DEVELOPMENT OF AN ONLINE DATABASE TOOL FOR QUICK ACCESS TO MASS ESTIMATING RELATIONSHIPS:

GOMER – GEORGIA TECH ONLINE MASS ESTIMATION RESOURCE ADMINISTRATOR/DEVELOPER'S GUIDE

Janssen Pimentel
Georgia Institute of Technology
Under the Academic Supervision of Dr. John R. Olds
AE 8900

April, 2004

TABLE OF CONTENTS

ABSTRACT

This document describes the methods and approach used in the development of an online database tool used for quick access to mass estimating relationships. The resulting tool is GOMER – Georgia Tech Online Mass Estimating Resource. This tool utilizes a typical three-tier database architecture employed by many web database applications. A unique relationship scheme was used to preserve the complex relationships found in the data sets. This scheme allows the user to search through the database using a variety of methods, while returning a list of results that can be examined in further detail. MySQL and PHP were used extensively throughout in the creation and implementation of the GOMER database.

1.0 - OVERVIEW

The objective for this project is to have an easily-accessible reference tool when performing a preliminary sizing analysis of a conceptual-level vehicle design. Though there has already been an effort to compile mass estimating relationships (MERs) for reusable launch vehicles (RLVs) into one downloadable volume, it was felt that placing this information in a more intuitive, dynamic online format would be more beneficial. As such, the previously mentioned MER information was compiled into an online database from which a user can search for equations, compare the relationship's predicted values to reference values, and submit MERs for inclusion into the database.

1.1 - MASS ESTIMATING RELATIONSHIPS (MERS)

Mass estimation relationships, or MERs, are commonly used to size aerospace vehicles in preliminary or conceptual design. These equations are typically parametric in nature, usually with the equation parameters defining certain vehicle design properties. It should also be mentioned that most MERs are regression fits based on limited data. In addition, not all the MERs for a certain weight group necessarily agree with each other due to the different data points and regression analysis with which each reference source author used to derive the relations. However, for conceptual design, MERs have the advantage in that they are quick and easy ways to determine the vehicle's size with a relatively sufficient degree of accuracy.

1.2 - GOMER

GOMER, or Georgia Tech Online Mass Estimation Resource, is a web application tool used to search for vehicle MERs. These MERs were obtained from a previous Master's project and transferred to a database format. This is advantageous in that the data in the database can be accessed by multiple users simultaneously. Another benefit to having a dynamic database is the relative ease with which search queries are performed.

2.0 - DATABASES

2.1 - INTRODUCTION

A database can be defined as a collection of related data. In this case, the data in the database are the actual equations, as well as other related information. (The relationships between data inside the database will be discussed in more detail in a later section). As with most databases, a typical three tier architecture was employed for this project.

2.2 - ARCHITECTURE

Like most typical web database applications, GOMER uses a three-tier architecture. The three-tier architecture consists of the database tier, the middle tier, and the client tier. The database tier contains the database itself and the database management system (DBMS) used in the handling and upkeep of the database. The middle tier contains the webserver and any scripts used to communicate between the browser and the database. The upper tier is the client tier, which accesses the database through the webserver and scripts in the middle tier. This architecture is presented in the figure below.

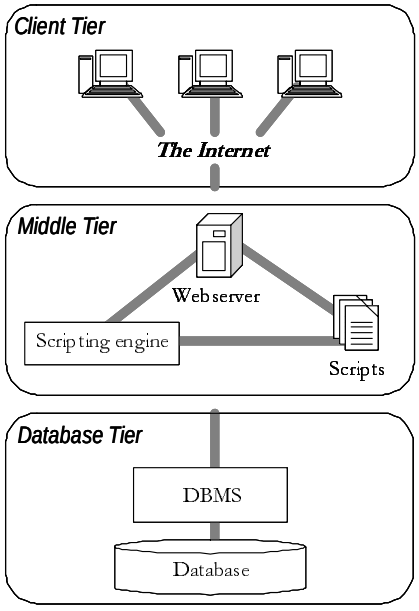


Figure 2-1: A simple representation of the three-tier architecture used in web database applications like GOMER.

In the simple representation depicted in the figure, a user retrieves information from the database through the internet. In reality, the client uses the hypertext transfer protocol (HTTP) to carry a request from the user's browser to the webserver. Scripts on the webserver translate the user's request into an interpreter language that can be understood by the DBMS, which then queries the database. Results are passed back to the DBMS, is translated into a browser-readable format by the scripting language, which is then parsed and viewed by the user on the web browser.

Even with a simplified architecture model, one can appreciate the complexity of the relationship between each tier. This is compounded even more with the number of scripts that must be run, and made even more difficult as the size and complexity of the database increases. However, the intricacy of this process can be reduced significantly by taking the time at the beginning to outline a proper database relationship scheme.

2.3 - RELATIONSHIP SCHEMA

The GOMER database consists of basically 3 different elements: the equations themselves, the parameters used in the equations, and the references associated with those equations. Those three elements are interrelated, as each reference source produces a set of equations which have their own unique set of parameters. Not all parameters are unique to a specific equation, however; some are common to all equations. Moreover, these common parameters also exist in multiple reference sources. Graphically, the relationship between these three elements is depicted in the following figure.

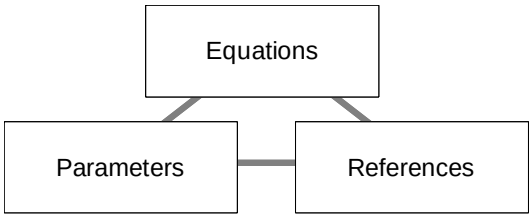


Figure 2-2: Simplified representation of GOMER database element relationships.

However, this is still a fairly basic representation of the problem. A better relationship schema which best captures the intricacies of relationships between various elements in the database (albeit one that is more complicated) is shown below:

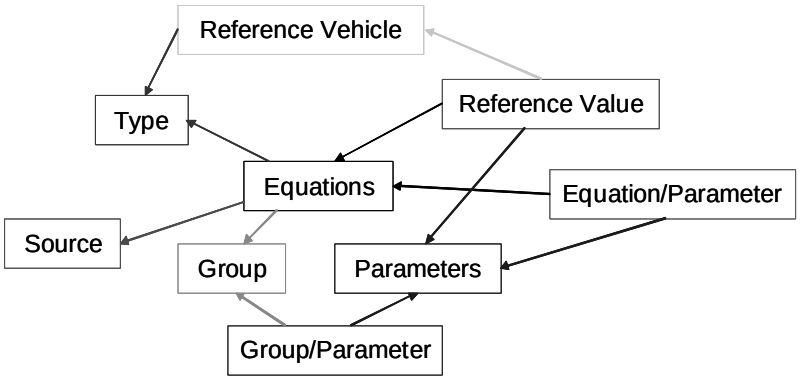


Figure 2-3: A better schema that captures interrelationships between elements.

With this scheme, one can easily identify the interdependency of one element on another. For instance, *equations* are composed of a certain set of *parameters*. Specific *equations* belong to a certain weight *group*, and are applicable to a certain *type* of vehicle. (The term "group" in this project refers to a weight group as defined by a weight breakdown structure, and "type" refers to a specific vehicle type—i.e., RLV, etc.) Each weight *group* contains a list of group-specific *parameters*, as well as common *parameters* and unique equation-specific *parameters*. Many of the *equations* are associated with multiple vehicle *types*. Many *parameters* are only associated with select *groups* (and not others), while applicable to a particular set of vehicle *types* (through the *equations)*... and so on and so forth. This is already a fairly heady example, and yet a third of the database elements have not yet been introduced into the fray. Not surprisingly, multiple iterations on this schema took place before the one used in GOMER was finalized.

3.0 - MySQL DBMS

3.1 - Introduction

Once the relationship schema was established, the next step was the actual creation of the database to be used in the project. There are multiple approaches to doing so; various DBMSs exist upon which GOMER could be based on. In the end, the MySQL DBMS was chosen due to the advantages it held over other systems:

- It is arguably the most popular DBMS used today. As such, there is a large development community dedicated to improving this system.
- It is relatively easy to learn over other systems.
- It is comes prepackaged with Mac OS X.
- It is free.

The first bullet clearly shows the strength of using such a popular system: it most likely won't die anytime soon, and there are plenty of resources available for help if it were needed. The advantage held by the third bullet was not particularly applicable for this project, as the version of OS X installed on the server on which GOMER resides did not have MySQL preinstalled and thus had to be manually installed. However, if and when the server does get upgraded to the next version of X, it can be assured that the database and its functionality will survive. Lastly, the final bullet is worth noting, if for nothing more than simple amusement.

3.2 - Command Line Interpreter

MySQL is normally accessed via the command line interpreter. A short example on how to use this interpreter to create and populate tables is described below. For further information on how to use the command line to create and manage databases, O'Reilly's "Web Database Applications with PHP and MySQL" is recommended reading. At this point, any text formatted like the following should be taken as a command line input to be typed in by the user: `command`

To begin with, the user must log onto the system. Using an SSH client such as SecureCRT, log onto *ssdl20.ae.gatech.edu* using the *ssdl* username. Enter the password when prompted. Once logged on, the screen should show the following:

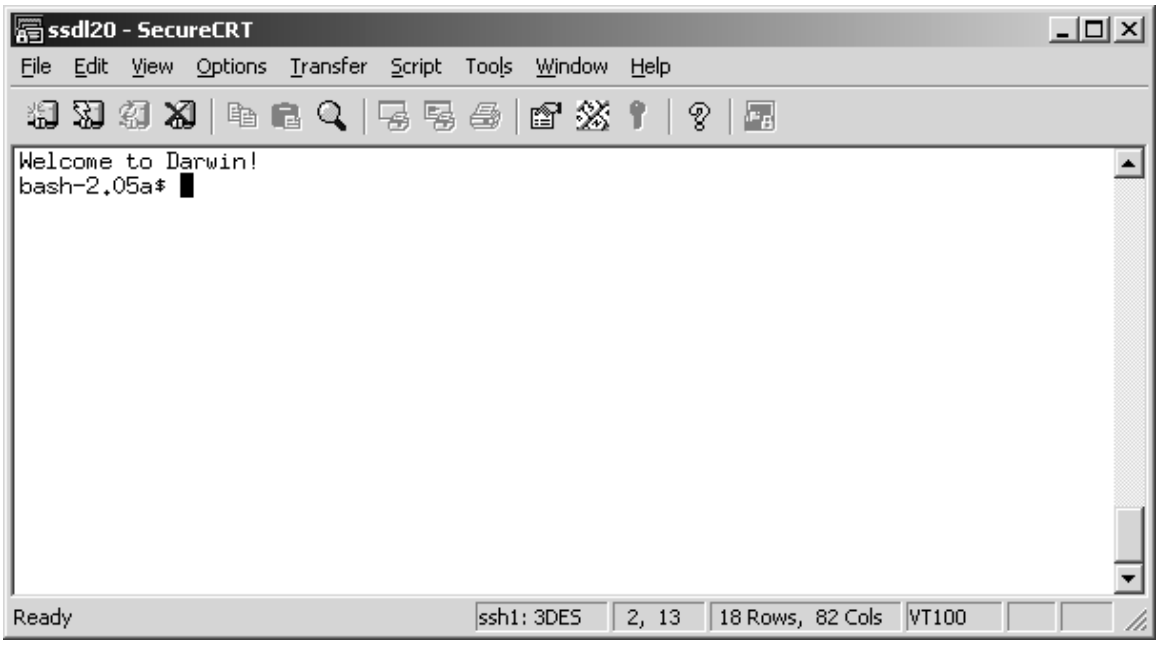


Figure 3-1: User *ssdl* logged onto host *ssdl20.ae.gatech.edu*.

To access the MySQL command line interpreter, type in the following at the prompt:

```
mysql -u merdb -p merdb
```

It will ask for the password, which is "*merdb*" (without the quotes). The window should now be something like this:

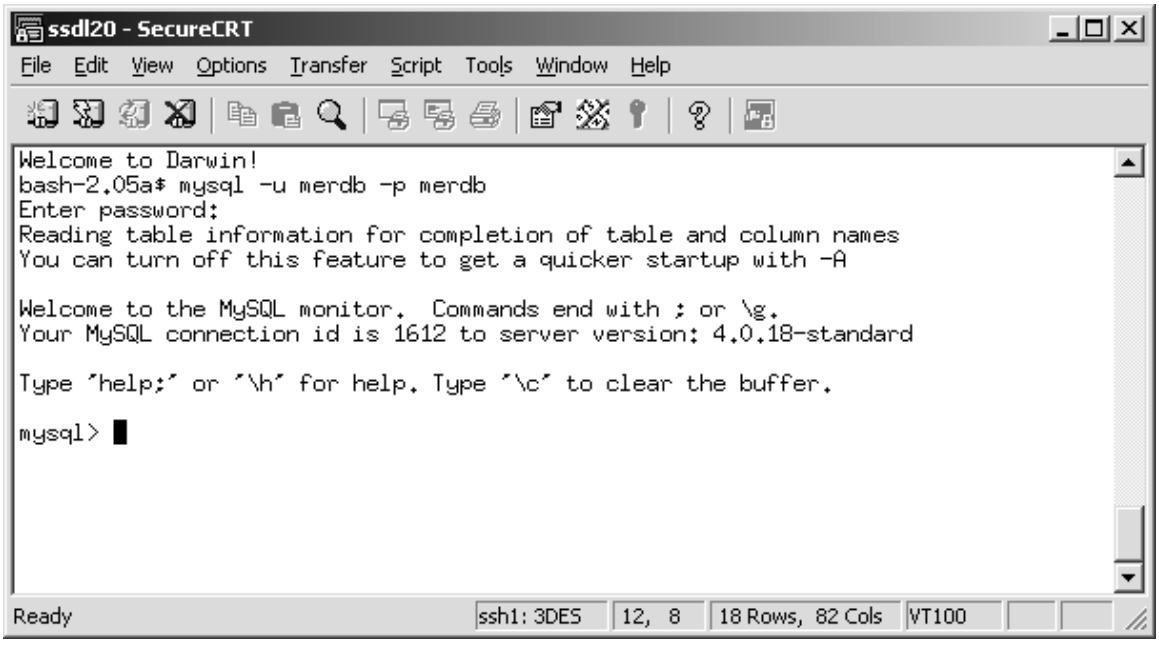


Figure 3-2: Logging onto the MySQL command line interpreter.

The user is now logged onto the MySQL interpreter and working in the *merdb* database. This is where management and handling of the database is performed. Data in a database is stored in tables, and so the next step would be to create an example table using the following commands and substituting when appropriate:

```
create table tablename (
id int(n),
col_1 vartype,
col_2 vartype,
primary key (id)
);
```

Note that the end of the command is terminated by a semicolon. Additionally, each of the lists is separated by a comma. In other words, the command doesn't need to be split into multiple lines; the same command typed on one line will run just the same. For example, if one were to type the following commands into the interpreter as such:

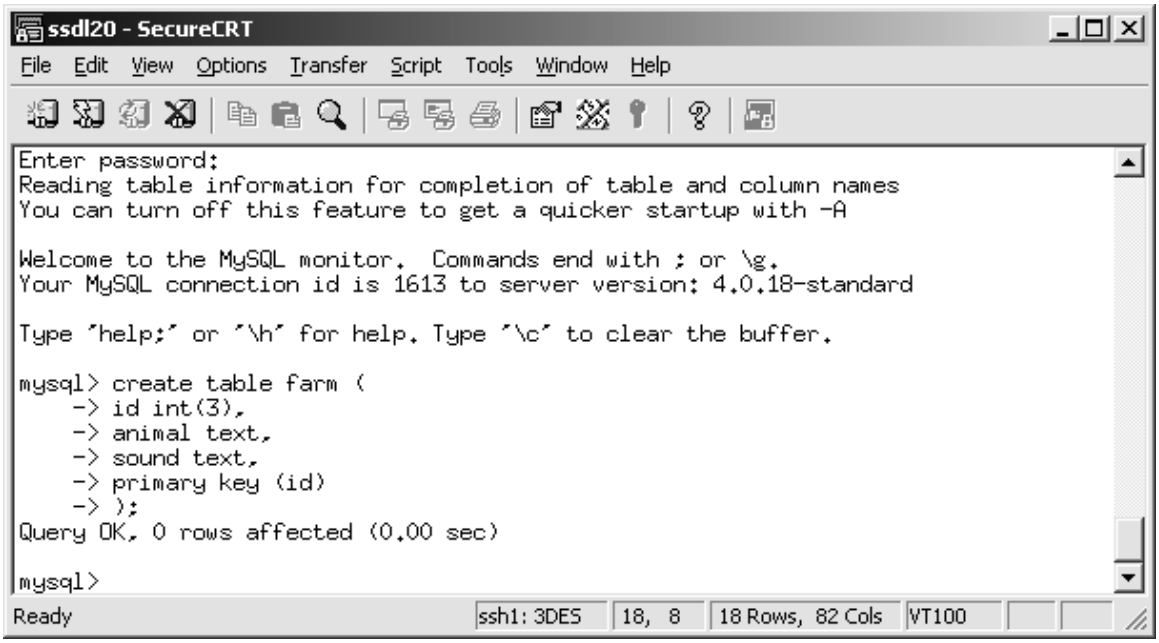


Figure 3-3: Creation of table *farm* in the *merdb* database using the MySQL interpreter.

A table called *farm* with three columns is created. This is an empty table (no rows) with the first column labeled as the id and identified as the primary key. This primary key would be used to relate this table to another table—in the same way that equations

would be related to parameters, for instance. Inserting values into the table rows is a relatively simple task. Using the following commands it can be achieved:

```
insert into tablename
values  ('val_for_column1','val_for_column2','val_for_column3');
```

The resulting table can then be viewed:

```
select * from tablename
```

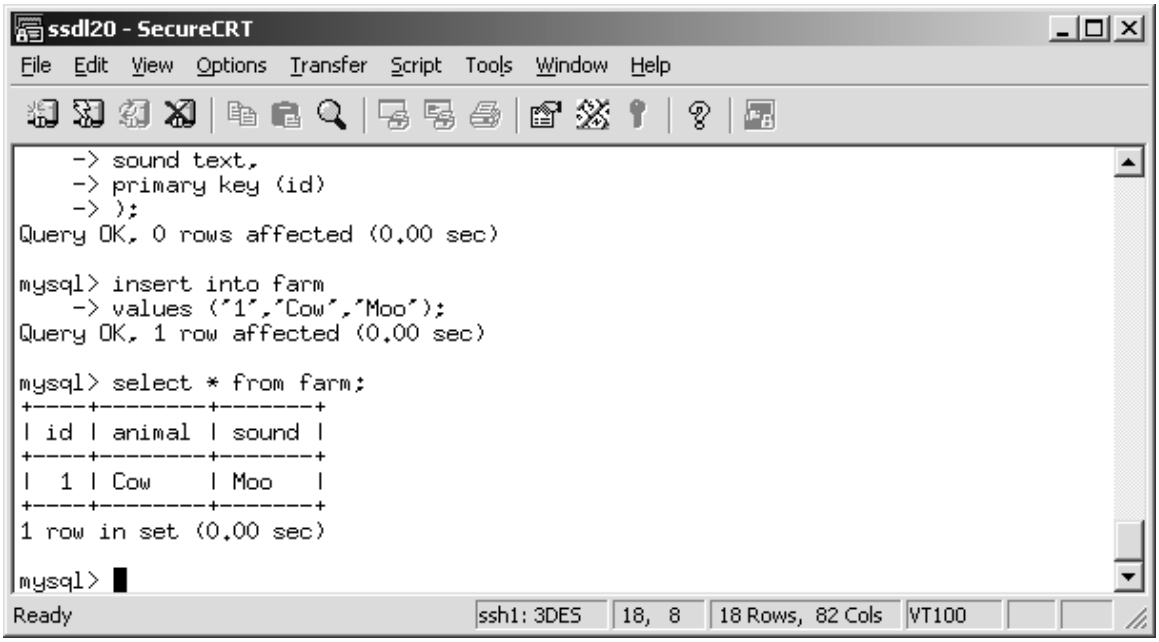


Figure 3-4: Populating the table *farm* and displaying the result.

That completes the short example using the command line interpreter. While not overly complicated, it should be noted that for a database as complex as GOMER, this method of creating tables/inserting data may not be the most efficient. Exiting the command line interpreter is performed by typing in `quit`.

4.0 - PHP SCRIPTING LANGUAGE

4.1 - INTRODUCTION

For this tool, PHP was the scripting language chosen to tie everything together. Recall that scripts are contained in the middle tier of the three-tier architecture (refer to Section 2.2) and are used to communicate between the database and the server. Fortunately, using PHP to talk to MySQL is common practice, and as such many resources exist that take full advantage of this relationship (see References i and ii).

Yet another advantage to using PHP is that it can be embedded in HTML code; that is, the script can be included inside a webpage's HTML source, much like JavaScript. However, unlike JavaScript, a PHP script can be used to create HTML code, therefore allowing the creation of dynamic webpages. To put this into perspective: for GOMER, PHP is used to create the results page after a search query is run, showing a table listing of MERs matching given search criteria.

However, one must keep in mind that PHP has limitations. Client-side scripting languages such as JavaScript allow changes on the client's web browser window (mouseovers, dynamically generate forms, resize the window, etc.) PHP, on the other hand, is not able to do those things. It is a scripting language which only executes after a specific event occurs, such as clicking on a "Submit" button. While this is not a particularly debilitating limitation, it is something to keep in mind when designing the functions and features of a webpage.

GOMER uses PHP extensively in order to communicate between the user and the database. It translates the information (i.e. – search parameters) supplied by the user's web browser into SQL commands, runs the query, then takes the SQL result and generates a table of search results. Two such scripts are located in the Appendix. Still, perhaps the best part to using PHP along with MySQL is the ability to employ phpMyAdmin.

4.2 - PHPMYADMIN

Introduced in an earlier section was the concept of using MySQL's command line interpreter to create and manage databases. However, one can easily perceive that as the

complexity of a database increases, using a command line to manage the database can be a rather tedious and trying task. Remember that one of the reasons that MySQL was chosen was due to the popularity of use, and the large development community that is associated with it. For this reason, there are multiple graphical user interfaces (GUI) out there that simulate the functions of MySQL's default command line interpreter. One such GUI is phpMyAdmin. In order for a user—in this case, the GOMER administrator/developer—would type the following web address into a browser: http://ssdl20.ae.gatech.edu/merdb/phpMyAdmin When prompted for the username and password, enter "*merdb*" for both fields (without the quotes).

Once logged on, the screen should show something similar to what is shown below.

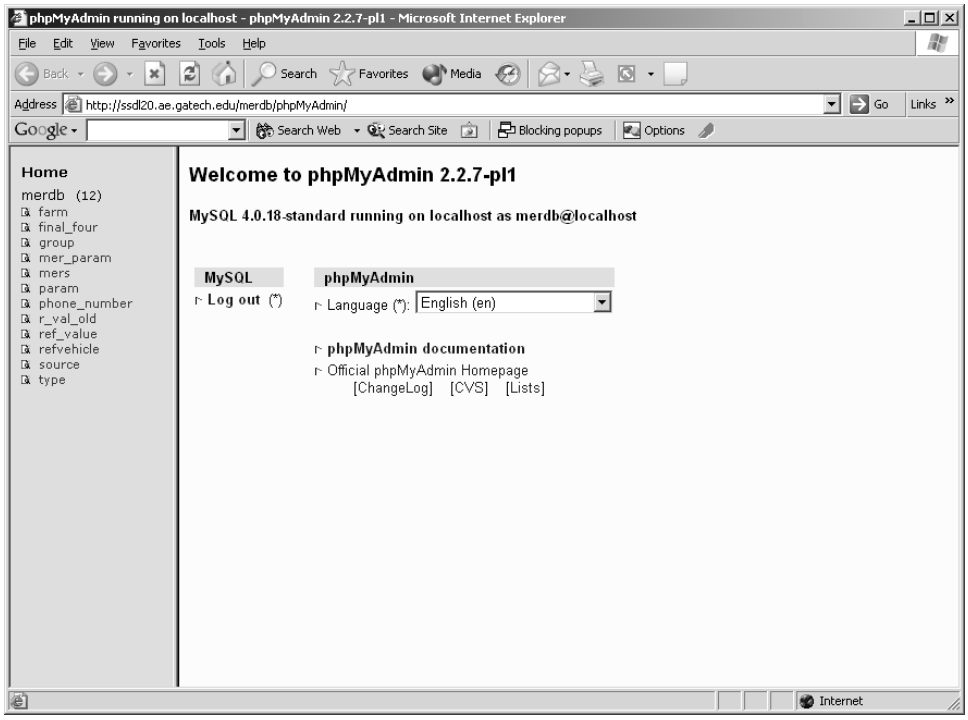


Figure 4-1: The main phpMyAdmin screen which first greets the user after logging in.

Using this interface, one can quickly and easily search through and add to the current database; which in this case is the *merdb* database. The tables that exist within this database are listed on the webpage's left frame, with management options listed on the main page frame. Clicking on any of the tables (for example, *farm*) brings up the following screen:

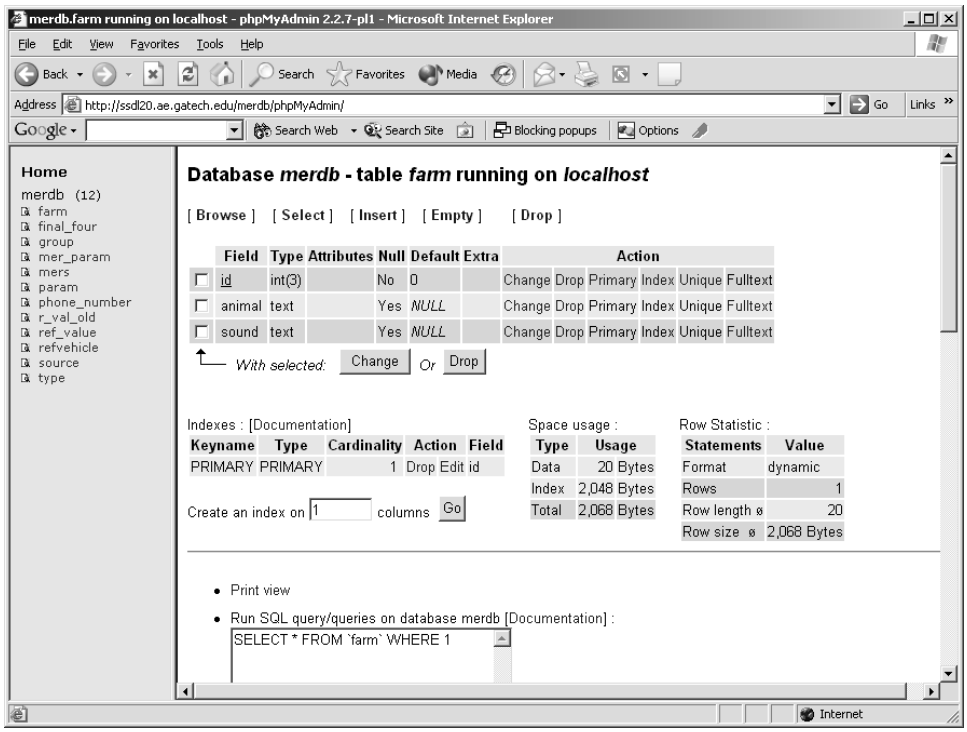


Figure 4-2: Screenshot of the table attributes page which allows the user to manage the selected table in the database.

Notice that the column (field) headings for the selected table are shown. In this example, the columns "id," "animal," and "sound" are listed, along with their variable type and other attributes. Using the navigation links above, one can then choose to browse the table (view the table's contents), manually insert values into the table, empty the table, or completely drop the table from the database. Word of caution: there is no Ctrl+Z undo option when using phpMyAdmin, so take care when dropping tables or deleting data. Clicking on the "Browse" link then allows the user to see the contents of the table.

Inserting a new row of values can be done by clicking on the "Insert new row" link located below the table. This is fine when adding a single, or a few, rows of data into the table. However, for GOMER, it was determined that bulk loading the information was the best route to take. This can easily be demonstrated using the *farm* table as an example.

To begin with, a comma separated value, or .csv, file is first created. This can be done in any number of ways—Excel, for instance, can be used. The screenshot below shows an Excel spreadsheet containing values that will be added to the *farm* table.

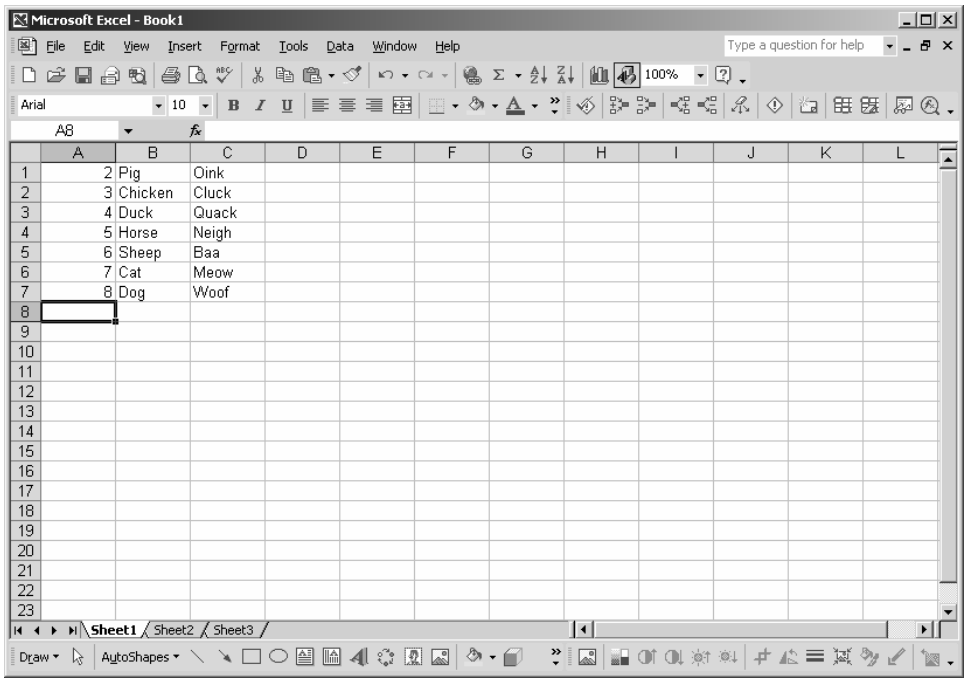


Figure 4-3: Sample data to be included in the *farm* table.

Notice that there are three columns in the spreadsheet, just as there are three columns in the table. The first column is associated with *farm*'s "id" field, the second column with "animal," and the third with "sound." It is important to remember this point, as there would inevitably be much cursing and frustration if this was neglected. The next step is to save the spreadsheet as a .csv file. This can be done by going to *File-Save As* and selecting "comma separated value" from the "Save as type" pull-down menu.

Back at the phpMyAdmin page, click on the table name that the data will be imported into (in this instance, *farm*). In the main page frame, the link to insert data using a data file is located beneath the table attributes:

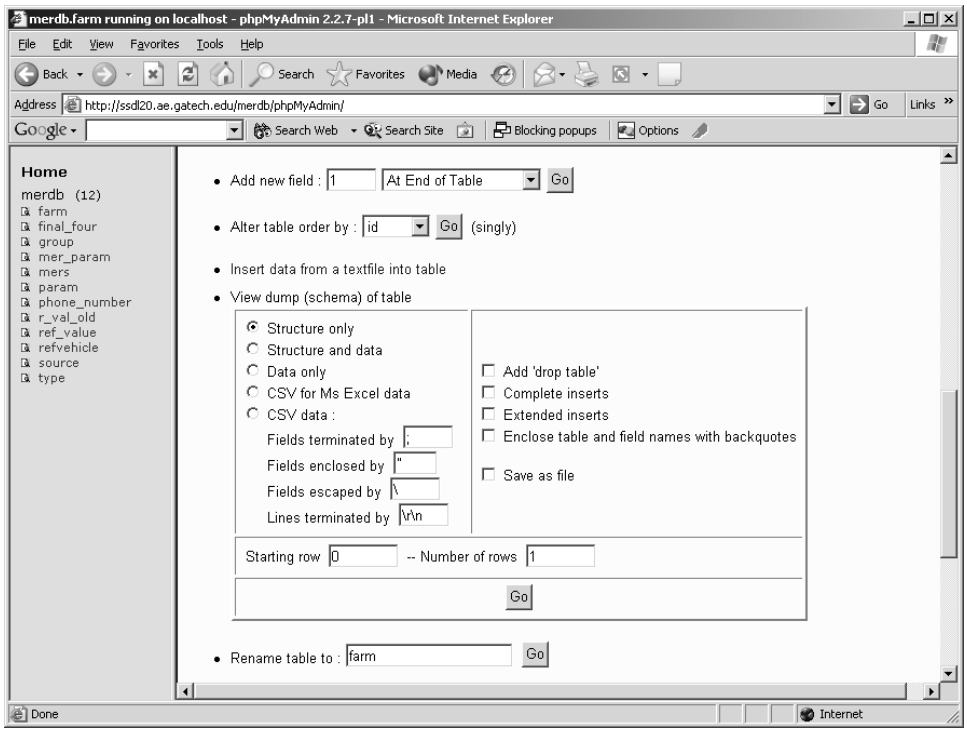


Figure 4-4: Back at the table attributes page, bulk loading of data from a data file can be located by scrolling down the page.

Clicking on the link then brings up the following page:

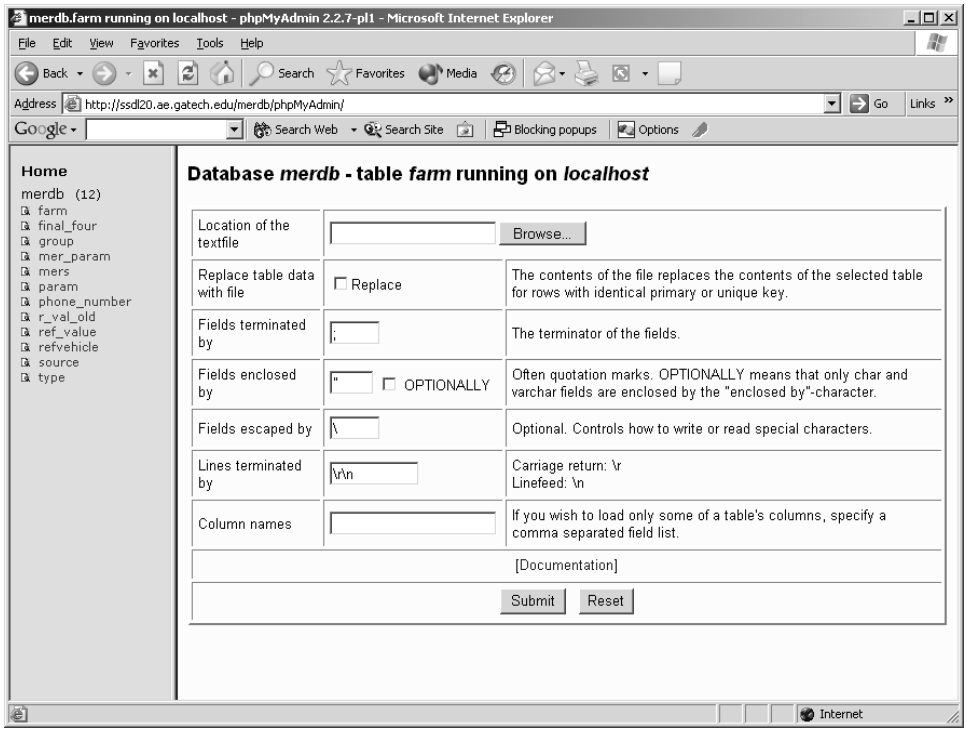


Figure 4-5: A screenshot of the data file upload page used to bulk load data into a table.

The .csv file can then be uploaded by clicking on the “Browse” button and locating the file. Also, it must be noted that since the file is a comma separated file, the “Fields terminated by” option must be changed from a semi-colon to a comma. Once this is set, clicking the “Submit” button brings the user back to the table attributes page. Viewing the updated contents of the table is done by clicking on the “Browse” link:

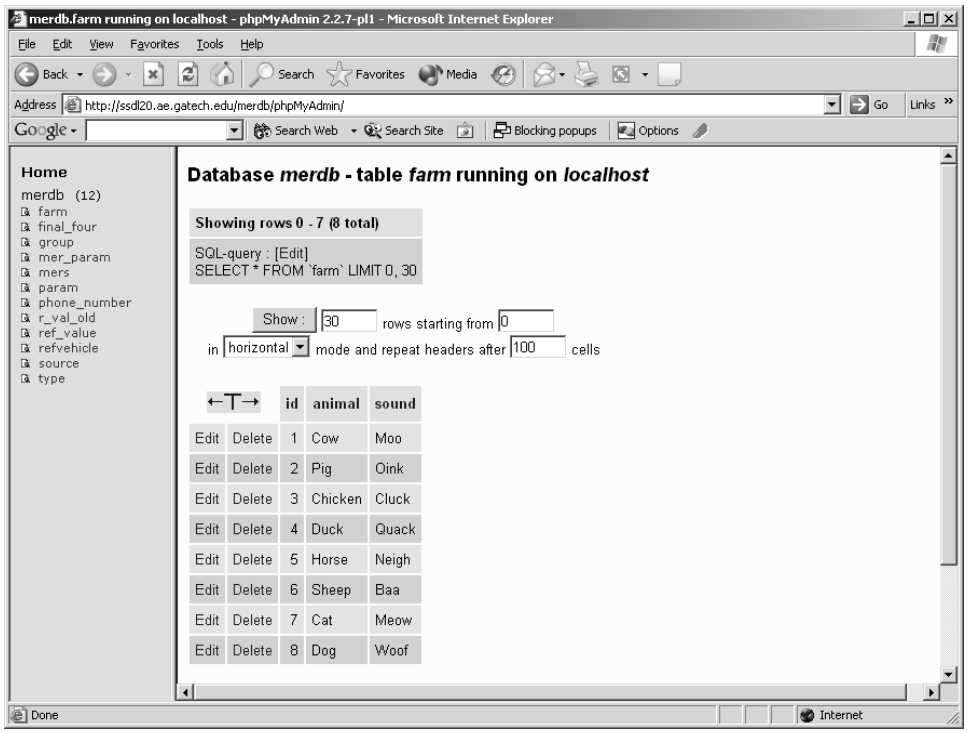


Figure 4-6: A view of the updated *farm* table including the sample data loaded from the .csv file.

It should also be noted that a .csv file containing all the elements of a table can be downloaded using this tool. This procedure is as simple as going to the table attributes page (Fig. 4-4). Notice that in Fig. 4-4 there is a bullet named “View dump (schema) of table.” Selecting the appropriate radio buttons/checkboxes will allow the user to view the table’s contents. All one has to do is then copy the data shown, paste it into a new .csv file, and then the user has the complete table data on his/her local machine.

That completes a quick overview of phpMyAdmin. Further exploration into the capabilities and functionality provided by this tool is highly recommended. Most of the functions and features are relatively self-explanatory; however, documentation abounds which greatly assist in understanding how to get certain things to work.

5.0 - GOMER DATABASE STRUCTURE

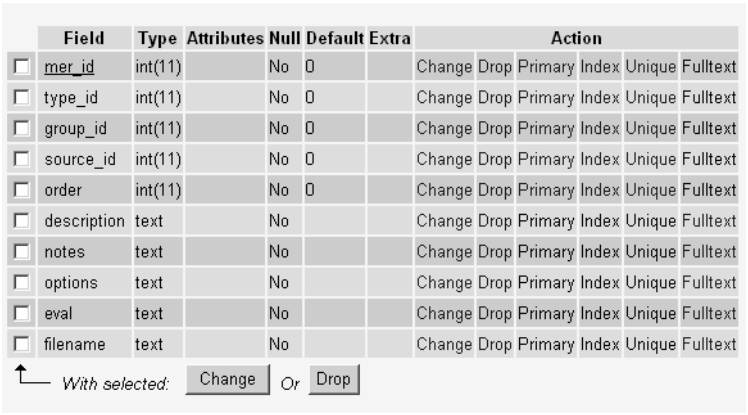
Until this point, this document gave a brief overview of the methods and approach used in the creation of GOMER. This section will focus on specific details regarding the GOMER database structure.

5.1 - TABLE DETAILS

All of the different database elements which were shown in the relationship schema figure (Figure 2-3) are individual tables in the GOMER database. Each table has unique attributes which are discussed in the following sections.

5.1.1 - *MERS* DETAIL

The figure below shows a screenshot of the *mers* table attributes as seen from phpMyAdmin:



	Field	Type	Attributes	Null	Default	Extra	Action
☐	mer_id	int(11)		No	0		Change Drop Primary Index Unique Fulltext
☐	type_id	int(11)		No	0		Change Drop Primary Index Unique Fulltext
☐	group_id	int(11)		No	0		Change Drop Primary Index Unique Fulltext
☐	source_id	int(11)		No	0		Change Drop Primary Index Unique Fulltext
☐	order	int(11)		No	0		Change Drop Primary Index Unique Fulltext
☐	description	text		No			Change Drop Primary Index Unique Fulltext
☐	notes	text		No			Change Drop Primary Index Unique Fulltext
☐	options	text		No			Change Drop Primary Index Unique Fulltext
☐	eval	text		No			Change Drop Primary Index Unique Fulltext
☐	filename	text		No			Change Drop Primary Index Unique Fulltext

With selected: Change Or Drop

Figure 5-1: Table attributes of the *mers* table within the GOMER database.

Notice that there are ten column fields. The first is the "mer_id" and serves as the primary key for this table—each MER in the table is assigned a unique mer_id. Two other fields in that table are used to refer to data in other tables (in effect, used to "link" data between tables): "type_id," "source_id." This ensures that the relationships between the *mers*, *type*, and *source* tables are preserved. Recall that the relationship schema presented in Section 2.2 shows that each equation is applicable to a certain *type* of vehicle (via the "type_id"), and comes from a given reference *source* (via the "source_id").

The "order" field describes in which order the equation is listed. For example, reference source *X* may have 3 equations that are related to the wing group, and are listed in a particular order. This field maintains that order when the results of the search query are displayed. The remaining fields in this table are the "description," "notes," "options," and "filename" fields. While the first three are fairly self-explanatory, some attention should be spent on the "filename" field.

The resulting equations that are displayed on the web browser after a successful search are image files. As such, each equation has an associated file name which is unique to only that equation. This filename follows this format: *a*.*b*.*c*-*d*.jpg where:

- *a* denotes the vehicle *type*—1 for RLV, 2 for expendable launch vehicles (ELV), 3 for space transportation vehicles (STV)
- *b* denotes the weight *group* (described by the weight breakdown structure, WBS)
- *c* denotes the reference *source* from which the equation is obtained from
- *d* denotes the order at which the MER should be displayed

So, for example, the second wing group equation applicable to RLVs that is obtained from reference source 3 would be listed as 1.1.3-2.jpg. Great care must be taken to ensure that each MER has the appropriate image filename.

5.1.2 - *GROUP* DETAIL

The *group* table is shown in the following figure:

	Field	Type	Attributes	Null	Default	Extra	Action					
☐	mer_id	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	type_id	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	group_id	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	source_id	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	order	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	description	text		No			Change	Drop	Primary	Index	Unique	Fulltext
☐	valeng	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	valsi	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	kind_id	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	filename	text		No			Change	Drop	Primary	Index	Unique	Fulltext

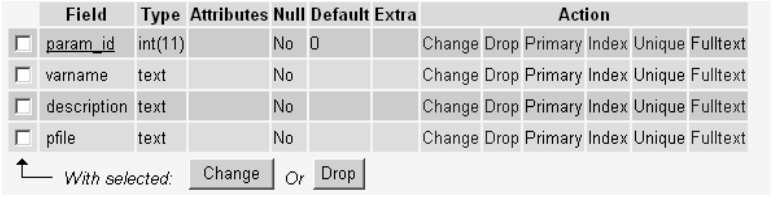
With selected: Change Or Drop

Figure 5-2: Table attributes of the *group* table within the GOMER database.

This table refers to the vehicle weight groups, which are outlined based on a typical vehicle WBS. The "group_id" field is serves as the primary and unique key for each entry in the table. This key is also what is used to refer back to the MERs in the *mers* table. The other two fields, "name" and "description," describe the name and short description associated with each weight group.

5.1.3 - *PARAM* DETAIL

The next table is one which contains the data regarding the different parameters used in the MERs:



	Field	Type	Attributes	Null	Default	Extra	Action					
☐	param_id	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	varname	text		No			Change	Drop	Primary	Index	Unique	Fulltext
☐	description	text		No			Change	Drop	Primary	Index	Unique	Fulltext
☐	pfile	text		No			Change	Drop	Primary	Index	Unique	Fulltext

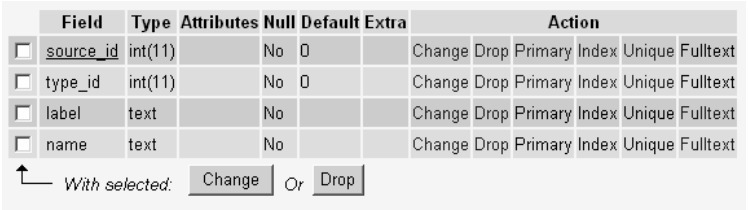
With selected: Change Or Drop

Figure 5-3: Table attributes of the *param* table within the GOMER database.

There are four fields in this table. The first, "param_id" is used as the unique key for each parameter listed in the *param* table. The following two fields—"varname" and "description" are text fields containing the variable name and description. Finally, each parameter has an associated image file called by "pfile." Unlike the "filename" field in the *mers* table, the pfile follows this format: p*n*.jpg, where *n* is the unique parameter id "param_id."

5.1.4 - *SOURCE* DETAIL

Below is a figure depicting the table attributes for the *source* table:



	Field	Type	Attributes	Null	Default	Extra	Action					
☐	source_id	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	type_id	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	label	text		No			Change	Drop	Primary	Index	Unique	Fulltext
☐	name	text		No			Change	Drop	Primary	Index	Unique	Fulltext

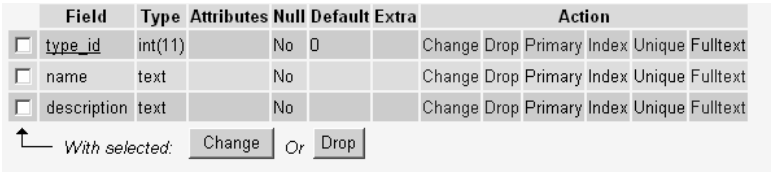
With selected: Change Or Drop

Figure 5-4: Table attributes of the *source* table within the GOMER database.

Notice that there are four fields in this table: “source_id” (which is the unique key for this table), “type_id” (which denotes which vehicle type this source is applicable to), “label” (which denotes the source label that is displayed along with the equation in the results page), and finally “name” which gives a name to the associated *source* label.

5.1.5 - *TYPE* DETAIL

The figure below illustrates the fields that compose the *type* table:



	Field	Type	Attributes	Null	Default	Extra	Action					
☐	type_id	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	name	text		No			Change	Drop	Primary	Index	Unique	Fulltext
☐	description	text		No			Change	Drop	Primary	Index	Unique	Fulltext

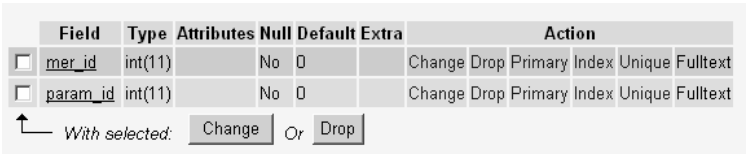
With selected: Change Or Drop

Figure 5-5: Table attributes of the *type* table within the GOMER database.

This is a simple table containing three fields. The first field, “type_id,” is used as the table’s unique key. The other two tables are fairly self-explanatory.

5.1.6 - *MER_PARAM* DETAIL

This next table is unlike the previous tables that have been shown. Whereas the previous tables show discrete database elements, the following table depicts the many-to-many relationship between data contained in the *mers* and *param* tables:



	Field	Type	Attributes	Null	Default	Extra	Action					
☐	mer_id	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext
☐	param_id	int(11)		No	0		Change	Drop	Primary	Index	Unique	Fulltext

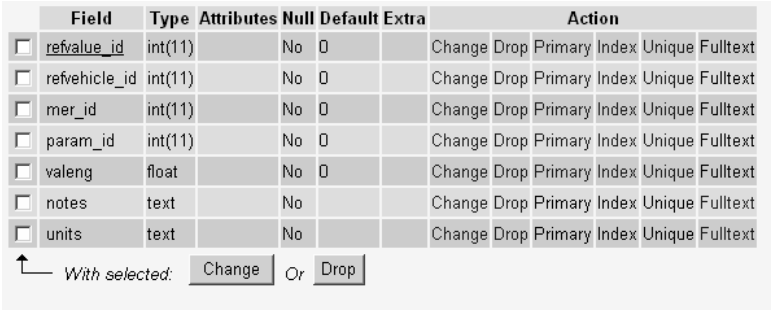
With selected: Change Or Drop

Figure 5-6: Table attributes of the *mer_param* table within the GOMER database.

One easily notices that this table only contains two fields, both of which are the primary keys to other tables. This table essentially defines which parameters go into which equations. A many-to-many relationship table such as this is one proof of why a relational database such as MySQL is useful.

5.1.7 - *REF_VALUE* DETAIL

The following figure illustrates the table attributes for the *ref_value* table:



	Field	Type	Attributes	Null	Default	Extra	Action
☐	refvalue_id	int(11)		No	0		Change Drop Primary Index Unique Fulltext
☐	refvehicle_id	int(11)		No	0		Change Drop Primary Index Unique Fulltext
☐	mer_id	int(11)		No	0		Change Drop Primary Index Unique Fulltext
☐	param_id	int(11)		No	0		Change Drop Primary Index Unique Fulltext
☐	valeng	float		No	0		Change Drop Primary Index Unique Fulltext
☐	notes	text		No			Change Drop Primary Index Unique Fulltext
☐	units	text		No			Change Drop Primary Index Unique Fulltext

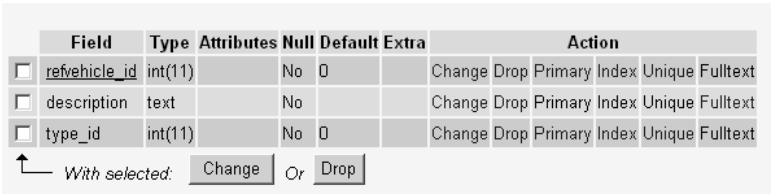
With selected: Change Or Drop

Figure 5-7: Table attributes for the *ref_value* table within the GOMER database.

This is essentially what populates the parameter input fields when the user selects an equation to view in more detail. It is also what is used to evaluate the selected equation. As with most of the other tables, the first field is defined as this particular table's primary key. Notice that the following three fields are the primary keys of other tables, namely the *refvehicle*, *mers*, and *param* tables. Again, we see the many-to-many relationship described in the database relationship schema. The next field, "valeng," contains the parameter values that are displayed in the details page. These are the parameter values associated with the selected reference vehicle geometry and configuration.

5.1.8 - *REFVEHICLE* DETAIL

The last table to be shown in detail is the *refvehicle* table:



	Field	Type	Attributes	Null	Default	Extra	Action
☐	refvehicle_id	int(11)		No	0		Change Drop Primary Index Unique Fulltext
☐	description	text		No			Change Drop Primary Index Unique Fulltext
☐	type_id	int(11)		No	0		Change Drop Primary Index Unique Fulltext

With selected: Change Or Drop

Figure 5-8: Table attributes for the *refvehicle* table within the GOMER database.

This is a simple table containing three fields. As with the other tables, the first field is the *refvehicle* table's primary key. The "description" field provides a short description each reference vehicle listed in the table, while "type_id" defines what type of vehicle it is in addition to linking this table to the *type* table.

6.0 - ADDING AN MER TO THE GOMER DATABASE

This next section will explain how to add an MER to the database. In order to better illustrate this procedure, the following MER will be used as an example (note: this equation already exists in the database):

$$M_{wing} = 2375\left[\frac{M_{entry} N_z b_{str} S_{ref}}{t_{root} \times 10^9}\right]^{0.584}$$

The GOMER developer should make a note of the following characteristics of this eqation: which weight group it belongs to, what source it comes from, what parameters are associated with the equation, and which vehicle types it is applicable to. In this particular example, the MER shown above is an equation provided by Dr. Ted Talay which gives the wing group weight of a reusable RLV. Using GOMER's terminology, this MER belongs in group 1, is applicable to a vehicle of type 1, and is obtained from source 3. The following subsections describe in detail the rather lengthy and tedious process that should be used when adding an MER to the database.

6.1 - UPDATING THE *MERS* TABLE

Once the MER's characteristics have been defined, the next step is to add the MER to the *mers* table. In this case, since only one MER will be added to the database, it is not necessary to download a .csv dump of the table data, update the .csv file, and then re-upload the .csv file containing the new MER using the phpMyAdmin tool. For the addition of one MER, that method is far too involved.

Instead, this example will outline how the user will add to the database using the graphical user interface of the phpMyAdmin tool. To begin, the user would access the tool (http://ssdl20.ae.gatech.edu/merdb/phpMyAdmin) and then proceed to log in with the proper username and password (refer to section 4.2). Once logged in, the user will see the welcome page as shown in Figure 4-1. Using the left sidebar, click on the *mers* link to get to the *mers* table attributes page. Once on that page, clicking on the "Browse" link will bring up the following page:

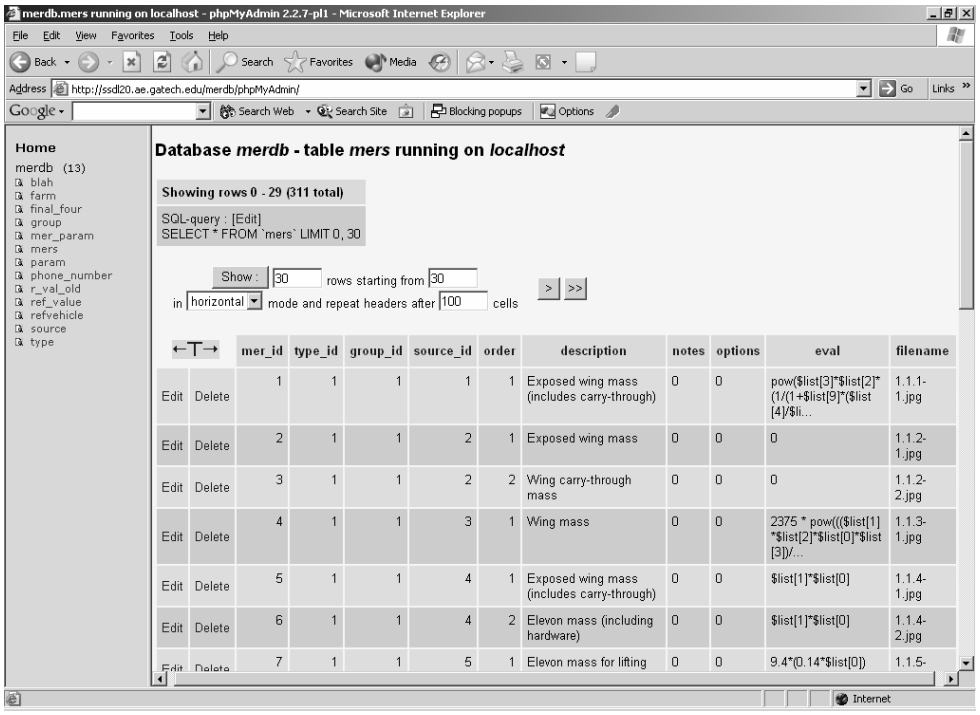


Figure 6-1: Browsing through the data contained in the *mers* table.

Notice that the *mers* table is already filled with entries. Towards the upper left portion of the figure is a small table stating that there are 311 total rows in this table. This is important to note, as that means that there are 311 different MERs in this table, all with their own unique mer_id's (section 5.1.1). One must take care to ensure that when adding MERs, each new MER added has its own separate and unique mer_id. (Remember that the equation used in the example already exists in the database: it is the MER located in row 4 of this table. For the purposes of this example, a duplicate of this MER will be added and assigned to a different mer_id).

The next step is to click on the ">>" button—doing so will lead the user to a page showing the last few rows of the table. Doing so will do two things: 1) ensure the user that there are indeed 311 (or however many) rows in the table, and 2) ensure that the user will use a unique mer_id for the new table entry. Scrolling down to the bottom of the page will reveal the "Insert new row" link. Clicking on this link will bring up the page shown in the following figure.

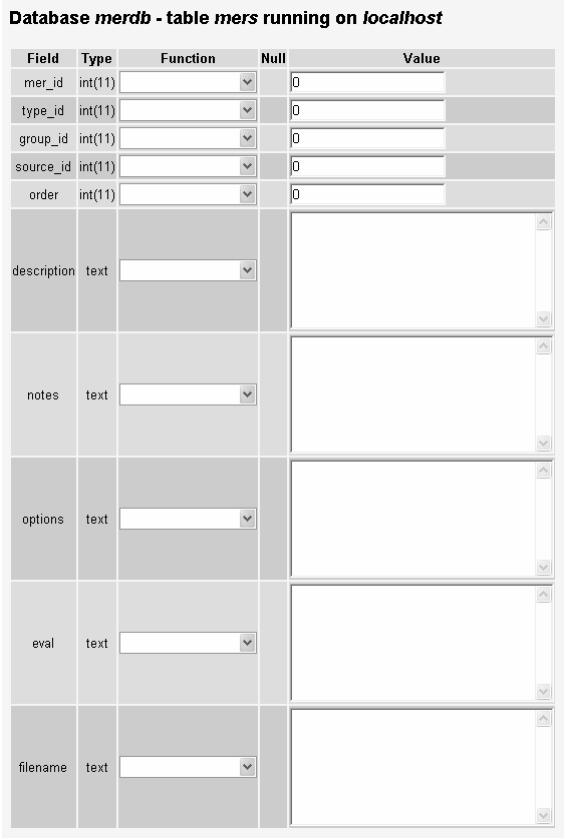


Figure 6-2: The myPhpAdmin form used to insert data into a new row of the *mers* table.

Adding the necessary information is then as easy as typing them in to the appropriate fields. For instance, the unique mer_id for this MER will be given as 312. The type, group, and source id's which have been previously identified for this MER will also be appropriately filled in. Next, a brief description of the MER, as well as any notes and options can be provided. The user should keep in mind that the keywords given to the *browsesearch.php* search script will be matched against the text provided in these fields. The "filename" field should be filled with the appropriate filename for the equation image, as described in section 5.1.1. At this point, all the fields except "eval" should be filled. This last field should be left empty for now—it will be revisited later. Clicking on the "Go" button will then add the data to the table and return the user to the table attributes page.

6.2 - UPDATING THE *MER_PARAM* TABLE

Now that the MER has been added to the database, the next step is to link the equation with the associated parameters. This is accomplished by updating the *mer_param* table. As described before, this table identifies which parameters are associated with which MERs. For the specific example at hand, the objective is to ensure that the following parameters are linked with this equation: b_{str}, M_{entry}, N_z, S_{ref}, and t_{root}. By virtue of convenience, it so happens that these parameters already exist in the *param* table—if this were not the case, then the obvious step is to add the data and other related information to the *param* table, making sure to stick to the convention described in section 5.1.3. With a quick look at the *param* table, one can see that the parameters listed above have the associated param_id's of 7, 15, 18, 26, and 28 for b_{str}, M_{entry}, N_z, S_{ref}, and t_{root}, respectively. Again, the convention to follow is that when adding new data, it must be added to the end of the table. Clicking on the ">>" button will send the user to the last set of data in that table, and then clicking on the "Insert new row" link will ensure that the new information will be appended to the end.

Adding a new row of data to this table is far simpler as there are only two fields to worry about (refer to section 5.1.6 for more detail). For this example, five new rows will be added to the table as there are five parameters in the equation. The new data is then added as follows: for the "mer_id" field in the first row—"312" as this identifies the given example equation, and for the "param_id" field—"7" as this identifies the first parameter listed above, b_{str}. This data is saved to the table by clicking the "Go" button. However, since there will be multiple entries added to this table, it is recommended that the user select the "Insert another new row" radio button before clicking the "Go" button. This option will save the current data to the table and then return the user to the "Insert new row" page (as opposed to sending the user back to the table attributes page). The data entry process continues until all the parameters have been entered into the table. As a final remark, it should be mentioned that the order in which the parameters are entered into this table should be noted, as this will come into play later.

6.3 - UPDATING THE *REF_VALUE* TABLE

The next table to which data must be added is the *ref_value* table (refer to section 5.1.7 for more details on this table). As before, click on the ">>" button to get to the last set of rows, and then click on the "Insert new row" link to get to the right page. Adding data to this table is somewhat similar to adding data to the *mer_param* table in that multiple rows of data will be added. However, some care must be exercised when entering data into the fields. This is best illustrated by looking at the figure below:

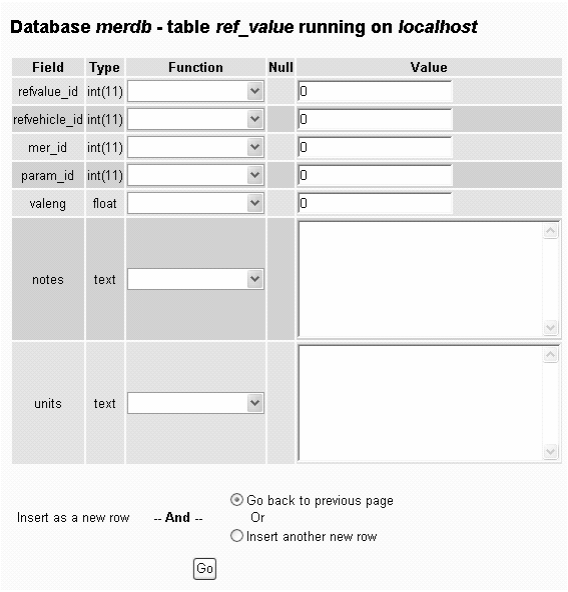


Figure 6-3: The phpMyAdmin form used to insert a new row to the *ref_value* table.

Apart from the "mer_id" and "param_id" fields (which are pretty self-explanatory, and follow the same reasoning as described in the preceding section), the two most important fields in this table are the "refvehicle_id" and "valeng" fields. Of course, last but certainly not least, make sure that there is a unique refvalue_id associated with each new row of data.

As described in section 5.1.7, the data in this table is used by GOMER to populate the parameter fields when a user chooses to view an equation in more detail (see the figure below).

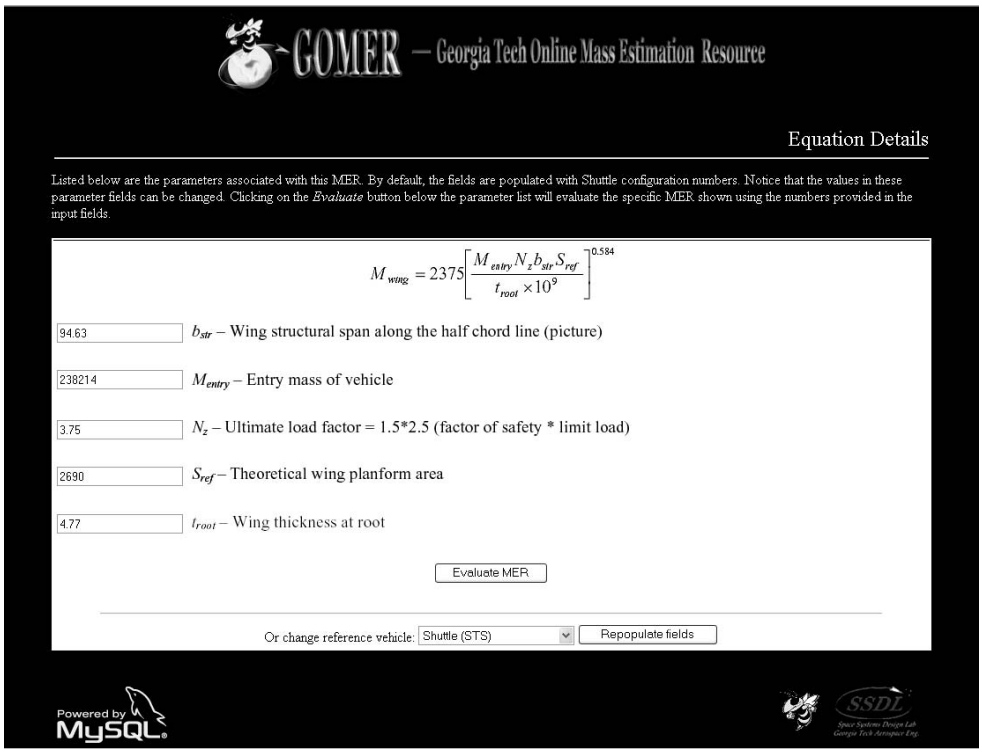


Figure 6-4: Screenshot of GOMER's details window for the example equation used in this section.

The next part is a fairly tricky aspect of the GOMER tool. Observe that in the page shown above, a user has the option to evaluate the given equation using the values listed in the parameter fields; in effect allowing the user to see the result that the MER would provide. The values that are placed by default in the parameter fields are the result of a query performed on the *ref_value* table. *By default*, the user will see the page above with the parameter fields populated with Shuttle numbers (i.e. – parameter values defining the Shuttle configuration) obtained from the data located in the "valeng" fields in the *ref_value* table.

As such, the tricky part here is to identify that these values are associated with the Shuttle; this is accomplished by entering a value of "1" in the "refvehicle_id" field of this table. If the developer wishes to add values associated with another reference vehicle, then the appropriate refvehicle_id would be used in place of "1" (i.e. – for the Vega RLV, a value of "2" would be used). The "valeng" field should then contain the appropriate parameter value (in English units).

6.4 - REVISITING THE *MERS* TABLE

The final step is to add the "actual" equation to the *mers* table so that it may be evaluated when a GOMER user clicks on the "Evaluate" button shown in Fig. 6-4. Going back to the *mers* table attributes page, click on the ">>" button to get to the last row (which was just recently inserted), then click on the "Edit" link. This step fills in the "eval" field that was left blank earlier in section 6.1.

Recall the recommendation that the order of the parameters as entered in the *mer_param* table be noted. This plays a key role as this is the order at which the parameters will be displayed in the details window (Fig. 6-4). Thus, in order to evaluate the equation, it must be ensured that the MER matches the "actual" equation that is evaluated by GOMER. Therefore, the following should be entered in the "eval" field, which really does match up with the example equation given earlier in section 6.0:

```
2375*pow((($list[1]*$list[2]*$list[0]*$list[3])/($list[4]*pow(10,9))),0.584)
```

Notice that the parameters were replaced with $list[x], where x denotes a number. With the way that the GOMER evaluate script is written, each value for each parameter is entered into an array labeled $list. PHP, by default, starts the index for each array at 0, and so therefore for this equation which has 5 parameters, the index runs from 0 to 4. Again, the parameter order matters, so b_{str} is equivalent to $list[0], M_{entry} is equivalent to $list[1], so on and so forth. This is one of the rather time-consuming portions of the process, as it should be made sure that no errors have been made and that the proper syntax is used. (For help on mathematical functions used in PHP, please refer to the PHP homepage, or to any of the references listed at the end of this report).

Completion of this step finishes the process and adds the MER to the database.

7.0 - MAINTENANCE AND UPGRADES

In closing, a few words should be spent on the process of maintaining and future upgrading of GOMER's database. Only routine checking is needed to make sure that the MySQL server is up and running. During the installation of the MySQL package, an application was installed and set up such that MySQL would automatically start up if the server were to ever reboot. If that application somehow fails to restart the MySQL server, a simple command in the OS X shell command line is all that is needed to get the server up and running.

As far as submissions of new MERs to be included in the database, all that needs to be done is to append the appropriate information into each of the tables described in the preceding sections. Updating the data in each of the tables contained within is as simple as downloading the table data into a .csv file, editing the file, and re-uploading the file to the table using phpMyAdmin.

Lastly, to address concerns about updating the MySQL software: upgrading to a new version should not destroy any functionality that is served by GOMER. This is because no real "exotic" commands were used to create GOMER, and as such the features provided by GOMER should be preserved. Additionally, as with any questions or concerns regarding MySQL or PHP, a quick visit and search through their online documentation should provide the needed answers.

8.0 - APPENDIX

8.1 - *KEYSEARCH.PHP* SOURCE CODE

The following is the PHP source code used to perform the keyword search method. Using the first form in the search page, it takes in two arguments: the search terms (as a string), and the search location.

```
<?php
…
      // Run SQL query: search param.varname and output
relevant eqns
      $query = "select m.group_id,
                  m.source_id,
                  m.description,
                  m.mer_id,
                  m.filename,
                  s.label
              from mers m, param p, mer_param mp, source s
              where m.mer_id = mp.mer_id
              and p.param_id = mp.param_id
              and m.source_id = s.source_id
              and match($search) against ('$description')
              group by m.mer_id, m.group_id, m.source_id";

// Open connection to DBMS
if (!($connection = @ mysql_connect("localhost", "merdb",
"merdb")))
      die("Could not connect to the database!");
if (!(mysql_select_db("merdb", $connection)))
      showerror();

// Run the query above
if(!($result = @mysql_query($query, $connection)))
      showerror();

// Find out how many rows in the result
$rowsfound = @mysql_num_rows($result);

// Check for data
if($rowsfound != 0)
{

echo "\n<font color=\"white\">" .
     $rowsfound .
     " records were found matching your query.</font><br><br>";

// Create the output table
echo "\n<table border=\"0\" align=\"center\" width=\"100%\">";

// Process the output data
```

```
while ($row = @ mysql_fetch_array($result))
{
      // Print heading
      echo "\n<tr>\n\t<td bgcolor=\"gray\">" .
           "<b><font color=\"white\">" .
           $row["description"] . " -  " .
           "Group" . " " .
           $row["group_id"] . ", " .
           "Source" . " " .
           $row["label"] .
           "</font></b></td>\n</tr>";

      // Link to parameter listing
      $eqn = $row["mer_id"];
      $name = $row["filename"];
      echo "\n<tr>\n\t<td bgcolor=\"silver\">" .
           "<b><font color=\"blue\">" .
           "<a href=\"paramlist.php?m_id=$eqn&img=$name\" " .
           "style=\"text-decoration:none\" target=\"new\">" .
           "View Details</a>" .
           "</font></b></td>\n</tr>";

      // Show the equation image
      echo "\n<tr>\n\t<td bgcolor=\"white\">" .
           "<img src = \"eqns/" .
           $row["filename"] .
           "\">" .
           "</td>\n</tr>";

      // Blank rows for formatting
      echo "\n<tr>\n\t<td><br></td>\n</tr>";
}

echo "\n</table>\n";

}  // End if rowsfound!= 0
else
{
      echo "<br>No results found matching your criteria.\n";
}

// Create link back to search page
echo "<br><a href=\"search.html\" " .
     "style=\"text-decoration:none\">" .
     "Back to search</a><br>";

if(!(mysql_close($connection)))
      showerror();
…
?>
```

8.2 - *BROWSESEARCH.PHP* SOURCE CODE

The following is the PHP source code used to perform the browse search method. Using the second form in the first page, it takes in three arguments: the vehicle *type*, the weight *group*, and the reference *source* in which to conduct the search query.

```
<?php
…
      // Run SQL query: search param.varname and output
relevant eqns
      $query = "select m.group_id,
                  m.source_id,
                  m.description,
                  m.mer_id,
                  m.filename,
                  s.label
            from mers m, param p, mer_param mp, source s
            where m.mer_id = mp.mer_id
            and p.param_id = mp.param_id
            and m.source_id = s.source_id";

      if ($browsetype != "All")
            $query .= " and m.type_id = $browsetype";
      if ($browsegroup != "All")
            $query .= " and m.group_id = $browsegroup";
      if ($browsesource != "All")
            $query .= " and m.source_id = $browsesource";
      $query .= " group by m.mer_id, m.group_id, m.source_id";

// Open connection to DBMS
if (!($connection = @ mysql_connect("localhost", "merdb",
"merdb")))
      die("Could not connect to the database!");
if (!(mysql_select_db("merdb", $connection)))
      showerror();

// Run the query above
if(!($result = @mysql_query($query, $connection)))
      showerror();

// Find out how many rows in the result
$rowsfound = @ mysql_num_rows($result);

// Check for data
if($rowsfound != 0)
{

echo "\n<font color=\"white\">" .
     $rowsfound .
     " records were found matching your query.</font><br><br>";
```

```
// Create the output table
echo "\n<table border=\"0\" align=\"center\" width=\"100%\">";

// Process the output data
while ($row = @ mysql_fetch_array($result))
{
      // Print heading
      echo "\n<tr>\n\t<td bgcolor=\"gray\">" .
           "<b><font color=\"white\">" .
           $row["description"] . " -  " .
           "Group" . " " .
           $row["group_id"] . ", " .
           "Source" . " " .
           $row["label"] .
           "</font></b></td>\n</tr>";

      // Link to parameter listing
      $eqn = $row["mer_id"];
      $name = $row["filename"];
      echo "\n<tr>\n\t<td bgcolor=\"silver\">" .
           "<b><font color=\"blue\">" .
           "<a href=\"paramlist.php?m_id=$eqn&img=$name\" " .
           "style=\"text-decoration:none\" target=\"new\">" .
           "View Details</a>" .
           "</font></b></td>\n</tr>";

      // Show the equation image
      echo "\n<tr>\n\t<td bgcolor=\"white\">" .
           "<img src = \"eqns/" .
           $row["filename"] .
           "\">" .
           "</td>\n</tr>";

      // Blank rows for formatting
      echo "\n<tr>\n\t<td><br></td>\n</tr>";
}

echo "\n</table>\n";

}  // End if rowsfound!= 0
else
{
      echo "<br>No results found matching your criteria.\n";
}

// Create link back to search page
echo "<br><a href=\"search.html\" " .
     "style=\"text-decoration:none\">" .
     "Back to search</a><br>";

if(!(mysql_close($connection)))
      showerror();
…
?>
```

9.0 - REFERENCES

i. Lane, David, Williams, Hugh E., "Web Database Applications with PHP and MySQL," O'Reilly, CA, 2002

ii. Ullman, Larry, "PHP for the World Wide Web," 2nd Edition, Peachpit Press, CA, 2004

iii. Rohrschneider, Reuben R., "Development of a Mass Estimating Relationship Database for Launch Vehicle Conceptual Design," AE 8900 Paper, Georgia Institute of Technology, April, 2002