achieve the desired result.

Before talking about the architecture itself, you must determine what you want from this architecture.

Meeting Long-Term Requirements with Minimal Effort

Apart from the fact that you want a fast web site, each of the phases of development we talked about in Chapter 1 brings new requirements that must be met.

Every time you proceed to a new stage, you want to be able to **reuse** most of the already existing solution. It would be very inefficient to redesign the whole site (not just the visual part, but the code as well!) just because you need to add a new feature. You can make it easier to reuse the solution by planning ahead, so any new functionality that needs to be added can slot in with ease, rather than each change causing a new headache.

When building the web site, implementing a **flexible architecture** composed of pluggable components allows you to add new features—such as the shopping cart, the departments list, or the product search feature—by coding them as separate components and plugging them into the existing application. Achieving a good level of flexibility is one of the goals regarding the application's architecture, and this chapter shows how you can put this into practice. You'll see that the flexibility level is proportional to the amount of time required to design and implement it, so we'll try to find a compromise that will provide the best gains without complicating the code too much.

Another major requirement that is common to all online applications is having a **scalable architecture**. Scalability is defined as the capability to increase resources to yield a linear increase in service capacity. In other words, ideally, in a scalable system the ratio (proportion) between the number of client requests and the hardware resources required to handle those requests is constant, even when the number of clients increases. An unscalable system can't deal with an increasing number of clients, no matter how many hardware resources are provided. Because we're optimistic about the number of customers, we must be sure that the site will be able to deliver its functionality to a large number of clients without throwing out errors or performing sluggishly.

Reliability is also a critical aspect for an e-commerce application. With the help of a coherent error-handling strategy and a powerful relational database, you can ensure data integrity and ensure that noncritical errors are properly handled without bringing the site to its knees.

The Magic of the Three-Tier Architecture

Generally, the architecture refers to splitting each piece of the application's functionality into separate components based on what they do, and grouping each kind of component into a single logical tier.

Almost every module that you'll create for your site will have components in these three tiers from the application server:

- The **presentation tier**
- The **business tier**
- The **data tier**

The **presentation tier** contains the user interface elements of the site, and includes all the logic that manages the interaction between the visitor and the client's business. This tier makes the whole site feel alive, and the way you design it has a crucial importance for the site's success. Because your application is a web site, its presentation tier is composed of dynamic web pages.

The **business tier** (also called the *middle tier*) receives requests from the presentation tier and returns a result to the presentation tier depending on the business logic it contains. Almost any event that happens in the presentation tier usually results in the business tier being called (except events that can be handled locally by the presentation tier, such as simple input data validation, and so on). For example, if the visitor is doing a product search, the presentation tier calls the business tier and says, "Please send me back the products that match this search criterion." Almost always, the business tier needs to call the data tier for information to be able to respond to the presentation tier's request.

The **data tier** (sometimes referred to as the *database tier*) is responsible for managing the application's data and sending it to the business tier when requested. For the TShirtShop e-commerce site, you'll need to store data about products (including their categories and their departments), users, shopping carts, and so on. Almost every client request finally results in the data tier being interrogated for information (excepting when previously retrieved data has been cached at the business tier or presentation tier levels), so it's important to have a fast database system. In Chapters 3 and 4, you'll learn how to design the database for optimum performance.

These tiers are purely logical—there is no constraint on the physical location of each tier. In theory, you are free to place all of the application, and implicitly all of its tiers, on a single server machine, or you can place each tier on a separate machine if the application permits this. In practice, PHP is not very suited to build applications that you want to split on different machines, but this is expected to change in the future. Chapter 16 explains how to integrate functionality from other web sites using XML Web Services. XML Web Services permit easy integration of functionality across multiple servers.

An important constraint in the three-layered architecture model is that information must flow in sequential order between tiers. The presentation tier is only allowed to access the business tier, and never directly the data tier. The business tier is the "brain" in the middle that communicates with the other tiers and processes and coordinates all the information flow. If the presentation tier directly accessed the data tier, the rules of three-tier architecture programming would be broken. When you implement a three-tier architecture, you must be consistent and obey its rules to reap the benefits.

Figure 2-1 is a simple representation of the way data is passed in an application that implements the three-tier architecture.

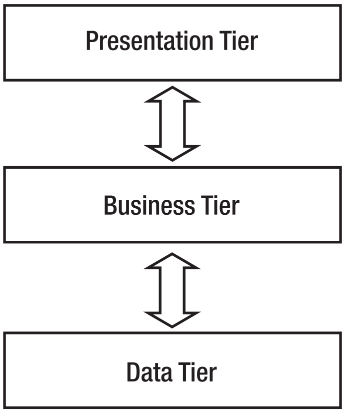


Figure 2-1. *Simple representation of the three-tier architecture*

A Simple Example

It's easier to understand how data is passed and transformed between tiers if you take a closer look at a simple example. To make the example even more relevant to our project, let's analyze a situation that will actually happen in TShirtShop. This scenario is typical for three-tier applications.

Like most e-commerce sites, TShirtShop will have a shopping cart, which we will discuss later in the book. For now, it's enough to know that the visitor will add products to the shopping cart by clicking an "Add to Cart" button. Figure 2-2 shows how the information flows through the application when that button is clicked.

When the user clicks on the "Add to Cart" button for a specific product (Step 1), the presentation tier (which contains the button) forwards the request to the business tier—"Hey, I want this product added to my shopping cart!" (Step 2). The business tier receives the request, understands that the user wants a specific product added to the shopping cart, and handles the request by telling the data tier to update the visitor's shopping cart by adding the selected product (Step 3). The data tier needs to be called because it stores and manages the entire web site's data, including users' shopping cart information.

The data tier updates the database (Step 4) and eventually returns a success code to the business tier. The business tier (Step 5) handles the return code and any errors that might have occurred in the data tier while updating the database, and then returns the output to the presentation tier.

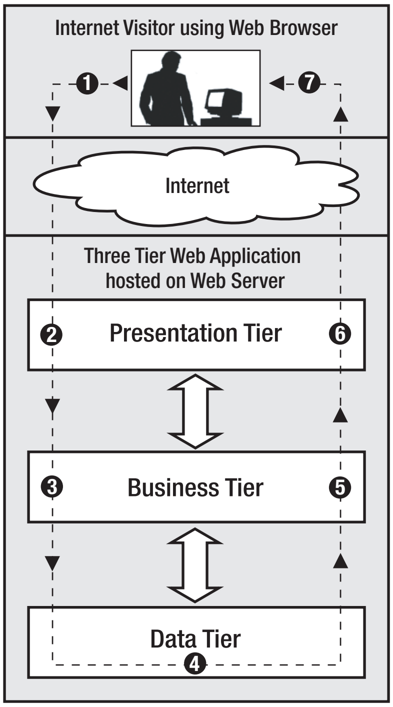

Internet Visitor using Web Browser
1
7
Internet
Three Tier Web Application hosted on Web Server
2
Presentation Tier
6
3
Business Tier
5
Data Tier
4

Figure 2-2. *Internet visitor interacting with a three-tier application*

Finally, the presentation tier generates an updated view of the shopping cart (Step 6). The results of the execution are wrapped up by generating an HTML (Hypertext Markup Language) web page that is returned to the visitor (Step 7), where the updated shopping cart can be seen in the visitor's web browser.

Note that in this simple example, the business tier doesn't do a lot of processing and its business logic isn't very complex. However, if new business rules appear for your application, you would change the business tier. If, for example, the business logic specified that a product could only be added to the shopping cart if its quantity in stock was greater than zero, an additional data tier call would have been made to determine the quantity. The data tier would only be requested to update the shopping cart if products are in stock. In any case, the presentation tier is informed about the status and provides human-readable feedback to the visitor.

What's in a Number?

It's interesting to note how each tier interprets the same piece of information differently. For the data tier, the numbers and information it stores have no significance because this tier is an engine that saves, manages, and retrieves numbers, strings, or other data types—not product quantities or product names. In the context of the previous example, a product quantity of 0 represents a simple, plain number without any meaning to the data tier (it is simply 0, a 32-bit integer).

The data gains significance when the business tier reads it. When the business tier asks the data tier for a product quantity and gets a "0" result, this is interpreted by the business tier as "Hey, no products in stock!" This data is finally wrapped in a nice, visual form by the presentation tier, such as a label reading, "Sorry, at the moment the product cannot be ordered."

Even if it's unlikely that you want to forbid a customer from adding a product to the shopping cart if the product is not in stock, the example (described in Figure 2-3) is good enough to present in yet another way how each of the three tiers has a different purpose.

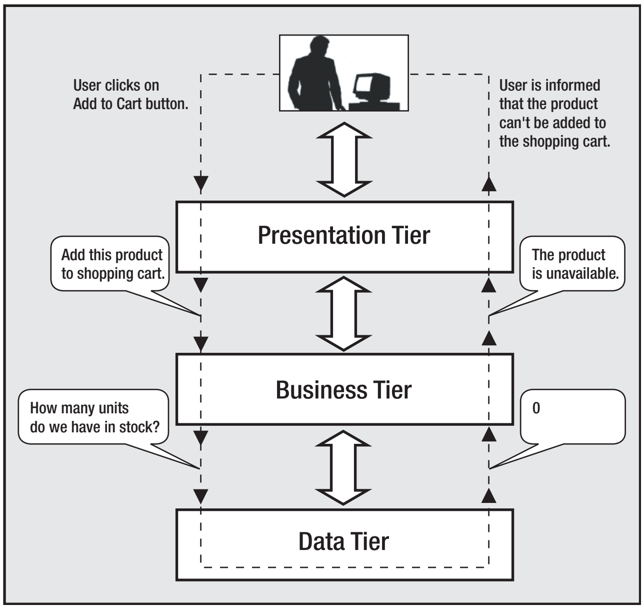

User clicks on Add to Cart button.
User is informed that the product can't be added to the shopping cart.
Presentation Tier
Add this product to shopping cart.
The product is unavailable.
Business Tier
How many units do we have in stock?
0
Data Tier

Figure 2-3. *Internet visitor interacting with a three-tier application*

The Right Logic for the Right Tier

Because each layer contains its own logic, sometimes it can be tricky to decide where exactly to draw the line between the tiers. In the previous scenario, instead of reading the product's quantity in the business tier and deciding whether the product is available based on that number (resulting in two data tier, and implicitly database, calls), you could have a single data tier method named `AddProductIfAvailable` that adds the product to the shopping cart only if it's available in stock.

In this scenario some logic is transferred from the business tier to the data tier. In many other circumstances, you might have the option to place same logic in one tier or another, or maybe in both. In most cases, there is no single best way to implement the three-tier architecture, and you'll need to make a compromise or a choice based on personal preference or external constraints.

Furthermore, there are occasions in which even though you know the *right* way (in respect to the architecture) to implement something, you might choose to break the rules to get a performance gain. As a general rule, if performance can be improved this way, it is okay to break the strict limits between tiers *just a little bit* (for example, add some of the business rules to the data tier or vice versa), *if* these rules are not likely to change in time. Otherwise, keeping all the business rules in the middle tier is preferable because it generates a "cleaner" application that is easier to maintain.

Finally, don't be tempted to access the data tier directly from the presentation tier. This is a common mistake that is the shortest path to a complicated, hard-to-maintain, and inflexible system. In many data access tutorials or introductory materials, you'll be shown how to perform simple database operations using a simple user interface application. In these kinds of programs, all the logic is probably written in a short, single file, instead of separate tiers. Although the materials might be very good, keep in mind that most of these texts are meant to teach you how to do different individual tasks (for example, access a database), and not how to correctly create a flexible and scalable application.

A Three-Tier Architecture for TShirtShop

Implementing a three-tiered architecture for the TShirtShop web site will help achieve the goals listed at the beginning of the chapter. The coding discipline imposed by a system that might seem rigid at first sight allows for excellent levels of flexibility and extensibility in the long run.

Splitting major parts of the application into separate smaller components encourages reusability. More than once when adding new features to the site, you'll see that you can reuse some of the already existing bits. Adding a new feature without needing to change much of what already exists is, in itself, a good example of reusability.

Another advantage of the three-tiered architecture is that, if properly implemented, the overall system is resistant to changes. When bits in one of the tiers change, the other tiers usually remain unaffected, sometimes even in extreme cases. For example, if for some reason the backend database system is changed (say, the manager decides to use Oracle instead of MySQL), you only need to update the data tier and maybe just a little bit of the business tier.

Why Not Use More Tiers?

The three-tier architecture we've been talking about so far is a particular (and the most popular) version of the *n*-Tier Architecture, which is a commonly used buzzword these days. *n*-Tier Architecture refers to splitting the solution into a number (*n*) of logical tiers. In complex projects, sometimes it makes sense to split the business layer into more than one layer, thus resulting in architecture with more than three layers. However, for our web site, it makes the most sense to stick with the three-layered design, which offers most of the benefits while not requiring too many hours of design or a complex hierarchy of framework code to support the architecture.

Maybe with a more involved and complex architecture, you could achieve even higher levels of flexibility and scalability for the application, but you would need much more time for design before starting to implement anything. As with any programming project, you must find a fair balance between the time required to design the architecture and the time spent to implement it. The three-tier architecture is best suited to projects with average complexity, like the TShirtShop web site.

You also might be asking the opposite question, "Why not use fewer tiers?" A two-tier architecture, also called *client-server* architecture, can be appropriate for less-complex projects. In short, a two-tier architecture requires less time for planning and allows quicker development in the beginning, although it generates an application that's harder to maintain and extend in the long run. Because we're expecting to have to extend the application in the future, the client-server architecture is not appropriate for our application, so it won't be discussed further in this book.

Now that the general architecture is known, let's see what technologies and tools you will use to implement it. After a brief discussion of the technologies, you'll create the foundation of the presentation and data tiers by creating the first page of the site and the backend database. You'll start implementing some real functionality in each of the three tiers in Chapter 3 when you start creating the web site's product catalog.

Choosing Technologies and Tools

No matter which architecture is chosen, a major question that arises in every development project is which technologies, programming languages, and tools are going to be used, bearing in mind that external requirements can seriously limit your options.

■ **Note** In this book we're creating a web site using PHP, MySQL, and related technologies. We really like these technologies, but it doesn't necessarily mean they're the best choice for any kind of project, in any circumstances. Additionally, there are many situations in which you must use specific technologies because of client requirements. The System Requirements and Software Requirements stages in the software development process will determine which technologies you must use for creating the application. See Appendix C for more details.

In this book, we'll work with PHP 5.0 and MySQL 4.x, the most popular combination for creating data-driven web sites on the planet. Although the book assumes some previous experience with each of these, we'll take a quick look at them and see how they fit into our project and into the three-tier architecture.

■ **Note** We included complete environment installation instructions (including Apache 2, PHP 5, and MySQL 4) in Appendix A.

Using PHP to Generate Dynamic Web Content

PHP is an open-source technology for building dynamic, interactive web content. Its short description (on the official PHP web site, `http://www.php.net`) is: "PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML."

PHP stands for PHP: Hypertext Preprocessor (yes, it's a recursive acronym), and is available for free download at its official web site. We included complete installation instructions for PHP in Appendix A. Because we're using PHP to build a dynamic web site, you'll also learn how to install Apache and how to integrate PHP with it in Appendix A.

The story of PHP, having its roots somewhere in 1994, is a successful one. Among the factors that led to its success are the following:

- PHP is free; especially when combined with Linux server software, PHP can prove to be a very cost-efficient technology to build dynamic web content.
- PHP has a shorter learning curve than other scripting languages.
- The PHP community is agile, many useful helper libraries or new versions of the existing libraries are being developed (such as PEAR and Smarty), and new features are added frequently.
- PHP works very well on a variety of web servers and operating systems (Unix-like platforms, Windows, Mac OS X).

However, PHP is not the only server-side scripting language around for creating dynamic web pages. Among its most popular competitors are JSP (Java Server Pages), Perl, ColdFusion, and ASP.NET. Between these technologies are many differences, but also some fundamental similarities. For example, pages written with any of these technologies are composed of basic HTML, which draws the static part of the page (the template), and code that generates the dynamic part.

■Note You might want to check out *Beginning ASP.NET 1.1 E-Commerce* (Apress, 2004) or *Beginning ASP.NET 2.0 E-Commerce* (Apress, 2005), which explain how to build e-commerce web sites with ASP.NET, Visual Basic .NET, and SQL Server.

Working Smart with Smarty

Because PHP is simple and easy to start with, it has always been tempting to start coding without properly designing an architecture and framework that would be beneficial in the long run.

What makes things even worse is that the straightforward method of building PHP pages is to mix PHP instructions with HTML, because PHP doesn't have by default an obvious technique of separating the PHP code from the HTML layout information.

Mixing the PHP logic with HTML has two important disadvantages:

- This technique often leads to long, complicated, and hard-to-manage code. Maybe you have seen those kilometric source files with an unpleasant mixture of PHP and HTML, hard to bear by the eye and impossible to understand after a week.
- These mixed files are the subject of both designers' and programmers' work, complicating the collaboration more than necessary. This also increases the chances of the designer creating bugs in the code logic while working on cosmetic changes.

These kinds of problems led to the development of template engines, which offer frameworks of separating the presentation logic from the static HTML layout. Smarty (`http://smarty.php.net`) is the most popular and powerful template engine for PHP. Its main purpose is to offer you a simple way to separate application logic (PHP code) from its presentation code (HTML).

This separation permits the programmer and the template designer to work independently on the same application. The programmer can change the PHP logic without needing to change the template files, and the designer can change the templates without caring how the code that makes them alive works.

Figure 2-4 shows the relationship between the Smarty Design Template file and its Smarty plugin file.

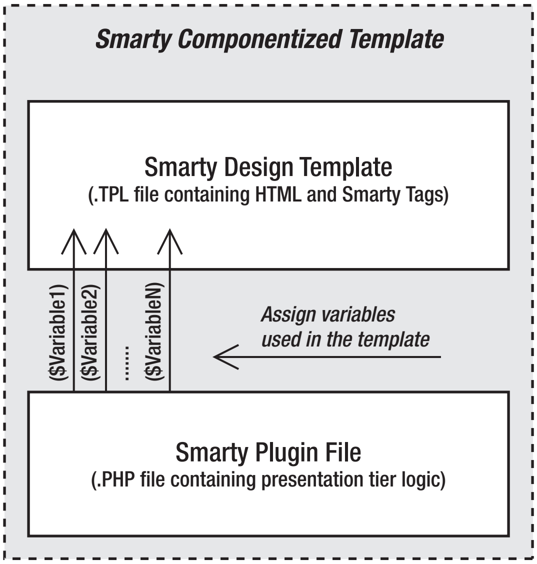

Smarty Componentized Template
Smarty Design Template
(.TPL file containing HTML and Smarty Tags)
($Variable1)
($Variable2)
....
($VariableN)
Assign variables used in the template
Smarty Plugin File
(.PHP file containing presentation tier logic)

Figure 2-4. *Smarty Componentized Template*

The Smarty Design Template (a .tpl file containing the HTML layout and Smarty-specific tags and code) and its Smarty plugin file (a .php file containing the associated code for the template) form a **Smarty Componentized Template**. You'll learn more about how Smarty works while you're building the e-commerce web site. For a fast introduction to Smarty, read the Smarty Crash Course at `http://smarty.php.net/crashcourse.php`.

What About the Alternatives?

Smarty is not the only template engine available for PHP. Other popular template engines are

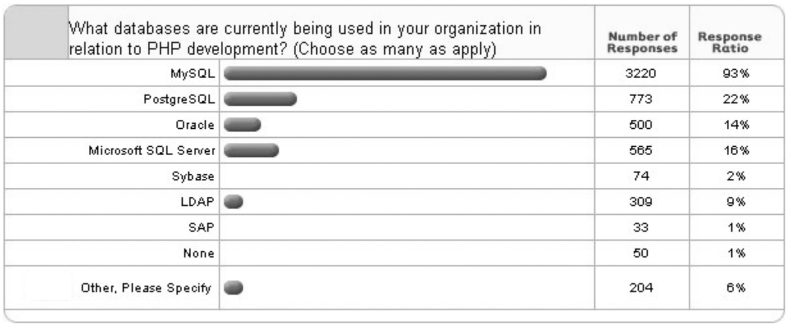
- Yapter (`http://yapter.sourceforge.net/`)
- EasyTemplate (`http://www.onlinetools.org/tools/easytemplate/index.php`)
- phpLib (`http://phplib.sourceforge.net/`)
- TemplatePower (`http://templatepower.codocad.com/`)
- FastTemplate (`http://www.thewebmasters.net/php/FastTemplate.phtml`)

Although all template engines follow the same basic principles, we chose to use Smarty in the PHP e-commerce project for this book because of its very good performance results, powerful features (such as template compilation and caching), and wide acceptance in the industry.

Storing Web Site Data in MySQL Databases

Most of the data your visitors will see while browsing the web site will be retrieved from a relational database. A Relational Database Management System (RDBMS) is a complex software program, the purpose of which is to store, manage, and retrieve data as quickly and reliably as possible. For the TShirtShop web site, it will store all data regarding the products, departments, users, shopping carts, and so on.

Many RDBMSs are available for you to use with PHP, including MySQL, PostgreSQL, Oracle, and so on. A 2003 survey of Zend (`http://www.zend.com/zend/php_survey_results.php`), with more than 10,000 respondents, revealed that the preferred database for PHP development is MySQL (see Figure 2-5).



What databases are currently being used in your organization in relation to PHP development? (Choose as many as apply)	Number of Responses	Response Ratio
MySQL	3220	93%
PostgreSQL	773	22%
Oracle	500	14%
Microsoft SQL Server	565	16%
Sybase	74	2%
LDAP	309	9%
SAP	33	1%
None	50	1%
Other, Please Specify	204	6%

Figure 2-5. *Survey results show that MySQL is popular among PHP developers.*

MySQL is the world's most popular open-source database, and it's a free (for noncommercial use), fast, and reliable database. Another important advantage is that many web hosting providers offer access to a MySQL database, which makes your life easier when going live with your newly created e-commerce web site. We'll use MySQL as the backend database when developing the TShirtShop e-commerce web site.

The language used to communicate with a relational database is SQL (Structured Query Language). However, each database engine recognizes a particular dialect of this language. If you decide to use a different RDBMS than MySQL, you'll probably need to update some of the SQL queries.

■ **Note** If you're using more database systems, we recommend you read *The Programmer's Guide to SQL* (Apress, 2003), which teaches the SQL standard with practical examples for SQL Server, Oracle, MySQL, DB2, and Access.

Getting in Touch with MySQL

You talk with the database server by formulating an SQL query, sending it to the database engine, and retrieving the results. The SQL query can say anything related to the web site data, or its data structures, such as “give me the list of departments,” “remove product no. 223,” “create a data table,” or “search the catalog for yellow T-shirts.”

No matter what the SQL query says, we need a way to send it to MySQL. MySQL ships with a simple, text-based interface, that permits executing SQL queries and getting back the results. The command-line interface isn’t particularly easy to use, but it is functional. However, there are alternatives.

A number of free, third-party database administration tools allow you to manipulate data structures and execute SQL queries via an easy-to-use graphical interface. In this book we’ll show you how to use phpMyAdmin, which is the most widely used MySQL admin tool in the PHP world (its interface is a PHP web interface). Many web-hosting companies offer database access through phpMyAdmin, which is another good reason for you to be familiar with this tool. However, you can use the visual client of your choice. You can find complete installation instructions for phpMyAdmin in Appendix A.

Apart from needing to interact with MySQL with a direct interface to its engine, you also need to learn how to access MySQL programmatically, from PHP code. This requirement is obvious, because the e-commerce web site will need to query the database to retrieve catalog information (departments, categories, products, and so on), when building pages for the visitors.

As for querying MySQL databases through PHP code, the tool you’ll rely on here is PEAR DB.

Implementing Database Integration Using PEAR DB

PEAR (PHP Extension and Application Repository) represents a structured library of open-source code for PHP users. PEAR DB is the database abstraction layer package in PEAR, and offers a very flexible and powerful means to interact with database servers.

PEAR DB layers itself on top of PHP’s existing database functionality, offering a uniform way to access a variety of data sources. Using PEAR DB increases your application’s portability and flexibility because if the backend database changes, the effects on your data access code are kept to a minimum (in many cases, all that needs to change is the connection string for the new database).

After you become familiar with the PEAR DB database abstraction layer, you can use the same programming techniques on other projects that might require a different database solution.

To demonstrate the difference between accessing the database using native PHP functions and PEAR DB, let’s take a quick look at two short PHP code snippets. If you aren’t familiar with how the code works, don’t worry—we’ll analyze everything in greater detail in the next chapter.

- Database access using PHP native (MySQL-specific) functions:

```
/* Connecting to MySQL */
$link = mysql_connect("mysql_host", "mysql_user", "mysql_password")
               or die("Could not connect: " . mysql_error());
/* Connect to database */
mysql_select_db("my_database") or die("Could not select database");
/* Execute SQL query */
$query = "SELECT * FROM product";
$result = mysql_query($query) or die("Query failed : " . mysql_error());
/* Close connection */
mysql_close($link);
```

- The same action, this time using PEAR DB:

```
/* reference PEAR DB library */
require_once 'DB.php';
/* Data Source Name */
$dsn = "mysql://user:pass@host/db_name";
/* Connect to database */
$db = DB::connect($dsn);
/* Test if we have a valid connection */
if (DB::isError($db)) die ($db->getMessage());
/* Execute SQL query */
$sql = "SELECT * FROM product";
$result = $db->query($sql);
/* Check if the SQL query executed successfully */
if (DB::isError($result)) die ($result->getMessage());
/* Closing connection */
$db->disconnect();
```

When using PEAR DB, you won't need to change the data access code if, for example, you decide to use PostgreSQL instead of MySQL. On the other hand, the first code snippet, which uses MySQL-specific functions, would need to change completely (use `pg_connect` and `pg_query` instead of `mysql_connect` and `mysql_query`, and so on). Also, some PostgreSQL-specific functions have different parameters than the similar MySQL functions.

When using a database abstraction layer (like PEAR DB), you'll probably only need to change the connection string when changing the database backend. Note that here we're only talking about the PHP code that interacts with the database. In practice, you might also need to update some SQL queries if the database engines support different dialects of SQL.

■**Note** To keep your SQL queries as portable as possible, keep their syntax as close as possible to the SQL-92 standard. You'll learn more about SQL details in Chapter 3.

MySQL and the Three-Tier Architecture

The code presented in this book was designed to work with MySQL 4.0 (the most recent stable version at the time of writing), and MySQL 4.1. MySQL consists of the data store in the e-commerce software project. Data access logic (the SQL queries) will be centralized in a number of PHP classes consisting of the data tier of the application.

Note MySQL 5 will support stored procedures. Stored procedures contain SQL code and are managed and stored by the MySQL database itself, and provide more clarity and improved performance over the traditional method of storing the SQL logic in PHP code.

When using MySQL 4.0 (or MySQL 4.1), we are going to compensate for the lack of stored procedures by keeping all the data logic in our data tier classes. We design this tier to make the eventual transition to MySQL 5 stored procedures as easy as possible, without needing to change any code in the presentation and business tiers. Figure 2-6 shows the technologies used in each tier, using MySQL 4.0 (or 4.1) and MySQL 5.

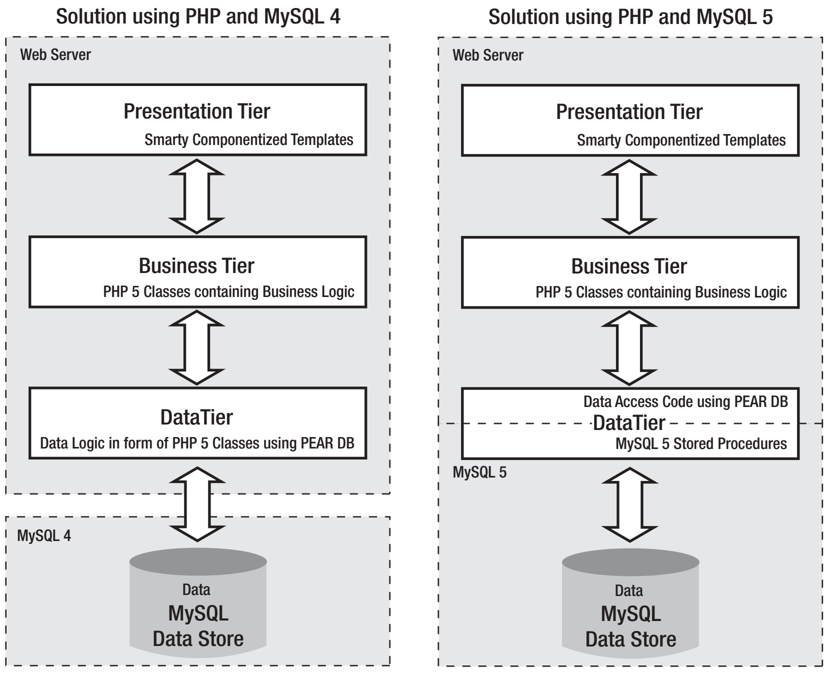

Solution using PHP and MySQL 4
Web Server
Presentation Tier
Smarty Componentized Templates
Business Tier
PHP 5 Classes containing Business Logic
DataTier
Data Logic in form of PHP 5 Classes using PEAR DB
MySQL 4
Data
MySQL
Data Store
Solution using PHP and MySQL 5
Web Server
Presentation Tier
Smarty Componentized Templates
Business Tier
PHP 5 Classes containing Business Logic
Data Access Code using PEAR DB
DataTier
MySQL 5 Stored Procedures
MySQL 5
Data
MySQL
Data Store

Figure 2-6. *The technologies you'll use to develop TShirtShop.*

Choosing Naming and Coding Standards

Although coding and naming standards might not seem that important at first, they definitely shouldn't be overlooked. Not following a set of rules for your code will almost always result in code that's hard to read, understand, and maintain. On the other hand, when you follow a consistent way of coding, you can almost say your code is already half documented, which is an important contribution toward the project's maintainability, especially when more people are working on the same project at the same time.

■ **Tip** Some companies have their own policies regarding coding and naming standards, whereas in other cases, you'll have the flexibility to use your own preferences. In either case, the golden rule to follow is *be consistent in the way you code*. Commenting your code is another good practice that improves the long-term maintainability of your code.

Naming conventions refer to many elements within a project, simply because almost all of a project's elements have names: the project itself, files, classes, variables, methods, method parameters, database tables, database columns, and so on. Without some discipline when naming all those elements, after a week of coding you won't understand a single line of what you've written.

When developing TShirtShop, we followed the naming conventions described at `http://alltasks.net/code/php_coding_standard.html`. These standards are followed by many professional PHP programmers and we'll try to keep our code close to it. Some of the most important rules are summarized here on the following piece of code:

```
class WarZone
{
   public $mSomeSoldier;
   public $mAnotherSoldier;

   function SearchAndDestroy($someEnemy,$anotherEnemy)
   {
     $master_of_war=”Soldier”;
     $this->mSomeSoldier = $someEnemy;
     $this->mAnotherSoldier = $anotherEnemy;
   }
}
```

- Class names and method names should be written using Pascal casing (uppercase letters for the first letter in every word), such as in `GetHtmlOuput` or `IsDataValid`.
- Class attribute names follow the same rules as class names, but should be prepended with the character "m". So, valid attribute names look like this: `$mGetData,$mWriteData`.
- Method argument names should use camel casing (uppercase letters for the first letter in every word except the first one), such as `$someEnemy`, `$anotherEnemy`.

- Variable names should be written in lowercase, with an underscore as the word separator, such as `$master_of_war`.
- Database objects use the same conventions as variable names (the `department_id` column).
- Try to indent your code using a fixed number of spaces (say, four) for each level. This book uses fewer spaces because of physical space limitations.

Starting the TShirtShop Project

So far, we have dealt with theory regarding the application you're going to create. It was fun, but it's going to be even more interesting to put into practice what you've learned up until now.

Start your engines!

Installing the Required Software

The code in this book has been tested with

- PHP 5.0
- Apache 2.0
- MySQL 4.0 and 4.1

The project should work with other web servers as well, as long as they're compatible with PHP 5.0 (see `http://www.php.net/manual/en/installation.php`). However, Apache is the web server of choice for the vast majority of PHP projects.

There are places where MySQL 4.1's capability to handle subqueries offers more possibilities to implement the same query. We'll try to highlight these situations, while keeping the main code compatible with MySQL 4.0.

See Appendix A for detailed installation instructions for PHP, Apache, and MySQL.

Getting a Code Editor

Before writing the first line of code, you'll need to install a code editor, if you don't already have one. A good choice is the freely available and cross-platform SciTe text editor that you can download at `http://scintilla.sourceforge.net/SciTEDownload.html`.

Alternatively, you can choose a professional commercial PHP IDE such as Zend Studio, or a simple text editor like Notepad. It's a matter of taste and money. You can find a list of PHP editors at `http://www.php-editors.com`.

Preparing the tshirtshop Virtual Folder

One of the advantages of working with open-source, platform-independent technologies is that you can choose the operating system to use for development. You should be able to develop and run TShirtShop on Windows, Unix, Linux, Mac, and others.

When setting up the project's virtual folder, a few details differ depending on the operating system (mostly because of the different file paths), so we'll cover them separately for Windows and for Unix systems in the following pages. However, the main steps are the same:

1. Create a folder in the file system named `tshirtshop` (we use lowercase for folder names), which will contain the TShirtShop project's files (such as PHP code, image files, and so on).

2. Edit Apache's configuration file (`httpd.conf`) to create a virtual folder named `tshirtshop` that points to the `tshirtshop` physical folder created earlier. This way, when pointing a web browser to `http://localhost/tshirtshop`, the project in the `tshirtshop` physical folder will be loaded. This functionality is implemented in Apache using aliases, which are configured through the `httpd.conf` configuration file. The syntax of an alias entry is as follows:

   ```
   Alias virtual_folder_name real_folder_name
   ```

■ **Tip** The `httpd.conf` configuration file is well documented, but you can also check the Apache 2 documentation available at `http://httpd.apache.org/docs-2.0/`.

If you're working on Windows, follow the steps in the following exercise. The steps for Unix systems will follow after this exercise.

Exercise: Preparing the tshirtshop Virtual Folder on Windows

Follow the steps in this exercise to set up the `tshirtshop` folder.

1. Create a new folder named `tshirtshop`, which will be used for all the work you'll do in this book. You might find it easiest to create it in the root folder (`C:\`), but because we'll use relative paths in the project, feel free to create it in any location.

2. The default place used by Apache to serve client requests from is usually something like `C:\Program Files\Apache Group\Apache2\htdocs`. This location is defined by the `DocumentRoot` directive in the Apache configuration file, which is located in the `APACHE_BASE/conf/httpd.conf` file (where `APACHE_BASE` is the Apache installation folder).

   Because we want to use our folder instead of the default folder mentioned by `DocumentRoot`, we need to create a virtual folder named `tshirtshop` that points to the `tshirtshop` physical folder you created in Step 1. Open the Apache configuration file (`httpd.conf`), find the Aliases section, and add the following lines:

   ```
   Alias /tshirtshop/ "c:/tshirtshop/"
   Alias /tshirtshop "c:/tshirtshop"
   ```

   After adding these lines, a request for `http://localhost/tshirtshop` or `http://localhost/tshirtshop/` will result in the application in the `tshirtshop` folder (if it existed) being executed.

■**Note** If the web server is not running on the default port (80), you need to manually supply the port number if you need to reach it using a web browser. So if you installed Apache to run on port 8080, you'll need to browse to `http://localhost:8080/tshirtshop` instead of `http://localhost/tshirtshop`. Also, keep in mind that instead of `localhost`, you can always use `127.0.0.1` (the computer's loopback address), or the network name of your machine.

3. Create a file named `test.php` in the `tshirtshop` folder, with the following line inside:

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

4. Restart the Apache web server, and load `http://localhost/tshirtshop/test.php` (or `http://localhost:8080/tshirtshop/test.php` if Apache works on port 8080) in a web browser.

Exercise: Preparing the tshirtshop Virtual Folder on Unix Systems

Follow the steps in this exercise to set up the `tshirtshop` folder.

1. Create a new folder named `tshirtshop`, which will be used for all the work you'll do in this book. You might find it easiest to create it in your home directory (in which case the complete path to your `tshirtshop` folder will be something like `/home/username/tshirtshop`), but because we'll use relative paths in the project, feel free to create it in any location.

2. The default place used by Apache to serve client requests from is usually something like `/var/www/html`. This location is defined by the `DocumentRoot` directive in the Apache configuration file, whose complete path is usually `/etc/httpd/conf/httpd.conf`

   Because we want to use our folder instead of the default folder mentioned by `DocumentRoot`, we need to create a virtual folder named `tshirtshop` that points to the tshirtshop physical folder you created in Step 1. Open the Apache configuration file (`httpd.conf`), find the Aliases section, and add the following lines:

   ```
   Alias /tshirtshop/ "/home/username/tshirtshop/"
   Alias /tshirtshop "/home/username/tshirtshop"
   ```

   After adding these lines, a request for `http://localhost/tshirtshop` or `http://localhost/tshirtshop/` will result in the application in the `tshirtshop` folder (if it existed) being executed.

■**Note** If the web server is not running on the default port (80), you need to manually supply the port number if you need to reach it using a web browser. So if you installed Apache to run on port 8080, you'll need to browse to `http://localhost:8080/tshirtshop` instead of `http://localhost/tshirtshop`. Also, keep in mind that instead of `localhost`, you can always use `127.0.0.1` (the computer's loopback address), or the network name of your machine.

3. Create a file named `test.php` in the `tshirtshop` folder, with the following line inside:

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

4. The last step in this exercise has to do with setting security options. You need to ensure that Apache has access to the `tshirtshop` folder.

   In a default Windows installation, Apache runs as the `SYSTEM` user so it has wide privileges locally (including write access to the `templates_c` directory, where the Smarty engine needs to save its compiled template files).

   If you're building your project under a Unix system, you should execute the following commands to ensure your Apache server can access your project's files, and has write permissions to the `templates_c` directory:

   ```
   chmod a+rx /home/mylinuxuser
   chmod -R a+rx  /home/mylinuxuser/tshirtshop
   chmod a+w /home/mylinuxuser/templates_c
   ```

   ■**Note** Setting permissions on Unix systems as shown here allows any user with a shell account on your Unix box to view the source code of any files in your folder, including PHP code and other data (which might include sensitive information such as database passwords, keys used to encrypt/decrypt credit card information, and so on). Running PHP in safe mode with a good configuration file solves the problem of the "custom PHP scripts," but if you have shell accounts on your box, you should go the suexec way, which is the most secure although not easy to configure. Setting up PHP for a shared server web hosting or for a Unix box with multiple shell accounts is beyond the scope of this book.

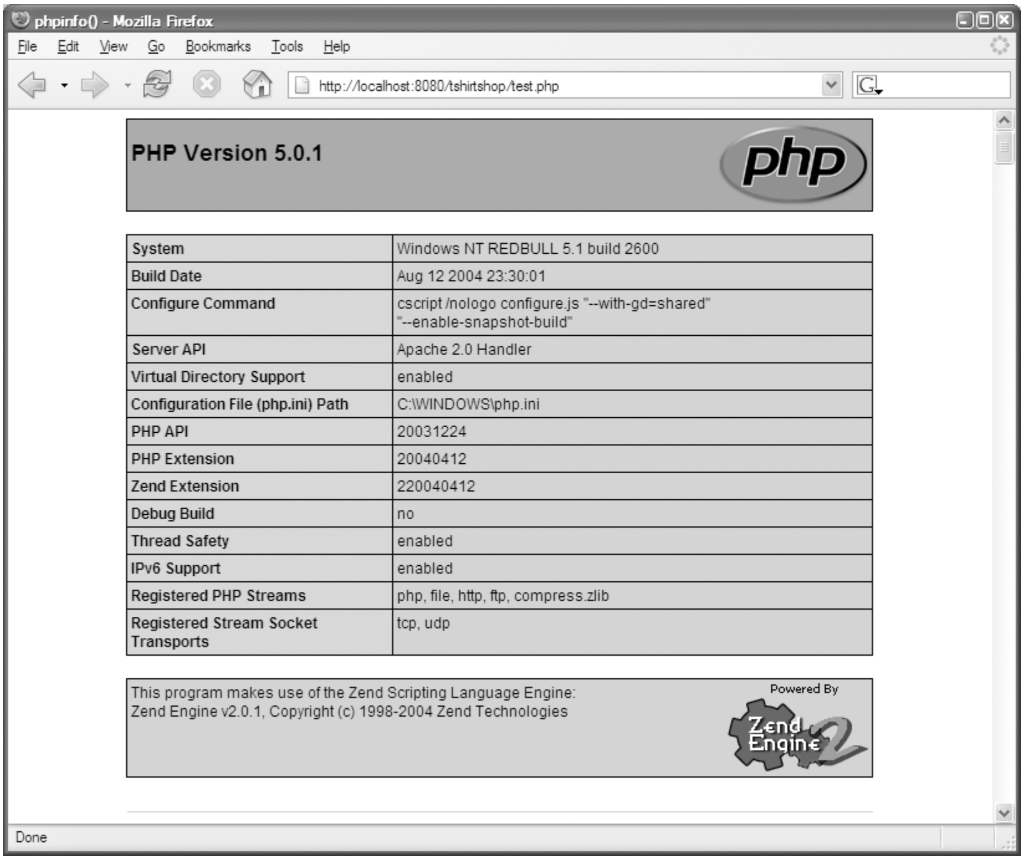
   ---

5. Restart the Apache web server, and load `http://localhost/tshirtshop/test.php` (or `http://localhost:8080/tshirtshop/test.php` if Apache works on port 8080) in a web browser.

How It Works: The Virtual Folder

This first step toward building the TShirtShop e-commerce site is a small, but important, one because it allows you to test that Apache, PHP, and the `tshirtshop` alias work okay. If you have problems running the test page, make sure you followed the installation steps in Appendix A correctly.

No matter whether you're working on Windows or a Unix flavor, loading `test.php` in a web browser should give you the PHP information returned by the `phpinfo` function as shown in Figure 2-7.


phpinfo() - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/tshirtshop/test.php
Done

PHP Version 5.0.1

System	Windows NT REDBULL 5.1 build 2600
Build Date	Aug 12 2004 23:30:01
Configure Command	cscript /nologo configure.js "--with-gd=shared" "--enable-snapshot-build"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS\php.ini
PHP API	20031224
PHP Extension	20040412
Zend Extension	220040412
Debug Build	no
Thread Safety	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, http, ftp, compress.zlib
Registered Stream Socket Transports	tcp, udp

This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.0.1, Copyright (c) 1998-2004 Zend Technologies

Powered By Zend Engine 2

Figure 2-7. *Testing PHP and the* `tshirtshop` *virtual folder*

■ **Tip** In case you were wondering, we use the FireFox web browser for taking our screenshots. Have a look at it at `http://www.mozilla.org/products/firefox/`.

Installing Smarty and PEAR

The Smarty and PEAR libraries are PHP classes you can simply install in your project's folder and use from there. Many web-hosting companies provide these for you, but it's better to have your own installation for two reasons:

- It's always preferable to make your project independent of the server's settings, when possible.
- Even if the hosting system has PEAR and Smarty installed, that company's version might be changed in time, perhaps without notice, possibly affecting your web site's functionality.

You'll install PEAR and Smarty into a subfolder of the `tshirtshop` folder named libs in the following exercise. The steps should work the same no matter what operating system you're running on.

Exercise: Installing Smarty and PEAR

Follow the steps to have Smarty and PEAR working in your project's folder:

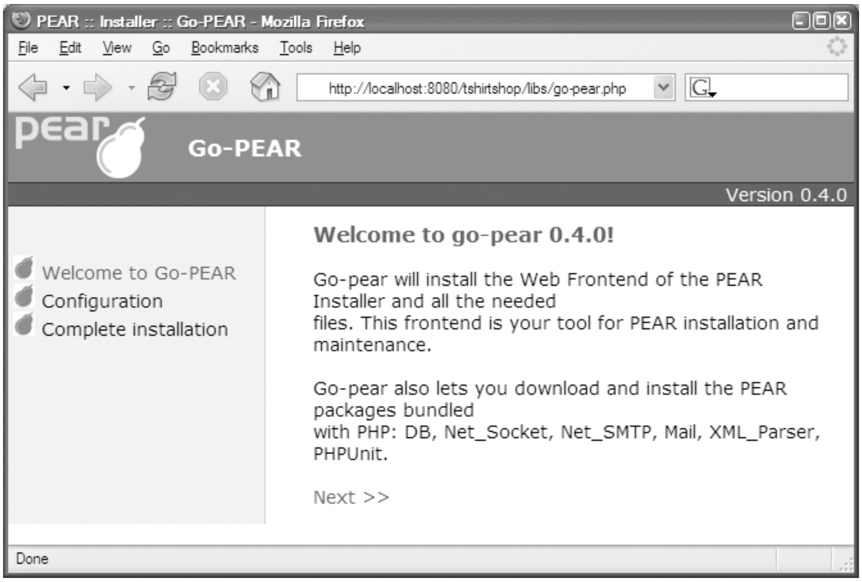
1. Create a folder named `libs` inside the `tshirtshop` folder, and then a folder named `smarty` inside the `libs` folder. Download the latest version of Smarty from `http://smarty.php.net/download.php`, and copy the contents of the `Smarty-2.x.x/libs` directory from the archive to the folder you created earlier (`tshirtshop/libs/smarty`).

2. To operate correctly, Smarty requires you to create three folders: `templates`, `templates_c`, and `configs`. Create these folders in your `tshirtshop` folder.

3. Now you can install PEAR. Use a web browser to navigate to `http://pear.php.net/go-pear`. This page contains the source code for a page. Save that code as a file named `go-pear.php` in your `tshirtshop/libs` folder, and then run it by loading it (`http://localhost/tshirtshop/libs/go-pear.php`) into a web browser (see Figure 2-8).

   Click Next, and then in the next screen, click Install. Clicking on the `Start Web Frontend of the PEAR Installer >>` link in the final screen forwards you to `http://localhost/index.php`, which might not exist on your system.

   After following these steps, you'll find the PEAR subdirectory in your `tshirtshop/libs` directory.


PEAR :: Installer :: Go-PEAR - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/tshirtshop/libs/go-pear.php
pear
Go-PEAR
Version 0.4.0
Welcome to Go-PEAR
Configuration
Complete installation
Next >>
Done

Welcome to go-pear 0.4.0!

Go-pear will install the Web Frontend of the PEAR Installer and all the needed files. This frontend is your tool for PEAR installation and maintenance.

Go-pear also lets you download and install the PEAR packages bundled with PHP: DB, Net_Socket, Net_SMTP, Mail, XML_Parser, PHPUnit.

Figure 2-8. *Installing PEAR*

How It Works: The Smarty and PEAR installation

In this exercise you created these three folders used by Smarty:

- The `templates` folder will contain the Smarty templates for your web site (.tpl files).
- The `templates_c` folder will contain the compiled Smarty templates (generated automatically by the Smarty engine).
- The `configs` folder will contain configuration files you might need for templates.

After adding these folders and installing PEAR, your folder structure should look like this:

```
tshirtshop/
    configs/
    libs/
        PEAR/
        smarty/
    templates/
    templates_c/
```

You also ensured that the `tshirtshop` directory and all its contents can be accessed properly by the web server.

Implementing the Site Skeleton

The visual design of the site is usually agreed upon after a discussion with the client, and in collaboration with a professional web designer. Alternatively, you can buy a web site template from one of the many companies that offer this kind of service for a reasonable price.

Because this is a programming book, we won't focus on web design issues. Furthermore, we want to have a pretty simple design that allows you to focus on the technical details of the site. Having a simplistic design also makes your life easier if you'll need to apply your layout on top of the one we're creating here.

All pages in TShirtShop, including the first page, will have the structure shown in Figure 2-9.

Although the detailed structure of the product catalog is covered in the next chapter, right now we know that a main list of departments needs to be displayed on every page of the site. When the visitor clicks on a department, the list of categories for that department will appear below the departments list. The site also has the search box that will allow visitors to perform product searches. At the top of the page, the site header will be visible in any page the visitor browses to.

To implement this structure as simply as possible, we'll use Smarty Componentized Templates (or simple Smarty Design Templates) to create the separate parts of the page as shown in Figure 2-10.

As Figure 2-10 suggests, you will create a Smarty componentized template named `departments_list`, and a simple Smarty design template file named `header.tpl`, which will help you populate the first page.

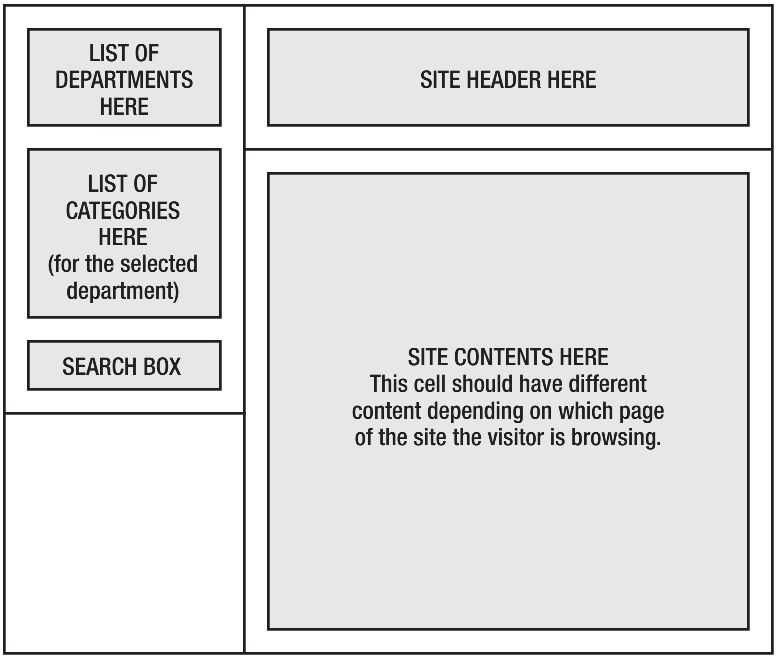

LIST OF DEPARTMENTS HERE
SITE HEADER HERE
LIST OF CATEGORIES HERE (for the selected department)
SEARCH BOX
SITE CONTENTS HERE This cell should have different content depending on which page of the site the visitor is browsing.

Figure 2-9. *Structure of web pages in TShirtShop*

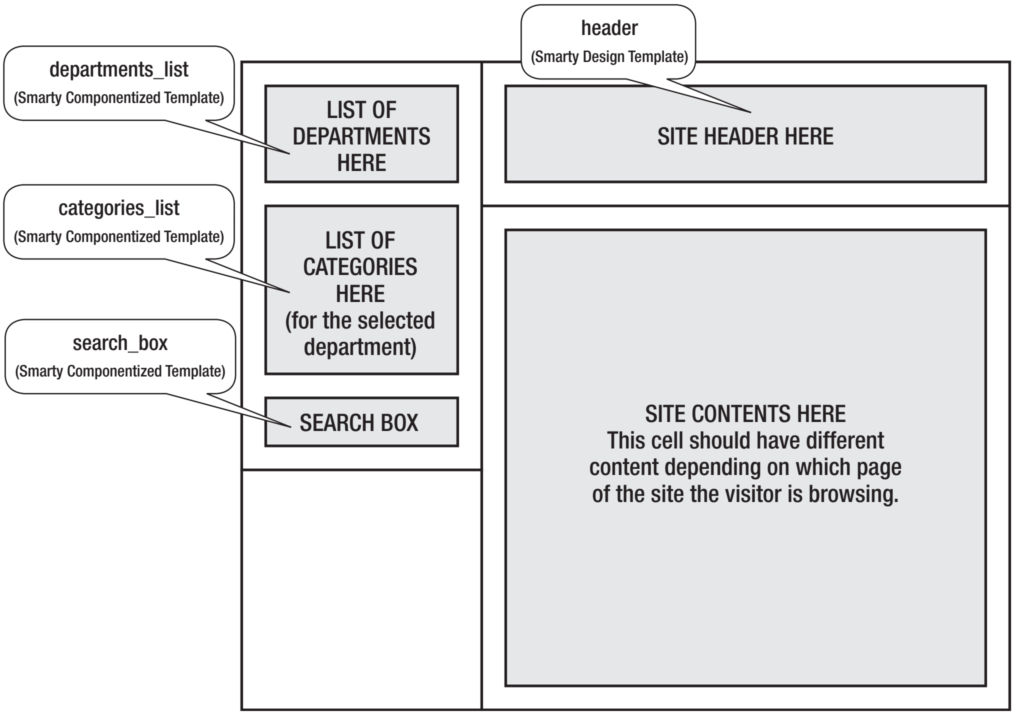

header (Smarty Design Template)
departments_list (Smarty Componentized Template)
LIST OF DEPARTMENTS HERE
SITE HEADER HERE
categories_list (Smarty Componentized Template)
LIST OF CATEGORIES HERE (for the selected department)
search_box (Smarty Componentized Template)
SEARCH BOX
SITE CONTENTS HERE This cell should have different content depending on which page of the site the visitor is browsing.

Figure 2-10. *Using Smarty to generate content*

Note We call **Smarty Componentized Template** the combination of a **Smarty Design Template** (the .tpl file) and its associated **Smarty Plugin file** which contains the presentation tier logic (a .php file). In cases of simple pages that don't need an associated .php code file, such as the header, we'll use just a Smarty Design Template file. You'll meet Smarty plugins in Chapter 3, and you can learn more about them at `http://smarty.php.net/manual/en/plugins.php`.

Using Smarty templates to implement different pieces of functionality provides benefits discussed earlier in the chapter. Having different, unrelated pieces of functionality logically separated from one another gives you the flexibility to modify them independently, and even reuse them in other pages without having to write their code again. It's also extremely easy to change the place in the parent web page of a feature implemented as a Smarty template.

The list of departments, the search box, and the site header are elements that will be present in every page of the site. The list of categories appears only when the visitor selects a department from the list. The most dynamic part of the web site that changes while browsing through the site will be the contents cell, which will update itself depending on the site location requested by the visitor. There are two main options for implementing that cell: add a componentized template that changes itself depending on the location or use different componentized templates to populate the cell depending on the location being browsed. There is no rule of thumb about which method to use, because it mainly depends on the specifics of the project. For TShirtShop, you will create a number of componentized templates that will fill that location.

In the remainder of this chapter, you will

- Create the main web page and the `header` componentized template.
- Implement the foundations of the error-handling system in TShirtShop.
- Create the TShirtShop database.

Building the First Page

The main page in TShirtShop will be generated by `index.php` and `index.tpl`.

You'll write the `index.tpl` Smarty template with placeholders for the three major parts of the site—the header, the table of departments, and the page contents cell. Implement the main page in the following exercise, and we'll discuss the details in the "How it Works" section thereafter.

Exercise: Implementing the First Page and Its Header

Follow these steps:

1. Create a new folder named `images` inside the `tshirtshop` folder.
2. Copy the files in `image folders/images` from the code download of the book (which you can find at `http://www.apress.com` or `http://www.CristianDarie.ro`) to `tshirtshop/images` (the folder you just created).

3. Create a file named `site.conf` in the `tshirtshop/configs` folder (used by the Smarty templates engine), and add the following line to it:

```
sitetitle = "TShirtShop - The Best E-Commerce Store on the Internet"
```

4. Create a file named `index.tpl` in tshirtshop/templates, and add the following code to it:

```
{* smarty *}
{config_load file="site.conf"}
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>{#sitetitle#}</title>
  </head>
  <body>
    <table cellspacing="0" cellpadding="0" width="750" border="0">
      <tr>
        <td width="200" height="100%" valign="top">
          <table  width="100%" cellspacing="0" cellpadding="0">
            <tr>
              <td valign="top" height="100%">
                Place list of departments here
              </td>
            </tr>
          </table>
        </td>
        <td>&nbsp;&nbsp;&nbsp;</td>
        <td valign="top" width="550"><br />
          {include file="header.tpl"}
          Place contents here
        </td>
      </tr>
    </table>
  </body>
</html>
```

5. Create a template file named `header.tpl` in `tshirtshop/templates` and add the following contents to it.

```
<table border="0" width="550" cellspacing="0" cellpadding="0">
  <tr align="right">
    <td>
     <a href="index.php">
      <img src="images/title.png" border="0" alt="site title" />
     </a>
    </td>
  </tr>
  <tr align="right">
```

```
    <td>
      <img src="images/1n.gif" border="0" width="350" height="1"
           alt="line here" />
    </td>
  </tr>
</table>
```

6. Create a folder named `tshirtshop/include` and add a file named `config.inc.php` to it with the following contents:

```
<?php
// SITE_ROOT contains the full path to the tshirtshop folder
define("SITE_ROOT", dirname(dirname(__FILE__)));
// Settings needed to configure the Smarty template engine
define("SMARTY_DIR", SITE_ROOT."/libs/smarty/");
define("TEMPLATE_DIR", SITE_ROOT."/templates");
define("COMPILE_DIR", SITE_ROOT."/templates_c");
define("CONFIG_DIR", SITE_ROOT."/configs");
?>
```

Before moving on, let's see what happens here. `dirname(__FILE__)` returns the parent directory of the current file; naturally, `dirname(dirname(__FILE__))` returns the parent of the current file's directory. This way our `SITE_ROOT` constant will be set to the full path of `tshirtshop`. With the help of the `SITE_ROOT` constant, we set up absolute paths of Smarty folders.

7. Create a file named `setup_smarty.php` in the `tshirtshop/include` folder, and add the following contents to it:

```
<?php
// Reference Smarty library
require_once SMARTY_DIR.'Smarty.class.php';
// Reference our configuration file
require_once 'config.inc.php';

// Class that extends Smarty, used to process and display Smarty files
class Page extends Smarty
{
  // constructor
  function __construct()
  {
    // Call Smarty's constructor
    $this->Smarty();
    // Change the default template directories
    $this->template_dir = TEMPLATE_DIR;
    $this->compile_dir = COMPILE_DIR;
    $this->config_dir = CONFIG_DIR;
  }
}
?>
```

In `setup_smarty.php`, you extend the Smarty class with a wrapper class named `Page`, which changes Smarty's default behavior. The `Page` class configures in its constructor the Smart folders you created earlier.

■ **Tip** As mentioned earlier, Smarty requires three folders to operate: `templates`, `templates_c`, and `configs`. In the constructor of the `Page` class, we set a separate set of these directories for our application. If you want to turn on caching, then Smarty also needs a directory named cache. We will not be using Smarty caching for TShirtShop, but you can read more details about this in the Smarty manual at `http://smarty.php.net/manual/en/caching.php`.

---

8. Create the `tshirtshop/include/app_top.php` file, and add the following contents to it:

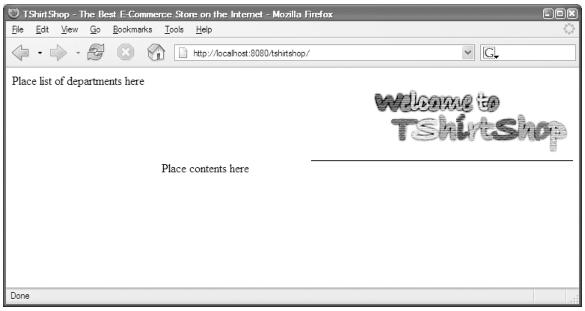
```
<?php
// include utility files
require_once 'config.inc.php';
require_once 'setup_smarty.php';
?>
```

This file (`app_top.php`) will be included at the top of the main web pages to perform the necessary initializations.

9. Add the `index.php` file to the `tshirtshop` folder. The role of this file is to load the `index.tpl` template by using the `Page` class you created earlier. Here's the code for `index.php`:

```
<?php
// load Smarty library and config files
require_once 'include/app_top.php';
// Load Smarty template file
$page = new Page();
$page->display('index.tpl');
?>
```

10. Now it's time to see some output from this thing. Load `http://localhost/tshirtshop/` in your favorite web browser and admire the results as shown in Figure 2-11.


TShirtShop - The Best E-Commerce Store on the Internet - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/tshirtshop/
Welcome to TShirtShop
Done

Place list of departments here

Place contents here

Figure 2-11. *Running TShirtShop*

How It Works: The First Page of TShirtShop

The main web page contains three major sections. There are two table cells that you'll fill with componentized templates—one for the list of departments and one for the page contents—in the following chapters.

Notice the departments list placed on the left side, the header at the top, and the contents cell filled with information regarding the first page. As previously mentioned, this contents cell is the only one that changes while browsing the site; the other two cells will look exactly the same no matter what page is visited. This implementation eases your life as a programmer and keeps a consistent look and feel of the web site.

Before you move on, it's important to understand how the Smarty template works. Everything starts from `index.php`, so you need to take a close look at it. Here's the code again:

```
<?php
// load Smarty library and config files
require_once 'include/app_top.php';
// Load Smarty template file
$page = new Page();
$page->display('index.tpl');
?>
```

At this moment, this file has very simple functionality. First it loads `app_top.php`, which sets some global variables, and then it loads the Smarty template file, which will generate the actual HTML content when a client requests `index.php`.

The standard way to create and configure a Smarty page is shown in the following code snippet:

```
<?php
// Load the Smarty library
require("smarty/Smarty.class.php");
// Create a new instance of the Smarty class.
$smarty = new Smarty();
$smarty->template_dir= TEMPLATE_DIR;
$smarty->compile_dir= COMPILE_DIR;
$smarty->config_dir= CONFIG_DIR;
?>
```

In TShirtShop, we created a class named `Page` that inherits Smarty, which contains the initialization procedure in its constructor. This makes working with Smarty templates easier. Here's again the code of the `Page` class:

```
class Page extends Smarty
{
  // constructor
  function __construct()
  {
    // Call Smarty's constructor
    $this->Smarty();
    // Change the default template directories
    $this->template_dir = TEMPLATE_DIR;
    $this->compile_dir = COMPILE_DIR;
    $this->config_dir = CONFIG_DIR;
  }
}
```

■ Note The notion of constructor is specific to object-oriented programming terminology. The constructor of a class is a special method that executes automatically when an instance of that class is created. In PHP, the constructor of a class is called `__construct()`. Writing that code in the constructor of the `Page` class guarantees that it gets executed automatically when a new instance of `Page` is created.

The Smarty template file (`index.tpl`), except for a few details, contains simple HTML code. Those details are worth analyzing. In `index.tpl`, before the HTML code begins, the configuration file `site.conf` is loaded.

```
{* smarty *}
{config_load file="site.conf"}
```

■ Tip Smarty comments are enclosed between `{*` and `*}` marks.

At this moment, the only variable set in `site.conf` is `sitetitle`, which contains the name of the web site. The value of this variable is used to generate the title of the page in the HTML code:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>{#sitetitle#}</title>
  </head>
```

Variables that are loaded from the config files are referenced by enclosing them within hash marks (`#`), or with the smarty variable `$smarty.config`, as in:

```
  <head>
    <title>{$smarty.config.sitetitle}</title>
  </head>
```

We loaded the `site.conf` config file using `{config_load file="site.conf"}` and accessed the `sitetitle` variable with `{#sitetitle#}`, which you'll use whenever you need to obtain the site title. If you want to change the site title, all you have to do is to edit `site.conf`.

Finally, it's important to notice how to include a Smarty template in another Smarty template. `index.tpl` references `header.tpl`, which will also be reused in a number of other pages:

```
<td valign="top" width="550"><br/>
  {include file="header.tpl"}
  Place contents here
</td>
```

Handling and Reporting Errors

Although the code will be written to run without any unpleasant surprises, there's always a possibility that something might go wrong when processing client requests. The best strategy to deal with these unexpected problems is to find a centralized way to handle these errors and perform certain actions when they do happen.

PHP is known for its confusing error messages. If you've worked with other programming languages, you probably appreciate the information you can get from displaying the stack trace when you have an error. Tracing information is not displayed by default when you have a PHP error, so you'll want to change this behavior. In the development stage, tracing information will help you debug the application, and in a release version, the error message must be reported to the site administrator. Another problem is the tricky E_WARNING error message, because it's hard to tell whether it's fatal or not for the application.

■ **Tip** If you don't remember or don't know what a PHP error message looks like, try adding the following line in your `include/app_top.php` file:

```
require_once 'inexistent_file.php';
```

Load the web site in your favorite browser, and notice the error message you get.

In the context of a live web application, errors can happen unexpectedly for various reasons, such as software failures (operating system or database server crashes, viruses, and so on) and hardware failures. It's important to be able to log these errors, and eventually inform the web site administrator (perhaps by sending an email message), so the error can be taken care of as fast as possible.

For these reasons, we'll start establishing an efficient error handling and reporting strategy. You'll create a user-defined error handler function named `tss_error_handler`, which will get executed anytime a PHP error happens during runtime. Serious error types (E_ERROR, E_PARSE, E_CORE_ERROR, E_CORE_WARNING, E_COMPILE_ERROR, and E_COMPILE_WARNING) cannot be intercepted and handled by `tss_error_handler`, but the other types of PHP errors (E_WARNING for example) can be.

■ **Note** You can find more info about PHP errors and logging in the PHP manual at `http://www.php.net/manual/en/ref.errorfunc.php`.

The error-handling method, `tss_error_handler`, will behave like this:

- It creates a detailed error message.
- If configured to do so, the error is emailed to the site administrator.
- If configured to do so, the error is logged to an errors log file.
- If configured to do so, the error is shown in the response web page.
- Serious errors will halt the execution of the page. The other ones will allow the page to continue processing normally.

Let's implement `tss_error_handler` in the next exercise.

Exercise: Implementing `tss_error_handler`

Follow these steps:

1. Add the following error-handling related configuration variables to `config.inc.php`:

```
// these should be true while developing the web site
define("IS_WARNING_FATAL", true);
define("DEBUGGING", true);
// settings about mailing the error messages to admin
define("SEND_ERROR_MAIL", false);
define("ADMIN_ERROR_MAIL", "admin_mail@localhost");
define("SENDMAIL_FROM", "errors@tshirtshop.com");
ini_set("sendmail_from", SENDMAIL_FROM);
// by default we don't log errors to a file
define("LOG_ERRORS", false);
define("LOG_ERRORS_FILE", "c:\\tshirtshop\\errors_log.txt"); // Windows
// define("LOG_ERRORS_FILE", "/var/tmp/tshirtshop_errors.log"); // Unix
// Generic error message to be diplayed instead of debug info
// (when DEBUGGING is false)
define("SITE_GENERIC_ERROR_MESSAGE", "<h2>TShirtShop Error!</h2>");
```

2. Add the `include/tss_error_handler.php` file:

```
<?php
// set the user error handler method to be tss_error_handler
set_error_handler("tss_error_handler", E_ALL);

// error handler function
function tss_error_handler($errNo, $errStr, $errFile, $errLine)
{
  /* the first two elements of the backtrace array are irrelevant:
     -DBG_Backtrace
     -outErrorHandler */
  $backtrace = dbg_get_backtrace(2);
  // error message to be displayed, logged or mailed
  $error_message = "\nERRNO: $errNo \nTEXT: " . $errStr . " \n" .
             "LOCATION: " . $errFile . ", line " . $errLine . ", at " .
             date("F j, Y, g:i a") . "\nShowing backtrace:\n" .
             $backtrace . "\n\n";
  // email the error details, in case SEND_ERROR_MAIL is true
  if (SEND_ERROR_MAIL == true)
     error_log($error_message, 1, ADMIN_ERROR_MAIL, "From: " .
               SENDMAIL_FROM . "\r\nTo: " . ADMIN_ERROR_MAIL);
  // log the error, in case LOG_ERRORS is true
  if (LOG_ERRORS == true)
    error_log($error_message, 3, LOG_ERRORS_FILE);
  // warnings don't abort execution if IS_WARNING_FATAL is false
  // E_NOTICE and E_USER_NOTICE errors don't abort execution
  if (($errNo == E_WARNING && IS_WARNING_FATAL == false) ||
     ($errNo == E_NOTICE || $errNo == E_USER_NOTICE))
    // if the error is non-fatal ...
  {
    // show message only if DEBUGGING is true
    if (DEBUGGING == true)
      echo "<pre>" . $error_message . "</pre>";
  }
  else
  // if error is fatal ...
  {
    // show error message
    if (DEBUGGING == true)
      echo "<pre>" . $error_message . "</pre>";
    else
      echo SITE_GENERIC_ERROR_MESSAGE;
    // stop processing the request
    exit;
  }
}
?>
```

3. Add the `dbg_get_backtrace` function at the end of `tss_error_handler.php`:

```
// builds backtrace message
function dbg_get_backtrace($irrelevantFirstEntries)
{
  $s = '';
  $MAXSTRLEN = 64;
  $traceArr = debug_backtrace();
  for ($i = 0; $i < $irrelevantFirstEntries; $i++)
    array_shift($traceArr);
  $tabs = sizeof($traceArr) - 1;
  foreach($traceArr as $arr)
  {
    $tabs -= 1;
    if (isset($arr['class']))
      $s .= $arr['class'] . '.';
    $args = array();
    if (!empty($arr['args']))
    foreach($arr['args']as $v)
    {
      if (is_null($v))
        $args[] = 'null';
      else if (is_array($v))
        $args[] = 'Array[' . sizeof($v).']';
      else if (is_object($v))
        $args[] = 'Object:' . get_class($v);
      else if (is_bool($v))
        $args[] = $v ? 'true' : 'false';
      else
      {
        $v = (string)@$v;
        $str = htmlspecialchars(substr($v, 0, $MAXSTRLEN));
        if (strlen($v) > $MAXSTRLEN)
          $str .= '...';
        $args[] = "\"" . $str . "\"";
      }
    }
    $s .= $arr['function'] . '(' . implode(', ', $args) . ')';
    $Line = (isset($arr['line']) ? $arr['line']: "unknown");
    $File = (isset($arr['file']) ? $arr['file']: "unknown");
    $s .= sprintf(" # line %4d, file: %s", $Line, $File, $File);
    $s .= "\n";
  }
  return $s;
}
```

4. Modify the `include/app_top.php` file to include the newly created `tss_error_handler.php` file:

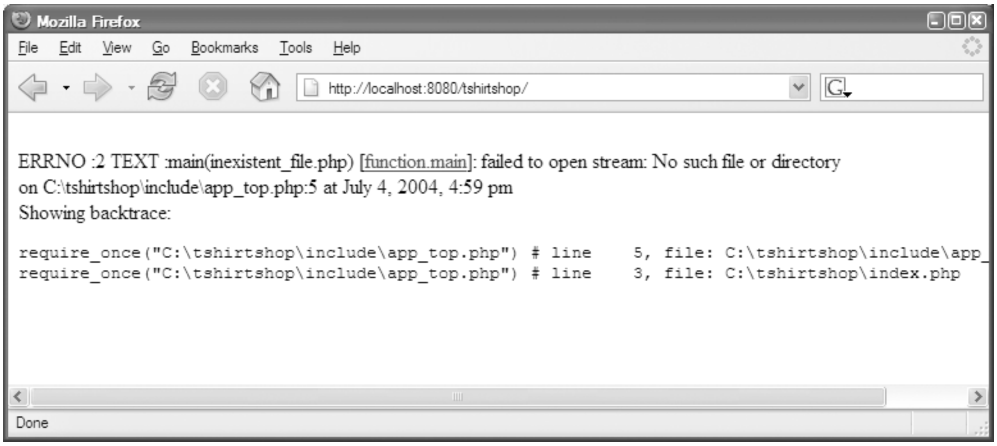
```
<?php
require_once 'config.inc.php';
require_once 'tss_error_handler.php';
require_once 'setup_smarty.php';
?>
```

5. Great! You just finished writing the new error-handling code. Let's test it. First, load the web site in your browser to see that you typed in everything correctly. If you get no errors, test the new error handling system by adding the following line to `include/app_top.php`:

```
<?php
require_once 'config.inc.php';
```

```
    require_once 'tss_error_handler.php';
    require_once 'setup_smarty.php';
    require_once 'inexistent_file.php';
    ?>
```

Now load again `index.php` in your browser, and admire your brand new error message as shown in Figure 2-12.


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/tshirtshop/
Done

ERRNO :2 TEXT :main(inexistent_file.php) [function.main]: failed to open stream: No such file or directory
on C:\tshirtshop\include\app_top.php:5 at July 4, 2004, 4:59 pm
Showing backtrace:

```
require_once("C:\tshirtshop\include\app_top.php") # line    5, file: C:\tshirtshop\include\app_
require_once("C:\tshirtshop\include\app_top.php") # line    3, file: C:\tshirtshop\index.php
```

Figure 2-12. *Error message showing backtrace information*

■**Note** Due to a bug in the PHP 5 engine (which is reported at `http://bugs.php.net/bug.php?id=28054`), this "improved" error message might be incorrect sometimes.

Don't forget to remove the buggy line from `app_top.php` before moving on.

How It Works: Error Handling

The method that intercepts web site errors and deals with them is `tss_error_handler` (located in `tss_error_handler.php`). The code that registers the `tss_error_handler` function to be the one that handles errors in your site is at the beginning of `tss_error_handler.php`:

```
// set the user error handler method to be tss_error_handler
set_error_handler("tss_error_handler", E_ALL);
```

■**Note** The second parameter of `set_error_handler` specifies the range of errors that should be intercepted. E_ALL specifies all types of errors, including E_NOTICE errors, which should be reported during web site development.

When called, `tss_error_handler()` constructs the error message with the help of a method named `dbg_get_backtrace`, and forwards the error message to the client's browser, to a log file, to the administrator (by email), or a combination of these, which can be configured by editing `config.inc.php`.

`dbg_get_backtrace()` gets the backtrace information from the `debug_backtrace` function (which was introduced in PHP 4.3.0), and changes its output format to generate an HTML error message similar to a Java error. It isn't important to understand every line in `dbg_get_backtrace` unless you want to personalize the backtrace displayed in case of an error. The `2` parameter sent to `dbg_get_backtrace` specifies that the backtrace results should omit the first two entries (the calls to `tss_error_handler` and `dbg_get_backtrace`).

You build the detailed error string in tss_error_handler, including the backtrace information:

```
$backtrace = dbg_get_backtrace(2);
// error message to be displayed, logged or mailed
$error_message = "\nERRNO: $errNo \nTEXT: " . $errStr . " \n" .
           "LOCATION: " . $errFile . ", line " . $errLine . ", at " .
           date("F j, Y, g:i a") . "\nShowing backtrace:\n" .
           $backtrace . "\n\n";
```

Depending on the configuration options from the `config.inc.php` file, you decide whether to display, log, and/or email the error. Here we use PHP's `error_log` method, which knows how to email or write the error's details to a log file:

```
// email the error details, in case SEND_ERROR_MAIL is true
if (SEND_ERROR_MAIL == true)
   error_log($error_message, 1, ADMIN_ERROR_MAIL, "From: " .
             SENDMAIL_FROM . "\r\nTo: " . ADMIN_ERROR_MAIL);
// log the error, in case LOG_ERRORS is true
if (LOG_ERRORS == true)
  error_log($error_message, 3, LOG_ERRORS_FILE);
```

■ **Note** If you want to be able to send an error mail to a localhost mail account (your_name@locahost), then you should have an SMTP (Simple Mail Transfer Protocol) server started on your machine. On a Red Hat (or Fedora) Linux distribution, you can start an SMTP server with the following command:

```
service sendmail start
```

On Windows systems, you should check in IIS (Internet Information Services) Manager for Default SMTP Virtual Server and make sure it's started.

While you are developing the site, the `DEBUGGING` constant should be set to `true`, but after launching the site in the "wild," you should make it `false`, causing a user-friendly error message to be displayed instead of the debugging information in case of serious errors, and no message shown at all in case of nonfatal errors.

The errors of type E_WARNING are pretty tricky because you don't know which of them should stop the execution of the request. The `IS_WARNING_FATAL` constant set in `config.inc.php` decides whether this type of error should be considered fatal for the project. Also, errors of type E_NOTICE and E_USER_NOTICE are not considered fatal:

```
// warnings don't abort execution if IS_WARNING_FATAL is false
// E_NOTICE and E_USER_NOTICE errors don't abort execution
if (($errNo == E_WARNING && IS_WARNING_FATAL == false) ||
   ($errNo == E_NOTICE || $errNo == E_USER_NOTICE))
  // if the error is non-fatal ...
{
  // show message only if DEBUGGING is true
  if (DEBUGGING == true)
    echo "<pre>" . $error_message . "</pre>";
}
else
// if error is fatal ...
{
  // show error message
  if (DEBUGGING == true)
    echo "<pre>" . $error_message . "</pre>";
  else
    echo SITE_GENERIC_ERROR_MESSAGE;
  // stop processing the request
  exit;
}
```

In the following chapters you'll need to manually trigger errors using the `trigger_error` PHP function, which lets you specify the kind of error to generate. By default, it generates `E_USER_NOTICE` errors which are not considered fatal, but are logged and reported, by `tss_error_handler` code.

Preparing the tshirtshop Database

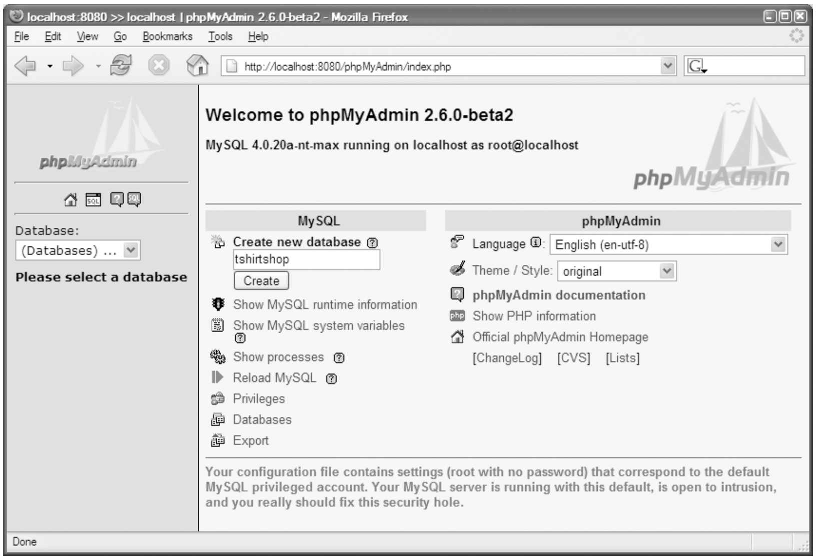
The final step in this chapter is to create the MySQL database, although you won't use it until the next chapter. We will show you the steps to create your database and create a user with full privileges to it using phpMyAdmin, but you can use other visual interfaces such as MySQL Control Center or even the MySQL text-mode console.

Before moving on, make sure you have MySQL 4.0 (or 4.1) and phpMyAdmin installed. Consult Appendix A for installation instructions.

■ **Note** For an excellent coverage of phpMyAdmin I recommend you read "Mastering phpMyAdmin for effective MySQL Management" (Packt Publishing, 2004)

Exercise: Creating the tshirtshop Database and a New User Account

1. Load the phpMyAdmin page in your favorite web browser, and type **tshirtshop** in the Create new database text box, as shown in Figure 2-13.


localhost:8080 >> localhost | phpMyAdmin 2.6.0-beta2 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/phpMyAdmin/index.php
Database:
(Databases) ...
Please select a database

Welcome to phpMyAdmin 2.6.0-beta2

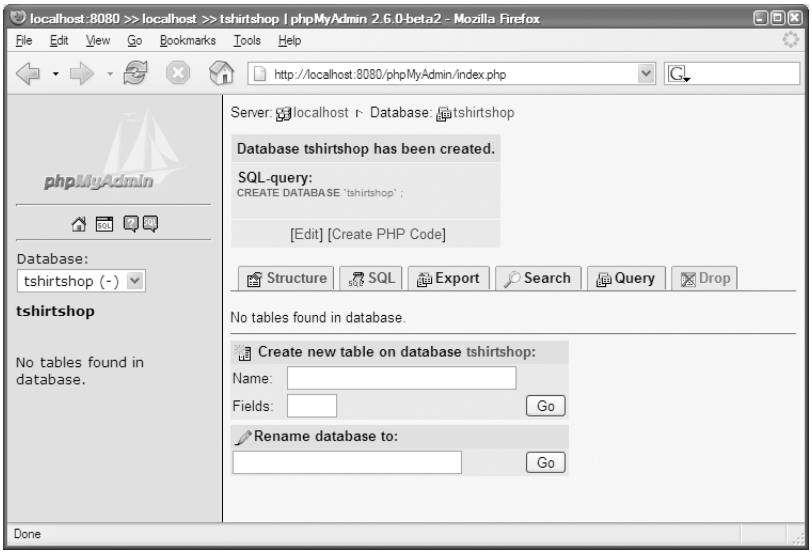
MySQL 4.0.20a-nt-max running on localhost as root@localhost

MySQL
Create new database
tshirtshop
Create
Show MySQL runtime information
Show MySQL system variables
Show processes
Reload MySQL
Privileges
Databases
Export
phpMyAdmin
Language: English (en-utf-8)
Theme / Style: original
phpMyAdmin documentation
Show PHP information
Official phpMyAdmin Homepage
[ChangeLog] [CVS] [Lists]
Done

Your configuration file contains settings (root with no password) that correspond to the default MySQL privileged account. Your MySQL server is running with this default, is open to intrusion, and you really should fix this security hole.

Figure 2-13. *The main phpMyAdmin page*

2. Click the Create button to create the new database. In the screen that follows (see Figure 2-14), you're shown the SQL query phpMyAdmin used to create your database.


localhost:8080 >> localhost >> tshirtshop | phpMyAdmin 2.6.0-beta2 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/phpMyAdmin/index.php
Database:
tshirtshop (-)
tshirtshop
No tables found in database.
Server: localhost
Database: tshirtshop

Database tshirtshop has been created.

SQL-query:

```
CREATE DATABASE `tshirtshop` ;
```

[Edit] [Create PHP Code]
Structure
SQL
Export
Search
Query
Drop

No tables found in database.

Create new table on database tshirtshop:
Name:
Fields:
Go
Rename database to:
Go
Done

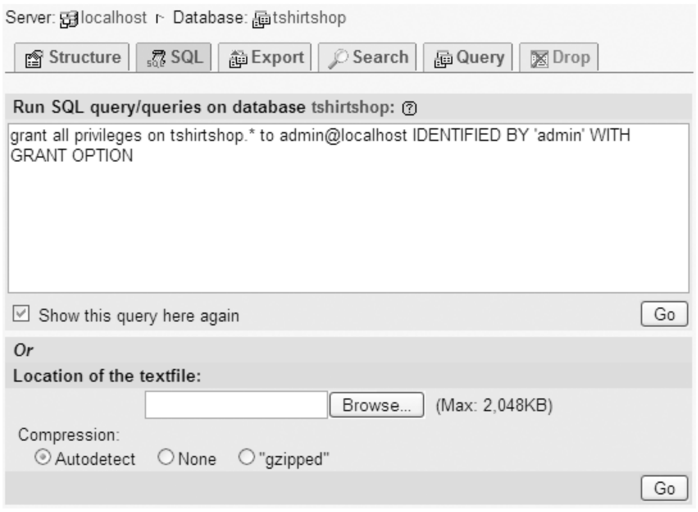
Figure 2-14. *Editing your database using phpMyAdmin*

■**Note** You'll learn more about SQL queries in Chapter 3. SQL is the language used to interact with the database. However, you can accomplish many tasks, such as creating databases and data tables, using a visual interface such as phpMyAdmin, which generates the SQL queries for you.

3. Now add a new user to the database. Our data tier access code will access the TShirtShop database using this user's credentials. You'll create a user named admin (with the password "admin"), who will have full access inside the TShirtShop database, but not to any other database.

   Because phpMyAdmin doesn't have visual interface tools to create new users or to manage security, you'll need to create the new user using an SQL query. You have more alternatives to achieve the same results, and for more details, see the MySQL documentation at `http://dev.mysql.com/doc/mysql/en/Account_management_SQL.html`.

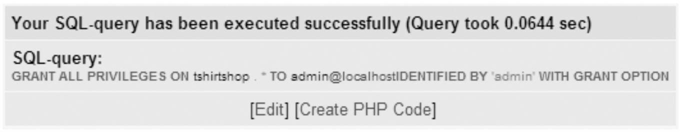
   Now click the SQL tab, and type the SQL query you can see in Figure 2-15.



```
grant all privileges on tshirtshop.* to admin@localhost IDENTIFIED BY 'admin' WITH
GRANT OPTION
```

Figure 2-15. *Creating a new database user*

4. After entering the SQL query as shown, click the Go button. You should be informed that the SQL query has been executed successfully (see Figure 2-16).



Your SQL-query has been executed successfully (Query took 0.0644 sec)

SQL-query:

```
GRANT ALL PRIVILEGES ON tshirtshop . * TO admin@localhostIDENTIFIED BY 'admin' WITH GRANT OPTION
```

Figure 2-16. *The new user has been successfully created in the database.*

Downloading the Code

You can find the latest code downloads and a link to an online version of TShirtShop at the author's web site, `http://www.CristianDarie.ro`, or in the Downloads section of the Apress web site at `http://www.apress.com`. It should be easy to read through this book and build your solution as you go; however, if you want to check something from our working version, you can. Instructions on loading the chapters are available in the `WELCOME.HTML` document in the download.

Summary

Hey, we covered a lot of ground in this chapter, didn't we? We talked about the three-tier architecture and how it helps you create great flexible and scalable applications. We also saw how each of the technologies used in this book fits into the three-tier architecture.

So far, we have a very flexible and scalable application because it only has a main web page, but you'll feel the real advantages of using a disciplined way of coding in the next chapters. In this chapter you have only coded the basic, static part of the presentation tier, implemented a bit of error-handling code, and created the TShirtShop database, which is the support for the data tier. In the next chapter you'll start implementing the product catalog and learn a lot about how to dynamically generate visual content using data stored in the database with the help of the middle tier and with smart and fast controls and components in the presentation tier.