

Using SAS, MySQL, and PHP to Empower End-Users to Generate Dynamic Descriptive and Analytic Statistical Web Reports

Jeremy Morris, Gary Levy & Ray Dahl
University of Utah, Office of Budget & Institutional Analysis (OBIA)

Abstract

Organizations typically post static and finished reports in HTML or PDF form to web pages. However, administrators and decision-makers are increasingly demanding more dynamic, end-user driven web applications for creating their own unique data queries and professional descriptive reports. In addition to simple descriptive reports such as those providing means or frequencies integrated use of SAS Stat software affords generation of analytic statistical reports and analyses.

This paper introduces and details the SAS programming and hardware, as well as other non-SAS software (e.g., PHP) used to create a dynamic, end-user driven web application that produces reports containing any SAS output.

1. Introduction

Although many for-profit and entrepreneurial businesses have developed and embraced advanced reporting and data warehousing applications, higher education settings have been slow to follow.

One would think that certain units in higher education, such as financial services and institutional research offices, would have incorporated such technologies in their day to day reporting

and analytic activities, but this has not been the case.

Unfortunately and partly as a result of large investments of resources in costly enterprise systems (e.g., PeopleSoft/Oracle, Banner) as well as the inherently poor reporting applications associated with such enterprise systems, many higher education institutional research offices have instead stuck with static web pages containing HTML and PDF report formats.

In contrast, our OBIA office has been aggressively developing web-database applications using what is commonly referred to as a LAMP (Linux-Apache-MySQL-PHP) architecture.

Our OBIA office also uses SAS for data manipulation, report generator, and statistical analysis. The potential of integrating the reporting and analytic powers of SAS with the strengths of the LAMP environment was very appealing.

2. Descriptive & Analytic Reports

Administrative and academic decision-makers frequently inquire about a similar set of general metrics or indices in assessing their operations. Examples of such indices include student credit hours, class enrollments, numbers of majors, student responses to surveys, etc.

Descriptive reports are those that present descriptive statistics. We can define descriptive statistics as statistical techniques used to easily summarize a set of data.

Typical descriptive statistics include tabular summaries of data and simple graphs and figures. Descriptive reports typically present summary information such as measures of central tendency (means, median, and mode), measures of data dispersion and distribution such as standard deviations, range, variance, and skew.

Analytic reports draw on analytic or inferential statistics. We can define these statistics as statistical techniques used to draw inferences about a sample or population (rather than merely describe them). Additionally, inferential statistics allow hypothesis testing, significance testing, correlative models, and sometimes even tests of cause and effect, something that descriptive statistics are not capable of.

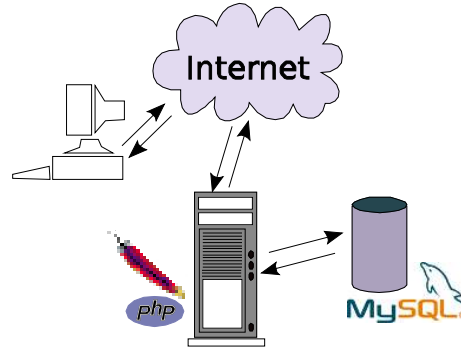
3. The Project at Hand

We had an initial project in mind when we embarked on using a PHP-MySQL-SAS system for reporting purposes. Our goal was to develop a simple web application that would permit end-users to generate simple descriptive statistics for student survey data collected by our OBIA office over a series of years. For example, end-users might be interested in what percentage of undergraduate transfer students report being married, or perhaps how satisfied graduating seniors were with courses in their major.

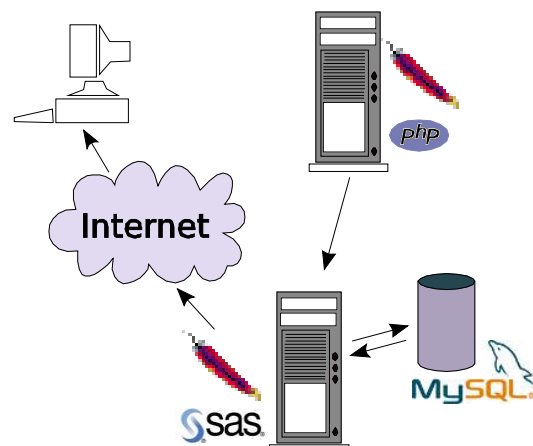
3.1 Web Architecture

The web application created to address our project goals operates in two stages.

The first stage is **Input Collection**. During Input Collection, the end-user follows a few steps and chooses a survey, a question from that survey and an alternate variable to break down the resulting statistics. This first stage of the solution is designed only using the LAMP architecture, as seen below.



After the end-user has chosen the inputs they are sent to SAS/IntrNet for the second stage, termed **Result Calculation and Display**. This stage is performed completely by SAS through the Application Dispatcher. Using HTML forms, we send the collected inputs from Stage One to a SAS system. This is illustrated in the following figure.



A benefit of this system design is that we are able to exploit the strengths of each technology and avoid using other

solutions that do not include the specific features that we need.

For example, PHP was designed specifically to create dynamic database driven web applications such as the one used in Stage One, and SAS has advanced capabilities to deliver reports and statistical analysis. By using our setup we are able to avoid building custom reports in PHP or designing a web application with SAS, either of which would be an enormous headache.

3.2 Survey Data Tables in MySQL

We needed to redesign our database to complete this project. We started with a table representing the result set from each survey administered. Each table followed a slightly different format with differing field names and Dynamic Web-based Reports Using SAS 3 types.

We decided to alter the structure of each table creating two fields (a numeric field and a character field) for each field that previously existed to take advantage of the full capabilities of SAS.

We also decided on a naming convention to standardize all tables. Then we developed two tables for use in the Input Collection phase. The first table is a simple list of surveys that the end-user is allowed to choose from together with some other data about each survey. The second table warehouses the text for each survey question asked along with an identifier that corresponds to the field names from each survey table. This allows us to only display question text and to tell SAS in what fields we are interested.

4. Input Collection (Stage 1)

The aim for stage one was to design a simple interface for end-users to interact with. Apache, MySQL and PHP are widely available and widely used for this purpose. All three are open source projects, meaning that there are no direct fiscal costs for the licensing or use of the software.

All three software products can be found on the Internet and include full instructions for installation. We recommend reading all documentation and installation instructions carefully before attempting to install or setup any of these software packages. Any problems encountered can usually be solved by using your favorite search engine (e.g., Google).

Apache is a web server and is the silent partner in the entire operation. Apache's primary job is to monitor all web activity and send requests back and forth through the Internet.

4.1 HTML Forms

The fundamental idea of the Input Collection stage is to allow the end-user to choose options from a website and then send those inputs to SAS. We needed to use HTML forms to do this.

The form object is a collection of HTML elements that allow the end-user to submit or input information which can then be processed by client or server based applications. Below are examples of form definitions.

```
<form name="AName" method="GET"
action="web address">
    insert sub-elements here
</form>
```

or

```
<form name="AName" method="POST"
action="web address">
```

```

    insert sub-elements here
</form>

```

PHP and SAS, in this instance, both qualify as server side applications. The form object has a set of sub-elements that allow the end-user to change the inputs. The inputs we used include select boxes, radio buttons, submit (buttons), and hidden inputs.

Select boxes (commonly known as drop downs) allow an option to be chosen from a list.

```

<select name="some name">
  <option value="">some
text</option>
</select>

```

Values in a form can be supplied and/or altered by end-user or can be supplied by a server side application. In our case we supply the values in select boxes and radio buttons via PHP and MySQL.

The submit button activates the form action. The names and values associated with each of the defined form sub-elements, or inputs, are passed from the client to the program on the server defined by the HTML form action attribute. This data is passed via the defined method attribute (GET or POST) of the form tag.

```

<input type="submit" value="Submit">

```

Notice that there are two options for the method attributes. `GET` sends the inputs into the address string and `POST` sends them through a special mechanism that can only be used by a server side language. Either method can be used when submitting inputs to the Application Dispatcher.

There are also two hidden fields that pertain to options required by the

Application Dispatcher that we will discuss in more detail.

```

<input type="hidden" name="_service"
value="name">

```

```

<input type="hidden" name="_program"
value="name">

```

4.2 PHP

We are using PHP to query a MySQL database and fill in some of the form's sub-elements, such as the select boxes and radio buttons. There are a few different versions of PHP and PHP modules.

We use PHP5 with the PEAR package enabled. PHP has an online manual with several tutorials on how to program using PHP and how to do simple things like connect to a database, perform a query and display or use the results. This manual can be found at:

<http://www.php.net/manual/>

PHP is straightforward to learn and for this reason we leave it up to the reader to find more on this topic.

4.3 MySQL

MySQL also has a strong presence on the web. A comprehensive manual can be found at:

<http://dev.mysql.com/doc/mysql/>

This manual includes a tutorial that can be used as a good start, and includes everything needed to use and administer a MySQL database. It can also be found in print form at your favorite local, privately owned, bookstore.

5. Result Calculation and Display (Stage 2)

After the end-user has clicked the submit button, all inputs are sent to SAS to a SAS/IntrNet program called the Application Broker. The main purpose of the Application Dispatcher is to provide a CGI gateway between a Web browser and SAS. Once the Dispatcher is set up and the service is running, the end-user only needs worry about the design of the SAS program.

The Application Dispatcher requires a web server for its setup. The Application Dispatcher is defined as a Compute Service. We are running Apache and SAS/IntrNet on a Windows machine.

Compute Services give the end-user full access to the analytical capabilities of the SAS server. End-users can access and use any non-visual functionality provided by the SAS server (from [SAS IntrNet: A Roadmap](#)).

Run the [Create a New IntrNet Service 9.1](#) wizard to set up the Application Dispatcher. When we did this on our Windows machine it created a folder in `C:\Program Files\SAS\IntrNet` with the chosen service name. We will need to alter some of the files in this folder for the service to work properly.

At the time of this publication we have done this setup in the Windows environment only. We have plans to try this in a Linux environment.

When the Application Dispatcher is configured, a CGI program called `broker.exe` is placed in the `cgi-bin` directory of your web server. You must specify this program when defining the `action` attribute of the `form` tag.

The `action` attribute can be seen where `address` refers to the web address of your SAS/IntrNet machine or the IP address of this machine.

```
action=http://address/cgi-
bin/broker.exe
```

5.1 Required Inputs for the Application Dispatcher

When a request is sent to the Application Dispatcher, it must include two inputs. These inputs tell the `broker.exe` which service to use and which SAS program to run. We have implemented both as hidden inputs.

5.1.1 `_service`

The `_service` input must be set to the name of the service you want called. Again, this is set up using the socket service wizard. On our system we start this service by running the `appstart.bat` batch file that is in the service directory.

5.1.2 `_program`

The `_program` input refers to the SAS program that you want to run. The program name must have three or four levels delimited by periods to work properly. If it does not, you will get an error message reminding you.

Lines must be added to the `appstart.sas` file in the service's directory to define the `library` section of this identifier.

`location` refers to the folder where all SAS programs will be located.

```
allocate file library `location';
proglibs library;
```

5.2 Extra Inputs

The Application Dispatcher can accept an arbitrary number of extra inputs. These can appear in your SAS program as variables. For example, if we were interested in allowing the end-user to specify the gender and marital status of a specific type of student, we could define three inputs named `gender`, `marital` and `studentType`.

Then we would refer to these inputs as the variables `&gender`, `&marital` and `&studentType` in the SAS program.

5.3 SAS Program

Here is an example of a SAS program that will run when called by the Application Dispatcher. Any SAS program that can be called on the command line can be executed by `broker.exe`, as we mentioned earlier.

We begin by using a `libname` statement to create a library to the database holding all of our survey data. Next, we create a temporary table using `proc sql` into the work library.

Notice that we use the variable `&survey` which is sent through GET/POST as specifying the data requested by the end-user.

```
libname SURVEYS mysql
  user='user' password='pswd'
  database='dbname'
  server='ip' port=3306;
run;

proc sql;
  create table work.my_survey as
  select * from SURVEYS.&survey;
run;

ods listing close;
ods html body=_webout
```

```
(dynamic title="&mytitle") rs
= none
  style = Brick;

title "&name";

proc tabulate
data=work.my_survey;
class &options &question;
var dummyN;
table (&options=' ') all,
  (&question=" ") *ROWPCTN
  dummyN=' '*n;
keylabel n = '# Responses'
  rowpctn = ' '
  all = 'Mean Response';
run;

ods html close;
ods listing;
```

Next we close `ods listing` and open `ods html`. Here we send the body to `_webout` which means the output will get sent to the end-user's browser. Use `dynamic title=""` to set the text in the title bar. Output style can be altered using the `style` command. SAS has several predefined styles to use, a custom style can be created using `proc template`.

You can also link to an external style sheet to change the look of your output. The `title` statement adds a title on your output, as opposed to the title in the title bar of the end-user's browser. Then we use `proc tabulate` to display our results.

Notice that we have two variables in use, `&options` and `&question`. These refer to two questions chosen by the end-user. Both show up in this case because as class variables they happen to have character data in them.

Since we want the character data to show up in the results we have defined a separate column in each of our data

tables called `dummyN`. This column is all 1's and only serves the purpose of giving us a numeric column to work with for the `var` statement.

We close with `ods html` and re-open `ods listing` at the end of the program. At this point the SAS program is finished and the results are displayed to the end-user's browser window.

6. Conclusions

This paper demonstrates and documents how we integrated the LAMP architecture with the power of SAS to create dynamic end-user driven web application that produces reports containing any SAS output.

Recall that we began this paper by mentioning that one of our goals was to create both analytic and descriptive reports. This is an example of creating a

descriptive report for the survey data that we warehouse.

It would be as simple to create a web interface for analytic reports. In this case, just replace `proc tabulate` with `proc glm` or any other analytic procedure that SAS provides.

Contact Information

Jeremy Morris
University of Utah
Office of Budget & Institutional Analysis
110 Park Building
201 S. Presidents Circle
Salt Lake City UT 84112

jeremydmorris@gmail.com

