Bayesian Web Filtering Clearinghouse

Jonathan Bulava
Department of Computer Science
The College of New Jersey
Ewing, NJ 08628
(609) 637-7255

bulava2@tcnj.edu

ABSTRACT

This paper describes the design and development of a system for the use of filtering web content by way of an online community. It will cover the many steps along the way that were taken to arrive at the project's current state. These include database design, scripting functionality, and security.

1. INTRODUCTION

1.1 Background

Not surprisingly, the availability of information via the Internet is growing at an exponential rate. This extensive accessibility creates an increase in responsibility to control or monitor what content should be viewed. There are several products on the market to sift through web content and decide for the consumer what is "good" or "bad." However, some of these products present issues. For example, most of the filtering process is left to the software and done behind the scenes without user involvement. Also, the users that purchase the product are not connected in any way to other consumers. This limits the amount of discussion between people concerning individual purposes for filtering content or helping each other to reach specific objectives.

1.2 Solution and Goals

This is where the Bayesian Web Filtering Clearinghouse proposal comes to life. The best way to solve this problem would be to create an online community where users can obtain accounts, edit "content managers," and publish them so multiple people are able to view and utilize which ones they believe would be beneficial. These changing objects are called content managers because they hold more than just a filter. They also contain a promotion list, quite the opposite of a filter. This allows users to promote certain websites for users to visit.

So our basic concern boils down to the question of how a system of multiple concurrent users can be supported with the intent to manage web-based content. My personal goal was to complete as much functionality of the web site as possible including database management and dynamic page functionality.

2. DATABASE DEVELOPMENT

2.1 Design

The backbone of the entire project is the database. The first step was to recognize what sort of information should be stored and then figure out an efficient way to save that information. This leads to the creation of schema, or the data to describe the data which is to be placed in the database. There are two main tables needed in this project; one for the accounts and one for the content managers. From here, several other tables can be added for efficiency reasons. Tables to keep track of account status, category, and abilities are created in order to be linked back to the main tables to eliminate redundancy of data storage. An Entity Relationship Diagram (ERD) of the database as well as the corresponding schema can be found in Appendix A.

2.2 Implementation

Once the design of the database is completed, it needs to be created. We chose to use MySQL as the main database language for our system. At this point, the schema needs to be converted to code that MySQL will understand. Simple text files are used to load the database with some initial information as well (See Appendix A.2 for examples).

The accounts table is where everything about a user is located whether it is stored directly or by links to other tables. Every account has a unique identification number and the user is able to choose his or her own user name and password for the site. Other fields hold the person's full name, email address and a description of what goals or aim that person has for filtering. There is a status identification number to tell the system what type of account the logged-in user has, i.e. administrator, publisher, or subscriber. For administrative purposes, this table also keeps track of who created the account and if it is "locked." Locking an account prohibits that specific user from entering and utilizing the site for whatever reason an administrator feels that it is necessary.

Once a user creates an account for his or her use, content managers can begin to be established. Numerous CMs can belong to one account and each content manager is assigned a unique identification number similar to the method used for user accounts. The CM is then linked to a user by storing the user's ID number in the content manager table. In order to store the filter and promotional list, Binary Large Objects (BLOBs) are employed. These allow large binary files to be stored in the database and retrieved for later use and editing. Even though users have much control over the CMs, administrators have a way to monitor them. All administrators can view the information in every content manager and control which ones are viewable by using a publishing flag located in the CM table.

The last set of important tables in the database is for logging user events. There are separate tables for the different types of users; one for administrators and one for publishers. Each table consists of five fields. There is a unique identification number for each

logged event along with the user ID of the person the event belongs to. MySQL's timestamp capabilities are a major part in the logging process. When an entry is made in the table, the timestamp function automatically inserts a fourteen character length timestamp in the format "YYYYMMDDHHMMSS" (year, month, day, hour, minutes, and seconds). The event is not actually stored in this table, but rather referenced by an integer to another table where all the events are stored as strings only once to avoid data redundancy. The last field holds onto a user ID if the event happens to affect another account. For example, if the user adminA with ID number 00012 decided to lock a particular publisher named publisherA with ID number 00154, then the row in the table could look similar to this:

```
00000004865   00012   20040507115959   16   00154
```

The first number is the unique ID for this specific logged event. The 16 in the 4th column represents the ID number in a common events table matching up with the string "Locked publisher account."

3. FUNCTIONALITY

3.1 Choosing a Scripting Language

Once the database is set up and ready to be accessed, a method to dynamically retrieve and display data is needed. Therefore, a scripting language for the web site needs to be chosen. Some of the most popular to pick from are CGI (Common Gateway Interface) scripts, ASP (Active Server Pages), ColdFusion, JSP (JavaServer Pages), and PHP (PHP: Hypertext Preprocessor). JavaScript is left out because it is a client-side language and a server-side technology is what the project needs. PHP was chosen as the web site's scripting language for several advantages. These include but are not limited to, a prior knowledge of programming languages is not necessary, PHP can carry out commands faster being designed for dynamic web page creation, and is well integrated with the MySQL database management system. Requesting database queries from one page to the next using PHP is easy and straightforward.

3.2 Creating Online Accounts

Now that the database and scripting language is ready to be deployed, dynamic web site functionality needs to be implemented. From here, we are going to assume that the online community has several PHP pages already written, mostly for logging in and out of the system (which will be discussed in greater detail in section 5). Creating accounts and having them stored in the database via the site is the first part of administrative functionality to develop.

When an administrator is logged in, he or she has the option to fill out a form to create an account for someone else. The fields in the form correspond to the information mentioned in the previous section about the accounts table. PHP supports the use of objects, so in this case, when the form is submitted, the page creates an instance of the `Account` class (see *Account Class* in Appendix B). The object allows an easy alternative for moving data within one entity instead of having all the variables passed around individually. Then this Account object is passed to a function to formulate a query for it to be entered into the database as seen below. `$acct` is the instance of the Account class.

```
mysql_query("INSERT INTO cm_accounts VALUES(NULL,
  '$acct->user','$acct->passwd',
  '$acct->full_name','$acct->description',
  '$acct->status_ID','$acct->email',
  '$acct->created_by','$acct->locked')");
  $message = "Account has been successfully
             created.";
```

A similar technique is used in the `requestNewAccount` function located in Appendix B under *General Functions*. Once this is placed in the accounts table in the database, that newly created user can then log into the site and utilize all of its utilities.

3.3 Requesting New Accounts

Administrators are the only clients with the ability to automatically create users for the community. Publishers, on the other hand, can only *request* that accounts be created. These pending accounts are placed in a table that resembles the one for active accounts. The same is true for people who are new to the site and are requesting an account via a signup page from the main site. When these requested accounts are entered, an automatic email is sent out to the supplied email address. The message welcomes the user to the site and contains a URL link to click in order to confirm their desire for an account. When the email is received back, an enum in the requested accounts table is changed to a "y". An enum is a variable in the database that can only be set equal to predefined values. In the case of confirming accounts, the enum in the table can be set to either "y" or "n". Pending users cannot be accepted and placed with the active users by any administrators until this email confirmation is complete.

3.4 Retrieving and Viewing Information

Once accounts are created, the next area to explore is the most efficient way to retrieve and view account and content management information. Before anything can be read from the database, a connection has to be made. `mysql_pconnent( ... )` is used to open our communication with the site. It establishes a persistent connection to remain open for calling more functions in the near future.

The Account class is put to use once again, but this time used in the opposite direction. In order to retrieve the information about one user, a query to the database is made based on a supplied username. This account's information is then stored into an Account object and sent back to the page that requested it for displaying purposes. This is the procedure that enables users of any type to view information about other users and contact them for whatever reason necessary.

3.5 Updating Community Data

There are certain times in the system where the content of an existing row in some arbitrary table needs to be changed or updated. Administrators possess the ability to change the attributes of any other account while lesser users, such as publishers, can only modify there own accounts. In order to update an account, the *updating.php* file must be required in the page and the function `updateAccountInfo( ... )` must be executed (See Appendix B.5). The attribute to be changed, the user's identification number, and the variable's new value are the three parameters for this update function. For

example, if my current full name in the database was "J Bulava," I could change it by dynamically activating the function to create this:

```
updateAccountInfo('FULLNAME','00001','Jon Bulava');
```

Other updating functions are responsible for switching the enums for locking and unlocking users as well as changing whether or not certain content managers are visible to the public.

4. LOGGING SYSTEM EVENTS

4.1 Use of Logging Functions

Once accounts are able to be created and the site's functionality increases, all of the events that belong to every user need to be logged in some way. The method used for storing timestamps was mentioned in section 2.2. There are functions that match up to every action that a user can take and their function names are very similar to the names given to the events. These functions can then be called right before or after the event occurs. For instance, if I were to log into the site, the `loggedInEvent($user_ID)` function would be called with my userID in place of that variable.

4.2 Report Generation

Viewing the logged events straight from the database can be quite tedious. Therefore, reports are generated dynamically to create more attractive and readable files of events. This was possible by finding a library that allows PHP to make PDFs on the fly. The library eventually used was called FPDF and can be found at http://www.fpdf.org/. This acquired code contains functions that make the creation of PDF files efficient and simple.

Within the actually PHP code, an instance of the FPDF file needs to be produced. Then, basic settings can be set such as font types, titles, and margins. Cells are used to place text and other objects around the document wherever necessary. However, the most important aspect of this PHP file is the ability to use conditionals to obtain and write out data to the PDF file. After the column headings for the log is added to the file, a connection to the database is made to retrieve every row from the respective log table. Cells are dynamically created and outputted until every row of the result set is used. Once this step is complete, the `Output()` function is called, which sends the newly created PDF file to the current browser. The file can also be set up to be downloaded instead of simply viewing it online.

5. WEB SITE SECURITY

5.1 Sessions

The solution to page security was sessions, an alternative to cookies. Sessions create a way to track user information when they travel from one page to the next. They were chosen over cookies simply because sessions can hold more information, are more secure without having to send data back and forth between the client and server, and they are still functional even if the user has disabled the use of cookies [1]. Without this security, any page in the file structure could be accessed if someone knew the exact URL. This means that anyone could have access to administrative pages or other pages that allow information in the database to be manipulated.

When a user logs into the web site successfully, a session array is created:

```
//begin session
session_start();
//extract info for session
$_SESSION['username'] = $_POST['username'];
$u = $_POST['username'];
$_SESSION['user_ID'] = getUserID($u);
$_SESSION['loggedin'] = time();
```

This array keeps track of the current user's username, user ID, and the time they logged in. Every PHP file after the login has the same code at the top. First, the function `session_start()` is called suggesting that a session is needed for the page. Then, `sessioncheck.php` is required in the file (See Appendix B.2). This code checks to see if a session exists when entering the page. If it does not exist, the user is redirected back to the login screen before any content is loaded. This eliminates the possibility of people reaching pages by simply typing the address for any PHP file in the browser instead of logging in.

5.2 Encryption

Encryption is another security measure that needed to be implemented. It would be unnecessary for anyone to see passwords except for the person it belongs to, including administrators. For this reason, whenever an account is created or modified, the password is encrypted and that string is the value stored in the database. The site utilizes md5, PHP's built in encryption function. It uses the RSA Data Security, Inc. MD5 Message-Digest Algorithm. After passing the function a string, the appropriate hash is returned as a 32-character hexadecimal number [2].

It is important to note that the site only calls encryption functions. Decryption is not used anywhere in the entire project, which creates a safer environment for the information in the database. The site can be utilized by employing encryption at the login page. The stored password of a given user is not decrypted and then compared to the string entered at the login page. Instead, the submitted password field is encrypted itself and is compared to the encryption string stored in the database. Of course, if the two hexadecimal values match, the user is allowed into the site.

6. SUMMARY AND CONCLUSIONS

All of these elements performing together have established the foundation for the Bayesian Web Filtering Clearinghouse. The back-end functionality has been set in place so that the project can only move forward. The next step includes an expansion of the current functionality. This would include archiving log reports after a set period of time and storing them in the database as well as executing more security measures such as session timeouts. However, the final leap is the integration of the filters and the online community. Up until this point, the filters have been constructed and tested as a separate project. It's time to put them to use by allowing the filters and promotional lists to be stored and downloaded from the content managers located in the database. Once this is achieved, beta testing of the online community can begin.

7. ACKNOWLEDGMENTS

I would like to thank my research mentor, Dr. P. DePasquale, for allowing me to be one of the first students to develop such a novel and interesting project in computer science. I also want to thank my research partners, Miss R. Lee and Mr. B. Wanner, for having the ability to work together efficiently so we could attain our goals throughout the term.

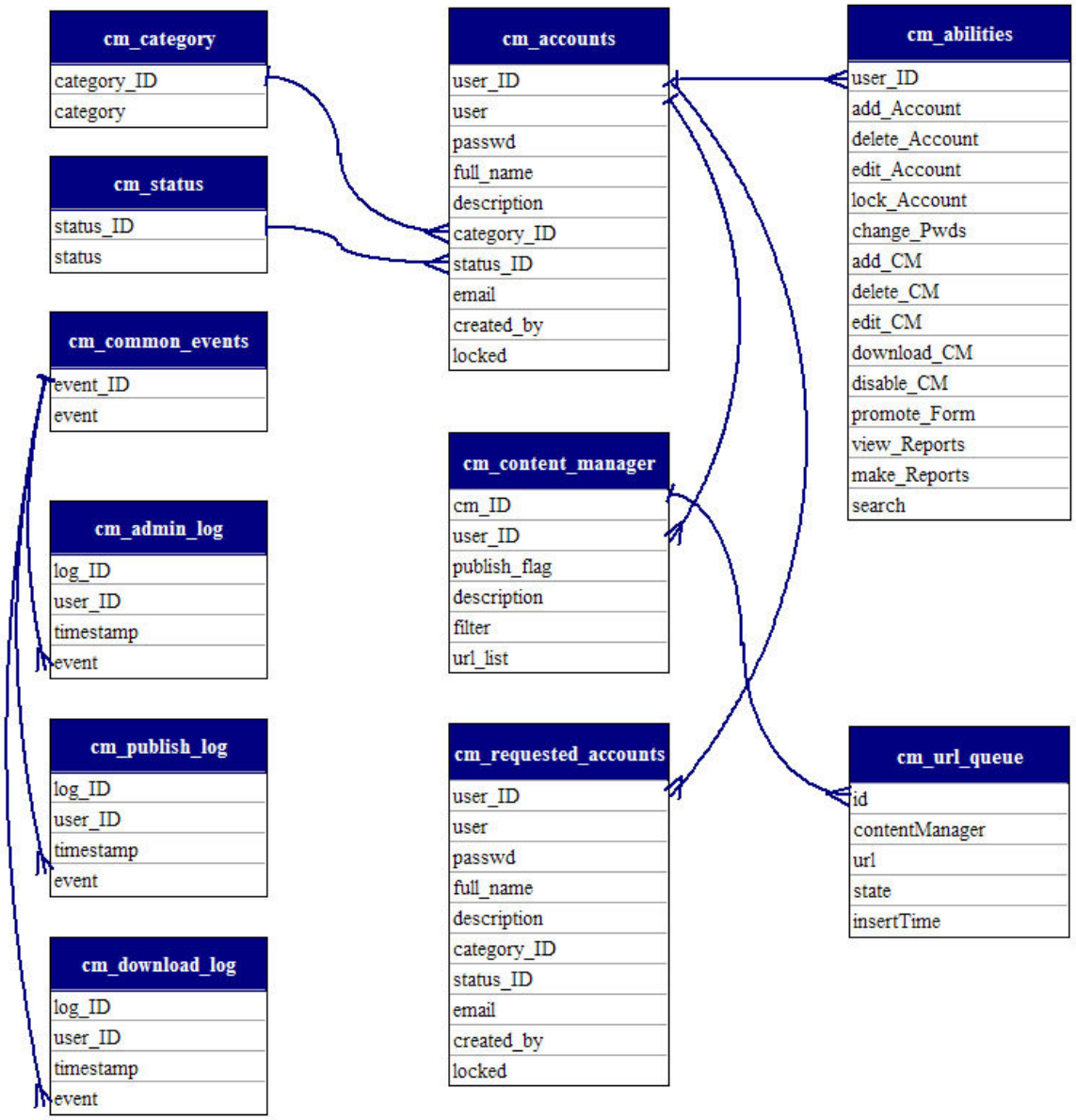
8. REFERENCES

[1] Ullman, Larry. PHP for the World Wide Web, Second Edition. Peachpit Press, Berkeley CA, 2004.
[2] PHP Manual. http://www.php.net/manual/en/.
[3] MySQL Manual. http://dev.mysql.com/doc/mysql/en/index.html.

APPENDIX A

A.1 Database ERD

The Entity Relationship Diagram for the database is shown below.



A.2 Database Schema in MySQL Code

The example code below shows tables for logging administration events, content managers, and the main accounts.

```
CREATE TABLE cm_admin_log (
   log_ID int(11) zerofill not null auto_increment,
   user_ID int(5) zerofill not null default '00000',
   timestamp timestamp(14),
   event int(3) not null default '0',
   affected_ID int(5) zerofill default '00000',

   PRIMARY KEY (log_ID)
)  TYPE=InnoDB;

CREATE TABLE cm_content_manager (
   cm_ID int(11) not null auto_increment,
   user_ID int(5) not null default '0',
   publish_flag enum("n", "y") not null,
   category_ID tinyint(2) not null default '0',
   description text,
   filter blob,
   url_list blob,

   PRIMARY KEY (cm_ID)
)  TYPE=InnoDB;

CREATE TABLE cm_accounts (
   user_ID int(5) zerofill auto_increment,
   user varchar(10) not null,
   passwd varchar(32) not null,
   full_name varchar(50) default ' ',
   description text,
   status_ID tinyint(2) not null default '0',
   email varchar(50) binary default 'none',
   created_by int(5) unsigned zerofill default '00000',
   locked enum("n", "y") not null,

   PRIMARY KEY (user_ID),
   FOREIGN KEY (user_ID) REFERENCES cm_Abilities(user_ID) ON DELETE CASCADE,
   FOREIGN KEY (user_ID) REFERENCES cm_content_manager(user_ID) ON DELETE CASCADE

)  TYPE=InnoDB;

load data local infile 'table_data/loadAccounts.txt' into table cm_accounts;
```

APPENDIX B

B.1 Login-Logout Functionality

```
/**********************
  Makes the connection
  to the database.
**********************/

function makeConnection() {
  $link = mysql_pconnect("localhost","*****","*******");
  mysql_select_db("contentproject");
  return $link;

}

/***********************************************
  Used to see if an account exists for the given
   user name.  Returns true if the user name
   already exists.
***********************************************/

function checkAccount($userName) {                //given user name
  $confirm = false;                               //confirmation is set to false

  //returns a result if the given username exists
  $result = mysql_query("SELECT user FROM cm_accounts WHERE user='$userName'");

  if (mysql_fetch_row($result)) {                 //if there is a result set,
    $confirm = true;                              //confirmation is set to true
  } //end if

  return $confirm;                                //return confirmation
} //end checkAccount

/************************************
  Checks to see if the given password
  is valid for a given user.
************************************/

function checkPasswd($userName, $pwd) {         //given user name and password
  $confirm = false;                             //confirmation is set to false

  $check = md5($pwd);                           //password is encrypted for comparison

  $result = mysql_query("SELECT passwd FROM cm_accounts WHERE user='$userName'");
  $row = mysql_fetch_object($result);
  $realpw = $row->passwd;
  if ($realpw == $check) {                      //if encryption matches,
    $confirm = true;                            //confirmation becomes true
  } //end if

  return $confirm;                              //return confirmation
} //end checkPasswd
?>
```

B.2 Session checking

This piece of code is used to check if a session exists before entering the page.

```
<?php

  // if a session does not exist with a set username...
  if ($_SESSION['username'] == NULL) {
    // redirect to the login page
    header ("Location: http://" . $_SERVER['HTTP_HOST'] .
    dirname($_SERVER['PHP_SELF']) . "/login.php");
    exit();
  }

  else {
    // debug
    while (true) {
      // show logged in user
      echo '(' . $_SESSION['username'] . ')';
    }
  }

?>
```

B.3 Account Class

This object is used when creating accounts for the database or for retrieving all the information of a user.

```
class account {

  var $user_ID;
  var $user;
  var $passwd;
  var $full_name;
  var $description;
  var $status_ID;
  var $email;
  var $created_by;
  var $locked;

  function account($uID, $u, $p, $f, $d, $s, $e, $cr, $l ) {

    $this->user_ID     = $uID;
    $this->user        = $u;
    $this->passwd      = $p;
    $this->full_name   = $f;
    $this->description = $d;
    $this->status_ID   = $s;
    $this->email       = $e;
    $this->created_by  = $cr;
    $this->locked      = $l;

  }
```

B.4 General Functions

This section demonstrates three functions that are constantly used in the online community. The first show how to retrieve a users name and ID number from the database. The second and third are examples of how to use the Account class to send information as an object rather than individual variables.

```
/**********************************
  Returns a set of all admin users
  and their user IDs.
**********************************/

function getAllAdmins() {
  $result = mysql_query("SELECT user, user_ID FROM cm_accounts WHERE status_ID=1");
  return $result;
}

/*********************************************************************
  Returns an account object that contains all the information for the
  requested user.  Used for already existing accounts.
*********************************************************************/

function getUserInfo($user) {
  $result = mysql_query("SELECT * FROM cm_accounts WHERE user='$user' LIMIT 1");
  $row = mysql_fetch_object($result);

  //creates an account object to pass back
  $requestedAcct = new account($row->user_ID, $row->user, $row->passwd, $row->full_name,
                               $row->description, $row->status_ID, $row->email,

$row->created_by, $row->locked);
  return $requestedAcct;
}

/*********************************************************************
  Used to request an account for those that are not already a member
      of the filtering community.
*********************************************************************/
function requestNewAccount($acct, $verify_num) {
  $error = mysql_query("INSERT INTO cm_requested_accounts VALUES (NULL,
    '$acct->user','$acct->passwd','$acct->full_name','$acct->description',
    '$acct->status_ID','$acct->email','$verify_num','$acct->locked')");

  return $error;
}
```

B.5 Updating Database Information

Here is an example of a function from the PHP file used to update account information.

```
/**********************************************************************
  A user's specified attribute is updated using this function.
      Used to update a user's password, full name, profile, or email
      address.
**********************************************************************/
function updateAccountInfo($attribute, $user_ID, $newValue) {
  $success = false;

  switch($attribute) {
        case 'PASSWORD':
         $pw = md5($newValue);
         $success = mysql_query("UPDATE cm_accounts SET passwd='$pw' WHERE
                    user_ID='$user_ID' LIMIT 1");
         break;
       case 'FULLNAME':
         $success = mysql_query("UPDATE cm_accounts SET full_name='$newValue'
                    WHERE user_ID='$user_ID' LIMIT 1");
         break;
       case 'PROFILE':
         $success = mysql_query("UPDATE cm_accounts SET description='$newValue'
                   WHERE user_ID='$user_ID' LIMIT 1");
         break;
       case 'EMAIL':
         $success = mysql_query("UPDATE cm_accounts SET email='$newValue' WHERE
                    user_ID='$user_ID' LIMIT 1");
         break;
       default:
         echo Error with updating!';
         break;
      }

      return $success;
}
```

B.6 Logging Event Functions

The following examples show some basic functions used to log events in the system.

```
/*****************************************************
  Used locally to decide which log table the data
      should be entered.
*****************************************************/

function getDestination($user_ID) {

      $destination = "";

  //get the user's status
  $status = returnStatus($user_ID);

      switch ($status) {
        case '1':
         $destination = "cm_admin_log";
              break;
       case '2':
         $destination = "cm_publish_log";
              break;
       default:
         $destination = "";
              break;
      }

      return $destination;
}

/*****************************************************
  Timestamps when the given user logs into the site.
      The data is placed in the appropriate table.
*****************************************************/

function loggedInEvent($user_ID) {

  //get which table for insertion
  $table = getDestination($user_ID);

      //insert the timestamp into the corrent table
  mysql_query("INSERT INTO " . $table . " VALUES (NULL, $user_ID, NULL, 1, NULL)");

}
```

```
/*************************************************
  Timestamps when the given user adds a new user to
      the database.  $new_status is the user
      status of the created account.
      NOTE: Admin's can only use this function.
*************************************************/

function addedUserEvent($adder_ID, $new_ID, $new_status) {

      //check the type of the new account,
      //if new account is an admin...
      if ($new_status == '1') {
       //insert 'Added new admin'
        mysql_query("INSERT INTO cm_admin_log VALUES (NULL, $adder_ID, NULL, 3, $new_ID)");
      }
      //else if new account is a publisher...
      else if ($new_status == '2') {
       //insert 'Added new publisher'
    mysql_query("INSERT INTO cm_admin_log VALUES (NULL, $adder_ID, NULL, 4, $new_ID)");
      }

}
```

B.7 Generating PDFs

Below is the full code used to generate a report based on the events of administrators.

```
<?php

require_once ("admin_login.php");

//defines the location of the fonts
define('FPDF_FONTPATH','/var/www/html/filterproj/requiredFiles/font/');
//requires this file for PDF functions
require('fpdf.php');

$pdf = new FPDF('L','mm');            //create FPDF object that is Landscape
                                      //and measured in millimeters

$pdf->Open();                         //start document
$pdf->AddPage();                      //add the first page

$pdf->SetMargins(10, 10);             //set left and top margins in centimeters
$pdf->SetFont('Times','',16);         //set top font, style, and size

$pdf->Cell(40,10,'BAYESIAN  FILTER  CLEARINGHOUSE'); //print out page title
$pdf->Ln();                                          //line break

$pdf->Cell(0,2,'',1);                     //draw empty cell as horizontal rule
$pdf->Ln();                               //line break

$pdf->Cell(207,10,'Administrator Event Log',0);      //print report title

//use lengths 140 and 50 for portrait pages
//use lengths 207 and 70 for landscape pages

$pdf->Cell(70,10,date("F d, Y", time()),0,0,'R');    //print current date
$pdf->Ln();                                          //line break
$pdf->Cell(0,2,'',1);                     //draw empty cell as horizontal rule
$pdf->Ln();                                          //line break
$pdf->Ln();                                          //line break

$pdf->SetFont('Arial','B',12);                       //set font for headings

//Table headings
```

```
$pdf->Cell(30,10,'Log ID');
$pdf->Cell(20,10,'User ID');
$pdf->Cell(30,10,'User Name');
$pdf->Cell(40,10,'Full Name');
$pdf->Cell(40,10,'Time');
$pdf->Cell(70,10,'Event');
$pdf->Cell(20,10,'Affected User');
$pdf->Ln();                                          //line break

$pdf->SetFont('Arial','',12);                        //set font for data

makeConnection();                                    //make database connection

$result = mysql_query("SELECT cm_admin_log.log_ID, cm_admin_log.user_ID, cm_accounts.user,
cm_accounts.full_name,            cm_admin_log.timestamp,            cm_common_events.event,
cm_admin_log.affected_ID      FROM      cm_admin_log,cm_accounts,cm_common_events      WHERE
cm_admin_log.user_ID=cm_accounts.user_ID   AND   cm_admin_log.event=cm_common_events.event_ID
ORDER BY cm_admin_log.log_ID");

while ($row = mysql_fetch_row($result)) {       //while there is another row...
  $pdf->Cell(30,10,$row[0]);                    //print log ID
  $pdf->Cell(20,10,$row[1]);                    //print user ID
  $pdf->Cell(30,10,$row[2]);                    //print user name
  $pdf->Cell(40,10,$row[3]);                    //print user's full name
  $pdf->Cell(40,10,$row[4]);                    //print timestamp
  //8:45pm March 3, 2004
  $pdf->Cell(70,10,$row[5]);                    //print logged event
  $pdf->Cell(20,10,$row[6]);                    //print affected user
  $pdf->Ln();                                   //line break
}

//We'll be outputting a PDF
header("Content-type: application/pdf");

$pdf->Output();                         //document is closed and sent to browser

?>
```