Call Center Forecasting Tools

by

Zachary D. Snider

Submitted to
the Faculty of the School of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

© Copyright Zachary D. Snider

The author grants to the School of Information Technology permission
to reproduce and distribute copies of this document in whole or in part.

Zachary D. Snider

Mark Stockman, Faculty Advisor

University of Cincinnati
College of
Education, Criminal Justice, and Human Services

April 2015
# Table of Contents

Abstract .................................................................................................................. 1
Introduction ........................................................................................................... 2
   Problem ............................................................................................................. 2
   Solution ............................................................................................................ 3
   Overview .......................................................................................................... 4
Discussion ............................................................................................................. 4
   Project Concept ................................................................................................. 4
   Design Objectives ............................................................................................ 5
   Technical Approach ......................................................................................... 6
   Security ........................................................................................................... 7
   Budget .............................................................................................................. 7
   Gantt Chart ...................................................................................................... 8
   Problems Encountered ..................................................................................... 8
Conclusion ........................................................................................................... 9
Works Cited ......................................................................................................... 10
Appendix A - Import Call Data Script .............................................................. 11
Appendix B - Import Agent Data Script ............................................................ 14
Appendix C - Export Call Data Script ............................................................... 16
Appendix D - Export Agent Data Script ............................................................ 19
Abstract

Accurate forecasting and data tracking can be a serious issue for call centers around the world. Having the correct tools will allow a call center to be able to predict how a day will go based on a number of factors: staffing planned, actual staffing, inbound calls offered, outbound calls made, email requests handled, etc. This is accomplished in three steps. First a MySQL database is created to store all of the necessary data long term. Second Dbeaver, an open source database front end, is used to make daily input of data easier for non-technical based users. Finally scripts are created for the input and export of data that will allow users to make reports.
Introduction

Problem

The leadership team of a medium to large scale call center needs to be able to conduct daily business efficiently and in a timely manner. In order to be more efficient and cost effective to the company as a whole, the leadership team needs to be able to predict, or forecast, how a shift will go. This is determined by past data: data from the same week of the previous year(s) and the average company growth, the average percentage at which the company grows year over year. By having this capability the members of the leadership team will then be able to properly staff each shift to meet maximum efficiency. As a result the company does not over spend on payroll such as unnecessary overtime to complete shift duties that could have been avoided if the proper staffing model was used.

The leadership team will also need to be able to pull out past employee performance data. This is used to monitor the employee’s performance and to make sure that each employee is meeting expectations. The performance of the agents is directly correlated with the shift service level agreements. The leadership team will also need to be able to pull months’ worth of past data to see trends for each employee. For example, this data trend might be used to calculate if an employee has met certain performance criteria over the year.

A problem that the leadership team will encounter while attempting to utilize all of the past data is that the software tool that manages the call center currently only holds a limited amount of data. This is done in order to minimize the amount of resources that
the tool utilizes. For example, only the past 2 months’ worth of data is able to be queried and anything past that time frame will return a null value.

**Solution**

The tools that will satisfy these needs while being extremely cost effective, compared to big company solutions, are: a database, a database front end, and scripts. The database should be an open source database, such as MySQL, to store all of the data necessary. The front end should also be open source, such as DBeaver, and will be used by the leadership team to import/export data and manage the database itself. Scripting will be created in order to automate frequent processes and limit human error when importing/exporting data. Both products mentioned above are open source, and therefore are both free. Both products support a wide range of server operating systems: Windows, Mac OS X, Ubuntu Linux, Debian Linux, Redhat Linux, etc. Both products also have user management, so that only the users permitted to use the system can, and is easily configured and scaled. DBeaver is not limited to just being used with MySQL either. It supports a wide range of databases including SQL Lite, Microsoft SQL Server, PostgreSQL, IBM DB2, Oracle, etc.

Satisfying these needs will allow leadership teams in call centers to be able to conduct daily business efficiently and in a timely manner. This is accomplished, while also being cost effective and continuing to perform at maximum efficiency, by giving the leadership team the ability to forecast how a shift will go, giving them the data necessary to staff all shifts accordingly, and being able to make sure all employees are performing at maximum efficiency.
Overview

The remainder of this final report outlines in detail how the project was completed. The report includes the following sections: Project Concept, Design Objectives, Technical Approach, Budget, Gantt Chart, Problems Encountered, and Recommendations.

Discussion

Project Concept

The concept of this project is to create a long term storage solution for a call center’s data, agent and call statistics, and allow members of the leadership team to import data, export data, and edit the database as needed. Figure 1 depicts the use case scenario.

Figure 1 Use Case Diagram
The leadership team will authenticate into the database, create queries, add/configure tables, input data, and execute queries. The administrator user will authenticate into the database, create queries, add/configure tables, create users, delete tables, and grant permissions to the database.

This problem was expressed to me by a friend who is a supervisor at a call center. He expressed how it was difficult to forecast how a shift would go because he didn’t have all of the data that he wanted to use since the tool his company uses only stores 2 months of data at a time.

The reason he wanted to go with a custom solution rather than a paid solution was based on the customization ability as well as price. There are multiple companies that offer this type of a solution to customers: Avaya, NetLert, VPI, Inova Solutions, etc. The problem with this is that the implementation alone is going to be extremely costly, not to mention the training and support that will be needed. The cost of implementation from one company was $10,000 per data source, and didn’t include importing of any past data not currently stored on the call servers.

**Design Objectives**

The initial goals of this project were as follows:

- Create a database for long term storage of data
- Use a user friendly database front end to easily manage the database
- Create scripts to automate the input and export of data to minimalize human error

All of these goals were completed in full by the end of the project.
Technical Approach

To complete this project I created a Windows based MySQL database server with two separate databases. One database was for agent data and had separate tables for each agent. The second database was for call statistics and had separate tables for each statistics that needed to be tracked. **Figure 2** shows the database architecture.

The server is then accessed by two client machines to simulate members of the leadership team accessing the server. All of this was done in the University of Cincinnati College of Education, Criminal Justice, and Human Services Sandbox, a virtualization platform accessible to students currently enrolled in the School of Information Technology. **Figure 3** depicts the network used in the sandbox.
Security

A big concern for many companies is security. How will this data stay secured?

Each leadership user will be granted the minimum amount of permissions, by the administrative user, necessary to do their job. This server will also be on an internal network. In order to access this remotely you will need a VPN client and domain credentials. Only the administrative user will be able to directly access the machine that the MySQL server is hosted on.

Budget

The budget for this project was free and only cost time; 20 hours per week for 20 weeks. In the real world setting, this type of task would most likely be done by a database administrator, and according to the Bureau of Labor Statistics the estimated average wage for a Database Administrator was $38.82 as of May 2013. If this were to be done for a business that had a server available (hardware, storage, and operating system) to install the database and the front end on, the cost to the company would just be for the labor. If
there is not a server available, the cost to the company would be a total of the cost of labor and the cost of the server. **Table 1** below breaks down the budget.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Labor</th>
<th>Server</th>
<th>Applications</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>This Project</td>
<td>$0</td>
<td>$0</td>
<td>$0</td>
<td>$0</td>
</tr>
<tr>
<td>Business w/</td>
<td>$15,528</td>
<td>$0</td>
<td>$0</td>
<td>$15,528</td>
</tr>
<tr>
<td>Server</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 1 Budget**

**Gantt Chart**

**Figure 4** shows the time line for this project.

![Gantt Chart](image)

**Problems Encountered**

The install of Dbeaver only works if you download it directly to that machine. If you try to use an install package that is stored on a network share, Dbeaver will not install
correctly. Also, Dbeaver is pretty picky when it comes to Java. Java Development Kit (JDK) or Java Runtime Environment (JRE) is required to be installed for Dbeaver to function properly. The only JDK or JRE I was able to utilize that actually let Dbeaver to properly run was Java 6 update 45.

**Conclusion**

This project strengthened my database skills as well as my scripting skills. While being primarily based in networking and systems management, it was good to broaden my horizons and work with SQL and Perl coding; two aspects that I am not really versed in.
Works Cited


<http://search.cpan.org/~timb/DBI/DBI.pm>.


<http://search.cpan.org/~makamaka/Text-CSV-1.33/lib/Text/CSV.pm>.
Appendix A - Dbeaver Application View

Appendix B - Perl Script Interface View
Appendix C - Import Call Data Script

use strict;
use warnings;
use Text::CSV;
use DBI;

#Variable declarations
#Database Variables
my $host = "10.71.64.220";
my $db = "calldatadlb";
my $uname = "";
my $pwd = "";
my $table = "";
my $query = "";

#Data Variables
my $file = "";
my $date = "";
my $values;
my @cols;
my @colnames = ();
my $colnum;
my $names = "";

#Establish a connection to the database
#First get the user name
print "Please connect to the database.\nUsername:\t";
chomp($uname=<>);

#Next get the password
print "Password:\t";
chomp($pwd=<>);

#Connect to the database using the user name and password
my $dsn = "DBI:mysql:database=$db;host=$host";
my $dbh = DBI->connect($dsn, $uname, $pwd, {RaiseError=>1}) or die "Couldn't
connect to database: " . DBI->errstr;

#Get file name from user to import data
print "What file do you want to upload data from?\nfile:\t";
chomp($file=<>);

#Get date from user that this data needs to be stored for
print "What date is the data from (YYYY-MM-DD)\ndate:\t";
chomp($date=<>);

#Print "Your file name is $file and the data is from $date."

#Open the csv file to get the data to upload
open my $fh, "<", $file or die "$file: $!";

my $csv = Text::CSV->new ( { binary => 1, auto_diag => 1 });

#Skip the line in the CSV file that contains the header names (1st line of CSV)
$csv->getline ($fh);

#Get the Data to insert into the database
while (my $row = $csv->getline ($fh))
{
    #Initialize/reset the column number variable
    $colnum = 0;

    #Initialize the array that holds the data values
    @cols = ();

    #Initialize/reset the string used to print out the array of data values
    $values = "";

    #Get the table name out of the first column of the CSV file
    $table = "$row->[0]";

    #Populate the column array with the values from the other columns of the
    #CSV file skipping the first column of the CSV file
    for(my $i = 1; $i <= 64; $i++) {
        $cols[$colnum] = "$row->[$i]";
        $colnum++;
    }

    #Set the value of the string variable used to print out the array of data values
    $values = join( ',', @cols);

    #Populate the SQL query string
    $query = "INSERT INTO $table VALUES('$date', $values)"

    #Print "Query: $query"

    #Prepare a SQL statement to input data
    my $sth = $dbh->prepare ($query) or die "prepare: " . $dbh->errstr;
    $sth->execute;
}

close $fh;
$dbh->disconnect;

Appendix D - Import Agent Data Script

use strict;
use warnings;
use Text::CSV;
use DBI;

#Variable declarations
#Database Variables
my $host = "10.71.64.220";
my $db = "agentcalldata";
my $uname = "";
my $pwd = "";
my $table = "";
my $query = "";

#Data Variables
my $file = "";
my $date = "";
my $values;
my @cols;
my @colnames = ();
my $colnum;
my $names = "";

#Establish a connection to the database
#First get the user name
print "Please connect to the database.\nUsername:\t";
chomp($uname=<>);

#Next get the password
print "Password:\t";
chomp($pwd=<>);

#Connect to the database using the user name and password
my $dsn = "DBI:mysql:database=$db;host=$host";
my $dbh = DBI->connect($dsn, $uname, $pwd, {RaiseError=>1}) or die "Couldn't connect to database: ". DBI->errstr;

#Get file name from user to import data
print "What file do you want to upload data from?\t";
chomp($file=<>);

#Get date from user that this data needs to be stored for
print "What date is the data from (YYYY-MM-DD)\n";
chomp($date=<>);

#print "Your file name is $file and the data is from $date.\n"

#Open the csv file to get the data to upload
open my $fh, "<", $file or die "$file: $!
";

my $csv = Text::CSV->new ({binary  => 1, auto_diag => 1});

#skip the line in the CSV file that contains the header names (1st line of CSV)
$csv->getline ($fh);

#Get the Data to insert into the database
while (my $row = $csv->getline ($fh)) {
    #Initialize/reset the column number variable
    $colnum = 0;

    #Initialize the array that holds the data values
    @cols = ();

    #Initialize/reset the string used to print out the array of data values
    $values = ""

    #Get the table name out of the first column of the CSV file
    $table = "$row->[0]"

    #Populate the column array with the values from the other columns of the
    #CSV file skipping the first column of the CSV file
    for(my $i = 1; $i <= 9; $i++) {
        $cols[$colnum] = "$row->[i]
        $colnum++;    
    }

    #Set the value of the string variable used to print out the array of data values
    $values = join(", ", @cols);

    #Populate the SQL query string
    $query = "INSERT INTO `Table` VALUES('$date', $values)\n"

    #print "Query: $query\n"

    #Prepare a SQL statement to input data
Appendix E - Export Call Data Script

use strict;
use warnings;
use Text::CSV;
use DBI;

#Variable declarations
#Database Variables
my $host = "10.71.64.220";
my $db = "calldatadbb";
my $uname = "";
my $pwd = "";
my $table = "";
my $ABSLquery = "";
my $CHquery = "";
my $COquery = "";
my $MDquery = "";
my $SLquery = "";
my $SAquery = "";

#Data Variables
my $fileName = "";
my $date = "";
my $values;
my @cols;
my @colnames = ();
my $colnum;
my $names = "";

#Establish a connection to the database
#First get the user name
print "Please connect to the database\nUsername:\t";
chomp($uname=<>);

#Next get the password
print "Password:\t";
chomp($pwd=<>);

#Connect to the database using the user name and password
my $dsn = "DBI:mysql:database=$db;host=$host";
my $dbh = DBI->connect($dsn, $uname, $pwd, {RaiseError=>1}) or die "Couldn't connect to database: " . DBI->errstr;

#Get file name from user to export data to
print "What file name do you want to export data to?\t";
chomp($fileName=<>);

#Get date from user that this data will be pulled from
print "What date do you want to export data from (YYYY-MM-DD)?\t";
chomp($date=<>);

#Print values of the user input initialized variables
#print "Your file is named $fileName, the data is from $date, and you want to export data for agent $table.\n";

#Populate the SQL query variables
$ABSLquery = "SELECT * FROM `answered_before_sl` WHERE Date = '$date'";
$CHquery = "SELECT * FROM `calls_handled` WHERE Date = '$date'";
$COquery = "SELECT * FROM `calls_offered` WHERE Date = '$date'";
$MDquery = "SELECT * FROM `max_delay` WHERE Date = '$date'";
$SLquery = "SELECT * FROM `service_level` WHERE Date = '$date'";
$SAquery = "SELECT * FROM `short_abandon` WHERE Date = '$date'";

#Print the value of the query variable
#print "Query1: $ABSLquery\n";
#print "Query2: $CHquery\n";
#print "Query3: $COquery\n";
#print "Query4: $MDquery\n";
#print "Query5: $SLquery\n";
#print "Query6: $SAquery\n";

my $ABSsth = $dbh->prepare ($ABSLquery) or die "prepare: ". $dbh->errstr;
$ABSsth->execute;

my $CHsth = $dbh->prepare ($CHquery) or die "prepare: ". $dbh->errstr;
$CHsth->execute;

my $COsth = $dbh->prepare ($COquery) or die "prepare: ". $dbh->errstr;
$COsth->execute;

my $MDsth = $dbh->prepare ($MDquery) or die "prepare: ". $dbh->errstr;
$MDsth->execute;

17
my $SLsth = $dbh->prepare ($SLquery) or die "prepare: ". $dbh->errstr;
$SLsth->execute;

my $SAsth = $dbh->prepare ($SAquery) or die "prepare: ". $dbh->errstr;
$SAsth->execute;

my $csv = Text::CSV->new ( { binary => 1, eol => "\n" } )
or die "Cannot use CSV: ".Text::CSV->error_diag();

#Open the connection to a .CSV file
open my $fh, "">", "$fileName";

#Write the headers once to the .CSV file
$csv->print($fh, $ABSsth->{NAME});

#Populate the .CSV file with the results from the SQL query variable
while (my $row = $ABSsth->fetchrow_arrayref){
  $csv->print($fh, $row);
}
$ABSsth->finish;

#Populate the .CSV file with the results from the SQL query variable
while (my $row = $CHsth->fetchrow_arrayref){
  $csv->print($fh, $row);
}
$CHsth->finish;

#Populate the .CSV file with the results from the SQL query variable
while (my $row = $COsth->fetchrow_arrayref){
  $csv->print($fh, $row);
}
$COsth->finish;

#Populate the .CSV file with the results from the SQL query variable
while (my $row = $MDsth->fetchrow_arrayref){
  $csv->print($fh, $row);
}
$MDsth->finish;

#Populate the .CSV file with the results from the SQL query variable
while (my $row = $SLsth->fetchrow_arrayref){
  $csv->print($fh, $row);
}
$SLsth->finish;
Appendix F - Export Agent Data Script

use strict;
use warnings;
use Text::CSV;
use DBI;

#Variable declarations
#Database Variables
my $host = "10.71.64.220";
my $db = "agentcalldata";
my $uname = "";
my $pwd = "";
my $table = "";
my $query = "";

#Data Variables
my $fileName = "";
my $date = "";
my $values;
my @cols;
my @colnames = ();
my $colnum;
my $names = "";

#Establish a connection to the database
#First get the user name
print "Please connect to the database.\nUsername:\t";
chomp($uname=<>);

#Next get the password
print "Password:\n";
chomp($pwd=<>);

#Connect to the database using the user name and password
my $dsn = "DBI:mysql:database=$db;host=$host";
my $dbh = DBI->connect($dsn, $uname, $pwd, {RaiseError=>1}) or die "Couldn't connect to database: ". DBI->errstr;

#Get file name from user to import data
print "What file name do you want to export data to?\n";
chomp($fileName=<>);

#Get the table name to export data from
print "What user do you want to export data for (66XXX)?\n";
chomp($table=<>);

#Get date from user that this data needs to be stored for
print "What date do you want to export data from (YYYY-MM-DD)?\n";
chomp($date=<>);

#Print values of the user input initialized variables
#print "Your file is named $fileName, the data is from $date, and you want to export data for agent $table.\n";

#Populate the SQL query variable
$query = "SELECT * FROM `$table` WHERE Date = '$date'";

#Print the value of the query variable
#print "Query: $query\n";

#Prepare a SQL statement to input data and execute
my $sth = $dbh->prepare ($query) or die "prepare: ". $dbh->errstr;
$sth->execute;

#Populate the .CSV file with the results from the SQL query variable
while (my $row = $sth->fetchrow_arrayref){
  #
  my $csv = Text::CSV->new ( { binary => 1, eol => "\n" } )
  or die "Cannot use CSV: ".Text::CSV->error_diag();

  #Open the connection to a .CSV file
  open my $fh, "">", "$fileName";

  #Write the .CSV file
  $csv->print($fh, $sth->{NAME});

  #Populate the .CSV file with the results from the SQL query variable
  while (my $row = $sth->fetchrow_arrayref){
    #
  }

  #Close the connection to the .CSV file
  close $fh;

  #Print values of the user input initialized variables
  #print "Your file is named $fileName, the data is from $date, and you want to export data for agent $table.\n";
}

#Populate the .CSV file with the results from the SQL query variable
while (my $row = $sth->fetchrow_arrayref){
  #
  my $csv = Text::CSV->new ( { binary => 1, eol => "\n" } )
  or die "Cannot use CSV: ".Text::CSV->error_diag();

  #Open the connection to a .CSV file
  open my $fh, "">", "$fileName";

  #Write the .CSV file
  $csv->print($fh, $sth->{NAME});

  #Populate the .CSV file with the results from the SQL query variable
  while (my $row = $sth->fetchrow_arrayref){
    #
  }

  #Close the connection to the .CSV file
  close $fh;

  #Print values of the user input initialized variables
  #print "Your file is named $fileName, the data is from $date, and you want to export data for agent $table.\n";
}
$csv->print($fh, $row);
}

# Close the connection to the CSV file
close $fh;

# Disconnect from the database
$dbh->disconnect;