Use PHP to build a search engine optimization app, Part 2: Creating reports and billing clients

Track your SEO efforts with PHP, Derby, and some ingenuity

Skill Level: Intermediate

Tyler Anderson (tyleranderson5@yahoo.com)
Freelance Writer
Stexar Corp.

21 Mar 2006

PHP, a dynamic Web-based programming language, takes a variety of input formats and has a built-in SOAP client to obtain information from the Web. PHP, combined with applications using search engine optimization (SEO), is a powerful tool for obtaining information from major search engines, allowing this information to guide a webmaster's online marketing and SEO strategies. In this final part of a two-part "Use PHP to build a search engine optimization app" series, we add two search venues and the functionality to search all or just one of the search engines. You will also extend the Apache Derby database to provide support for billing clients.

Section 1. Before you start

This tutorial is for PHP programmers and webmasters interested in extending the application from Part 1 of this two-part series. The example application will be extended in the following ways:

- To allow a user to choose which of the supported search engines to obtain positions on
- To allow an SEO provider to bill clients based on positions obtained for clients, while providing a positions summary, as well as a billing summary in CSV format

© Copyright IBM Corporation 2006. All rights reserved.

- To add Ask.com and the Open Directory Project as possible search venues

About this series

This series creates a search engine optimization application that connects to search engines to obtain client positions for a given list of URLs and keywords. Matching results will be stored in an Apache Derby database for later processing.

Part 1 sets up and builds the back-end database and application in PHP. This includes code to retrieve positions from the three top search engines.

Part 2 adds functionality to search Ask.com and the Open Directory Project. You'll also extend the database and application to automatically bill clients, and provide two summary files in CSV format.

About this tutorial

This tutorial extends the application written in Part 1 to allow a client or SEO company to choose which of four search engines and one directory to search from. A new search engine and directory, Ask.com and the Open Directory Project, will be added as search venues. Lastly, we implement functionality in PHP that will automatically create two CSV file reports, a positions and billing summary, that can be imported into spreadsheet software.

Prerequisites

The following tools are needed to follow along:

Web server
Any operating system and any Web server can be used. Feel free to use Apache V2.X or the IBM HTTP Server.

PHP
Due to the use of PHP data objects, PHP V5.1 or higher is required for this tutorial. Be sure to configure PHP with the following option to include support for Derby and the SOAP extensions:
`--with-pdo-odbc=ibm-db2,/home/db2inst1/sqllib`
`--enable-soap`. See Resources for information about configuring Apache or the IBM HTTP Server with PHP.

Database
This tutorial uses Apache Derby, which is open source and lightweight, the IBM

© Copyright IBM Corporation 2006. All rights reserved.

DB2 JDBC Universal Driver, and the DB2 runtime client from IBM. Make sure that you have set your classpath appropriately by following the given instructions on each page. You can follow either the Linux® or Windows® instructions for installing and downloading the DB2® runtime client. Cloudscape may also be used for this tutorial. The internals of Cloudscape are the same as Derby, however, the DB2 JDBC Universal Driver and other things are packaged into Cloudscape, and it is supported by IBM. Download Cloudscape V10.1, and the DB2 runtime client from IBM.

Java™ technology
Derby requires Java technology from Sun Microsystems or from IBM.

Developer and application tokens
Each major search engine requires that you pass along some sort of ID with your SOAP or REST request for search results. You need to get one from each of them: Google, Yahoo! and MSN.

This tutorial assumes basic PHP knowledge of PHP syntax, including assignments, `for` loops and functions, and so forth.

Section 2. Application overview

In Part 1, you developed the Derby database and coded functionality for searching three of the major search engines in PHP.

Next, we see how to add Ask.com and the Open Directory Project. We will also provide new program functionality, and the ability for billing clients and providing summaries, all within the example application.

The current state

In Part 1, you built up the PHP application and implemented a CSV file parser that the PHP application imports.

This CSV file feeds keywords and domain pairs to your SEO application, allowing your application to search and obtain positions for discovering the status of your Web sites with the top search engines.

Take a look at Figure 1 to see how the application looked after completing Part 1.

Figure 1. The SEO application

© Copyright IBM Corporation 2006. All rights reserved.



And Figure 2 illustrates the example browser output from inputting a CSV file.

Figure 2. Obtaining matched positions

© Copyright IBM Corporation 2006. All rights reserved.



Now let's take a look at the additions we will add to the application.

Ask and the Open Directory Project

Here in Part 2, we add Ask as a fourth major search engine. This is provided as an extra example because not all search engines provide Web services as a means to obtain positions. Ask requires manual parsing of HTML via PHP.

The Open Directory Project (ODP) is also being added because it is the single most important directory to get listed in. Its fame grew because it was one of the first human-edited directories. Inclusion(s) within ODP are like gold because several Web sites get their positions directly from ODP, meaning one link on the ODP is like having hundreds of links Web-wide. Now getting your Web site included in ODP couldn't be a more difficult task, but it's definitely worth trying. ODP, like Ask, doesn't provide Web services, and we will only look at ODP for how many links exist, not positions.

© Copyright IBM Corporation 2006. All rights reserved.

New functionality

Searching ODP for links of large corporations can take a while, as there could be several pages of links, so wouldn't it be nice to pick what search engines or directories your application searches? It certainly would, and so that's what we do. Extend the form by providing five check boxes, one for each engine and ODP. The new form looks like Figure 3.

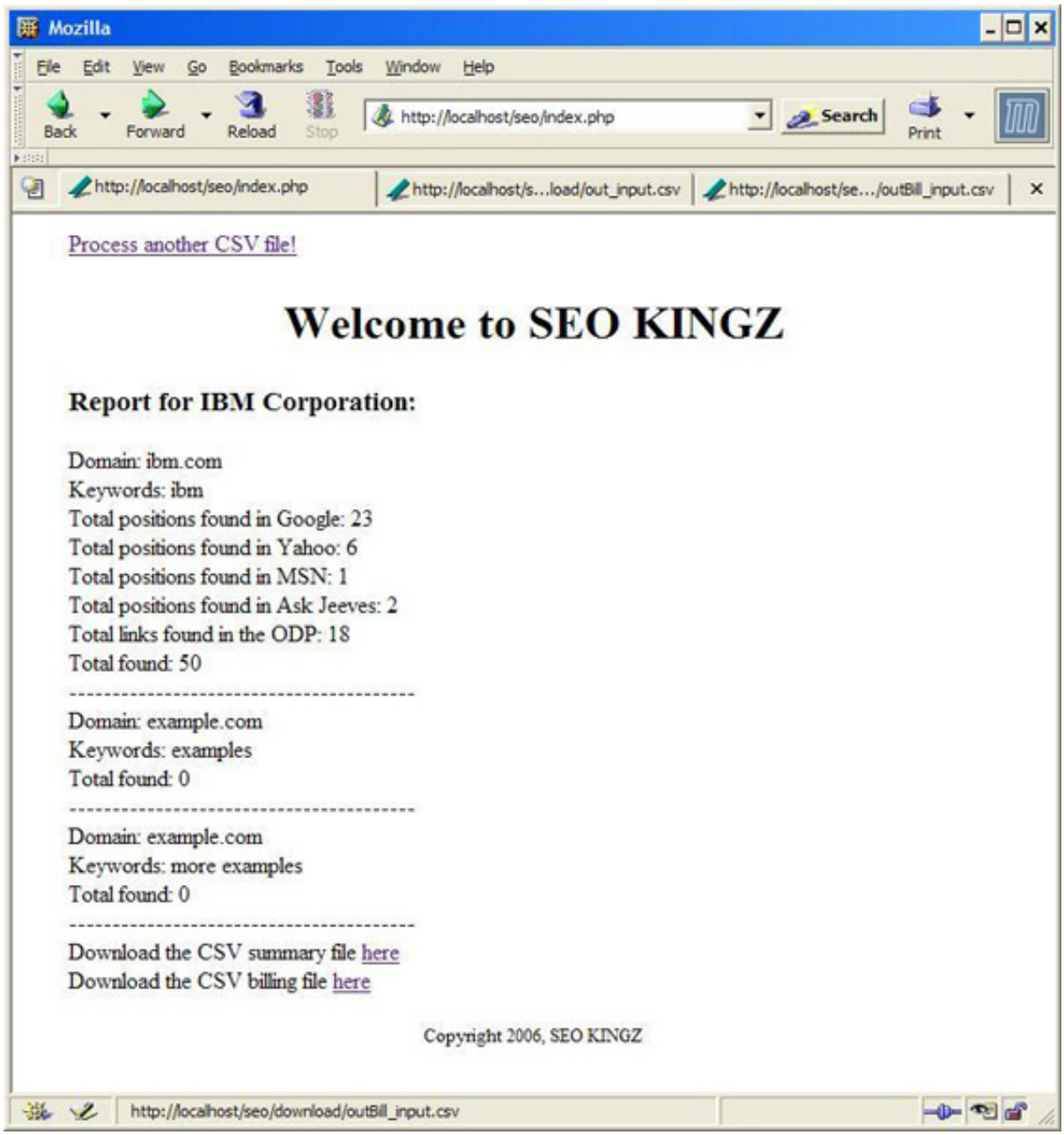
Figure 3. The new form



Example browser output after searching with all four search engines and ODP using the completed application is shown in Figure 4. The remainder of this tutorial will walk through adding this functionality to the example application.

Figure 4. The complete application

© Copyright IBM Corporation 2006. All rights reserved.



Billing clients

Billing clients requires more database functionality, which we set up in the next section. We use the database to save how to bill a given position range, as well as to save how much a client is getting charged for the positions obtained in each search engine. Also, because results are always good to see, we provide simple summaries for yourself and your client to download, which we look at next.

© Copyright IBM Corporation 2006. All rights reserved.

Providing summaries

Summaries provide a confirmation of the task, as well as a means to display the results of the search in your favorite reporting and spreadsheet software. Here is example browser output of the positions summary CSV file (see Figure 5), and the billing summary (see Figure 6).

Figure 5. Positions summary

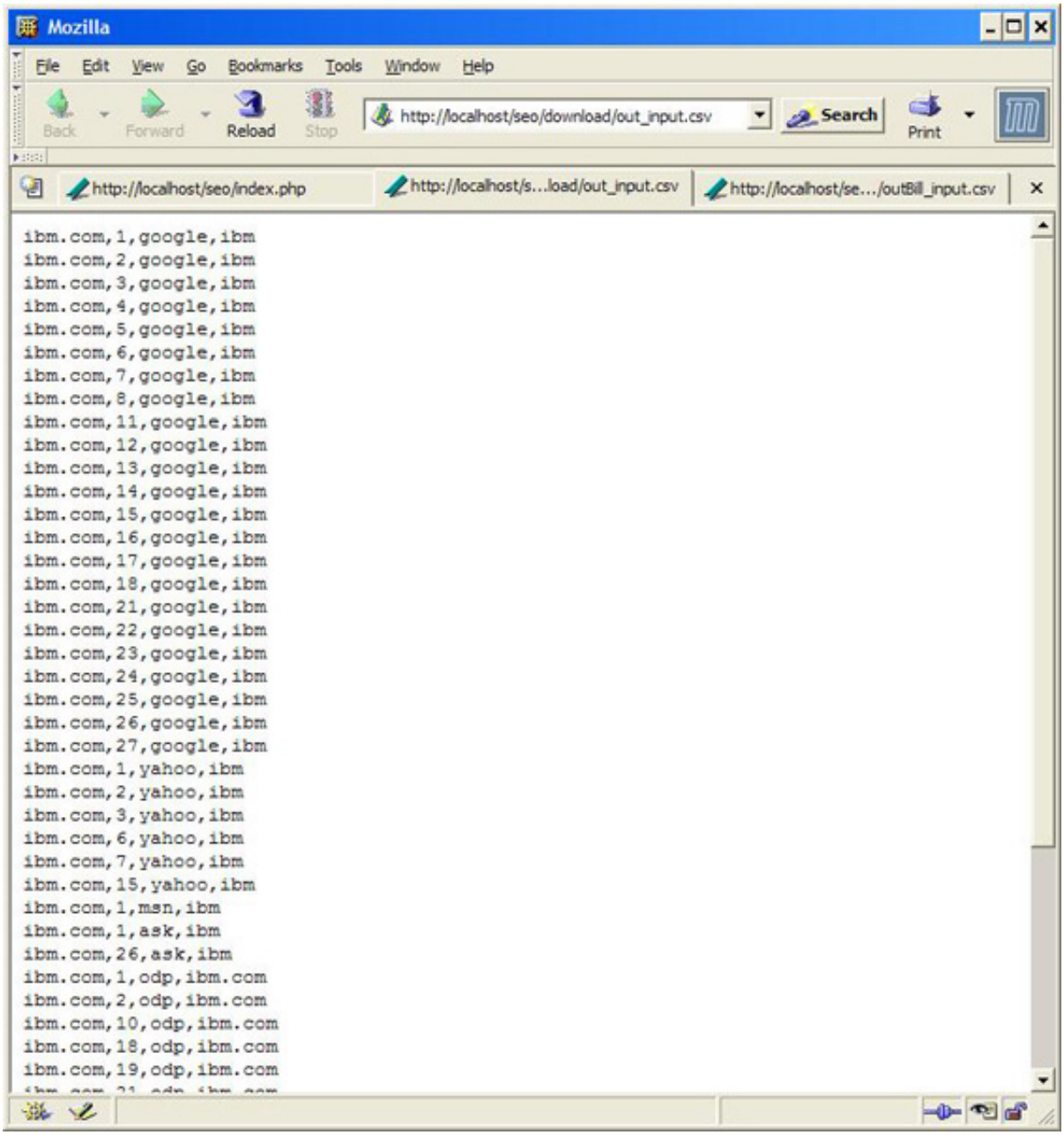
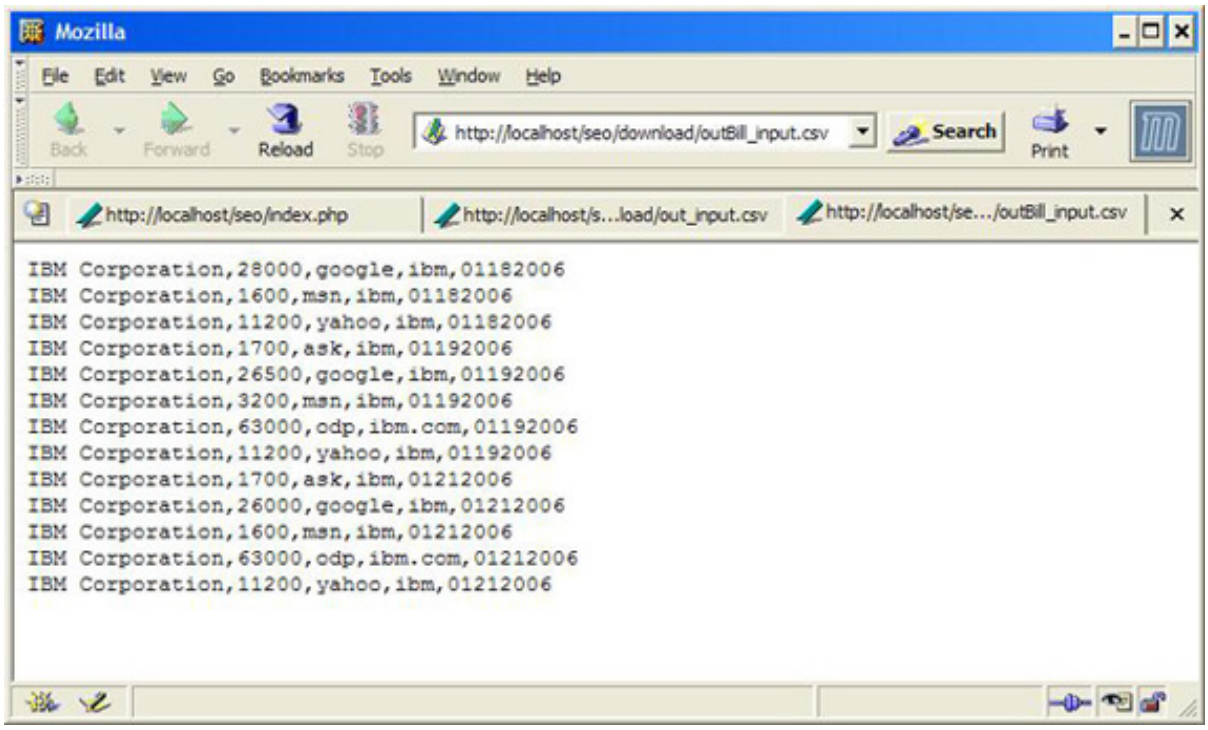


Figure 6. Billing summary

© Copyright IBM Corporation 2006. All rights reserved.



Section 3. Extending Derby and setting up

Storing billing data into the database requires more Derby functionality by adding a couple more tables. Here we learn about the new functions we create. Also, since searching online for various keywords on several domains can take a while, we learn about which PHP ini properties you need to set to overcome this restriction.

New tables

Billing clients requires two new tables:

1. `billToValues` -- Contains the price attached to a given position within a given keyword range. So, if the range is 1 through 5 for a price of $2,000 on Google, and you get your client spot 3, that'll earn you $2,000.
2. `billing` -- This is where we store the billing results, attached with a date.

These are all the new tables. Let's create them using the Derby `ij` tool.

© Copyright IBM Corporation 2006. All rights reserved.

Creating the new tables

Before getting started, the new tables need to be created. Remember to start the network server, as you did in Part 1:

```
java org.apache.derby.drda.NetworkServerControl start
```

Now open a new console and start the `ij` tool:

```
java org.apache.derby.tools.ij
```

Next, connect to the database you created in Part 1:

```
connect
'jdbc:derby:net://localhost:1527/SEO;create=true:user=seouser;
password=seopass;';
```

Create the `billToValues` table by typing the following, as shown in Listing 1.

Listing 1. Creating the billToValues table

```
create table billToValues (price integer,
                           searchengine varchar(255),
                           positionLow integer,
                           positionHi integer);
```

This creates the `billToValues` table with the price, search engine, and the obtained range specified by `positionHi` and `positionLow`. Next, create the billing table, as shown in Listing 2.

Listing 2. Creating the billing table

```
create table billing (clientname varchar(255),
                      price integer,
                      searchengine varchar(255),
                      keywords varchar(255),
                      date varchar(8));
```

This creates the billing table with the fields, client name, price, search engine, keywords, and date.

Initializing the billToValues table

© Copyright IBM Corporation 2006. All rights reserved.

The `billToValues` table is the only of the two tables that needs to be initialized. Fill in the billing table with values obtained from billing clients.

To initialize the table with the following hypothetical billing prices, type the following using `ij`, as shown in Listing 3.

Listing 3. Initializing the billToValues table

```
insert into billToValues values (2000, 'google',  1,  5),
                                (1500, 'google',  6, 10),
                                (1000, 'google', 11, 20),
                                ( 500, 'google', 21, 30),
                                (2000,  'yahoo',  1, 10),
                                (1200,  'yahoo', 11, 20),
                                ( 750,  'yahoo', 21, 30),
                                (1600,    'msn',  1,  5),
                                (1300,    'msn',  6, 10),
                                ( 800,    'msn', 11, 20),
                                ( 500,    'msn', 21, 30),
                                (3500,    'odp',  1,999),
                                (1400,    'ask',  1,  5),
                                (1000,    'ask',  6, 10),
                                ( 600,    'ask', 11, 20),
                                ( 300,    'ask', 21, 30);
```

Notice that ODP is just looking for included links, and each link within ODP is worth the same amount. The search engines cover prices from $1 to $30, or the first three pages of results, which are statistically the most viewed. Next, we see what the new functions are.

New functions

The new functionality in the application requires new functions. Add the functions (as shown in Listing 4) to your functions.php file, since you will then later edit each one when applicable throughout the remainder of this tutorial.

Listing 4. New functions

```
function billForPositions($client, $pdo){
}

function writeBillingCSV($client, $filename, $pdo){
}

function writeSummaryCSV($filename, $pdo){
}

/********** Open Directory Project *****************/
function domainCountODP($domain, $pdo){
}

function getResultODP($result){
```


© Copyright IBM Corporation 2006. All rights reserved.

```
}

function getTotalODP($results){
}

/********** Ask Jeeves *****************/
function domainMatchAskJeeves($domain, $query, $pdo){
}

function getStartAskJeeves($results){
}

function getResultAskJeeves($result){
}

function getTheURL($getURL){
}

function queryAskJeeves($query, $ct){
}

function queryODP($domain, $ct){
}
```

Setting PHP ini properties

Because searching Web sites through Web services can take a while to get results back, we need to extend the maximum execution time allowed by PHP, among other settings. Add the following to the index.php file, as shown in Listing 5.

Listing 5. Setting PHP ini properties

```
...
require('header.php');
ini_set('max_execution_time', '600');
ini_set('max_input_time', '600');
ini_set('memory_limit', '8M');
```

Now the application is free to take the time it needs without returning a max execution time-exceeded error. The next section covers the new extensions to your application.

Section 4. Extending the application

It's time to add the new form previewed in Application Overview.

© Copyright IBM Corporation 2006. All rights reserved.

Selecting which search engines to search

The new form allows you to search only the search engines you wish. Modify the current form in index.php to match what's shown in Listing 6.

Listing 6. The new form in PHP

```
     print "<form
enctype='multipart/form-data' \
     action='index.php'".
         " method='POST'>".
         "<input type='hidden'
name='maxSize' \
         value='100000' />".
         "Enter a CSV (comma separated file)
you".
         " would like me to check positions
on: ".
         "<input name='filename' type='file'
/><p />".
         "Check which search
engines/directory \
         to search on:<br>".
         '<input type=checkbox
checked="true" '.
         'name="google">Google<br>'.
         '<input type=checkbox
checked="true" \
         name="yahoo">Yahoo!<br>'.
         '<input type=checkbox
checked="true" \
         name="msn">MSN<br>'.
         '<input type=checkbox
checked="true" \
         name="ask">Ask '.
         'Jeeves<br>'.
         '<input type=checkbox
checked="true" \
         name="odp">'.
         'The Open Directory Project<p>'
         "<input type='submit' value='Find
me Positions!' \
         /></form>";
```

You can now view the revised form, as shown in Figure 7.

Figure 7. Example browser output of the new form

© Copyright IBM Corporation 2006. All rights reserved.



Next, we cover what to do within PHP to implement this form.

Processing the new form

With the new form in place, clicking on it will set new values in the `POST` array. Thus, we need to pass it along to the `domainMatch()` function. Modify the `if` statement in index.php, as shown by the bold code in Listing 7.

Listing 7. Passing the POST array to the processFileCSV function

```
if($_FILES['filename']['name'] != ''){
    processFileCSV($_FILES['filename']['name'],
                   $_FILES['filename']['tmp_name'],
                   $post);
}
```

Next, modify the `processFileCSV()` function to pass the `POST` array to the `domainMatch()` function, as shown in Listing 8.

Listing 8. Passing the POST array to the domainMatch function

© Copyright IBM Corporation 2006. All rights reserved.

```
function processFileCSV($filename, $tmpname,
                        $post){
...
        for($i = 0; $i < $row; $i++){
            print "Domain: ".$data[$i][0]."<br>Keywords: ".
                $data[$i][1]."<br>";
            $totalFound = domainMatch($data[$i][0],
                                      urlencode($data[$i][1]), $pdo,
                                      $post);
...
```

The POST array has been successfully passed. Next, we modify the domainMatch() function.

Modifying the domainMatch function

The domainMatch() function takes the POST array and searches only those search engines/directories that got checked in the form. Modify the domainMatch() function, as shown in Listing 9.

Listing 9. Implementing the new form in the domainMatch function

```
function domainMatch($domain, $query, $pdo,
                     $post){
    $totalFound = 0;
    if($post['google'] === "on"){
        $found = domainMatchGoogle($domain, $query, $pdo);
        if($found){
            print "Total positions found in Google: $found<br>";
            $totalFound += $found;
        }
    }
    if($post['yahoo'] === "on"){
        $found = domainMatchYahoo($domain, $query, $pdo);
        if($found){
            print "Total positions found in Yahoo: $found<br>";
            $totalFound += $found;
        }
    }
    if($post['msn'] === "on"){
        $found = domainMatchMSN($domain, $query, $pdo);
        if($found){
            print "Total positions found in MSN: $found<br>";
            $totalFound += $found;
        }
    }
    return $totalFound;
}
```

Thus, if a checkbox was checked, the value for that search engine will be "on", where you will then call the individual domainMatch functions for each respective search engine.

© Copyright IBM Corporation 2006. All rights reserved.

Adding a keywords field to the positions table

One minor extension we make to the functions.php file is the modification of the positions table. To bill by the position, we need to know which keyword you billed, so here we modify the positions table to reflect this addition. Modify the `processFileCSV()` function, as shown in Listing 10.

Listing 10. Modifying the positions table

```
...
        $pdo->exec("drop table positions");
        $pdo->exec("create table positions (domain varchar(1024),".
                   " position integer, searchengine varchar(255),".
                   " keywords varchar(255))");
...
```

Now whenever you insert values into this table, we need to also add what keywords were used in the query. Modify the `domainMatchGoogle()`, `domainMatchYahoo()`, and `domainMatchMSN()` functions, as shown by the following three segments of code (see Listing 11).

Listing 11. Modifications for the domainMatch functions of Google, MSN, and Yahoo!

```
$pdo->exec("insert into positions values ('".$domain.
           "', ".$position.", 'google', '".$query."')");

$pdo->exec("insert into positions values ('".$domain.
           "', ".$position.", 'yahoo', '".$query."')");

$pdo->exec("insert into positions values ('".$domain.
           "', ".$position.", 'msn', '".$query."')");
```

We're all set to query the new search engines.

Section 5. Querying new search engines

Obtaining positions from Ask and ODP is fairly simply accomplished. However, we soon see that parsing the results isn't as simple as it was in Part 1, when we parsed the SOAP structure returned from Google or MSN's Web services, or the XML document returned from Yahoo!

© Copyright IBM Corporation 2006. All rights reserved.

Helper function to retrieve URLs

Obtaining the Web page of positions from Ask and ODP requires a helper function. Define the `getTheURL()` function, as shown in Listing 12.

Listing 12. defining the getTheURL function

```
function getTheURL($getURL){
    $fp = fopen($getURL,"rb");
    $results = "";
    while(!feof($fp)){
        $results .= fread($fp, 8192);
    }
    fclose($fp);
    return $results;
}
```

This function simply takes in a URL (for example, http://www.example.com) and returns its Web page. We use it to retrieve the HTML full of search results for both Ask and ODP next.

Ask

Ask has two parameters for its search: q and `page`. q holds the keywords for the search, and `page` holds the page number of results, with the first page being 1, the second 2, and so on. Define the `queryAskJeeves()` function, as shown in Listing 13.

Listing 13. Querying Ask

```
function queryAskJeeves($query, $ct){
    $start = $ct;
    $getURL = "http://www.ask.com/web?q=".$query."&page=".$start;

    return getTheURL($getURL);
}
```

Notice it calls the `getTheURL()` function you just defined to return the HTML results from querying Ask. Next, we do the same for ODP.

Open Directory Project (ODP)

ODP also has a search-like variable in the URL, but this time, it's simply search. For multipage results, we need to specify start as well. Thus, for page 1, enter 1, and for page 2, enter 21 because each page holds 20 results. Define the `queryODP()` function, as shown in Listing 14.

© Copyright IBM Corporation 2006. All rights reserved.

Listing 14. Defining the queryODP function

```
function queryODP($domain, $ct){
    $start = ($ct-1)*20 + 1;
    $getURL = "http://search.dmoz.org/cgi-bin/search?search=".
              $domain."&start=".$start;

    return getTheURL($getURL);
}
```

Again, like Ask, HTML is returned by calling the `getTheURL()` function. The next section deals with parsing out the positions from the returned HTML.

Section 6. Obtaining positions

As mentioned, obtaining each URL from the search engine is not as easy as it was in Part 1. Here, we learn how to manually parse each search result using the string functions of PHP -- a major strength of PHP -- to obtain positions from Ask and ODP.

Integrating Ask and ODP

Before parsing the HTML, you need to have the `domainMatch()` function call the Ask `domainMatch` code and the ODP domain count code defined throughout this section. Finish defining the `domainMatch()` function, as shown in Listing 15.

Listing 15. Adding hooks into the Ask and ODP domainMatch code

```
function domainMatch($domain, $query, $pdo){
...
            print "Total positions found in
MSN: $found<br>";
            $totalFound += $found;
        }
    }
    if($post['ask'] === "on"){
        $found = domainMatchAskJeeves\
        ($domain, $query, $pdo);
        if($found){
            print "Total positions found in
\
            Ask Jeeves: $found<br>";
            $totalFound += $found;
        }
    }
    if($post['odp'] === "on"){
        $found = domainCountODP($domain,
$pdo);
```


© Copyright IBM Corporation 2006. All rights reserved.

```
        if($found){
            print "Total links found in \
            the ODP: $found<br>";
            $totalFound += $found;
        }
    }
    return $totalFound;
}
```

The functions are called `domainMatchAskJeeves()` because, like the other search engines, you're looking for the positions matched with the search engine results. For ODP, the function's called `domainCountODP()` because you're only interested in the number of inclusions in ODP for a given domain. And, like the other entries present in this function, if Ask or the ODP were checked in the new form, the respective domain match/count code will be called.

Matching domains for Ask

Ask is similar to Google in that you can't retrieve more than 10 positions with each query, and so we have to query Ask three times. Define the `domainMatchAskJeeves()` function, as shown in Listing 16.

Listing 16. Obtaining positions within Ask

```
function domainMatchAskJeeves($domain, $query, $pdo){
    $position = 1;
    $found = 0;

    for($i = 1; $i <= 3; $i++){
        usleep(1000);
        $results = queryAskJeeves($query, $i);
        $start = getStartAskJeeves($results);
        if($start < (($i-1)*10))
            break;

        $results = explode('_t" class="L4" href="', $results);
        for($j = 1; $results[$j]; $j++, $position++){
            $url = getResultAskJeeves($results[$j]);
            $d = getDomain($url);
            if($d['d1'] === $domain || $d['d2'] === $domain){
                $pdo->exec("insert into positions values ('".$domain.
                    "', ".$position.", 'ask', '".$query."')");
                $found++;
            }
        }
    }
    return $found;
}
```

Differences between the other search engines and Ask are noted in bold:

- First, the HTML is obtained by calling `queryAskJeeves()`.
- Then the position of the first entry is obtained by calling

© Copyright IBM Corporation 2006. All rights reserved.

`startAskJeeves()`.

- If the start of the first position isn't what it should be, this page is a duplicate of the previous page, meaning that the previous page was the last page of results.
- The results are then split into an array using the `explode()` function. Now each result is contained within `$results` and looped through in the `for` loop, starting at position 1. The unique value in bold, `"_t" class="L4" href=""`, occurs just before a valid query.

Next, we look at the HTML returned from Ask and the supporting functions used here.

Looking at HTML returned from Ask

The HTML returned from Ask contains a single unique string for each value returned from the search results. Take a look at this snippet of HTML returned from Ask:

Listing 17. Example HTML from Ask

```
...
                Results 1-10 of  36,920,000
...
<a id="r0_t" class="L4" href="http://www.ibm.com/"
onmousedown="return fp(this,'te','0
...
```

There are 10 or fewer such results obtained for each page. Next, we look into how to obtain the start position within the HTML for Ask.

Retrieving the start position for Ask

We know the start position because in Ask, if the first page contains four results, you'd think the second page would return zero results. However, it returns the same four results from the first page. Therefore, you need to know the position of the first result in obtaining positions and break out of the `for` loop, as shown in Listing 16. Define the `getStartAskJeeves` function, as shown in Listing 18.

Listing 18. Obtaining the first position in the results obtained from Ask

```
function getStartAskJeeves($results){
    $results = explode('  Results ', $results);
    $results = explode('-', $results[1]);
    return $results[0];
}
```


© Copyright IBM Corporation 2006. All rights reserved.

The string directory after `Results` contains `1-10 ...` for the first page, `11-20 ...` for the second page, and so on. Now if you explode with `"-"` as the delimiter, we get the start position for the current page in `$results[0]`. Next, we see how the code to get a single result from Ask works.

Retrieving individual results for Ask

With each result in the `$results` array shown in Listing 16, you still need to parse only the URL. The code for this is in the `getResultAskJeeves()` function, as shown in Listing 19.

Listing 19. Parsing a single result for Ask

```
function getResultAskJeeves($result){
    $url = explode('" onmousedown="', $result);
    return $url[0];
}
```

The unique string directly after a URL, as shown in Listing 17, is simply `" onmousedown="`. Thus, we now have only the URL in `$url[0]`. Next, we see how it's done for ODP.

Matching domains for ODP

ODP, similar to Ask, caps the number of returned results to 20. This time, you want to search all the results, so searching ODP could potentially take a while since you need to keep searching until you've looked at all the results. For example, ibm.com at dmoz.org returns 660 results. Define the `domainCountODP()`, as shown in Listing 20.

Listing 20. Counting domain matches within ODP

```
function domainCountODP($domain, $pdo){
    $found = 0;

    $results = queryODP($domain, 1);
    $total = getTotalODP($results);
    for($i = 1; $i <= $total;){
        if($i > 1)
            $results = queryODP($domain, $i);

        $results = explode('<li><a href="', $results);
        for($j = 1; $j < 20 && $i <= $total; $j++, $i++){
            $url = getResultODP($results[$j]);
            $d = getDomain($url);
            if($d['d1'] === $domain || $d['d2'] === $domain){
                $pdo->exec("insert into positions values ('".$domain.
                        "', ".$i.", 'odp', '".$domain."')");
                $found++;
```


© Copyright IBM Corporation 2006. All rights reserved.

```
            }
        }
        usleep(1000);
    }
    return $found;
}
```

Code unique to ODP is noted in bold.

- We obtain your first batch of results and determine how many total links match your query by calling `getTotalODP()`.
- Next, we loop through them all with the first `for` loop.
- Then you split up the results into an array using the unique tag `<li><a href=""` and the `explode()` function.
- The second `for` loop loops through the entire first page and the second page until all the results have been processed, denoted by the equality `$i <= $total`.

Next, we look at the HTML returned from ODP and the supporting functions used here.

Looking at HTML returned from ODP

Looking at the example HTML from the ODP will help you see how each result is parsed from the returned HTML. Take a look at Listing 21.

Listing 21. HTML returned from ODP

```
...
<font size="+1"><b>Open Directory
Sites<b></font> (1-20 of 660)<p>;
<ol start=1>
<li><a href="http://www.ibm.com/servers/eserver/zseries/os/">;
...
```

Thus, you can see the unique strings in the HTML we need to get the total number of results, as well as each individual result. Twenty or fewer URLs exist for each page. Next, we see the code that obtains the total number of ODP results from the HTML.

Retrieving the total number of ODP results

Retrieving the total number of ODP results is needed because if the first page contains 20 results (`start=1`), and the second page contains five results (`start=21`), then the third page will contain one result (`start=41`). However, its

© Copyright IBM Corporation 2006. All rights reserved.

position is only 25, the position of the last valid result. Define the `getTotalODP()` function, as shown in Listing 22.

Listing 22. Extracting the total number of ODP results from the HTML

```
function getTotalODP($results){
    $results = explode('Open Directory Sites', $results);
    $results = explode(' of ', $results[1]);
    $results = explode(')<p>', $results[1]);
    return $results[0];
}
```

The first line of code gets you past the listing of the irrelevant matching categories. The next line gets you past the range of results on the current page and right before the total results. The third call to `explode()` gives you the total number of results in `$results[0]`.

Retrieving individual ODP results

As shown by the `$results` variable in Listing 20, you now have each result. The next step is to separate out just the URL. Define the `getResultODP()` function, as shown in Listing 23.

Listing 23. Retrieving an ODP result

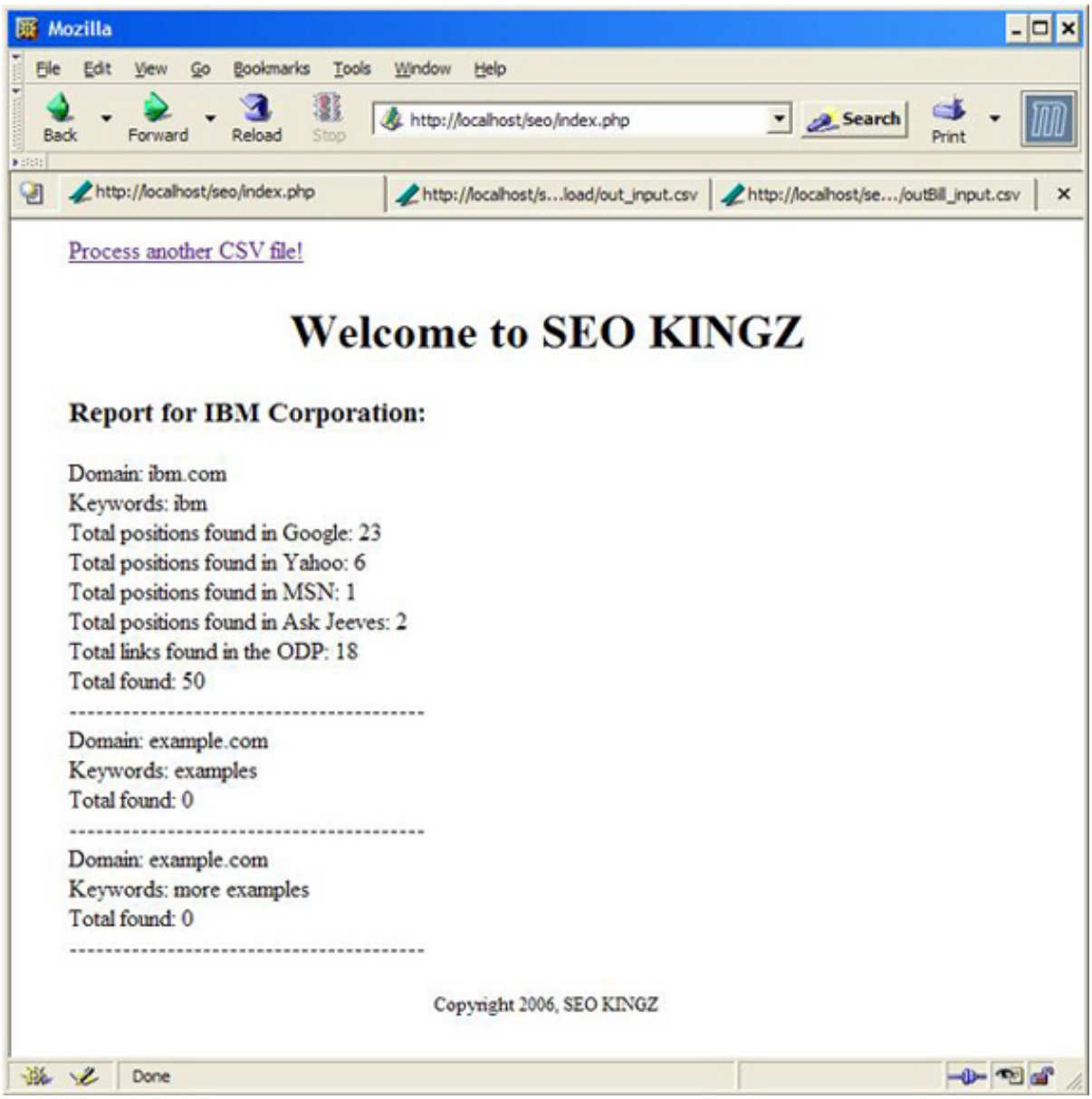
```
function getResultODP($result){
    $url = explode('">', $result);
    return $url[0];
}
```

The unique string after the URL, as you can see from Listing 21, is `'">'`, and after the call to `explode()`, we have the URL in `$url[0]`.

Take a look at Figure 8 for example browser output for the results achieved from each search engine and ODP.

Figure 8. Current browser results output

© Copyright IBM Corporation 2006. All rights reserved.



Next up is billing and reports.

Section 7. Billing and reports

Your PHP application is now successfully querying Google, MSN, Yahoo!, Ask and ODP. Now we save client billing information in Derby, as well as create a positions and billing summaries in CSV format for your clients.

© Copyright IBM Corporation 2006. All rights reserved.

New input CSV file format

To get the name of your client in the database, the input CSV file format from Part 1 is being modified. Simply add the client to the first line of the input CSV file, as shown in Listing 24.

Listing 24. The new input CSV file format

```
IBM Corporation
ibm.com, ibm
example.com, examples
example.com, more examples
```

Now the first line will always be the client name with each succeeding entry being the standard domain/keywords pair.

Setting up

Here, we parse out the client name from the input CSV file and call the summary functions defined throughout the rest of this section. Finish defining the `processFileCSV()` function, as shown in Listing 25.

Listing 25. Completing the processFileCSV function

```
...
        $fp = fopen($tmpname, 'rb');
        $client = stream_get_line($fp, 1024, "\r\n");
        $row = 0;
        while (($data[$row] = fgetcsv($fp, 1024, ",")) != false){
...
        fclose($fp);

        print "<h3>Report for $client:</h3>";// new
        for($i = 0; $i < $row; $i++){
...
            print "<br>----------------------------------------<br>";
        }

        writeSummaryCSV($filename, $pdo);
        billForPositions($client, $pdo);
        writeBillingCSV($client, $filename, $pdo);

        $pdo->commit();
...
```

New code is noted in bold. The first line grabs the line containing the client name from the CSV file adhering to the new CSV input file format. The next line displays a line announcing to the user which client the following report is for. Last, calls to the three new functions are made:

© Copyright IBM Corporation 2006. All rights reserved.

- `writeSummaryCSV` -- Displays a link to the positions summary file
- `billForPositions` -- Queries Derby and obtains billing information about each position and keyword and stores it in the billing table
- `writeBillingCSV` -- Displays a link to the billing summary file

Creating a summary CSV file

With all the cool functionality of your PHP application, now you need to be able to extract some results from it. Define the `writeSummaryCSV()` file, as shown in Listing 26.

Listing 26. Writing a position summary CSV file

```
function writeSummaryCSV($filename, $pdo){
    $filename = "out_".$filename;
    if(!file_exists(DOWNLOAD_DIR))
        mkdir(DOWNLOAD_DIR);
    $fp = fopen(DOWNLOAD_DIR."/$filename", "w");

    $sql = "select * from positions";
    foreach($pdo->query($sql) as $row){
        $domain = $row['DOMAIN'];
        $position = $row['POSITION'];
        $searchEngine = $row['SEARCHENGINE'];
        $query = $row['KEYWORDS'];
        fwrite($fp,
               $domain.','.
               $position.','.
               $searchEngine.','.
               $query."\r\n");
    }
    fclose($fp);
    print "Download the CSV summary file <a".
          " href='download/$filename'>here</a><br>";
}
```

First, we create a new file in the download directory. Next, we query the database for all the entries in the positions table. For each entry, we write a line into the file containing the domain, position, search engine the position was obtained from, and what keywords were used in the query. Finally, a link is displayed to view or download the file. Look at an example positions summary in Figure 5.

Adding billing data to Derby

Here, we store billing data into Derby for later use, if desired. This function queries the database by joining the `billToValues` table with the `positions` table. Define the `billForPositions()` function, as shown in Listing 27.

Listing 27. Storing billing values in Derby

© Copyright IBM Corporation 2006. All rights reserved.

```
function billForPositions($client, $pdo){
    $sql = "select positions.searchengine,".
        "          positions.keywords,".
        "          sum(billtovalues.price) as total ".
        "from billtovalues, positions ".
        "where billtovalues.searchengine = positions.searchengine".
                                                    " and ".
        "      billtovalues.positionlow <= positions.position and ".
        "      billtovalues.positionhi >= positions.position ".
        "group by positions.searchengine, positions.keywords";

    $date = date("mdY");
    foreach($pdo->query($sql) as $row){
        $searchengine = $row['SEARCHENGINE'];
        $query = $row['KEYWORDS'];
        $total = $row['TOTAL'];
        $sql = "delete from billing where ".
            "clientname='".$client."' and ".
            "searchengine='".$searchengine."' and ".
            "keywords='".$query."' and ".
            "date='".$date."'";
        $pdo->exec($sql);
        $sql = "insert into billing values ('".
            $client."',".
            $total.",'".
            $searchengine."','".
            $query."','".
            $date."')";
        $pdo->exec($sql);
    }
}
```

The SQL is set up in the first lines. It's essentially a join of the two tables on `searchengine` and `position`, where `position` is less than `positionhi` and greater than `positionlow`. Next, the date is obtained, and matching entries in the billing table for the same client/searchengine/query/date combination are removed and replaced with a new entry containing the updated total price.

Creating a billing summary CSV file

This function takes the entries from the billing table that match the client name so your client can view history. Define the `writeBillingCSV()` function, as shown in Listing 28.

Listing 28. Writing a billing summary CSV file

```
function writeBillingCSV($client, $filename, $pdo){
    $filename = "outBill_".$filename;
    if(!file_exists(DOWNLOAD_DIR))
        mkdir(DOWNLOAD_DIR);
    $fp = fopen(DOWNLOAD_DIR."/$filename", "w");

    $sql = "select * from billing where clientname='".$client."'";
    foreach($pdo->query($sql) as $row){
        $searchengine = $row['SEARCHENGINE'];
        $query = $row['KEYWORDS'];
```


© Copyright IBM Corporation 2006. All rights reserved.

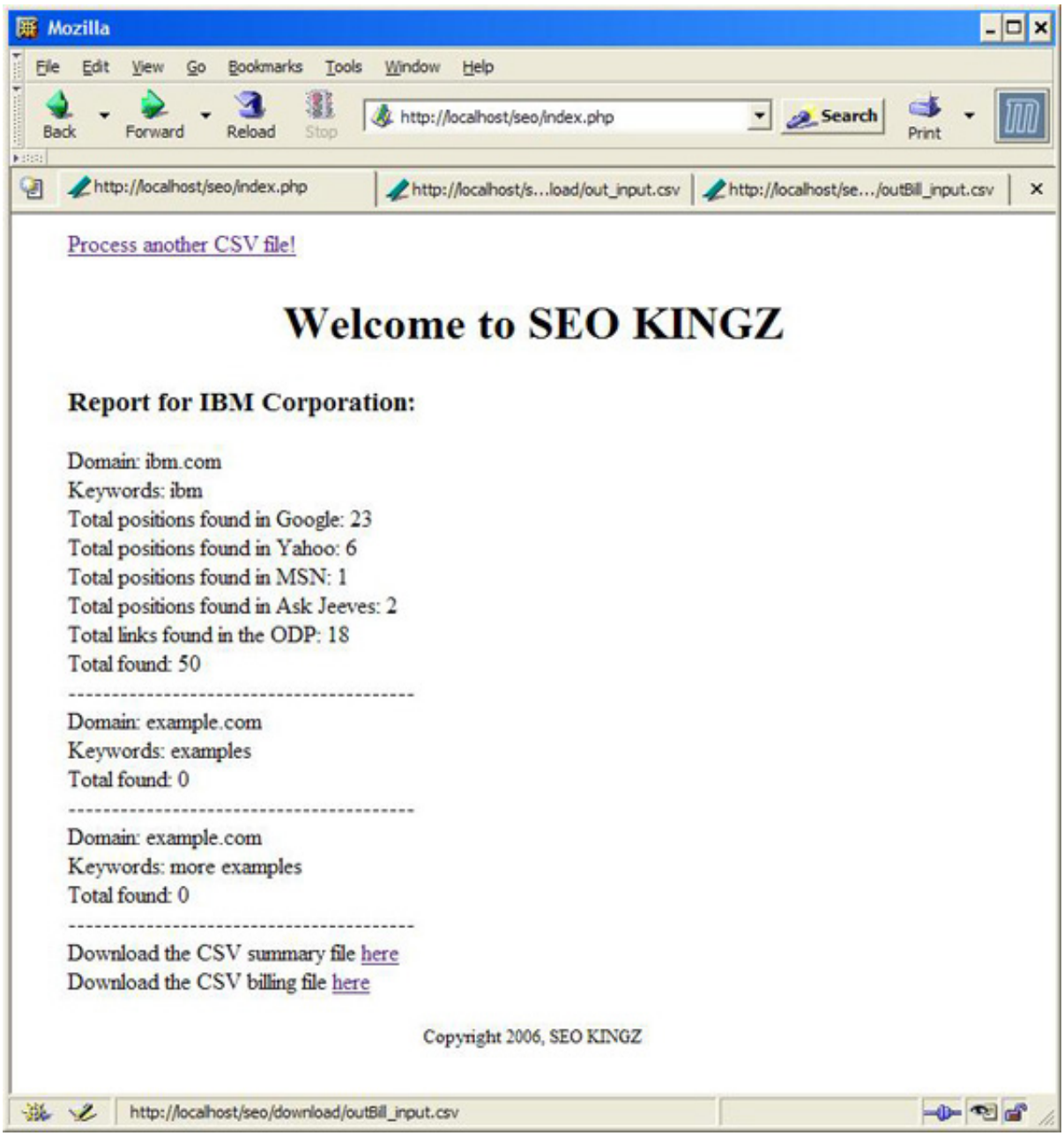
```
        $total = $row['PRICE'];
        $date = $row['DATE'];
        fwrite($fp,
               $client.','.
               $total.','.
               $searchengine.','.
               $query.','.
               $date."\r\n");
    }
    fclose($fp);
    print "Download the CSV billing file <a".
          " href='download/$filename'>here</a><br>";
}
```

First, create the file in the download directory. Next, query the billing table for all entries for the given client. Then write the resulting client, total, search engine, query, and date entries to the CSV file. Last, write a link to the billing file so the user can download it. Take a look at an example billing summary in Figure 6.

Example browser output of the completed search is shown in Figure 9.

Figure 9. The finished application

© Copyright IBM Corporation 2006. All rights reserved.



Now we can automatically, thanks to PHP, query four search engines and the No. 1 directory.

Section 8. Summary

We completed our first search engine optimization application in PHP successfully!

© Copyright IBM Corporation 2006. All rights reserved.

We can search four search engines automatically for obtained positions and check one directory for inclusion. Check out Resources to learn more about search engine optimization.

© Copyright IBM Corporation 2006. All rights reserved.

Downloads

Description	Name	Size	Download method
Part 2 source code	os-php-seo2.source.zip	7KB	HTTP

Information about download methods

© Copyright IBM Corporation 2006. All rights reserved.

Resources

Learn

- Read "Search engine optimization basics, Part 1" to get a foundation in SEO to organically optimize your Web site and create pages that are usable, accessible, and friendly to search engines.
- See "Use PHP to build a search engine optimization app, Part 1" for Part 1 of this series.
- Browse all of the PHP content on developerWorks.
- See "Hosting PHP applications on the IBM HTTP Server" to learn more about the IBM HTTP Server.
- To learn how to get Apache V2 and PHP V4.x working together on Linux, see Apache 2 and PHP (mod_php) on Linux.
- Search for search engine optimization books at the Safari Bookshelf.
- For a description, examples, and options for the `getcsv` command, which gets line from file pointer and parse for CSV fields, see fgetcsv.
- To learn how to install and configure PHP on Windows, see "Connecting PHP Applications to Apache Derby."
- PHP.net has links to documentation, source, mailing lists, and news groups.
- Be sure to check out the Search Guild for SEO forums and articles.
- Learn the basics of PHP with the "Learning PHP" series of tutorials from developerWorks.
- ALT-PHP-FAQ.ORG provides an excellent resource for all PHP-related issues.
- Expand your PHP skills by visiting IBM developerWorks PHP project resources.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- For a library that allows you to easily parse XML into a PHP data structure and to serialize PHP data structures into XML, visit keithdevens.com.
- Innovate your next open source development project with IBM trial software, available for download or on DVD.

Discuss

- Get involved in the developerWorks community by participating in

© Copyright IBM Corporation 2006. All rights reserved.

developerWorks blogs.

About the author

Tyler Anderson

Tyler Anderson formerly worked for DPMG.com, an SEO company, for whom he wrote proprietary SEO software. He graduated with a degree in computer science from Brigham Young University in 2004 and has just graduated with a Master of Science degree in Computer Engineering in December 2005, also from Brigham Young University. He is currently an engineer for Stexar Corp., based in Beaverton, Oregon. You can reach Tyler at tyleranderson5@yahoo.com.

© Copyright IBM Corporation 2006. All rights reserved.