E-COMMERCE WITH OPEN SOURCE TECHNOLOGY*: Apache, PHP, and PostgreSQL

STICLARU, Gabriel; DAN, Radu, Necolae; ISPAS, Marius
Faculty of Mathematics-Informatics
Spiru Haret University
master.fmi.ush@gmail.com

Abstract

Building an E-Commerce site requires a significant investment: time, money and lots of code (hundreds or thousands files). For a complete tutorial, we must write a book, but in this paper, we present only technical details. We concentrate on establishing the basic framework for the site, technologies, programming languages, and tools to reduce the initial investment. Our solution is to use only open source technology and multitier architecture.

Key-words: *open-source technology, E-Commerce, three-tier architecture, PHP, PostgreSQL.*

Computing Reviews classification: K.4.4

1. Introduction

This paper provides you with a technical guide to build an e-commerce web site, implemented in five stages.

First phase focuses on how to design the web site architecture to allow for long-term development flexibility.

The second phase concentrates on building the database for storing the product catalog, containing sections, categories and items.

In the last three steps, we will show how to implement the three-tier architecture, using PHP and PostgreSQL.

For a complete tutorial to building a full e-commerce site with PHP and PostgreSQL we refer to [1].

For PHP knowledge base, we refer to [2] and for PostgreSQL we refer to [3] and [4].

* This material presents the results obtained by the authors in their master team-project for the course "Advanced databases". The project was presented at the *Conference on Applied Mathematics and Informatics,* 14 December 2008, Faculty of Mathematics-Informatics, *Spiru Haret* University, Bucharest, Romania.

2. **Open source technology**

In the twenty-first century, much is being made of open-source software. Possibly the best thing about open source is that it's free-its source code is freely available on the Web, and developers can install and use it without paying licensing fees or investing in expensive hardware or software. Using open source we can thus significantly reduce the development costs of a software application, without compromising on either reliability or performance.

In this paper, we are creating a web site using open source technology Apache, PHP, PostgreSQL, and related technologies (a PHP editor and a graphical interface for PostgreSQL). For more information on what open source is, check out http://www.opensource.org.

PHP is useful tool 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". Many RDBMSs are available to use with PHP, including PostgreSQL, MySQL, Oracle. PHP is a server-side, cross-platform scripting language for writing web-based applications. PHP contain a native data-access abstraction library, PDO (PHP Data Objects) and PostgreSQL-specific functions.

PostgreSQL is world's most advanced open source database, and it's a fast and reliable database. PostgreSQL is an award-winning ORDBMS (Object Relational Database Management System) with a huge global community of developers keeping the project up to date with the latest standard database technologies. It provides an alternative to other open-source database management systems like MySQL and Firebird, as well as to proprietary commercial database management systems such as Microsoft SQL Server, IBM's DB2 and Oracle.

Apache is the most popular web server on the Internet, with millions of live sites using it every day. In fact, as the Apache website says, it is more widely used than all the other web servers combined.

pgAdmin III is a full-featured graphical interface for PostgreSQL databases, community-maintained at http://www.pgadmin.org. According to the web site, pgAdmin is "a powerful administration and development platform for the PostgreSQL database, free for any use".

phpPgAdmin is an application (written in the PHP programming language) that is installed on a web server and provides a browser-based interface for administration of database servers. The project home page is at http://phppgadmin.sourceforge.net.

PHPEclipse is an open-source PHP IDE, which takes advantage of a well-designed, robust and widely used application framework (see http://www.eclipse.org or http://www.php-editors.com for other editors like **NetBeans PHP Editor**).

3. **Building the web site**

This paper shows you a logical way to build an e-commerce site that will deliver what it needs to be profitable. To accomplish these goals, we will develop a virtual store that sells products over the Internet. E−Commerce project, 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.

From our experience, the best solution is to build product catalog on three levels: sections, categories and items. In order to assure an easy future maintenance and further adjustments, we have to design in fact two applications, with files in two different folders. The first one, contained in the catalog folder, is the actual online shop, which is the client-facing part of the application. The second is the administration tool, held in the admin folder, hidden for the client. We must password-protect this folder to keep it secure from potentially malicious users. We can think our site as two applications in one, an online retail store and an administration tool.

Another principle in our design is to implement a multitier architecture, to discipline the programming and to assure the reusability. Each layer will have different responsibility in the overall project and will contain one or more related components. To implement this architecture, the catalog folder must contain three sub-folders: Database, Business and Presentation. The presentation logic can be handled by PHP, JavaScript, HTML and stylesheets, the business logic handled by PHP, and Database by PostgreSQL.

3.1. *The web site architecture*

n-Tier Architecture refers to splitting the project into n number of logical tiers. 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. Because we are expecting to extend the application in the future, the client-server architecture is not appropriate for our application.

We prefer the three-layered design:

- 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. Because our 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. 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.

The Data Tier (sometimes referred as the Database Tier) is responsible for managing the application's data and sending it to the Business Tier when requested.

These tiers are logical modules, 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. An important constraint in the three-layered architecture model is that information must flow in sequential order between tiers. The Presentation Tier only allows accessing the Business Tier, and it can never directly access the Data Tier. The Middle Tier communicates with the other tiers and processes and coordinates all the information flow. Splitting major parts of the application into separate components permit reusability. In addition, the system is resistant to changes (when one of the tiers change, the other tiers remain unaffected).

3.2. *Creating Database in PostgreSQL*

We will create a database called mydb and a super user account called myadmin with the password mypassword.

```
CREATE USER myadmin  PASSWORD 'mypassword'  SUPERUSER;
CREATE DATABASE mydb WITH OWNER = myadmin;
```

We have the user account created, now let us create the tables. What we want is an easy way to manage the catalog in our database. Product categories are arranged in a tree. For our products, we want to be able to put a item into one or many categories to make it easy for the customer to find. Because categories entity is an one-to-many relation upon itself, we can describe this table with fields: category_id, parent_id(the ID of the parent category), name, and description. We recommend another solution, to build another table, "section" with fields: section_id, name, description and to replace parent_id field in the table category with section_id.

After this, we know that:

- a section contains zero or more categories;
- a category belongs to one and only one section;
- a category contains zero or more items;
- an item belongs to one or more categories.

Since we have a many-to-many relation between items and categories, we need a new table "item_category" to keep track of these relations.

We have selected here a minimal set of tables and fields for these, like id, name, description, image, and price. In real life, you will have more table and fields to hold other information about the product, but for the purposes of this guide, this is all we need. The field "image" stores the name of the product's picture file. You could keep the picture directly in the table, but in most cases, it is much more efficient to store the picture files in the file system and have only their names stored into the database. If you have a high-traffic web site, "thumbnail" field is useful to store the name of the product's thumbnail picture.

Now let us write the SQL code to create PostgreSQL database.

```
CREATE TABLE section
(
section_id            SERIAL NOT NULL PRIMARY KEY,
name                  VARCHAR (40) NOT NULL,
description           VARCHAR (255)
);
```

```
CREATE TABLE category
(
category_id           SERIAL NOT NULL PRIMARY KEY,
section_id            INTEGER NOT NULL,
name                  VARCHAR (40) NOT NULL,
description           VARCHAR (255),
FOREIGN KEY (section_id)
REFERENCES section (section_id)
ON UPDATE RESTRICT ON DELETE RESTRICT
);

CREATE TABLE item
(
item_id               SERIAL NOT NULL PRIMARY KEY,
name                  VARCHAR (40) NOT NULL,
description           VARCHAR (255) NOT NULL,
price                 NUMERIC (10, 2) NOT NULL,
image                 VARCHAR (100),
thumbnail             VARCHAR (100),
search                TSVECTOR
);
```

The field "search" will hold a string of the data you want to be searched, such as product names and descriptions.

```
CREATE INDEX idx_search ON item USING gist (search);
```

Gist is the engine that performs the searches, and it is an implementation of the Berkeley Generalized Search Tree (more details about gist can be found at http://gist.cs.berkeley.edu).

Next table simply keeps pairs of item IDs and category IDs that relate to each other.

```
CREATE TABLE item_category
(
item_id          INTEGER NOT NULL,
category_id      INTEGER NOT NULL,
PRIMARY KEY      (item_id, category_id),
FOREIGN KEY      (item_id)
REFERENCES       item (item_id)
ON UPDATE RESTRICT ON DELETE RESTRICT,
FOREIGN KEY      (category_id)
REFERENCES       category (category_id)
ON UPDATE RESTRICT ON DELETE RESTRICT
);
```

We recommend creating many types, to easy handle returned objects by functions in data tier.

```
CREATE TYPE section_list AS
(
section_id      INTEGER,
name            VARCHAR(40)
);
CREATE TYPE section_details AS
(
name            VARCHAR(40),
description     VARCHAR(255)
);
```

Now that we have created the tables, we must put some data into them.

```
INSERT INTO section (section_id, name, description)
   VALUES(1, ' flowers ', 'flowers for all the
seasons');
INSERT INTO section (section_id, name, description)
   VALUES(2, ' fruits ', 'fruits for all the seasons');
INSERT INTO category (category_id, section_id, name,
description)
   VALUES (10, 1, ' 'Decoration flowers ', 'bouquets');
INSERT INTO category (category_id, section_id, name,
description)
   VALUES (20, 2, ' Citrus ', ' Citrus fruits ');
INSERT INTO item (item_id, name, description, price,
image, thumbnail, search)
   VALUES (100, 'Spring bouquet'', 'Bouquet of roses ',
'10.00', 'roses.jpg', 'roses.thumb.jpg',’’);
INSERT INTO item (item_id, name, description, price,
image, thumbnail, search)
   VALUES (200, 'orange', 'orange fruits ', '5.00',
'orange.jpg', 'orange.thumb.jpg',’’);
INSERT INTO item_category VALUES (100, 10);
INSERT INTO item_category VALUES (200, 20);
```

3.3. *Implementing the Data Tier*

Using functions allows for better maintainability of the data access and manipulation code, which is stored in a central place, and permits easier implementation of the three-tier architecture (the database functions forming the data tier). Using functions for data operations has the following advantages:

• the performance can be better because PostgreSQL generates and caches the function’s execution plan when it is first executed.

• security can be better controlled because PostgreSQL permits setting different security permissions for each individual function.

• SQL queries created in PHP code are more vulnerable to SQL injection attacks, which is a major security threat. (for this security subject, visit www.sitepoint.com)

Now, we write two usual functions that query the database and than we can call from other tiers.

```
CREATE FUNCTION  get_sections_list()
RETURNS SETOF section_list LANGUAGE plpgsql AS $$
  DECLARE
    output   section_list;
  BEGIN
    FOR output IN
      SELECT section_id, name
      FROM section
      ORDER BY section_id
    LOOP
      RETURN NEXT output;
    END LOOP;
  END;
$$;
```

The return type is SETOF section_list, which means the function is supposed to return one or more records that have the structure defined by the section_list type.

```
CREATE FUNCTION get_section_details(INTEGER)
RETURNS section_details LANGUAGE plpgsql AS $$
DECLARE
input    ALIAS FOR $1;
output  section_details;
BEGIN
SELECT INTO output
name, description
FROM section
WHERE section_id = input;
RETURN output;
END;
$$;
```

The classical solution frequently used to implement searching, consists of using LIKE in the WHERE clause of the SELECT statement. We recommend implementing the search functionality by using the full-text searching functionality of PostgreSQL. The following query performs a search on the “orange fruits” search string:

SELECT item_id, name FROM item WHERE search_vector @@ to_tsquery('orange & fruits') ORDER BY item_id;

Another suggestion is to work with PL/pgSQL stored procedures and triggers. The combination of stored procedures and triggers gives us the power to enforce quite sophisticated business rules directly in the database. The trigger fires when the specified event occurs (DELETE, INSERT, or UPDATE). The power of triggers and stored procedures comes when your declarative constraints become very complex, or you wish to implement a constraint that is too complex for the declarative form.

3.4. *Implementing the Business Tier*

The Middle Tier manages the application's business logic. For simple tasks such as getting a list of items from catalog, the Business Tier just requests the data from the database and passes it to the Presentation Tier.

The business layer includes other complex task such as open and close database connections, store SQL logic as PostgreSQL functions and accesses these functions from PHP.

We can access the catalog using PHP native, PostgreSQL-specific functions.

```
<?php
$db = pg_connect('host=localhost dbname=mydb' .
'user=' . $username . ' password=' . $password) or
die('no connection');
$sql = 'SELECT * FROM items';
$result = pg_query($db, $sql) or die('Query failed');
pg_close($db);
?>
```

Another solution is to access database with PDO (PHP Data Objects).

```
<?php
try {
$db =
new PDO('pgsql:host=localhost;dbname=mydb', $username,
$password);
$sql =
$db->prepare('SELECT * FROM items');
$sql->execute();
$result = $sql->fetchAll();
$db = null;
}
catch (PDOException $e)
{
print 'Error! <br />' . $e->getMessage() . '<br />';
exit; }
?>
```

The PDO code includes a powerful error-handling mechanism and prepared statements, which protect the site from injection-based attacks.

3.5. *Implementing the Presentation Tier*

First, we have to make a visual design of the site. Usually we use a set of templates for the pages.

The simplest case is when all the pages including the first page, will have this minimal template:

- site header: will be visible in any page;
- boxes for sections, categories and items: some will be invisible;
- search box: allow visitors to perform product searches;
- page content: different aspect for the pages of the site.

Sections will be displayed on every page of the site, but the list of categories appears only when the visitor selects a section from the box.

When the visitor clicks on a category, the list of items for that category will appear.

To implement user interface, we must write many php scripts: index.php for the main page, header.php, sections.php, categories.php, items.php, contents.php and so on (located in the "Presentation" sub-folder).

Another solution uses a template for all pages with the following structure:

- site header: display the company logo and provide a link to the site's homepage;
- the page header;
- the navigation bar: to jump to any of the parent pages up to the homepage;
- quick links: quickly access to the more important pages on the site
- left-hand information column: boxes for product catalog;
- right-hand information column: contain a variety of information boxes, which allow the user to perform a variety of tasks;
- page content: display the content of each page.

4. **Summary**

Most open source web projects are based on AMP technology (Apache, MySQL, PHP) and a two-tier architecture.

This article is original due to the reconsideration of the classical AMP technology by replacing MySQL with PostgreSQL, on a three-tier architecture.

We moved to a more powerful database server that has many features, including referential integrity, transactions, sub-selects, unions, full-text searching, etc.

We suggested that many of business rules to be written in data tier with stored procedures and triggers. In addition, because SQL queries created in PHP code are more vulnerable to SQL injection attacks, we recommend the use of SELECT inside the functions in Data Tier.

On multitier architecture we can isolate database connectivity and provide a centralized repository for business logic.

The user interface handles only input and display and the database engine handles database issues. Other advantages are high performance, scalability, code reuse and programming discipline.

REFERENCES

1. Gilmore J. and Treat R., *Beginning PHP and PostgreSQL 8: From Novice to Professional*, Apress, 2006.
2. Powers D., *PHP Solutions: Dynamic Web Design Made Easy*, Apress, 2006.
3. Matthew N. and Stones R., *Beginning Databases with PostgreSQL: From Novice to Professional*, Second Edition, Apress, 2005.
4. Douglas K. and Douglas S., *PostgreSQL*, Sams Publishing, 2003.
5. Web links:
   - http://www.postgresql.org
   - http://www.php.net
   - http://www.php-editors.com
   - http://www.opensource.org