DataVerity

Statistical Analysis Enhancement Report

Colorado School of Mines Computer Science Field Session 2008
Data Verity #1

By
Curtis Fleming
Seth Goings

Summer 2008
Presentation: Friday June 20

1. Abstract

DataVerity is a consulting firm that helps banks make decisions on how to increase revenue by studying sales and referral data. The consultants use a web-based system called ESP (Elevated Service Performance) which collects and presents data which is critical to the recommendations DataVerity offers. The consultants would like to improve the quality and confidence of the advice they provide to their clients by adding the ability to investigate relationships between sales decisions and actual sales performance.

We will begin improving the ESP system by designing database procedures that will generate statistical data such as correlations, linear regressions, and confidence intervals. Once this functionality has been added, we will enhance the existing interface to include this information.

2. Introduction

2.1 Project Topic

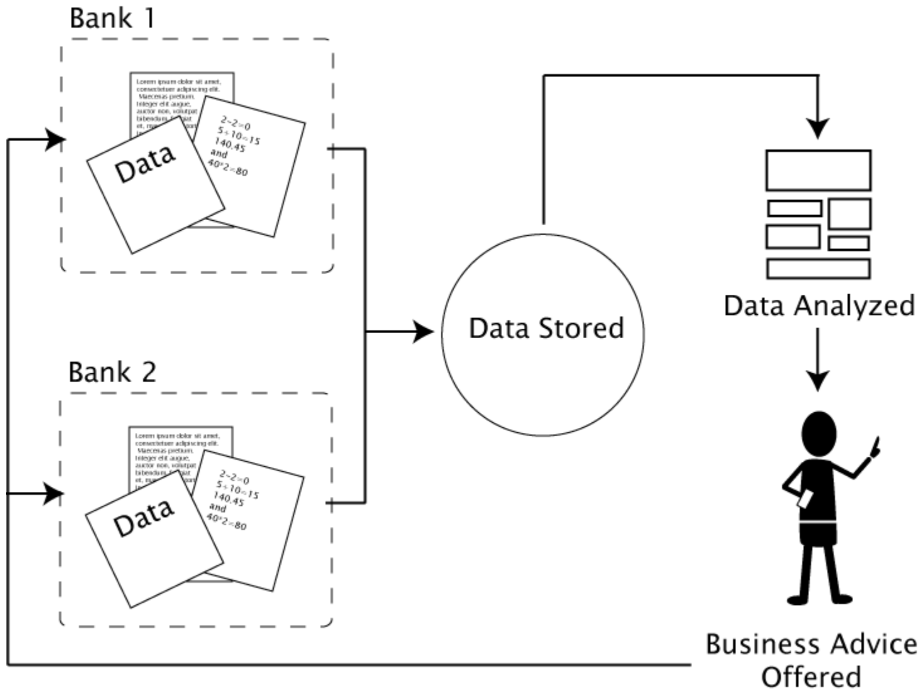
This project requires us to investigate relationships and manipulate data by using statistical calculations and complex algorithms. This project makes heavy use of database programming and some statistical knowledge.

2.2 Client

The client is DataVerity (dataverity.net), a consulting firm for financial institutions. DataVerity aspires to improve the data mining capabilities of their software in order to offer more useful advice to their clients.

For each client DataVerity serves, data is collected, manipulated, analyzed, and then placed into reports for the DataVerity consultants to use when offering business suggestions to the client. (See Figure 1)

Figure 1: DataVerity Operations



3. Functional Project Requirements

The following tasks are required for this field session project.

- Recode several existing procedures for optimization and familiarization with the code structure.
- Investigate relationships between data using the following statistical methods:
    - Correlation Coefficients
    - Best-Fit Lines / Linear Regressions
    - Error / Confidence Intervals
- Create functionality to show (in percentage form) the connectivity between variables, such as referrals and sales. In this referrals and sales example, the client requests that the percentage be defined as: *"What percentage of customers who opened a certain number of account type <variable> were referred a certain number of times to account type <variable>?"* This functionality will be called the "cross-tabbing" procedure from this point.
- Extend the ESP (Elevated Service Performance) web system to use these calculations.
    - The new statistical calculations will produce a line of best fit and confidence intervals through a scatter plot of the selected dataset. (See Figure 2.0)
    - The cross-tabbing functionality will be expressed as a bar chart. Using the above cross-tabbing example, the account type which was sold will be on the independent axis, with a percentage of referrals stacked vertically. (See Figure 2.1 Version 1) There is also a second version of chart which uses a similar bar graph style, but instead of stacking the "referred-to" account percentages, they are distributed along the independent axis from tallest to shortest. (See Figure 2.1 Version 2)

Figure 2.0: Scatter Plot + Linear Regression + Confidence Intervals

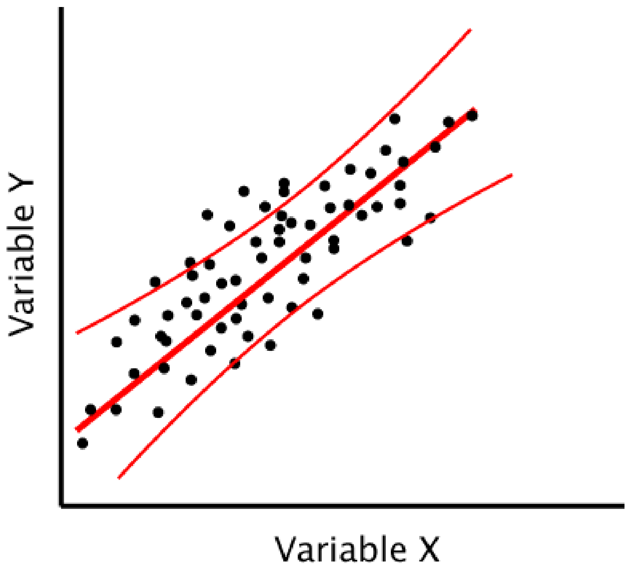
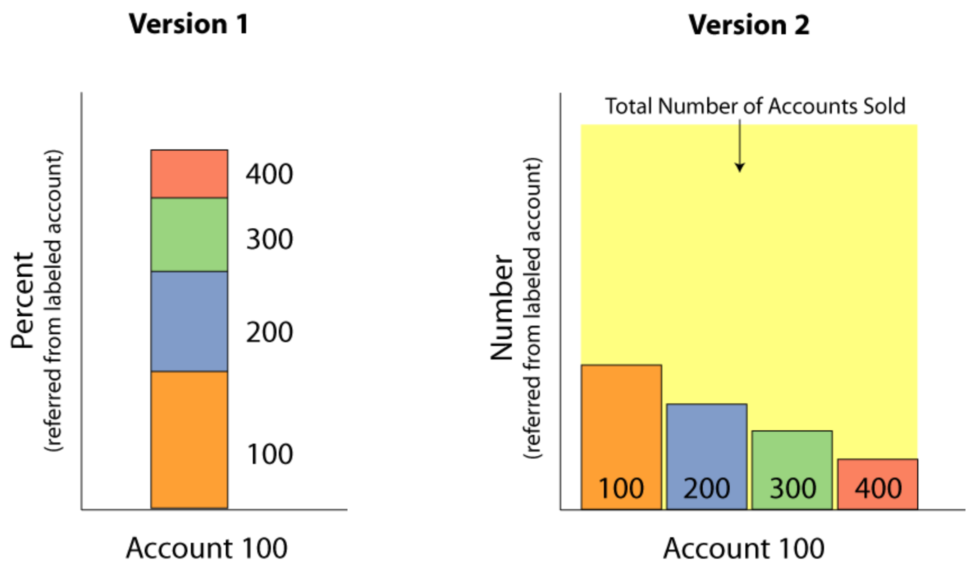


Figure 2.1: Two Versions of the Cross-Tab Graphs



4. Deliverables

We will deliver code which performs the functional requirements as previously defined along with ample documentation which will aid in maintenance. The code deliverables include:

- improved/recoded existing procedures (.sql)
- statistical procedures (.sql)
- cross-tabbing procedure (.sql)
- the code and files needed to update DataVerity's ESP interface to show the output of the procedures listed above (.inc.php and .php)

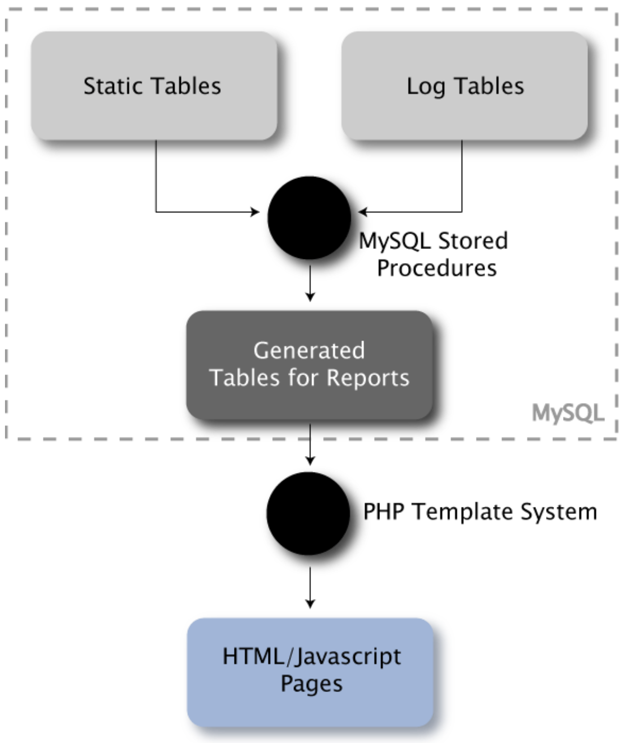
5. Non-functional Project Requirements

- Program the statistical calculations required and cross-tabbing functionality as MySQL stored procedures; MySQL stored procedures were chosen to maintain code portability and speed.
- Use PHP when modifying the ESP system for uniformity.

6. Scope

The main scope of this project is our client's data processing system. The code base of DataVerity is split into two main groups: the ESP web-based system and the database management. This project will first focus on adding database functionality and then on enhancing the ESP interface with these new database improvements. In Figure 3, the two areas of improvement are the black circles.

Figure 3: Existing ESP structure



7. System Design and Implementation

7.A Statistics Sub Project

7.A.1 Design of Statistical Calculations

Our task is to calculate the least squares linear regression line, correlation coefficient, and standard deviation of two data columns. The client requested that we design our database-heavy work as stored procedures in order to increase efficiency and keep execution times to a minimum. Our approach to solving this problem is to break down the mathematical equations into operations and find appropriate MySQL commands to implement. See appendix A for the implemented equations.

7.A.2 Design of the *Statistical Procedure*

The core procedure is called "proc-statistics." It will act on a single pair of columns. It will output six values: the regression slope, regression intercept, correlation coefficient, standard error of the slope, standard error of the intercept, and standard deviation. There are three inputs, strings of the two column names for comparison and a string specifying the data source.

In the procedure two of the inputs, xFunction and yFunction, are strings of MySQL formatted functions that operate on columns. This is more powerful than simply passing in only the two column names. This allows a caller of the procedure to transform non-linear data into linear data.

The third and last input is a string specifying the data source. This can simply be the name of the table like "myTableName." The string can also be a valid SQL sub-query giving great flexibility with what data is actually used. This means the WHERE clause can be used in this sub-query to filter out unnecessary rows of data.

Originally the design called for the ability to specify what the slope or intercept should be and then continue on with the calculations based on this assignment. This purposed feature will not be implemented at this time because the mathematical calculations are too involved.

Example call:
```
CALL statistics (“employeesColumn”, “sqrt(profitColumn)”,
“bankingTable”, @slope, @intercept, @correlationCoefficient,
@slopeError, @interceptError, @standardDeviation);
```

7.A.3 Design of the *Mean Procedure*

The mean procedure takes many columns in and create combinations of the columns to compare and then output the statistical values to a table. This procedure, “proc-mean”, relies on the previous proc-statistics procedure. There are five inputs: a list if columns to act on, a primary delimiter for the list (mainDelimiter), a secondary delimiter (pairDelimiter), the data source, and the output table name. All of these inputs are strings.

The list (called itemList) can be in one of two formats, which will be automatically detected:

1) itemList doesn't contain a substring of pairDelimiter. Every substring delimited by mainDelimiter will be considered elements of a set such that every possible ordered pair will be made and sent to the statistics calculator.
2) itemList does have at least one substring of pairDelimiter. The string will be parsed based on the primary and then those substrings will be split on the single pairDelimiter within each giving ordered pairs, (independent, dependent)

The actual elements of itemList can be SQL functions that act on the desired columns.
For each pair, the six statistical values will be calculated by proc-statistics and saved as a row to the output table. The output table has nine columns, two for the pair of columns being compared, the data source, and the six statistical values.

Example call:
```
CALL mean ("numAccounts-_-totAmount-_-numCustomers",
"-_-", NULL, "(SELECT * FROM salesTable WHERE
accountID LIKE '2_a_%') AS dummyAlias",
"statsOutputTable");
```

7.B Cross Tabbing Sub Project

7.B.1 Design of the *Cross-Tabbing Procedure*

Before diving into the details of the cross-tabbing MySQL stored procedure, a restatement of the functionality is needed. The general form of the client-defined cross-tabbing functionality is: (where <text> stands for a variable)
“What percentage of <commonFactors> did <subject> <certain number of times> and also did <object> <certain number of times>?”

A more specific, easier to understand and implement question derived from the abstract question posed above is:

“What percentage of customers who opened an account of any type at least once were referred at least once to another (or the same) account type?”

A small amount of unsuccessful research was conducted to try to find existing code that served the client’s purpose. This procedure had to be created from scratch, especially with the client’s original data structure. To create the cross tabbing stored procedure, a very complex sequence of queries were created around the specific example. Essentially, two tables (the sales and referral tables) were joined with advanced criteria to create the output desired. Once the specific example output was determined to be what the client had requested and the data was checked for errors, the queries were generalized. This complied with the client’s original demand to answer the abstract question presented above which allowed for more flexibility in the execution of the procedure. For more information on the design of this procedure, please refer to the Programmer’s Manual.

7.B.2 Results from the *Cross-Tabbing Procedure*

Even though the cross tab procedure contains multiple joins, temporary and permanent table creations, as well as several GROUP BY statements, the procedure is much faster than we originally predicted. In a trial run, the procedure operated on approximately 20,000 rows between two tables, creating the output desired (about 300 rows) in 1.1 seconds. The longest encountered run time for the cross tabbing procedure was a little less than 5 seconds. This run time happened when the procedure operated on one table and performed various types of joins normally used on two separate tables. A series of self joins would be faster in this case, but since it is a rare occurrence and the procedure still ran relatively quickly, the design of the procedure was not altered.

The indexes that are added dynamically during the cross tab procedure help a lot when tables are joined. Without indexes, the procedure took about 50 seconds to create the same output. Combining multiple SELECT, GROUP BY, and WHERE statements into sub queries also helped keep runtime to a minimum since sub queries create tables in memory, not on the hard drive.

7.C Graphs Sub Project

7.C.1 Graph Creation Overview

After generating data using the new stored procedures, work on the project shifted to represent this data graphically. Recall from Section 3: the client's requested a combined scatter plot/linear regression/confidence interval graph for the statistical calculations and the two styles of bar graphs for the cross tabbing procedure.

Creating graphs can be difficult task, especially when dealing with graphs on web pages. Thankfully, the client already had the tools to implement graphs into the ESP reports using PHP and JPGraph (a graphing library for PHP). After learning a bit about JPGraph, we began to create static versions of each style of graph the client requested. By using static datasets, we were able to confirm that our graphs were operating properly. The implementation of these graphs is unique enough for further explanation.

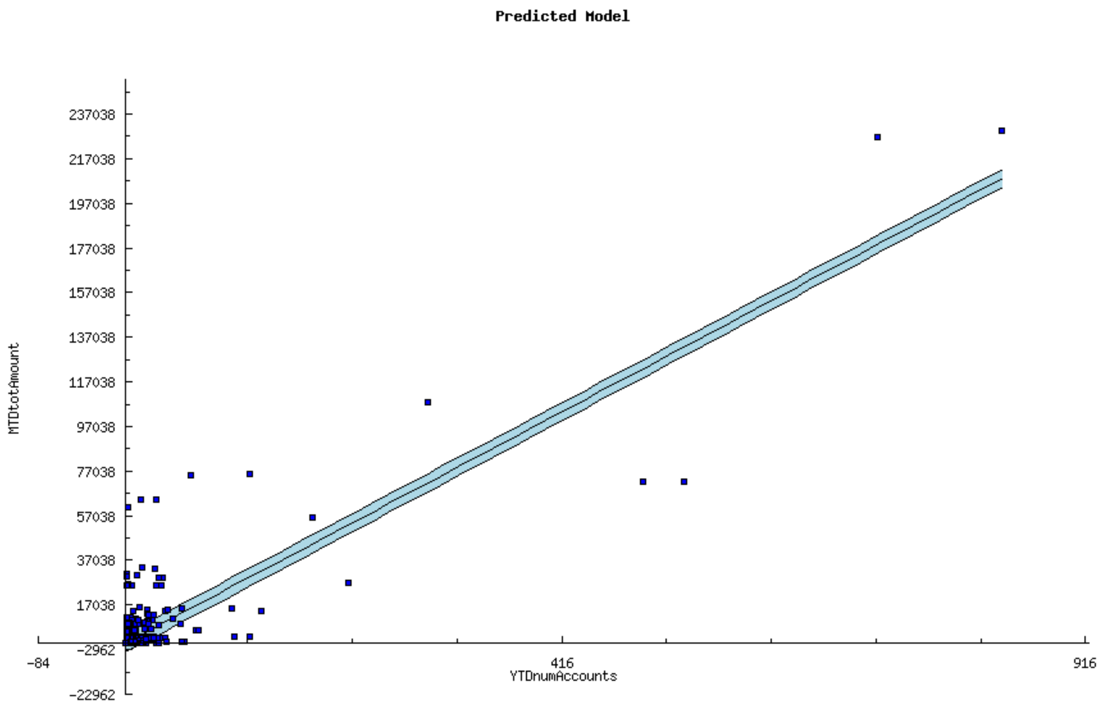
7.C.2 Statistics/Mean Graph

There are two parts to the statistics graph script regressionGraph.inc.php, a scatter plot and best fit lines. There are three lines and a shaded region between them to be more visually appealing. The center line is the calculated linear regression line of best fit. The two surrounding lines indicate a certain spread of data about the line. The more data points within the shaded region and the narrower the region the better the fit. A good fit may indicate the two data sets being compared are related in some fashion. The script regressionGraph.inc.php queries a database to grab the regression line data.

The scatter plot is simply all of the (x,y) data points being compared. When the script queries the database for the regression data it also grabs a stored query string specifying the raw data's location. After executing the second query the resulting data is graphed as the scatter plot.

Figure 4.0 is example output from the script. It has the raw data plotted along with the regression line. This graph depicts data clustered around lower values and so there is a lack of linearity even though the outlying points make a general linear trend.

Figure 4.0: Statistical Graph



7.C.3 Cross-Tabbing Graphs

Though there were two graphs requested by the client to represent data from the cross tabbing stored procedure, these two graphs had a similar backbone: the array manipulation function.

7.C.3.1 Array Manipulation Function

When the cross tabbing stored procedure is run, it creates a table. This table is essentially dumped into an array by existing code the client supplied, but these arrays were still not in the correct format that JPGraph needed in order to create a bar chart. The client's code that created PHP arrays from MySQL tables created three arrays for us to deal with. From these three arrays, the array manipulation function creates a 2 dimensional array with:

- Array 1 as the column names,
- Array 2 as the row names,
- Array 3 values as the cell data.

Then the array manipulation function takes the 2D array and loops through it to create an array for each row of data. Each entry in this new array will define the height (Array 3) of the unique account label (Array 2) and color associated with each x axis label section (Array 1), as shown in Figure 4.2. These row arrays are pushed into JPGraph's graph generating functions, and with a little style, a visual representation of the cross tabbing stored procedure is created!

7.C.3.2 Graph Formatting and Extra Features

The "Grouped Bar Plot" is one of the styles requested by the client (Figure 4.1). The second style, the "Accumulated Bar Plot," (Figure 4.2) is a different style of a "Grouped Bar Plot" so just a few changes were required to create one graph style from the other. However, in the Accumulated Bar Plot style, extra work was needed to create the *referred Account ID* labels on top of each bar section. JPGraph did not have the ability to create labels on the individual bar sections, so some fairly complex code was added to enable this functionality. These labels, though tricky to create, make the graph a whole lot more understandable.

Figure 4.1 – Grouped Bar Plot

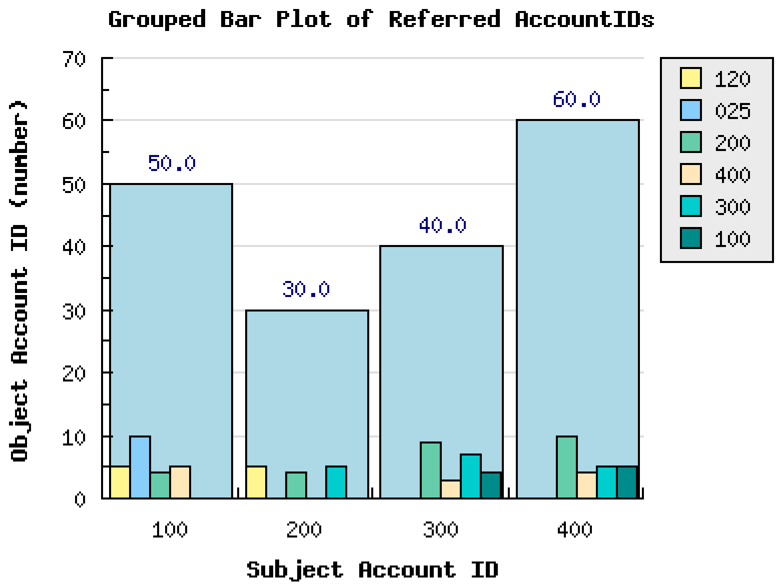
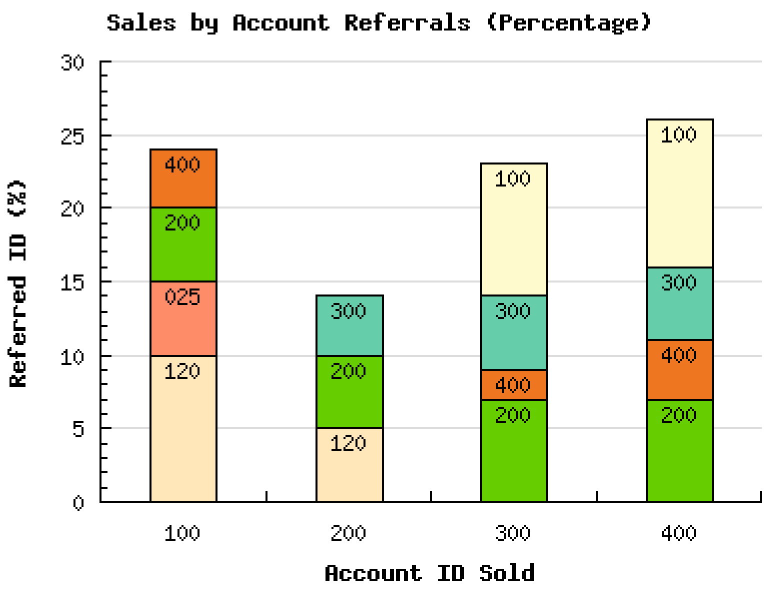


Figure 4.2 – Accumulated Bar Plot



A feature that was implemented into these two graphs that cannot be shown in the report figures is the embedded image map. This allows the user to hover the mouse pointer over each bar and see its height and account number values. The image map makes the graph less cluttered (instead of having a huge legend or bulky labels) and also has the added effect of making the graphs semi-interactive.

8. Design Issues

8.1 Design Issue: MySQL Stored Procedure Limitations

The stored procedure was introduced March 2005. Stored procedures are relatively new and are not feature rich. There are numerous misconceptions of its capabilities especially when compared to standard languages like Java. A stored procedure is a routine saved in its own file and expands the SQL language by using flow control and the opportunity to reuse SQL code.
MySQL does have another routine called a function. Functions have very limited capabilities because they cannot dynamically access tables. Functions are best used for numeric calculations rather than involved table access.

8.2 Design Issue: MySQL Variables

The introduction of stored procedures presents a problem involving variable collisions. MySQL has always implemented "session variables"; they are undeclared variables that exist for the duration of your connection with the server. Stored procedures can use session variables like normal however the variables are global and persist after the procedure call. This means a called procedure will overwrite the caller's session variables of the same name because both procedures have the same variable scope.

A new type of variable was introduced to combat this scope problem. Declared variables exist only during the procedure's statement block. However, MySQL declared variables cannot replace and be used like session variables in every case. The poor implementation of the new variable type forces use to use session variables, creating an unavoidable opportunity for our variables to collide with similarly named variables in other stored procedures. This issue is ongoing and a constant concern for stored procedure maintenance.

8.3 Design Issue: No SELECT . . . INTO changes in the near future

We requested an additional feature to MySQL: using declared variables in a SELECT . . . INTO statement instead of just session variables, but the MySQL team said that they are not willing to add this functionality.

9. Obstacles Overcome

MySQL is designed to be a database/query language, not a programming language. This discrepancy makes programming stored procedures a little different than in your normal procedural programming language. Unlike most procedural languages, MySQL stored procedure code contains heavy amounts of string concatenations. The reason for so many string concatenations is to combine variables with static text in the creation of dynamic queries. Additionally the programmer must also realize that a query doesn't return any values. To access, set, and manipulate values while programming a MySQL stored procedure, one must create a table by query and then act on that table by executing other queries. In short, MySQL doesn't allow direct interaction with individual table cells, so stored procedures require a unique style of programming.

GROUP BY statements do not use indexes, so it is important to group rows when the table size is as small as possible. There were also some cases where JOINS could be used instead of multiple GROUP BY statements, which increased speed and efficiency greatly.

It is also important to note that GROUP BY statements are a little unpredictable. Consider two rows that are about to be grouped by accountID. If there is a column that holds the time that the row was created, which timestamp will MySQL retain when the two rows are combined to one? In the crosstabbing procedure, extra SELECT criteria was imposed before rows were grouped, so that the GROUP BY statements always yielded consistent but adjustable results.

10. Future Work

The cross tabbing stored procedure's functionality is general enough that just by altering its parameters, the client is able to manipulate the detailed operations almost entirely. In the future however, the client may choose to add more parameters to the stored procedure to allow even more flexibility. Adding parameters to a procedure of this size could be difficult, but the code is well documented and organized to make these future changes a bit easier.

The statistics procedures have not been optimized. A call to the mean program that found the combinations of 14 columns of a few thousand rows took 40 seconds. Besides not using indexes, one sub-query is executed twice. If an intermediate structure is introduced, the program may speed up. Give how infrequent the program is run and operating in fractions of a minute this optimization may be unnecessary.

Another consideration would be to output intermediately calculated statistical values such as the means of the columns.

11. Conclusion

We have produced nine code files that generate statistical data between data columns, find the cross relations between data columns, and graphical representations of the output data. The stored procedures have been successfully run on several data sets from the client and the output stored in their database. Our php graphing scripts create informative depictions of our calculated data and allow for quick analysis of the data.

12. Glossary

- **Bash Files** - A script that runs on Linux systems for fundamental operating system features. We used this to rapidly compile our MySQL scripts in the database.
- **ESP** (**E**levated **S**ervice **P**erformance) - ESP is the name for the website used by DataVerity. It can view and manipulate client data and provides a view for data reports among other features. It is coded in PHP.
- **Linux**- An open source computer operating system that performs well on servers.
- **MySQL** (**S**tructured **Q**uery **L**anguage) - Is a multithreaded, multi-user SQL database management system; it has its own programming language. A *stored procedure* is a block of code that can be considered a subroutine to be executed at a later time.
- **PHP**- (Hypertext Pre-Processor) is a computer scripting language, originally designed for producing dynamic web pages. It is for server-side scripting, but can be used from a command line interface or in standalone graphical applications.
- **SVN** (**S**ub**v**ersio**n**) - Is a control system to manage versions of code and files.
- **Query**- Given two computers or even more general two programs, a query involves one program 'asking' the other to perform a task, namely a request for data.

Appendix A: Mathematical Equations Describing Linear Regression

Table 1 shows the equations used in proc-statistics to calculate the six outputs. The standard deviation equation is not shown because it's a built-in MySQL function. The table mentions x list and y list, which are the two passed in columns. The equations were found at the website http://mathworld.wolfram.com/LeastSquaresFitting.html

Table 1: Mathematics of Linear Regression and Correlation

$\mathbf{x_i}$	A datum of the list x
$\mathbf{y_i}$	A datum of the list y
$\mathbf{X}$	The mean of list x
$\mathbf{Y}$	The mean of list y
$s_{xx} = \sum_{i=1}^{n} (x_i - X)^2$	An intermediate value which describes how far the x values are from the mean of list x. The list has length n.
$s_{yy} = \sum_{i=1}^{n} (y_i - Y)^2$	An intermediate value which describes how far the y values are from the mean of list y. The list has length n.
$s_{xy} = \sum_{i=1}^{n} (x_i - X) \cdot (y_i - Y)$	This is the covariance of x and y.
$m = \frac{s_{xy}}{s_{xx}}$	The slope of the least squares regression line
$b = Y - m \cdot X$	The intercept of the least squares regression line
$r = \frac{s_{xy}}{\sqrt{s_{xx} \cdot s_{yy}}}$	The correlation coefficient of the regression line. Its range is [-1, 1].
$s = \sqrt{\frac{s_{yy} - (m \cdot s_{xy})}{n - 2}}$	Intermediate value used for the following standard error calculations
$m_{SE} = \frac{s}{\sqrt{s_{xx}}}$	Standard Error of the regression line slope.
$b_{SE} = s \cdot \sqrt{\frac{1}{n} + \frac{X^2}{s_{xx}}}$	Standard Error of the regression line's intercept.

Appendix B: Files Produced

csm-proc-crossRef.sql – optimized existing procedure that is a simpler version of the added crossTab stored procedure
func-listCount.sql – new procedure that returns the number of items in a list.
csm-proc-crossTab.sql – new procedure that answers a question like: "What percentage of customers who opened an account of any type at least once were referred at least once to another (or the same) account type?"
crossTabAccumulated.inc.php – graphing script that creates the accumulated bar chart style of graph
crossTabGrouped.inc.php – graphing script that creates the grouped bar chart style of graph
proc-mean.sql – procedure that runs the statistics procedure a number of times and creates a table with all of the statistical values
proc-statistics.sql – procedure that calculates statistical information such as linear regression coefficients, error, and more from two columns of data
regressionGraph.inc.php – graphing script that combines a scatter plot, best fit line, and confidence intervals into a single graph.
arrayFormatter.inc.php - includes several functions to aid in the manipulation of 3 arrays into one 2 dimensional array