

PHP Development

Building an E-Commerce Site Part 1: Building a Product Catalog

Contributed by Ying Zhang

2000-04-25

[Send Me Similar Content When Posted]

[Add Developer Shed Headlines To Your Site]



DISCUSS



NEWS



SEND



PRINT



PDF

MYSQL
Don't do it yourself.

advertisement

Article Index: Welcome to E-Commerce Step-by-Step. This guide is meant to show you how to build a basic online store complete with (1) a product catalog, (2) user accounts for customers, and (3) the ability for your customers to order products from your catalog.

After you complete this three part series, you will

- (part 1) understand the fundamentals components of an E-Commerce site
- (part 1) learn to create a product catalog
- (part 1) learn to create maintenance screens for the product catalog
- (part 2) learn to use the session management capabilities in PHP4
- (part 2) learn to manage users and user privileges
- (part 3) learn to display the product catalog
- (part 3) learn to allow customers to order from the product catalog
- (part 3) understand how credit card authorization and payment works

To accomplish these goals, we will develop a fictitious store called MyMarket that sells products over the Internet. For this we will use the freely available Apache, MySQL, and PHP programs.

In this installment, we will cover the administrative aspects of building a product catalog. After completing this installment, you have:

- created the MyMarket database in MySQL
- created the underlying tables for the product catalog
- created PHP scripts to add product categories
- created PHP scripts to edit product categories
- created PHP scripts to delete product categories
- created PHP scripts to add products

- created PHP scripts to edit products
- created PHP scripts to delete products

If you are reading this article, you should the following software installed and working:

- Apache 1.3.9 or higher
- MySQL 3.22.20 or higher
- PHP4 RC1 or higher with MySQL support compiled in

We are using MySQL because it is quick and easy. We are using PHP4 because it provides session management functions. By the time you read this article, the stable version of PHP4 should be just about ready for public use.

This article also assumes that you are familiar and comfortable with programming in PHP, and that you are able to write SQL queries. If you need to install PHP or MySQL, please read the follow the instructions in the article:

- [Setting Up Database Driven Websites](#)

Also, you may want to check out:

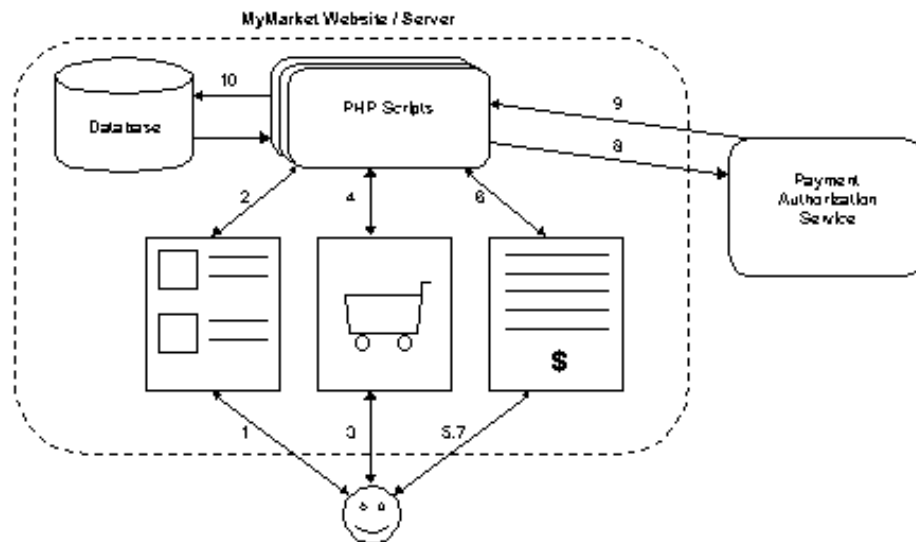
- PHP Homepage, <http://www.php.net>
- MySQL Homepage, <http://www.mysql.com>

E-Commerce systems, no matter how fancy or how simple, involve three basic functions:

- Displaying a product catalog
- Allowing customers to browse through the product catalog
- Allowing customers to buy items from the product catalog

What do you do when you visit an online merchant? You look through their product catalog to see what they have for sale. Let's say you find something that you like and would like to buy, you'd add the item into your shopping cart and then eventually complete the order by supplying payment information.

Here is a simple diagram to illustrate the process:



1. John Doe visits the MyMarket website and access the product catalog
2. The server generates the product catalog by reading the items from the database
3. John browses the catalog and adds items into his shopping cart
4. The server updates John's shopping cart with the items he has selected
5. John goes to the checkout to complete his order
6. The server generates his order summary by calculating the price of the order
7. John verifies the order, then supplies his credit card number for validation
8. The server talks to a payment authorization service to validate the credit card
9. The payment is authorized and the result is sent back to the server
10. If all goes well, and John's transaction is saved into the database and John waits for the socks

That's the big picture of the entire process. For now it may be a little confusing, but as we go through each of the steps everything will make more sense. We will begin by creating the product catalog, and the maintenance scripts that will us to manage it.

We will create a database called mymarket and then a user account called myuser with the password mypassword. Fire up MySQL and login as the root user by issuing this command from the shell prompt:

```
$ mysql -u root -p
```

You should see MySQL started:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 412 to server version: 3.22.30 Type 'help' for help. mysql>
```

From here, issue the following commands:

```
mysql> CREATE DATABASE mymarket;
```

This creates a database called mymarket. Now let's create the myuser account and specify the privileges that he has in the mymarket database. Issue this command:

```
mysql> GRANT select, insert, update, delete -> ON mymarket.* TO myuser@localhost
IDENTIFIED BY 'mypassword';
```

This will add an entry into the appropriate MySQL security tables that tell MySQL to create a user called myuser who uses the password mypassword. myuser can only connect from localhost, and once properly authenticated will be able to issue SELECT, INSERT, UPDATE and DELETE queries on all the tables in the mymarket database (mymarket.*).

Next we have to make mymarket the current database:

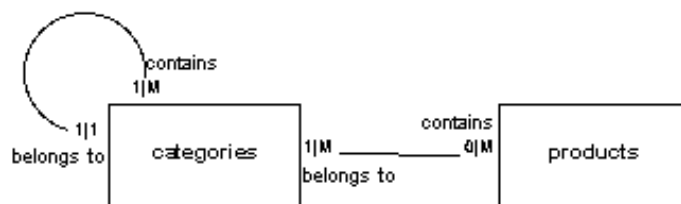
```
mysql> USE mymarket;
```

We have the user account created, now let's create the product catalog tables. Let's say that our product catalog looks like this (yes we have a very small catalog :)

```
[Top] | +---[Fruits] ||| +---[Apples] |||| +--- Granny Smith || +--- Red Delicious || +--- Rome
Beauty || +--- Apple Chips ||| +---[Citrus] |||| +--- Orange || +--- Lemons || +--- Grapefruit |||
+---[Berries] |||| +--- Blueberries || +--- Strawberries || +--- Raspberries ||| +--- Bananas |
+---[Vegetables] ||| +--- [Lettuce] |||| +--- Butterhead || +--- Cutting || +--- Crisp || +---
[Mushrooms] |||| +--- Truffles || +--- Shiitake || +--- Enoki ||| +--- [Potatoes] |||| +--- Sweet
Potatoes || +--- White Rose || +--- Russet || +--- Potatoe Chips ||| +--- Cucumbers | +---[Snacks]
| +--- [Chips] ||| +--- Potato Chips | +--- Apple Chips | +--- Corn Chips | +--- [Icecream] ||
| +--- Vanilla | +--- Chocolate | +--- Popcorn
```

What we want is an easy way to manage this catalog in our database. Product categories are arranged in a tree. There is one root category called Top, under which all other categories will sit. Next, go the subcategories under those, etc. For our products, we want to be able to put a product into one or many categories to make it easy for the customer to find.

So in summary, we have what looks like this:



Reading this clockwise, we say that:

- A category contains zero or more (0|M) products
eg. We have 3 products in the [Apples] category
- A product belongs to one or more (1|M) categories
eg. Potato chips are in the [Potatoes] and [Chips] category
- A category contains one or more (1|M) categories
eg. We have 3 categories under the [Fruits] category
- A category belongs to one and only one (1|1) category
eg. The [Chips] category can only belong under snacks

NOTE:

The 0|M, 1|M are just my own notations to represent the optionality and cardinality, if that doesn't make sense to you don't worry about it too much, you can look these terms up in a database book later.

NOTE

: We can make categories a many-to-many relation upon itself, such that a category can have multiple sub-categories and belong to multiple parent-categories, but it's a too much to explain for this howto :) so we won't do it like that.

In terms of the database entity relations, we have 2:

1. the categories entity is a one-to-many relation upon itself
2. the products entity is a many-to-many relation with the categories entity

Now based on the relations that we have defined, we will need three tables to drive our product catalog:

1. categories

This table will hold the names of our categories, as well as telling us where this category sits in our catalog.

	Fieldname	Type	Description
PK	id	int auto_increment	An internal identifier (ID) for this category
IDX	parent_id	int	The ID of the parent category
IDX	name	varchar(25)	A name for this category
	description	varchar(255)	A description for this category

Since the ID will only be used internally, we will be using a MySQL autonumber to generate them. Also, since the ID field will be used to identify records it will be the primary key of this table. We want to index the parent_id and name fields because we may want to perform searches or look up categories by them.

2. products

This table will hold the information about our products, for example the name, description and price, etc. In real life you will have a lot more fields to hold other information about the product, but for the purposes of this guide, this is all you need.

	Fieldname	Type	Description
PK	id	int auto_increment	An internal identifier (ID) for this product
IDX	name	varchar(25)	A name for this product

description	varchar(255)	A description for this product
price	float(5,2)	The price of this product

3. products_categories

Since we have a many-to-many relation between products and categories, we need a table to keep track of these relations. This table simply keeps pairs of product IDs and category IDs that relate to each other.

	Fieldname	Type	Description
PK	product_id	int	The ID for the product
PK	category_id	int	The ID for the category

Okay, so we know what tables we want, now let's write the SQL CREATE TABLE queries to create them in MySQL. Run the following commands (lines in `/* */` are comments and will be ignored by MySQL):

```
mysql> /* create the categories table */ -> CREATE TABLE categories ( -> id int auto_increment
not null, -> parent_id int not null, -> name varchar(25) not null, -> description varchar(255) not
null, -> PRIMARY KEY (id), -> INDEX parent_id (parent_id), -> INDEX name (name) -> );
mysql> /* create the products table */ -> CREATE TABLE products ( -> id int auto_increment
not null, -> name varchar(25) not null, -> description varchar(255) not null, -> price float(5,2)
not null, -> PRIMARY KEY (id), -> INDEX name (name) -> );
mysql> /* create the products_categories table */ -> CREATE TABLE -> products_categories (
-> product_id int not null, -> category_id int not null, -> PRIMARY KEY (product_id,
category_id) -> );
```

Let's take a look at what we've just created, let's list the tables with the following command:

```
mysql> SHOW TABLES; +-----+ | Tables in mymarket |
+-----+ | categories | | products_categories | | products |
+-----+
```

Now let's look each table with the DESCRIBE command:

```
mysql> DESCRIBE categories;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| id | int(11) | | PRI | 0 | auto_increment | |
| parent_id | int(11) | | MUL | 0 | | |
| name | varchar(25) | | | | |
| description | varchar(255) | | | | |
+-----+
mysql> DESCRIBE products;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| id | int(11) | | PRI | 0 | auto_increment | |
| name | varchar(25) | | MUL | | | |
| description | | | | | |
+-----+
```

```

varchar(255) || || || || price | float(5,2) || || 0.00 ||
+-----+-----+-----+-----+-----+-----+
mysql> DESCRIBE products_categories;
+-----+-----+-----+-----+-----+-----+ | Field | Type | Null
| Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+ | product_id |
int(11) || PRI | 0 || | category_id | int(11) || PRI | 0 ||
+-----+-----+-----+-----+-----+-----+

```

Now that we have all our tables, let's move on and add some data. Now that we've created the tables, we will go put some data into them. Starting with the categories table, we need to add the [Top] category, and we want it to have the ID of 0:

```

mysql> INSERT INTO categories (id, parent_id, name, description) -> VALUES (0, 0, 'Top',
'This is the top level category.')

```

Let's see what we just added by running a SELECT query:

```

mysql> SELECT * FROM categories;
+-----+-----+-----+-----+-----+-----+ | id | parent_id |
name | description |
+-----+-----+-----+-----+-----+-----+ | 1 | 0 | Top |
This is the top level category. |
+-----+-----+-----+-----+-----+-----+

```

Everything is in order here, or is it? Looking carefully we see that the ID is 1 instead of 0 — which is what we wanted it to be. What gives? This is MySQL's autonumber, it automatically assigned the number 1. Normally this would be okay, but we want our [Top] category to have the ID of 0 (just because) so let's issue an UPDATE statement to fix this:

```

mysql> UPDATE categories SET id = 0 WHERE id = 1;

```

Now let's see it again:

```

mysql> SELECT * FROM categories;
+-----+-----+-----+-----+-----+-----+ | id | parent_id |
name | description |
+-----+-----+-----+-----+-----+-----+ | 0 | 0 | Top |
This is the top level category. |
+-----+-----+-----+-----+-----+-----+

```

That's much better :). Now let's create a few more categories, feel free to come up with more creative descriptions:

```

mysql> INSERT INTO categories (name, description) -> VALUES ('Fruits', 'Fresh and tasty
fruits. '); mysql> INSERT INTO categories (name, description) -> VALUES ('Snacks', 'Tasty
snacks. ');

```

Notice that this time around we didn't specify the "id" and "parent_id" fields. That's because the "id" field is an autonumber and the "parent_id" field defaults to 0. We've already seen that

specifying a value for autonumber fields doesn't do anything in MySQL so we don't have to bother. As for the "parent_id" field, the default value is 0, which is the ID of the [Top] category so okay for now because we are creating top level categories.

Now we should create some sub-categories, but before we do that let's find out what ID's MySQL assigned to the two that we've just created:

```
mysql> SELECT * FROM categories;
```

parent_id	name	description	id
0	Top	This is the top level category.	0
1	Fruits	Fresh and tasty fruits.	1
2	Snacks	Tastely snacks.	2

So we've got 1 for [Fruits] and 2 for [Vegetables]. Now let's make some Fruit and Vegetable categories:

```
mysql> INSERT INTO categories (parent_id, name, description) -> VALUES (1, 'Apples', 'Yummy crunchy apples.');
```

```
mysql> INSERT INTO categories (parent_id, name, description) -> VALUES (1, 'Berries', 'Yummy berries.');
```

```
mysql> INSERT INTO categories (parent_id, name, description) -> VALUES (2, 'Chips', 'Crunchy Greasy Treats.');
```

```
mysql> INSERT INTO categories (parent_id, name, description) -> VALUES (2, 'Icecream', 'Great on a hot summer day.');
```

Okay, now let's see what we have again:

```
mysql> SELECT * FROM categories;
```

parent_id	name	description	id
0	Top	This is the top level category.	0
1	Fruits	Fresh and tasty fruits.	1
2	Snacks	Tasty snacks.	2
3	Apples	Yummy crunchy apples.	3
4	Berries	Yummy berries.	4
5	Chips	Crunchy Greasy Treats.	5
6	Icecream	Great on a hot summer day.	6

Let's practice writing some more interesting SELECT queries. For example, let's find all the sub-categories under the [Fruits] category:

```
mysql> SELECT cat.id, cat.name, cat.description -> FROM categories cat, categories parent -> WHERE cat.parent_id = parent.id -> AND parent.name = 'Fruits';
```

id	name	description
3	Apples	Yummy crunchy apples.
4	Berries	Yummy berries.

Note that was a little unnecessary because we already knew that the ID of the [Fruits] category was 1, we could have just run the query:

```
mysql> SELECT * FROM categories WHERE parent_id = 1;
```

to do the same thing, but that was for fun so that we could practice our joins :) Okay, now that we've got some category data in there let's put in some products.

```
mysql> INSERT INTO products (name, description, price) -> VALUES ('Granny Smith', 'Yummy
Granny Smith Apples', 1.00); mysql> INSERT INTO products (name, description, price) ->
VALUES ('Strawberries', 'Fresh Strawberries', 1.50); mysql> INSERT INTO products (name,
description, price) -> VALUES ('Apple Chips', 'Crunch Dried Apple Chips', 2.00);
```

Let's see what we have in the products table now:

```
mysql> SELECT * FROM products;
+-----+-----+-----+ | id | name |
description | price |
+-----+-----+-----+ | 1 | Granny Smith
| Yummy Granny Smith Apples | 1.00 | | 2 | Strawberries | Fresh Strawberries | 1.50 | | 3 | Apple
Chips | Crunch Dried Apple Chips | 2.00 |
+-----+-----+-----+
```

Okay, we've got some products in the system, now we have to categorize them. We will start off by putting the Granny Smith apples into the [Apples] category (id:3) and the Strawberries into the [Berries] category (id:4).

```
mysql> INSERT INTO products_categories (product_id, category_id) VALUES (1, 3); mysql>
INSERT INTO products_categories (product_id, category_id) -> VALUES (2, 4);
```

We have Apple Chips showing up in both the [Apples] category (id:3) and the [Chips] category (id:5), so we need two entries for it in the products_categories table:

```
mysql> INSERT INTO products_categories (product_id, category_id) VALUES (3, 3); mysql>
INSERT INTO products_categories (product_id, category_id) -> VALUES (3, 5);
```

Let's take a look at our products_categories table now:

```
mysql> SELECT * FROM products_categories; +-----+-----+-----+ |
product_id | category_id | +-----+-----+ | 1 | 3 | | 2 | 4 | | 3 | 3 | | 3 | 5 |
+-----+-----+-----+
```

Wow, that's so informative! Let's write a more useful query, we will start by writing a query to display the category, name, and price of all the products in the system:

```
mysql> SELECT cat.name, prod.name, prod.price -> FROM categories cat, products prod,
products_categories pc -> WHERE cat.id = pc.category_id -> AND prod.id = pc.product_id
+-----+-----+-----+ | name | name | price |
+-----+-----+-----+ | Apples | Granny Smith | 1.00 | | Berries |
Strawberries | 1.50 | | Apples | Apple Chips | 2.00 | | Chips | Apple Chips | 2.00 |
+-----+-----+-----+
```

That's much more useful than just looking at a bunch of numbers isn't it? Okay, let's try another one, find the name and price of all the products under the [Apples] category:

```
mysql> SELECT prod.name, prod.price -> FROM categories cat, products prod,
products_categories pc -> WHERE cat.id = pc.category_id -> AND prod.id = pc.product_id ->
```

```

AND cat.name = "Apples"; +-----+-----+ | name | price |
+-----+-----+ | Granny Smith | 1.00 | | Apple Chips | 2.00 |
+-----+-----+

```

Getting the hang of it? Practice writing some queries on your own before moving on to the next step. When you're ready to move on, type QUIT to exit the MySQL client. The next step is to create some PHP scripts for us to do catalog maintenance through a web interface.

Now that you have a good understanding of how the products and categories are going to work, we will create some maintenance screens to allow administrative users to manage them. Download the [mymarket1.tar.gz](#) file and extract it into your web root directory. For example, if your web root is in

/home/httpd/html

type

```
$ cd /home/httpd/html $ tar -zxf /tmp/Commerce1_mymarket1.tar.gz
```

Assuming that you've downloaded mymarket1.tar.gz into /tmp. Now, open up the file application.php and change the \$CFG->wwwroot and \$CFG->dirroot paths to match your server configuration.

Directory Structure

Before we dive into the source code, let me explain how I've setup the directories and files. Once you go into the **mymarket** directory, you will see four directories and one file:

```

drwxrwsr-x 3 ying ying 1024 Apr 20 01:38 admin/ drwxrwsr-x 2 ying ying 1024 Apr 20 01:38
images/ drwxrwsr-x 2 ying ying 1024 Apr 20 01:38 lib/ drwxrwsr-x 2 ying ying 1024 Apr 20
01:38 templates/ -rw-rw-r-- 1 ying ying 1732 Apr 20 01:39 application.php

```

The **admin** directory is where we will be working in for the rest of this howto. It holds all the administrative scripts for us to manage the data.

The **images** directory is where we would store pictures if we had any. You can fill it up with neat graphics and icons when you start playing with the site.

The **lib** directory is where our library files will go. These are standard functions that we will use throughout the site, so it makes sense for us to store them in one location.

The **templates** directory will hold template files for the site. When building dynamic web pages, my personal preference is to separate the processing code from the display code. In our case here, the processing code consists of all the PHP commands we write to perform the actions. The display code consists mainly of HTML code, as well as some PHP commands to aid in the creation of the HTML code. More on this as we work through the code.

The **application.php** file is where our configuration and settings reside. We will include this at the top of all our scripts so we can have a common script to define all our settings. This is similiar to the Application.cfm files in ColdFusion, except ours isn't loaded up automatically. If that makes no

sense to you then don't worry about it.

Before we go too far, let's take a look to see what standard libraries are available. In every PHP application that I write, I always have a set of functions that I reuse over and over. These are just some handy functions that make my life easier when I write more complicated code.

lib/dblib.php

This is a database abstraction library that I use when writing code. One thing about PHP that I find lacking is the consistency of their database functions, or the lack thereof. Every database that you use has a different set of function names. I wrote a simple library that wraps the MySQL functions that I call, so that if I ever need to switch over to a different database I only have one place to change all the database calls.

NOTE:

In practice, it's not all that fun adapting your code to different databases. It takes good planning to make your application easily portable to different databases. Having a database abstraction library is nice, but then you have a bigger problem in that things you take for granted in one database are not available in another.

lib/stdlib.php

This is a subset of my standard library of functions, it contains mainly string functions that help when I'm displaying output or when I'm dealing with variables. Take a look through the comments in this file to familiarize yourself with functions. Now on with the show.

admin/

Now let's move into the admin directory, since that's where all the action.

If you take a directory listing of the admin directory, you will see this:

```
drwxrwsr-x 2 ying ying 1024 Apr 20 01:38 templates/ -rw-rw-r-- 1 ying ying 5069 Apr 20
01:38 categories.php -rw-rw-r-- 1 ying ying 1727 Apr 20 01:38 index.php -rw-rw-r-- 1 ying
ying 6881 Apr 20 01:38 products.php
```

You will notice that there is a **templates** directory here as well, it will hold template files that are specific to the administrative area.

The **categories.php** script is the one that will handle all of our category management functions. It contains all the processing code and logic to add, edit, delete, and list categories. There is no display code in this script, it uses templates from the templates directory for displaying output.

The **products.php** script is the one that will handle all of our product management functions. As with the categories.php script, it contains all the processing code and logic to add, edit, delete, and list products. There is no display code in this script, it uses template from the templates directory for displaying output.

The **index.php** script is just a simple "Welcome to the Administrative Area" screen. Now let's go through these scripts to give you an idea of what is going on.

admin/index.php

Load up the index.php file into your favorite text editor. It looks something like this (with terms of usage agreement stripped out):

```
<? /* index.php (c) 2000 Ying Zhang (ying@zippydesign.com) */
/*****
* MAIN
*****/
include("../application.php"); $DOC_TITLE = "MyMarket Administrator";
include("templates/header.php"); ?> <p class=normal> Welcome to the MyMarket Administrative
menu! <pre> TERMS OF USAGE: </pre> <? include("templates/footer.php"); ?>
```

Aside from the displaying a Terms of Usage statements (which I've cut out here) the script really just boils down to a few commands:

```
include("../application.php");
```

This includes the application.php file and sets up the program settings, and more importantly, it creates the configuration object \$CFG. We've already been introduced to the \$CFG->wwwroot and \$CFG->dirroot properties, you can take a look at the application.php file to see what others there are.

```
$DOC_TITLE = "MyMarket Administrator";
```

Sets up the name of this page into the \$DOC_TITLE, and the command:

```
include("templates/header.php");
```

displays a standard Administrative header page that prints out the name of the document and some standard administrative links. The rest of the file is pretty self-explanatory, but you should be able to see that with the bulk of the display code (HTML) going into separate template files, the file is much cleaner to read. You can easily see where your logic is and what you are trying to do without sifting out the HTML. This file was actually a bad example since there is no logic in here, so on to the **categories.php** file.

admin/categories.php

The categories.php file is a bit more interesting because it handles all the category management functions. If you pull up the file, you will (hopefully) appreciate the cleanliness and simplicity of the code. With all the display code moved to a template file, all you are left with is the core logic that drives this script:

```
switch (nvl($mode)) { case "add" : print_add_category_form(nvl($id, 0)); break; case "edit" :
print_edit_category_form($id); break; case "del" : delete_category($id); print_category_list();
break; case "insert" : insert_subcategory($id, $HTTP_POST_VARS); print_category_list(); break;
case "update" : update_category($id, $HTTP_POST_VARS); print_category_list(); break; default :
print_category_list(); break; }
```

There's all the logic that drives all the category management functions, isn't that nice and compact? Where are the functions? They are down below, because in PHP 4 you can call functions before you define them, this further allows us to keep our code neat and tidy because we can put the main logic up front and show our function declarations later.

I won't go into the specifics of the functions, because you should be able to follow the code and comments to figure out what is going on. Instead I will cover the general steps involved in each of the functions.

Adding a Category

To add a category into the database, we just have to issue a simple INSERT statement with the ID of the parent category, the name we want to give the new category, and a description. We have to ask the user this information, so we present them with a form (mode=add) and then run the INSERT statement after the form has been submitted (mode=insert).

Editing a Category

To edit a category, we must first pull the existing information from the database. Once we've done this, we can pre-fill the category form with the existing information (mode=edit) and then update the database after the form has been submitted (mode=update).

Deleting a Category

Deleting a category is a bit more complicated an operation, the reason is that we have to do something with its subcategories and its products. The solution will be to reassign all the subcategories and products under this category to this its parent. For example, let's say we have:

[Top] | +---[Fruits] | +---[Apples] +--- Kiwi +--- Banana

Now we want to delete the [Fruits] category, so what we should end up with is:

[Top] | +---[Apples] +--- Kiwi +--- Banana

Everything that was under the [Fruits] category was shifted up one category, so they are under the [Top] category now. To do this in the database we must delete the category in question, then reassign the products and then the sub-categories.

Listing the Categories

This is a simple SELECT on the database, and then printing out the information into an HTML format. On the listings, we will want to provide links for people to add, edit, and delete categories, as well as adding products into categories.

admin/products.php

The products page is very similar in layout and functionality to the categories page. They both

offer the same features, just that they operate on different tables. One thing of interest to product maintenance is the one-to-many relation between categories and products. Since one product can belong to many categories, our product entry screens have to allow us to select multiple categories.

Adding and Editing Products

To assist in this, I've written a function to create a multi-select listbox that attempts to print out the category tree. With the categories that we've entered so far, the listbox looks like this:

Sub-categories are automatically indented (notice Apples and Berries), and the categories to which this product belongs are automatically selected. To build this, I recursively make queries to the database to print each category level. This isn't efficient, but it is easy to write and understand. You should try to come up with a better algorithm, especially if you are going to have a lot of products.

The queries to INSERT and UPDATE products more complicated than with categories because of the one-to-many relation. When we perform INSERT and UPDATES, we have to make sure the relation ship table **products_categories** has the correct records to link this product with the appropriate categories.

When we are creating a new product, we have to INSERT one entry into the products_categories table for each category this product belongs to. On UPDATES, we must first delete all records in the products_categories table for this product and then rebuild them as we do with INSERTs.

We've covered the maintenance screens, now let's put everything together. Open up **mymarket/admin** with your browser and you should see the administrative welcome screen. Click on the **category** and **product** links on the side and start playing with the system.

Go over the code and comments while you play with the system to get a good feel for what is happening. If you are relatively new to PHP or MySQL, this might be a bit hard to digest at first, but it will be worth it! To reinforce your understanding of what's going on, you should run SELECT queries on the database tables to see that the data in there matches what you see on the pages.

We've covered a lot of topics in this part of the E-Commerce guide. Stay tuned for the next part where we look at the PHP 4 sessions, and user tracking!