Voice traffic real-time cost and revenue monitoring with Excel

Emin Gabrielyan
2013-10-01

# Table of contents

[1. Table of contents 1](#_Toc368652197)

[2. Introduction 2](#_Toc368652198)

[3. Data worksheet 2](#_Toc368652199)

[4. Parameters of the periodic retrieval subroutine 3](#_Toc368652200)

[5. Refresh VBA script 10](#_Toc368652201)

[6. Parameters for the missing record detection subroutine 14](#_Toc368652202)

[7. Re-Sync VBA script 17](#_Toc368652203)

[8. Breakout into exponentially growing intervals 23](#_Toc368652204)

[9. Collecting the statistics per interval 25](#_Toc368652205)

[10. Constructing the chart range 28](#_Toc368652206)

[11. Time axis labels 30](#_Toc368652207)

[12. Chart construction 31](#_Toc368652208)

[13. Evolution of the chart 36](#_Toc368652209)

[14. Parameters for chart building and emailing 38](#_Toc368652210)

[15. Chart emailing VBA script 40](#_Toc368652211)

[16. The scheduler 44](#_Toc368652212)

[17. Installation 48](#_Toc368652213)

[18. ADODB Reference 50](#_Toc368652214)

[19. Further work 51](#_Toc368652215)

[20. Issues with range calculation in Excel 2013 51](#_Toc368652216)

[21. References 52](#_Toc368652217)

[22. Acronyms 54](#_Toc368652218)

[23. Legal 55](#_Toc368652219)

# Introduction

The previous version of the cost/revenue monitoring application was based on a bash script retrieving the data from the remote MySQL server into local html files. The local html files were linked to an Excel file, which periodically constructed the cost/revenue charts from the content of these html files. The same bash script, responsible for the MySQL connection, is also responsible for the distribution of the chart images generated by the Excel file to its recipients via email [[130915 i](http://switzernet.com/3/public/130915-mysql2html2excel2chart2smtp/)].

This document presents the new version of the application entirely scripted in Excel. There is no bash script for supporting the inflow from the MySQL server and the outgoing email to the users. All components are scripted with VBA in the Excel file. The new version of the Excel application does not rely to data source names defined in the ODBC data source administrator of the local computer. All credentials of remote data connection are stored in the Excel file making it portable across computers. Therefore, the Excel file must be kept securely and the sensitive information must be removed before transmitting it to other users.

The version presented in this document is also equipped with subroutines periodically verifying the consistency of the data. It regularly compares and checks whether there are missing calls in the data chunks retrieved in the past. Our application is not connected to the main billing server but to its replication. Therefore the fresh records may arrive to our MySQL server with delays. Normally, the Excel application selects data incrementally and retrieves only fresh calls following from the stop time of the previous retrieval until the current time minus a gap of about two minutes. If at the moment of the retrieval of the previous chunk some calls had not yet arrived to the database server, the retrieval of the next chunk would not correct the calls lacking in the previous chunk. The new version of the application compares, for quite old periods located back in the past, the local numbers of calls with the numbers of calls on the remote server. A large gap of 6 hours is considered to recover all calls even under scenarios when the replication temporarily breaks. Whenever a mismatch in the number of calls is detected, the hourly chunks of the past are re-downloaded until the Excel file is totally re-synced with the remote database.

Additional optimization is achieved in the data treatment. Compared to the previous version [[130915 i](http://switzernet.com/3/public/130915-mysql2html2excel2chart2smtp/)] receiving first the MySQL data flow into HTML text files, in the new version, the data is streamed from the MySQL server directly into the binary Excel file’s worksheet. There are no intermediary HTML files storing the data records in text format.

The next sections present the scripts and the formulas of the Excel file in details.

# Data worksheet

The Excel file is organized as it is shown on the screenshot below. Columns from A to F are reserved for a retrieval of data from the remote database server. Columns from H to I contain all parameters and variables of the application. Column H holds the name of the variable for the user. The columns from K and on contain the formulas building the input data for the chart. The chart is stored in a separate sheet named ‘chart sheet’.



# Parameters of the periodic retrieval subroutine

The key function of our Excel workbook is its capacity to connect to the remote database server and download the call records in successive adjacent but not overlapping chunks. The computation of the dynamically changing MySQL requests and the areas where the chunks must land as well as the areas which must be removed when exceeding the limits, are carried out with the Excel formulas that will be discussed in this section. In the next section we will present the VBA script using the data computed by the Excel formulas and carrying out the transfer of records from the remote database server to the local worksheet.

Parameters shown in the column I of the table below are used for the establishment of the MySQL connection and for a periodic incremental retrieval of fresh call records from the remote database.

|  |  |  |  |
| --- | --- | --- | --- |
|  | H | I |  |
|   | Name | Value |   |
| $I$2 | driver | {MySQL ODBC 5.2 Unicode Driver} |   |
| $I$3 | server | rep-db-1.switzernet.com |   |
| $I$4 | database | porta-billing |   |
| $I$5 | user | emin |   |
| $I$6 | password | \*\*\*\* |   |
| $I$7 | [db\_connect](#RANGE!I7) | driver={MySQL ODBC 5.2 Unicode Driver}; server=rep-db-1.switzernet.com; database=porta-billing; user=emin; password=\*\*\*\*; option=3 | {$I$7} =H2&"="&I2&"; "&H3&"="&I3&"; "&H4&"="&I4&"; "&H5&"="&I5&"; "&H6&"="&I6&"; option=3" |
| $I$8 | [db\_error](#RANGE!I8) |   |   |
| $I$9 | time zone hours | 2 |   |
| $I$10 | margin | 0:01:00 |   |
| $I$11 | limit time | 20:00:00 |   |
| $I$12 | limit per select | 49000 |   |
| $I$13 | limit lines | 50000 |   |
| $I$14 | [stop\_did](#RANGE!I14) | 2013-10-01 12:16:27.000 |   |
| $I$15 | now | 2013-10-01 14:55:12.880 | {$I$15} =NOW() |
| $I$16 | now in UTC | 2013-10-01 12:55:12.000 | {$I$16} =FLOOR(I15-I9/24,TIME(0,0,1)) |
| $I$17 | [stop\_now](#RANGE!I17) | 2013-10-01 12:54:12.000 | {$I$17} =I16-I10 |
| $I$18 | start | 2013-10-01 12:16:28.000 | {$I$18} =IF(I17-I14<=I11,I14,I17-I11)+TIME(0,0,1) |
| $I$19 | range select | '2013-10-01 12:16:28' and '2013-10-01 12:54:12' | {$I$19} ="'"&TEXT(I18,"yyyy-mm-dd hh:mm:ss")&"' and '"&TEXT([stop\_now](#RANGE!I17),"yyyy-mm-dd hh:mm:ss")&"'" |
| $I$20 | mysql\_query | (select 1 as 'table', setup\_time, disconnect\_time as 'time', revenue, charged\_amount, charged\_quantity from CDR\_Vendors where disconnect\_time between '2013-10-01 12:16:28' and '2013-10-01 12:54:12' limit 49000) union all (select 2, setup\_time, connect\_time, null, null, null from CDR\_Vendors\_Failed where connect\_time between '2013-10-01 12:16:28' and '2013-10-01 12:54:12' limit 49000) order by time | {$I$20} =IF([stop\_now](#RANGE!I17)>=I18,"(select 1 as 'table', setup\_time, disconnect\_time as 'time', revenue, charged\_amount, charged\_quantity from CDR\_Vendors where disconnect\_time between " & I19 & " limit " & I12 & ") union all (select 2, setup\_time, connect\_time, null, null, null from CDR\_Vendors\_Failed where connect\_time between " & I19 & " limit " & I12 & ") order by time") |
| $I$21 | data\_lines | 50000 | {$I$21} =COUNT(A:A)+1 |
| $I$22 | data\_land | 'data sheet'!A50001 | {$I$22} =I24&"!A"&([data\_lines](#RANGE!I21)+1) |
| $I$23 | [data\_scroll](#RANGE!I23) | FALSE | {$I$23} =IF(I21>I13,I24&"!"&"A2:F"&(2+I21-I13-1)) |
| $I$24 | [worksheet\_name](#RANGE!I24) | 'data sheet' | {$I$24} ="'"&REPLACE(CELL("filename",I24),1,SEARCH("]",CELL("filename",I24)),"")&"'" |
| $I$25 | [period\_refresh](#RANGE!I25) | 'data sheet'!$I$15:$I$20 | {$I$25} =I24&"!"&ADDRESS(ROW(I15:I20),COLUMN(I15:I20))&":"&ADDRESS(ROW(I15:I20)+ROWS(I15:I20)-1,COLUMN(I15:I20)) |

The column H contains the name of the parameter (for the user only). Blue color of the parameter indicates that the cell under I column is named according to the value of the H column.

All cells marked in yellow are input cells where the user must provide data. These cells are unlocked. All other cells are locked for the user. The worksheet is protected, but without a password. The purpose of the protection is to avoid accidental errors. You can unprotect the worksheet any time to access the other cells.

In cell [[db\_connect](#RANGE!I7)] we construct the string for opening a MySQL connection. This connect string is used by the VBA script presented in the next section.

This version of the application does not rely on a user defined DSN connection which must be defined via an ODBC Data Source Administrator interface of your computer. All data connection parameters are provided locally in the Excel worksheet. When you deal with numerous remote databases, no need to create in advance on your computer the data connections for each database. The credentials for each remote database are stored in the Excel file. Therefore the Excel file is also portable, and can be executed on any machine without need to configure the ODBC data sources of the PC for matching your Excel needs. However a special attention must be paid when transferring these Excel files. All sensitive information must be removed.

The MySQL ODBC driver is needed to be installed on the PC.



Cell [driver] contains the name of the MySQL ODBC driver installed on your computer {MySQL ODBC 5.2 Unicode Driver}.

In [[db\_error](#RANGE!I8)] cell the VBA script accumulates all connection errors by concatenating them one after the other. The content of this cell is deleted by the email script, after a successful transmission of the chart and all so far accumulated error reports to the users of the distribution list.

The yellow cell [time zone hours] is an input to be provided by the user. It is the current time zone in which the chart’s horizontal axis must be shown. Note that the raw data received from the server is in UTC. The time zone is important also to construct the MySQL call retrieval requests. The current time must be converted into UTC before building a MySQL request.

Input cell [margin] indicates until which moment with respect to the current time the data chunk must be retrieved. The MySQL request retrieving the data does not request calls until the current time but it stops a few minutes before the current time. In the above example this margin is equal to 1 minute. It means that at each retrieval request, the stop time of the MySQL selection will be 1 minute older than the current time. This one minute gap is reserved for leaving the remote MySQL server sufficient time margin to replicate the original flow from the main billing server. If no sufficient time is left, the re-synchronization script will detect and correct the errors after a few hours.

The value of [limit time] in our example is equal to 20 hours. It is the maximal period of the time allowed for downloading via the Refresh script. If no [[stop\_did](#RANGE!I14)] time available, or if [[stop\_did](#RANGE!I14)] is too far in the past the [limit time] will be used to determine the duration of the requested chunk.

The value of cell [limit per select] indicates the maximal number of records in a single MySQL select query. The MySQL string parsed to the server contains two embedded select requests, one retrieving the records from the table of answered calls, and the other one retrieving the calls from the table of failed calls, both for the same period of time.

The value of input cell [limit lines] contains the maximal number of lines allowed in the local worksheet. In the above example it is equal to 50000. If the data retrieved from the remote server results in the excess of this limit, the VBA script of the Refresh subroutine will remove from the Excel file the older records to fit into the limit of lines.

Cell [[stop\_did](#RANGE!I14)] is being updated by the VBA script. Its value is equal to the stop time of the last successfully downloaded chunk of calls. The user must delete this value if he/she wishes to start the synchronization with the database server from the scratch. The next chunk will be downloaded starting from [[stop\_did](#RANGE!I14)] plus 1 second (inclusively). The times of call records are rounded to seconds.

Cell [now] contains the current time in the current time zone.

Cell [now in UTC] contains the current time in the UTC time zone. As its formula shows, the milliseconds are cut out.

Cell [[stop now](#RANGE!I17)] contains the stop time (in UTC) of the MySQL request being prepared. The stop time is a little past from the current time by the interval defined in cell [margin].

The start time of the current MySQL request being prepared is the previous stop time plus one second if the interval between the previous stop and new stop [[stop\_now](#RANGE!I17)] does not exceed the maximally allowed time. Otherwise the value of [start] cell is set to the current stop time minus the maximally allowed interval [limit time] (20 hours in this example).

Cell [range select] contains the time criteria used in the MySQL string for selecting the calls both from the tables of answered and failed calls.

In cell [[mysql\_query](#RANGE!I20)] we construct the request to be sent to the MySQL server. The value of this cell can be FALSE when the stop time is before the start. Such a value tells to the VBA script to skip the parsing to the remote server. The stop time can be effectively fall before the start time if the user increases the value of [margin] pushing thus the [[stop\_now](#RANGE!I17)] value further into the past (even before [[stop\_did](#RANGE!I14)]). The processing of the retrieval request must be suspended until the current time advances enough to bring the stop time [[stop\_now](#RANGE!I17)] ahead the start [[stop\_did](#RANGE!I14)].

As you see, the MySQL query consists of a union of two select requests. The union is accompanied with a keyword “all”, ensuring that the duplicate records are not deleted. The chances of duplicate records are high as we retrieve only the strict minimum of fields (for example the calling and called party phone numbers are not downloaded). The outputs are limited in each individual select request making the requests more optimal as otherwise the server would create first two large datasets to cut them down to the global limit only after their union. The columns from both tables are arranged so as to form a single output table of all calls (both answered and failed). The first column contains a value of 1 or 2 indicating the source table on the server where the data is taken from. For the answered calls the disconnect time is used as the time criteria field when selecting the records. The table of failed calls does not have the disconnect time field and we rely to the connect time field (which is essentially the same for failed calls with no duration). All records are ordered by the value of time after the union. The value of time is defined to be the second field of the union table, which is disconnect time for the original table of answered calls and the connect time for the original table of failed calls. The MySQL manual <http://dev.mysql.com/doc/refman/5.6/en/union.html> provides more insights on the UNION syntax.

Cell [[data lines](#RANGE!I21)] shows how many lines are already occupied in the Excel file.

Cell [[data land](#RANGE!I22)] computes the landing point of the record set retrieved from the database server.

Cell [[data scroll](#RANGE!I23)] computes the area on the top of the data table (columns from A to F being alimented by the remote server) to be removed whenever the new arrivals result into an excess of the allowed number of local [data lines](#RANGE!I21).

Cell [[worksheet name](#RANGE!I24)] computes the current worksheet name. This value is used by other cells constructing the range names. The use of the worksheet name in the range references is obligatory in VBA as our Excel file has multiple worksheets.

Cell [[period refresh](#RANGE!I25)] contains the range of all cells involved in the dynamic calculation of the period of the time and the corresponding MySQL request. The VBA script which is downloading the data disables the automatic calculation of cells during its execution. It is necessary to avoid useless computation throughout the entire worksheet with thousands of data lines while executing the VBA subroutine. For instance the computation of the chart’s input data range will be required only after execution of the VBA script. As the automatic calculations are disabled, the Excel formulas computing the MySQL request to parse to the remote server will not produce up to date results. The VBA script therefore refers to the cells covered by the [[period refresh](#RANGE!I25)] range in order to force their calculation. It is important to note that when you are forcing the calculation of cells, the order in which the cells are computed is very important for the cross referred cells. First you have to update the source cell then the cell referring to the source, and not the other way around. Updating the last cell in the dependency chain will not force a calculation of the cells this cell is depending on. When forcing a calculation of a range of cells in one single operation, the coherence of the update (within the limit of the range) is ensured by the Excel’s calculation engine. Therefore, it is important to cover in the range all cells that the final resulting cells may depend on, and it is important to update them in a single calculation operation. Excel 2013 however has serious issues when calling the range calculation method while the automatic calculation is disabled (see section 20).

Below is the full list of all named parameters of this worksheet. This list contains the parameters introduced in this section as well as all other named parameters that will be discussed in the following sections.



All named cells are the ranges that are accessed by VBA scripts. Changing the location of a parameter in the worksheet would require modification of the VBA script accessing the parameter via its address. Named cells do not cause such inconveniences.

# Refresh VBA script

Below is the VBA script responsible for an incremental downloading of adjacent CDR chunks of fresh calls from the remote database server.

|  |  |
| --- | --- |
| Code\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Comments |
| Sub Oval\_Refresh\_Click() Refresh\_CallsEnd Sub | Update the Excel file with fresh call records when clicking on the oval “Refresh”. The subroutine of new calls is otherwise called automatically on periodical basis by VBA script responsible for scheduling (see section 16). |
| Sub Refresh\_Calls()  Worksheets(1).Unprotect Application.Calculation = xlCalculationManual  range(range("[period\_refresh](#RANGE!I25)").Value).Calculate  If range("[mysql\_query](#RANGE!I20)").Value <> False Then | Unprotect the worksheet. Disable the automatic calculation of cells in the worksheet. Calculate the values of the cells involved in the calculation of the MySQL request string and the period of the time the calls must be retrieved for. Execute the body of the if-block if the period of time is valid. |
|  Dim conn1 As ADODB.Connection Set conn1 = New ADODB.Connection  On Error GoTo Error\_Handler\_1 conn1.Open range("[db\_connect](#RANGE!I7)").Value On Error GoTo 0 | Opening a MySQL connection according to the connection string combined (by Excel worksheet formulas) from the input elements of the user. The input data includes the driver, server, database, username, and the password. |
|  Dim [stop\_now](#RANGE!I17) As Date [stop\_now](#RANGE!I17) = range("[stop\_now](#RANGE!I17)").Value | Save the stop value of the current request in a [[stop now](#RANGE!I17)] variable. Upon a successful completion of the subroutine, the value of [[stop did](#RANGE!I14)] cell will be replaced by new [[stop now](#RANGE!I17)]. |
|  Dim rs1 As ADODB.Recordset  On Error GoTo Error\_Handler\_2 Set rs1 = conn1.Execute(range("[mysql\_query](#RANGE!I20)").Value) On Error GoTo 0 | Execute the MySQL request on the remote server. If error is occurred, process it under Error Handler 2. |
|  range("[data\_lines](#RANGE!I21),[data\_land](#RANGE!I22)").Calculate  range(range("[data\_land](#RANGE!I22)").Value).CopyFromRecordset rs1  rs1.Close Set rs1 = Nothing  conn1.Close Set conn1 = Nothing | If record set is successfully obtained, compute the number of lines in the current Excel worksheet and then, the place the downloaded record set must be copied to. Copy the received data at the next available row and close the connection with the remote database server. |
|  range("[data\_lines](#RANGE!I21),[data\_scroll](#RANGE!I23)").Calculate  If (range("[data\_scroll](#RANGE!I23)").Value <> False) Then range(range("[data\_scroll](#RANGE!I23)").Value).Delete Shift:=xlUp End If | Compute the [data lines](#RANGE!I21) again (after the new records are copied) and the area in the top of the data (i.e. the oldest rows) to be deleted from the worksheet in case the allowed limit of lines is exceeded. |
|  range("[stop\_did](#RANGE!I14)").Value = [stop\_now](#RANGE!I17) | As the data chunk is successfully downloaded write in cell [[stop did](#RANGE!I14)] the new stop time value that is successfully used in the current MySQL request. This stop time value in UTC will serve as a reference point for the next MySQL request. |
|  Else  MsgBox "Wait as Stop is behind Start"  End If | If the Excel worksheet returns a False value for the MySQL string that means the range of the time to be retrieved is not valid (the stop time is behind the start due to an increase of the margin by the user). No request must be sent to the server. |
| Do\_Clean\_Up: Application.Calculation = xlCalculationAutomatic Worksheets(1).Protect Password:=""Exit Sub | Before exiting from the subroutine, whether upon its successful completion or after an error, switch back on the automatic calculation of the worksheet and protect the worksheet. |
| Error\_Handler\_1: range("[db\_error](#RANGE!I8)").Value = range("[db\_error](#RANGE!I8)").Value \_ & " On " & Format(Now(), "yy-mm-dd hh:mm:ss") \_ & " error " & Err.Number & " occured" \_ & " while opening MySQL connection" \_ & " {" & Err.Description & "}." On Error GoTo 0 Resume Do\_Clean\_Up | If an error is occurred while opening the database connection, append the current time, the error code, and the error description into the [[db error](#RANGE!I8)] Excel cell. The content of this cell will be communicated to recipients via the next successful email. Go to the cleanup section for exiting from the subroutine. |
| Error\_Handler\_2: range("[db\_error](#RANGE!I8)").Value = range("[db\_error](#RANGE!I8)").Value \_ & " On " & Format(Now(), "yy-mm-dd hh:mm:ss") \_ & " error " & Err.Number & " occured" \_ & " while retrieving MySQL data" \_ & " {" & Err.Description & "}." On Error GoTo 0 conn1.Close Resume Do\_Clean\_UpEnd Sub | If error is occurred upon the execution of the MySQL request, append the current time, error code, and the error description into [[db error](#RANGE!I8)] cell. The value of this cell will be communicated to the users via email. Close the database connection and prepare to exit from the subroutine. The [[stop did](#RANGE!I14)] value obviously will not be updated, and therefore the next time when this subroutine will be called, the interval start time will be the same. |

# Parameters for the missing record detection subroutine

Below we present the formulas calculating the parameters for the second important VBA script of this Excel file which is responsible for detecting the missing calls in the local worksheet and resynchronizing the data chunks downloaded in the past when such mismatches in the number of records detected.

|  |  |  |  |
| --- | --- | --- | --- |
|  | H | I |  |
| $I$69 | [chunk\_calculate](#RANGE!I69) | 'data sheet'!$I$70:$I$93 | {$I$69} =I24&"!"&ADDRESS(ROW(I70:I93),COLUMN(I70:I93))&":"&ADDRESS(ROW(I70:I93)+ROWS(I70:I93)-1,COLUMN(I70:I93)) |
| $I$70 | [chunk\_last](#RANGE!I70) | 2013-10-01 05:00:00 |   |
| $I$71 | chunk min | 1 |   |
| $I$72 | chunk max | 5 |   |
| $I$73 | chunk gap | 6 |   |
| $I$74 | chunk start | 2013-10-01 05:00:00 | {$I$74} =CEILING(MAX(MIN(C:C),I70),1/24) |
| $I$75 | [chunk\_stop](#RANGE!I75) | 2013-10-01 06:00:00 | {$I$75} =I74+MIN(-I74+FLOOR(MAX(C:C),1/24)-I73/24,I72/24) |
| $I$76 | chunk begin | 42137 | {$I$76} =COUNTIF(C:C,"<"&I74)+1+1 |
| $I$77 | chunk end | 42399 | {$I$77} =COUNTIF(C:C,"<"&I75)+1 |
| $I$78 | chunk area | C42137:C42399 | {$I$78} =IF(I77>=I76,"C"&I76&":C"&I77) |
| $I$79 | chunk count1 | 263 | {$I$79} =MAX(0,I77-I76+1) |
| $I$80 | chunk count2 | 263 | {$I$80} =IF(I78=FALSE,0,COUNT(INDIRECT(I78))) |
| $I$81 | chunk count3 | 263 | {$I$81} =IF(I78=FALSE,0,COUNTIF(INDIRECT(I78),">="&I74)) |
| $I$82 | chunk count4 | 263 | {$I$82} =IF(I78=FALSE,0,COUNTIF(INDIRECT(I78),"<"&I75)) |
| $I$83 | chunk valid | TRUE | {$I$83} =MIN(I79:I82)=MAX(I79:I82) |
| $I$84 | chunk range | '2013-10-01 05:00:00' and '2013-10-01 05:59:59' | {$I$84} ="'"&TEXT(I74,"yyyy-mm-dd hh:mm:ss")&"' and '"&TEXT(I75-TIME(0,0,1),"yyyy-mm-dd hh:mm:ss")&"'" |
| $I$85 | [chunk\_measure](#RANGE!I85) | select count(1) from CDR\_Vendors where disconnect\_time between '2013-10-01 05:00:00' and '2013-10-01 05:59:59' union all select count(2) from CDR\_Vendors\_Failed where connect\_time between '2013-10-01 05:00:00' and '2013-10-01 05:59:59' | {$I$85} =IF(AND(ROUND([chunk\_stop](#RANGE!I75)-I74,5)>=ROUND(I71/24,5),I83),"select count(1) from CDR\_Vendors where disconnect\_time between " & I84 & " union all select count(2) from CDR\_Vendors\_Failed where connect\_time between " & I84) |
| $I$86 | [chunk\_answered](#RANGE!I86) |   |   |
| $I$87 | [chunk\_failed](#RANGE!I87) |   |   |
| $I$88 | chunk missing | -263 | {$I$88} =SUM(I86:I87)-I79 |
| $I$89 | [chunk\_insert](#RANGE!I89) | FALSE | {$I$89} =IF(I88>0,I24&"!A"&I76&":F"&(I76+I88-1)) |
| $I$90 | [chunk\_status](#RANGE!I90) | detecting -263 missing calls between '2013-10-01 05:00:00' and '2013-10-01 05:59:59'. | {$I$90} ="detecting "&I88&" missing calls between "&I84&"." |
| $I$91 | [chunk\_report](#RANGE!I91) |   |   |
| $I$92 | [chunk\_retrieve](#RANGE!I92) | (select 1 as 'table', setup\_time, disconnect\_time as 'time', revenue, charged\_amount, charged\_quantity from CDR\_Vendors where disconnect\_time between '2013-10-01 05:00:00' and '2013-10-01 05:59:59' limit ) union all (select 2, setup\_time, connect\_time, null, null, null from CDR\_Vendors\_Failed where connect\_time between '2013-10-01 05:00:00' and '2013-10-01 05:59:59' limit ) order by time | {$I$92} ="(select 1 as 'table', setup\_time, disconnect\_time as 'time', revenue, charged\_amount, charged\_quantity from CDR\_Vendors where disconnect\_time between " & I84 & " limit " & I86 & ") union all (select 2, setup\_time, connect\_time, null, null, null from CDR\_Vendors\_Failed where connect\_time between " & I84 & " limit " & I87 & ") order by time" |
| $I$93 | [chunk\_land](#RANGE!I93) | 'data sheet'!A42137 | {$I$93} =I24&"!A"&I76 |

Similarly to the table of formulas computing the parameters for the incremental update subroutines (see section 4), here also, the cells marked in blue are named cells which are accessed from within the VBA subroutine responsible for re-synching. This VBA script will be presented in the next section (see section 7).

The value of [[chunk calculate](#RANGE!I69)] cell provides to the VBA script the range of cells on which the calculation must be forced in order to refresh the results of the formulas. Note that our VBA script disables the automatic calculation of the worksheet while running. This is to avoid heavy updates of the worksheet in the areas not needed by this particular script (e.g. in the construction of the chart) upon the arrival of large chunks of data.

Cell [[chunk last](#RANGE!I70)] contains the time until which the hourly chunks are verified. The formulas are using the value of this cell to compute the next set of hourly chunks to verify. The VBA script, upon each successfully carried out synchronization, overwrites this cell shifting its value forward in time.

Cell [chunk min] is the minimal duration of a chunk (in hours) to compare with the server. In our example its value is equal to 1 hour.

Cell [chunk max] contains the maximal duration in hours, of a chunk to compare with the remote server. The value of this parameter is equal to 5 hours in our example.

Cell [chunk gap] contains the gap in hours to respect between the current time and the chunk in the past susceptible for the comparison. This parameter ensures that fresh chunks are not compared. In our example, the chunk must be at least 6-hour old to be a subject of a comparison. Such a gap leaves to the remote database server enough time to recover from the replication delays or failures.

Cell [chunk start] contains the value of [[chunk last](#RANGE!I70)] or, if no value is provided yet, the earliest record rounded to the next hour. Value of [chunk start] is the hour, starting from which the comparison of record numbers with the remote server must be carried out. The user may wish to delete the value of [[chunk last](#RANGE!I70)] if he/she needs to initiate the comparison for the whole data set.

Cell [[chunk stop](#RANGE!I75)] represents the end time of the chunk to be compared with the remote server. It is computed as the time of the last available call, rounded down to the beginning of the hour, minus the gap of 6 hours dictated by the [chunk gap]. However [[chunk stop](#RANGE!I75)] cannot be more than [chunk max] hours away from the [chunk start].

Cell [chunk begin] is the number of the first row of the chunk of the worksheet being compared.

Cell [chunk end] is the number of the last row of the chunk of the worksheet being compared.

Cell [chunk area] is only the time value column the chunk being compared.

Cells [chunk count1], [chunk count2], [chunk count3], and [chunk count4], must all contain the same value, which is the number of local records in the chunk. If the values mismatch, something important went wrong in the past (e.g. records were not sorted or they contained empty lines).

Cell [chunk valid] shows true if all four methods of computation of the number of records in the chunk match.

Cell [chunk range] is the time criteria (lower and upper bounds) of both select MySQL requests. Note that the stop value is reduced by one second as between-criteria is inclusive in MySQL.

Cell [[chunk measure](#RANGE!I85)] contains the MySQL request if the time between the stop and the start is more than or equal to the minimal duration of the chunk [chunk min]. If not, the value of the request will be equal to a Boolean false, which will indicate to the VBA script to skip the comparison until the next time. This MySQL request returns us a single-column table with two values in it, the number of answered calls, and the number of failed calls within the same period of time.

Cells [[chunk answered](#RANGE!I86)] and [[chunk failed](#RANGE!I87)] are updated by the VBA script upon the execution of the MySQL request and the retrieval of its results.

Cell [chunk missing] contains the number of records missing in the local worksheet when comparing with the remote database.

If the value of [chunk missing] is positive, then cell [[chunk insert](#RANGE!I89)] computes the address of the data area to be added in the data space for holding the larger record set available on the remote server (for the same period of the time of the chunk). The VBA script uses this address string to insert new data rows by shifting down the rows below. After extending the space, the new data can now land to the extended space without a risk to overwrite the records outside of the current chunk.

Cell [[chunk status](#RANGE!I90)] computes the message to be appended to [[chunk report](#RANGE!I91)] string by the VBA script. The value of [[chunk status](#RANGE!I90)] is appended to [[chunk report](#RANGE!I91)] whenever the status shows a mismatch. The report string grows every time a mismatch is detected until its content is emailed, after which it is emptied.

Cell [[chunk retrieve](#RANGE!I92)] is the MySQL command requesting from the server all call records with the period of the time of the chunk. The syntax is identical to the MySQL request updating the worksheet in small incremental chunks. The difference is in the period of the time, here corresponding to the period of the large chunk being compared, and in the limit values, corresponding to the exact count of answered and failed calls retrieved and stored previously in [[chunk answered](#RANGE!I86)] and [[chunk failed](#RANGE!I87)] cells.

Cell [[chunk land](#RANGE!I93)] contains the landing position of the new data set. It’s used by the VBA script to overwrite the old records of the worksheet chunk by those obtained from the server.

# Re-Sync VBA script

The worksheet parameters presented in the previous section are employed by the VBA script [Sync Calls]. We provide comments to this code in the following table.

|  |  |
| --- | --- |
| Code\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Comments |
| Sub Oval\_Resync\_Click() Sync\_CallsEnd Sub | The re-synchronization of old chunks of calls can be forced by clicking on the circle with “Re-Sync” label. The subroutine is however called also periodically by VBA script “Periodic” (see section 16). |
| Sub Sync\_Calls()  Worksheets(1).Unprotect Application.Calculation = xlCalculationManual | Before the execution, we unprotect the worksheet as the VBA script needs to access to the areas locked for the user (particularly to the data chunks). We also disable the automatic calculation in the worksheet to avoid heavy updates in places not needed by this code (for example in chart data range). |
|  range(range("[chunk\_calculate](#RANGE!I69)")).Calculate If range("[chunk\_measure](#RANGE!I85)").Value <> False Then | Forcing the calculation of the cells only in a limited area supplying information to this scrip. If there are still chunks in the past to be compared with the remote servers proceed with the body of the if-statement. Cell [[chunk measure](#RANGE!I85)] will return False if there are no chunks to compare (i.e. if the next chunk to compare is not sufficiently old). |
|  Dim conn1 As ADODB.Connection Set conn1 = New ADODB.Connection On Error GoTo Error\_Handler\_1 conn1.Open range("[db\_connect](#RANGE!I7)").Value On Error GoTo 0 | Open the MySQL connection to the remote server. If failed, handle the error under label [Error Handler 1] |
|  Dim rs1 As ADODB.Recordset  On Error GoTo Error\_Handler\_2 Set rs1 = conn1.Execute(range("[chunk\_measure](#RANGE!I85)").Value) On Error GoTo 0 range("[chunk\_answered](#RANGE!I86)").CopyFromRecordset rs1 rs1.Close Set rs1 = Nothing | We execute the MySQL request provided in [[chunk measure](#RANGE!I85)] cell. This request retrieves the numbers of answered and failed calls on the remote server for a period of the time of the chunk being compared. If error occurs during the execution of the MySQL request go to the corresponding Error Handler. Otherwise copy the retrieved two numbers into cells [[chunk answered](#RANGE!I86)] and [[chunk failed](#RANGE!I87)]. |
|  range(range("[chunk\_calculate](#RANGE!I69)")).Calculate If range("[chunk\_insert](#RANGE!I89)").Value <> False Then range("[chunk\_report](#RANGE!I91)").Value = range("[chunk\_report](#RANGE!I91)").Value \_ & " On " & Format(Now(), "yy-mm-dd hh:mm:ss") & " " \_ & range("[chunk\_status](#RANGE!I90)").Value & " After re-syncing..." | At this moment we already know the numbers of calls on the remote server. We recalculate the formulas of the worksheet refreshing information to this script in order to see whether we have missing calls or not. Cell [[chunk insert](#RANGE!I89)] is equal to False if there is no mismatch. Otherwise we enter into the if-block. We append the detected discrepancy report to the text stored in [[chunk report](#RANGE!I91)] cell. |
|  On Error GoTo Error\_Handler\_3 Set rs1 = conn1.Execute(range("[chunk\_retrieve](#RANGE!I92)").Value) On Error GoTo 0 | All calls of the chunk are retrieved thanks to the MySQL request computed in [[chunk retrieve](#RANGE!I92)] cell. The control is forwarded to Error Handler 3 label if error occurs upon the execution of this request. |
|  range(range("[chunk\_insert](#RANGE!I89)").Value).Insert Shift:=xlDown | Making space for missing calls in the local worksheet so as to fit the new data chunk retrieved from the remote server. |
|  range(range("[chunk\_land](#RANGE!I93)").Value).CopyFromRecordset rs1 rs1.Close Set rs1 = Nothing | Overwriting the worksheet’s corresponding area with the new data chunk collected from the server. |
|     range(range("[chunk\_calculate](#RANGE!I69)")).Calculate range("[chunk\_report](#RANGE!I91)").Value = range("[chunk\_report](#RANGE!I91)").Value \_ & " " & range("[chunk\_status](#RANGE!I90)").Value  | Calculating the current comparison status and appending it to the report string. The status must be ok except an exotic error occurred and the data retrieved from the server has a different number of records than the counts reported by the previous MySQL request. As we deal with sufficiently old chunks such errors must not occur. |
|  End If | End of the if-block executed when a discrepancy in the local and remote numbers of calls is found |
|  range("[chunk\_answered](#RANGE!I86),[chunk\_failed](#RANGE!I87)").Clear range("[chunk\_last](#RANGE!I70)").Value = range("[chunk\_stop](#RANGE!I75)").Value  conn1.Close Set conn1 = Nothing | The values of [[chunk answered](#RANGE!I86)] and [[chunk failed](#RANGE!I87)] cells are cleaned. The value of [[chunk last](#RANGE!I70)] is updated to store a reference to the last processed chunk. The MySQL connection is closed. |
|  End If | End of the if-block executed when there is an old chunk which is a subject of comparison. The numbers of calls on the remote server are retrieved at the beginning of this if-block. |
| Do\_Clean\_Up: Application.Calculation = xlCalculationAutomatic Worksheets(1).Protect Password:=""Exit Sub | These two commands are carried out irrespectively how we quit the subroutine, i.e. by a normal completion without errors, or due to errors occurred while opening MySQL connection or executing MySQL requests. |
| Error\_Handler\_1: range("[chunk\_report](#RANGE!I91)").Value = range("[chunk\_report](#RANGE!I91)").Value \_ & " On " & Format(Now(), "yy-mm-dd hh:mm:ss") \_ & " error " & Err.Number & " occured" \_ & " while connecting to the remote database" \_ & " {" & Err.Description & "}." On Error GoTo 0 Resume Do\_Clean\_Up | We arrive here if an error occurred while opening the MySQL connection with the remote server. Append the current time, error number, and the error description to the string stored in [[chunk report](#RANGE!I91)] cell. The content of this string is cleaned when successfully emailed to users. |
| Error\_Handler\_2: range("[chunk\_report](#RANGE!I91)").Value = range("[chunk\_report](#RANGE!I91)").Value \_ & " On " & Format(Now(), "yy-mm-dd hh:mm:ss") \_ & " error " & Err.Number & " occured" \_ & " while obtaining record counts" \_ & " {" & Err.Description & "}." On Error GoTo 0 conn1.Close Resume Do\_Clean\_Up | This error handler takes control when an error occurs while attempting to execute the MySQL request computing the numbers of answered and failed calls on the remote server. We append to cell [[chunk report](#RANGE!I91)] the time, error code, and error description. Further, we proceed with the cleanup commands before quitting the subroutine. |
| Error\_Handler\_3: range("[chunk\_report](#RANGE!I91)").Value = range("[chunk\_report](#RANGE!I91)").Value \_ & " On " & Format(Now(), "yy-mm-dd hh:mm:ss") \_ & " error " & Err.Number & " occured" \_ & " while re-downloading CDR chunks" \_ & " {" & Err.Description & "}." On Error GoTo 0 conn1.Close Resume Do\_Clean\_UpEnd Sub | If error occurs upon the execution of the MySQL request retrieving the data of the chunk, we land here. We append the error report to the string stored in [[chunk report](#RANGE!I91)] cell. Then we close the MySQL connection and proceed with the cleanup and quit. The [[chunk last](#RANGE!I70)] cell is obviously not updated, so we have to process the same chunk the next time this subroutine is called. |

# Breakout into exponentially growing intervals

In the following table we compute the breakout of the time axis into a fixed number of exponentially growing intervals. The value of [time intervals] is a parameter which is equal to 120 in this example. The intervals are indexed from 0 to 120. The breakout into 121 intervals must be done such that the oldest interval is longer the earliest interval by a fixed factor. In this example the ratio between the oldest (i.e. the longest) interval over the earliest (i.e. the shortest) is equal to 60. As the intervals are growing exponentially, each interval is longer than the next one by a constant factor (to be found by formulas).

|  |  |  |  |
| --- | --- | --- | --- |
|  | H | I |  |
| $I$27 | time intervals | 120 |   |
| $I$28 | largest/smallest | 60 |   |
| $I$29 | time factor | 1.034708286 | {$I$29} =I28^(1/I27) |
| $I$30 | first call | 2013-09-28 14:09:52 | {$I$30} =MIN(C:C) |
| $I$31 | last call | 2013-10-01 12:16:25 | {$I$31} =MAX(C:C) |
| $I$32 | period | 70:06:33.000 | {$I$32} =I31-I30 |
| $I$33 | smallest | 0:02:23.415 | {$I$33} =I32\*(I29-1)/(I28\*I29-1) |
| $I$34 | largest | 2:23:24.882 | {$I$34} =I33\*I28 |
| $I$35 | [chart range](#RANGE!I35) | 'data sheet'!$AH$1:$AO$122 | {$I$35} =I24&"!"&ADDRESS(ROW(AH1:AO1),COLUMN(AH1:AO1))&":"&ADDRESS(ROW(AH1:AO1)+MAX(I27,10)+1,COLUMN(AH1:AO1)+COLUMNS(AH1:AO1)-1) |
| $I$36 | max labels | 60 |   |
| $I$37 | label every | 2 | {$I$37} =CEILING($I$27,$I$36)/$I$36 |
| $I$38 |   |   |   |
| $I$39 | spoken | 10 |  |

[time intervals] is the number of intervals minus 1.

[largest/smallest] is the ratio of the earliest time interval over the most recent. Such a breakout permits to have a fine granulation for the fresh data while dealing with averaged data over long intervals for the older data. This approach permits to monitor the past daily statistics while at the same time on the same chart the recent hourly activity is being simultaneously monitored.

[time factor] is the growth rate from an interval to interval.

[first call] is the date and time of the earliest record in the worksheet

[last call] is the date and time of the most recent record in the worksheet

[period] is the duration of the time from the earliest to the most recent call

[smallest] is the duration of the most recent shortest interval

[largest] is the duration of the earliest oldest and the longest interval

The formulas for computing the value of [smallest] are presented in the previous document [[130915 i](http://switzernet.com/3/public/130915-mysql2html2excel2chart2smtp/)]. Below is a quick reminder.

$$a\_{0}=P∙\frac{f-1}{k∙f-1}$$

$$a\_{n}=a\_{0}∙k$$

$$f=k^{\frac{1}{n}}$$

[[chart range](#RANGE!I35)] is the range of the input cells of a chart. As the number of intervals can be changed by the user, there is a VBA script which is adjusting the chart range before emailing it.

[max labels] is the maximal number of labels to appear on the horizontal axis

[label every] is the number of intervals per one label

[spoken] is the minimal number of seconds to consider a call as a successful conversation

# Collecting the statistics per interval

As the [first call], [largest], and [time factor] are known thanks to the Excel formulas of the previous section, we can start building the time scale by starting with the time of the first call and with a delta equal to the largest interval. The beginning of the next interval is the time of the previous interval plus its delta. The delta of the next interval is the previous delta divided by factor [time factor] found in the previous section.

The table below constructs the times of the intervals with their deltas.

Column L contains the interval index from 0 to [time intervals] inclusively. According to the formula shown below, an underline is displayed if the index is out of the range.

The initial values of columns M and N are the time of the first call and the length of the longest interval. M2=I30 and N2=I34. M3=M2+N2 and N3=N2/[time factor] and so on for the rest. If the index is out of range an underline is displayed.

For computing statistics within each interval, Excel database functions DSUM and DCOUNT are used. The data range is represented by the area from $A to $F where to the data from the remote server lands, while the criteria of selection must change for each row of the table shown below. For defining criteria, the Excel database functions need a criteria table with its headers corresponding to the headers of the data range. When collecting statistics, for each interval we need to separate the answered calls from the failed calls (one criteria on Table field), bound the calls within the limits of the current interval (two criteria on the Time filed), and identify the long calls (one criteria on Duration field). Therefore for each interval we need a table of criteria with two rows (first one being reserved for the headers) and four columns. As we cannot fit a two-row criteria table in a single-row of the interval, we double the number of columns of criteria (i.e. 8 instead of 4) to interleave the criteria tables in the first 4-column half with the criteria tables in the second 4-column half. The first set of criteria columns (from O to R) contains the criteria to be used by the odd rows and the second set of columns (from S to V) contains the criteria for the even rows.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | L | M | N | O | P | Q | R |
| 1 |   | time | delta | Table | Time | Time | Duration |
| 2 | 0 | 13-09-28 14:09:52.00 | 2:23:24.882 | =1 | >=41545.5901851852 | <41545.6897787325 | >=10 |
| 3 | 1 | 13-09-28 16:33:16.88 | 2:18:36.240 | Table | Time | Time | Duration |
|  | {$L$3} =IF(ISNUMBER(L2),IF(L2<$I$27,L2+1,"\_"),"\_") | {$M$3} =IF(L3="\_","\_",M2+N2) | {$N$3} =IF(L3="\_","\_",N2/$I$29) | {$O$3} =IF($L3="\_","\_",IF(MOD(ROW()-1,2)=1,"=1",O$1)) | {$P$3} =IF($L3="\_","\_",IF(MOD(ROW()-1,2)=1,">="&$M3,P$1)) | {$Q$3} =IF($L3="\_","\_",IF(MOD(ROW()-1,2)=1,IF($L3<$I$27,"<"&($M3+$N3),">=0"),Q$1)) | {$R$3} =IF($L3="\_","\_",IF(MOD(ROW()-1,2)=1,">="&$I$39,R$1)) |

The formulas of the table above and below show that the choice of showing the header name or the criteria’s definition text in columns from S to V are simply the inverse of the choice used in the columns from O to R. Time columns P and T define the lower bound computed in the M column. The columns Q and U define the upper bound which is the lower bound plus the delta. The lower bound is inclusive (more-than or equal-to sign is used). To avoid overlaps, the upper bound is therefore exclusive (lest-than sign is used). As you can note in the formulas of Q and U columns, the upper bound is omitted for the last interval (because otherwise the last call record would not be included in DSUM and DCOUNT statistics).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | S | T | U | V |
| 1 | Table | Time | Time | Duration |
| 2 | Table | Time | Time | Duration |
| 3 | =1 | >=41545.6897787325 | <41545.7860315109 | >=10 |
|  | {$S$3} =IF($L3="\_","\_",IF(MOD(ROW()-1,2)=0,"=1",S$1)) | {$T$3} =IF($L3="\_","\_",IF(MOD(ROW()-1,2)=0,">="&$M3,T$1)) | {$U$3} =IF($L3="\_","\_",IF(MOD(ROW()-1,2)=0,IF($L3<$I$27,"<"&($M3+$N3),">=0"),U$1)) | {$V$3} =IF($L3="\_","\_",IF(MOD(ROW()-1,2)=0,">="&$I$39,V$1)) |

For computing the numbers of all calls, of only answered calls, of only spoken conversations, the corresponding criteria table is used. For instance, for the total number of calls only the time criteria must be used (under columns P and Q or T and U). The time criteria are in the P to Q columns for even rows and in T to U columns for the odd rows. See the formula of W column. Answered calls (see the X column) are computed similarly but they involve also the criteria on the table column. Finally the spoken calls (those which last at least 10 seconds) use also the criteria of Duration column.

|  |  |  |  |
| --- | --- | --- | --- |
|  | W | X | Y |
| 1 | count | answered | spoken |
| 2 | 2471 | 1214 | 1039 |
| 3 | 2111 | 1119 | 972 |
|  | {$W$3} =IF($L3="\_","\_",DCOUNT($A:$F,"Time",IF(MOD(ROW()-1,2)=1,$P2:$Q3,$T2:$U3))) | {$X$3} =IF($L3="\_","\_",DCOUNT($A:$F,"Time",IF(MOD(ROW()-1,2)=1,$O2:$Q3,$S2:$U3))) | {$Y$3} =IF($L3="\_","\_",DCOUNT($A:$F,"Time",IF(MOD(ROW()-1,2)=1,$O2:$R3,$S2:$V3))) |

The sums of PDD, Revenue, Cost, and Duration per interval are computed similarly. The criteria of table and time bounds are used, i.e. either the columns from O to Q or the columns from S to U. The criterion defining the long calls is not used as we need to compute the cost, revenue, and duration of all answered calls.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Z | AA | AB | AC |
| 1 | PDD | Revenue | Cost | Duration |
| 2 | 21811000 | 279.85578 | 181.69786 | 435327 |
| 3 | 19690000 | 303.78519 | 191.62865 | 487745 |
|  | {$Z$3} =IF($L3="\_","\_",DSUM($A:$F,Z$1,IF(MOD(ROW()-1,2)=1,$O2:$Q3,$S2:$U3))) | {$AA$3} =IF($L3="\_","\_",DSUM($A:$F,AA$1,IF(MOD(ROW()-1,2)=1,$O2:$Q3,$S2:$U3))) | {$AB$3} =IF($L3="\_","\_",DSUM($A:$F,AB$1,IF(MOD(ROW()-1,2)=1,$O2:$Q3,$S2:$U3))) | {$AC$3} =IF($L3="\_","\_",DSUM($A:$F,AC$1,IF(MOD(ROW()-1,2)=1,$O2:$Q3,$S2:$U3))) |

Now when statistics are collected in the columns from W to AC we will do a validation of totals. In the first part of the following table we compute the numbers of all input records, of only answered calls, of failed calls, of calls with a duration longer than or equal to 10 seconds ($I$39), and of calls shorter than 10 seconds. The computations are carried out over the raw data input located in columns from A to F. We compute also the total PDD of answered calls, the total revenue, the total cost, and the total duration also over the raw data in A to F columns.

If the breakout of the counts and totals across the intervals is computed correctly, and if the intervals do not overlap and are adjacent, then the total number, the counts of answered and spoken calls, as well as the totals of PDD, Revenue, Cost, and Duration must match. The matching is demonstrated in the last 7 rows of the below table.

|  |  |  |  |
| --- | --- | --- | --- |
|  | H | I |  |
| $I$41 | calls count | 49999 | {$I$41} =COUNT(C:C) |
| $I$42 | calls answered | 26057 | {$I$42} =COUNTIF(A:A,"=1") |
| $I$43 | calls failed | 23942 | {$I$43} =COUNTIF(A:A,"=2") |
| $I$44 | calls spoken | 22769 | {$I$44} =COUNTIF(F:F,">="&I$39) |
| $I$45 | calls short | 3288 | {$I$45} =COUNTIF(F:F,"<"&$I$39) |
| $I$46 | calls PDD | 432685000 | {$I$46} =SUMIF(A:A,"=1",B:B) |
| $I$47 | calls revenue | 5608.08533 | {$I$47} =SUM(D:D) |
| $I$48 | calls cost | 3572.95514 | {$I$48} =SUM(E:E) |
| $I$49 | calls duration | 10790506 | {$I$49} =SUM(F:F) |
| $I$50 |   |   |   |
| $I$51 | ok count | TRUE | {$I$51} =I41=SUM(W:W) |
| $I$52 | ok answered | TRUE | {$I$52} =I42=SUM(X:X) |
| $I$53 | ok spoken | TRUE | {$I$53} =I44=SUM(Y:Y) |
| $I$54 | ok PDD | TRUE | {$I$54} =I46=SUM(Z:Z) |
| $I$55 | ok Revenue | -9.492196565 | {$I$55} =LOG(ABS(I47-SUM(AA:AA)),10) |
| $I$56 | ok Cost | -10.18386733 | {$I$56} =LOG(ABS(I48-SUM(AB:AB)),10) |
| $I$57 | ok Duration | TRUE | {$I$57} =I49=SUM(AC:AC) |

The values in front of [ok Revenue] and [ok Cost] show the degree of the error (in the number of zeroes after the decimal point). The negligible discrepancy with the Revenue and Cost values (an error of nine zeroes after the decimal point) is only a result of the floating point arithmetic.

# Constructing the chart range

In columns from AH to AO we compute the input range of the Excel chart. The values are not absolute sums but are rates. PDD header represents the average setup time per an answered call. All remaining values are converted into hourly rates or rates per minute. For instance, instead of displaying the absolute cost and absolute margin within the interval, we display the cost per hour and the margin per hour. Thus the chart values will be relatively constant and will not vary proportionally to the length of the interval.

The first formula shows how the header is computed. The headers of cost/h and margin/h are accompanied (for the sake of information) by the absolute total of the cost and the absolute total of the margin throughput the entire period of the data.

|  |  |  |  |
| --- | --- | --- | --- |
|  | AH | AI | AJ |
| 1 |   | cost/h [CHF3573] | margin/h [CHF2035] |
| 2 | ┌13-09-28 16:09:52┐ Sep 28 Saturday 17:21 Δ2h | 76.02 | 41.07 |
| {1} |   | {$AI$1} ="cost/h [CHF"&TEXT(I48,"0")&"]" | {$AJ$1} ="margin/h [CHF"&TEXT(I47-I48,"0")&"]" |
| {2} |   | {$AI$2} =IF($L2="\_","\_",AB2/($N2\*24)) | {$AJ$2} =IF($L2="\_","\_",(AA2-AB2)/($N2\*24)) |

The first row of formulas, shown in the table below computes the values of the header row. The second row of formulas computes the average value of PDD, the number of spoken and failed calls per minute, and the number of traffic’s minutes per hour. The last column is used for managing the labels on the horizontal axis. The default horizontal axis does not suit our needs. We show on the time axis the time and the dates are shown only once, when the day changes occur. We wish to display all day changes even when the labels are skipped due to the density. The default time axis does not allow this control. This is the reason we introduced the last curve values of which replace the labels of the horizontal axis.

PDD is computed by dividing the sum of setup times over the number of answered calls. The values of [spoken/m] and [failed/m] are the numbers of spoken and failed calls in the interval divided over the duration of the interval in minutes. The value of [minutes/h] is the sum of duration of calls in minutes divided over the length of the interval in hours. For values of PDD as well as for the values of spoken and failed calls (per minute), two factors, 50 and 100, are introduced to scale their curves to a degree of grandeur comparable to the curve of minutes per hour. These factors are shown in the headers. For instance PDD x50 header means that if the PDD value is equal to 20 seconds, on the chart, the curve will be on the level of 1000 points of the vertical axis. Similarly a level of 1800 points on the vertical axis for the curve of failed calls means 18 failed calls per minute, because the display factor of failed/m is of x100.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | AK | AL | AM | AN | AO |
| 1 | PDD x50 [16.6s] | spoken/m x100 [22769] | failed/m x100 [27230] | minutes/h [179842m] | UTC+2h [2d22h6m33s] |
| 2 | 898.3113674 | 724.47 | 998.50 | 3035.44 | 0 |
| 1 | {$AK$1} ="PDD x"&$I$59&" ["&TEXT(I46/1000/I42,"0.0\s")&"]" | {$AL$1} ="spoken/m x"&$I$60&" ["&I44&"]" | {$AM$1} ="failed/m x"&$I$60&" ["&(I43+I45)&"]" | {$AN$1} ="minutes/h ["&TEXT(I49/60,"0\m")&"]" | {$AO$1} ="UTC"&TEXT($I$9,"+0\h;-0\h;+0\h")&" ["&IF(I31-I30>1,FLOOR(I31-I30,1)&"d","")&TEXT(I31-I30-INT(I31-I30),"[h]\hm\ms\s")&"]" |
| 2 | {$AK$2} =IF($L2="\_","\_",IF(X2=0,NA(),Z2/X2/1000\*$I$59)) | {$AL$2} =IF($L2="\_","\_",Y2/($N2\*24\*60)\*$I$60) | {$AM$2} =IF($L2="\_","\_",(W2-Y2)/($N2\*24\*60)\*$I$60) | {$AN$2} =IF($L2="\_","\_",(AC2/60)/($N2\*24)) | {$AO$2} =IF($L2="\_","\_",IF(MOD(ROW()-2,$I$37)=0,0,NA())) |

The labels of the horizontal axis are displayed as values of a curve having a constant level equal to 0. This curve corresponds to column AO of the table above. The header of this column shows the time zone of the chart followed with the duration of the time scale in days, hours, minutes, and seconds. Recall that $I$37 contains the [label every] value. The values of the curve under AO column are equal to 0 once $I$37 number of times and are [Not Available] the rest of the time. The not-available values skip the points while drawing the curve, and therefore also the label associated to the curve. This is the way how we manage the density of labels. By knowing at which point the label is displayed and where they are skipped, we can make sure that labels carrying extra information (when the day change occurs) are always displayed. The label construction formulas will be shown in the next section.

Now when the input range of the chart is created, we will introduce formulas for validation of the values of this table by comparing them again with the raw data.

Sum of the products of the durations of the intervals with their costs per hour gives us the total cost. We compare this total with the one obtained from the raw data. The table below shows a negligible precision error of 10 zeroes after the decimal point.

Sum of the products of the durations of the intervals (i.e. of the delta values located under the column N) with their margins per hour gives us the total margin which must be equal to the total revenue (obtained from the raw input table) minus the total cost (also from the raw table).

Similarly the sum of products of deltas with the rate of spoken calls, with the rate of failed calls, and with the rate of traffic minutes per hour, will give the total of spoken calls, the total of failed calls, and the total of the traffic. Note that we compensate the sum of the product of the number of calls with the [call factor] stored in $I$60 cell.

Finally the total of all PDD values (in milliseconds) is sum of products of the average PDD of the interval with the number of answered calls in the interval (stored under X column).

|  |  |  |  |
| --- | --- | --- | --- |
|  | H | I |  |
| $I$59 | PDD factor | 50 | {$I$59} 50 |
| $I$60 | call factor | 100 | {$I$60} 100 |
| $I$61 |   |   |   |
| $I$62 | ok Cost | -10.21189605 | {$I$62} =LOG(ABS(I48-SUMPRODUCT(AI:AI,N:N)\*24),10) |
| $I$63 | ok Margin | -9.590950718 | {$I$63} =LOG(I47-I48-SUMPRODUCT(AJ:AJ,N:N)\*24,10) |
| $I$64 | ok PDD | TRUE | {$I$64} =I46=SUMPRODUCT(AK:AK,X:X)\*1000/$I$59 |
| $I$65 | ok spoken | TRUE | {$I$65} =I44=SUMPRODUCT(N:N,AL:AL)\*24\*60/$I$60 |
| $I$66 | ok failed | TRUE | {$I$66} =(I43+I45)=SUMPRODUCT(N:N,AM:AM)\*24\*60/$I$60 |
| $I$67 | ok duration | TRUE | {$I$67} =I49=SUMPRODUCT(AN:AN,N:N)\*24\*60 |

# Time axis labels

The time axis label set under column AH is constructed as follows. If it is the first visible label (checking under column AO), then display the earliest date in the local time zone ($I$30+$I$9/24) in yy-mm-dd hh:mm:ss format. If it is the last visible label (checking again in AO column), then display the latest date in the local time zone ($I$31+$I$9/24) in the same full format. If it is not the first or last dates of the time interval, display nothing. Append to the above the time computed in AE column, which is the middle of the interval. Display the time in month day and day of week format followed by hour and minutes if the current visible date is a new day (compared with the previous visible day), otherwise display the time in hh:mm format only. Whether the time corresponds to a new day or no is computed under column AF. In column AF we check whether the date’s day in AE column is new (INT(AE2)<>MAX(AF$1:AF1) or whether the current label is going to be the last visible one (COUNT(AO$1:AO2)=COUNT(AO:AO)). In these cases we display the month, the day of the month, and the day of the week in addition to the time. Finally the time label ends with the value of the delta computed in days, hours, months, or seconds depending on its width. The text corresponding to the width of the interval is computed under column AG.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | AE | AF | AG | AH | AO |
| 1 |   |   |   |   | UTC+2h [2d22h6m33s] |
| 2 | 13-09-28 17:21:34.44 | 13-09-28 | Δ2h | ┌13-09-28 16:09:52┐ Sep 28 Saturday 17:21 Δ2h | 0 |
| 2 | {$AE$2} =IF($L2="\_","\_",M2+N2/2+$I$9/24) | {$AF$2} =IF($L2="\_","\_", IF(ISNUMBER(AO2), IF( OR(INT(AE2)<>MAX(AF$1:AF1),COUNT(AO$1:AO2)=COUNT(AO:AO)),INT(AE2) ) ) ) | {$AG$2} =IF($L2="\_","\_","Δ" & IF( N2>1, TEXT(N2,"0\d"), IF( N2>1/24, TEXT(N2\*24,"0\h"), IF( N2>1/(24\*60), TEXT(N2\*24\*60,"0\m"), TEXT(N2\*24\*60\*60,"0\s" ))))) | {$AH$2} =IF($L2="\_","\_", IF(COUNT(AO$1:AO2)=1, TEXT($I$30+$I$9/24,"┌yy-mm-dd hh:mm:ss┐ "), IF(COUNT(AO$1:AO2)=COUNT(AO:AO), TEXT($I$31+$I$9/24,"└yy-mm-dd hh:mm:ss┘ "), "")) & TEXT(AE2, IF(AF2,"mmm d dddd hh:mm", "hh:mm")) & REPT(" ",MAX(3\*(5-LEN(AG2)),1)) &AG2 ) | {$AO$2} =IF($L2="\_","\_",IF(MOD(ROW()-2,$I$37)=0,0,NA())) |

The columns between columns AH and AO are discussed in the previous section and are skipped in the above table.

# Chart construction

The capture below shows the data source of the chart. The data source range of the chart will be changed by a VBA script before generating and mailing the image file. This step is carried out for the cases when the user changes the number of intervals. The mailing VBA script will be presented later in this document (see sections 14 and 15).



The chart looks as follows:



The vertical axis on the left side is associated to the areas of cost (in pink) and margin (in green). All other curves, minutes per hour, PDD, spoken per minute, and failed per minute, are associated to the blue vertical axis on the right.

Gradients are used for the areas to underline the lack of precision of the fresh data with pale colors and more consistent statistics for the old data with more opaque colors.



Similar gradients are used for the curves.



The labels on the curve of minutes per hour are shown when they exceed the level of 6000 minutes per hour.



Labels of cost are shown when the cost exceeds CHF 150 per hour.



On the above chart, we see an unusually high cost of 400 CHF/h on October 1 around 9 o’clock. This is turned to be a fraud attempt, successfully detected and blocked by the anti-fraud system.



The chart with a longer history looks as follows.



The October 1st morning peak is still visible but is less noticeable as it melted down in the surrounding normal pattern of calls when the interval became sufficiently long (2h on the above chart while it was of 12 minutes on the previous chart).

# Evolution of the chart

The following GIF animation shows the 13 frames representing about 12 hours of an evolution of the chart from 2013-10-01 21:36:03 through 2013-10-02 09:53:05.



The GIF file is generated with Image Magick’s convert command line tool out of the pool of the PNG files created and emailed by the Excel application.

$ ls -1 \*.png

130922'165343\_131001'213603\_131001'213733.png

130922'165343\_131001'223725\_131001'223848.png

130922'165343\_131001'233804\_131001'234001.png

130922'165343\_131002'003955\_131002'004128.png

130922'165343\_131002'014012\_131002'014244.png

130922'165343\_131002'024146\_131002'024437.png

130922'165343\_131002'034454\_131002'034620.png

130922'165343\_131002'044611\_131002'044757.png

130922'165343\_131002'054613\_131002'054928.png

130922'165343\_131002'064630\_131002'065039.png

130922'165343\_131002'075038\_131002'075203.png

130922'165343\_131002'085152\_131002'085312.png

130922'165343\_131002'095305\_131002'095423.png

$ convert -delay 25 \*.png out.gif

$

# Parameters for chart building and emailing

Here we present the Excel worksheet area which computes and provides the data to the VBA script responsible for the generation and mailing of the traffic charts. In the range from I95 through I100 we compute the current chart name and the folder name where the chart images are stocked. The chart name consists of three date and time values separated by the underline signs. The first date/time value is that of the first call appearing in the raw data input, the second one corresponds to the last call in the raw data input, and the third one is the date and the time of the generation of the chart.

|  |  |  |  |
| --- | --- | --- | --- |
|  | H | I | Formula |
| $I$95 | sheet | J:\run\folders\130916 Excel VBA ADODB ODBC MySQL\3\[Book40.xlsm]data sheet | {$I$95} =CELL("filename",I95) |
| $I$96 | folder | J:\run\folders\130916 Excel VBA ADODB ODBC MySQL\3\ | {$I$96} =LEFT(I95,SEARCH("[",I95)-1) |
| $I$97 | file | Book40.xlsm | {$I$97} =REPLACE(LEFT(I95,SEARCH("]",I95)-1),1,SEARCH("[",I95),"") |
| $I$98 | charts | Book40.xlsm\_Charts | {$I$98} =I97&"\_Charts" |
| $I$99 | [chart\_path](#RANGE!I99) | J:\run\folders\130916 Excel VBA ADODB ODBC MySQL\3\Book40.xlsm\_Charts | {$I$99} =I96&I98 |
| $I$100 | [chart\_name](#RANGE!I100) | 130928'160952\_131001'141625\_131003'114822.png | {$I$100} =TEXT(I30+I9/24,"yymmdd'hhmmss")&"\_"&TEXT(I31+I9/24,"yymmdd'hhmmss")&"\_"&TEXT(NOW(),"yymmdd'hhmmss")&".png" |
| $I$101 |   |   |   |
| $I$102 | smtp way | $I$114 | {$I$102} =CHOOSE(FLOOR(RAND()\*3,1)+1,I114,I122,I130) |
| $I$103 | [smtp\_server](#RANGE!I103) | smtp.mail.yahoo.com | {$I$103} =OFFSET(INDIRECT(I$102),ROWS(I$103:I103),0) |
| $I$104 | [smtp\_port](#RANGE!I104) | 25 | {$I$104} =OFFSET(INDIRECT(I$102),ROWS(I$103:I104),0) |
| $I$105 | [smtp\_authenticate](#RANGE!I105) | 1 | {$I$105} =OFFSET(INDIRECT(I$102),ROWS(I$103:I105),0) |
| $I$106 | [smtp\_ssl](#RANGE!I106) | TRUE | {$I$106} =OFFSET(INDIRECT(I$102),ROWS(I$103:I106),0) |
| $I$107 | [smtp\_user](#RANGE!I107) | d9a.monitor@yahoo.com | {$I$107} =OFFSET(INDIRECT(I$102),ROWS(I$103:I107),0) |
| $I$108 | [smtp\_password](#RANGE_I108) | \*\*\*\* | {$I$108} =OFFSET(INDIRECT(I$102),ROWS(I$103:I108),0) |
| $I$109 | [smtp\_from](#RANGE!I109) | d9a Monitor Yahoo.com <d9a.monitor@yahoo.com> | {$I$109} =OFFSET(INDIRECT(I$102),ROWS(I$103:I109),0) |
| $I$110 | [smtp\_to](#RANGE!I110) | emin.gabrielyan@switzernet.com;emin.gabrielyan@gmail.com;intarnet2@yahoo.com | {$I$110} emin.gabrielyan@switzernet.com;emin.gabrielyan@gmail.com;intarnet2@yahoo.com |
| $I$111 | [smtp\_subject](#RANGE!I111) | [5'd98'1 traffic] | {$I$111} [5'd98'1 traffic] |
| $I$112 | [smtp\_error](#RANGE!I112) |   |   |
| $I$113 |   |   |   |
| $I$114 | smtp way | $I$114 | {$I$114} =CELL("address",I114) |
| $I$115 | smtp server | smtp.mail.yahoo.com |   |
| $I$116 | smtp port | 25 |   |
| $I$117 | smtp authenticate | 1 |   |
| $I$118 | smtp ssl | TRUE |   |
| $I$119 | smtp user | d9a.monitor@yahoo.com |   |
| $I$120 | smtp password | \*\*\*\* |   |
| $I$121 | smtp from | d9a Monitor Yahoo.com <d9a.monitor@yahoo.com> |   |
| $I$122 | smtp way | $I$122 | {$I$122} =CELL("address",I122) |
| $I$123 | smtp server | smtp.googlemail.com |   |
| $I$124 | smtp port | 25 |   |
| $I$125 | smtp authenticate | 1 |   |
| $I$126 | smtp ssl | TRUE |   |
| $I$127 | smtp user | d9a.monitor@gmail.com |   |
| $I$128 | smtp password | \*\*\*\* |   |
| $I$129 | smtp from | d9a Monitor Gmail.com <d9a.monitor@gmail.com> |   |
| $I$130 | smtp way | $I$130 | {$I$130} =CELL("address",I130) |
| $I$131 | smtp server | smtp.switzernet.com |   |
| $I$132 | smtp port | 587 |   |
| $I$133 | smtp authenticate | 1 |   |
| $I$134 | smtp ssl | FALSE |   |
| $I$135 | smtp user | d9a.monitor@smtp.switzernet.com |   |
| $I$136 | smtp password | \*\*\*\* |   |
| $I$137 | smtp from | d9a Monitor Switzernet.com <d9a.monitor@switzernet.com> |   |

The meanings of the blue SMTP fields are self-explanatory.

[[smtp\_server](#RANGE!I103)] is the server the Excel application must connect to in order to transmit the email

[[smtp\_port](#RANGE!I104)] the port of the SMTP daemon on the server, usually 25 or 587

[[smtp\_authenticate](#RANGE!I105)] equal to 1 or 0 depending whether an authentication (login) is required or not

[[smtp\_ssl](#RANGE!I106)] equals to True or False depending whether the connection must be encrypted or not

[[smtp\_user](#RANGE!I107)] the user name of the SMTP session if the authentication is required

[[smtp\_password](#RANGE_I108)] the password of the SMTP session if the authentication is required

[[smtp\_from](#RANGE!I109)] the email address and the display name to be provided in the email header for the current SMTP server

[[smtp\_to](#RANGE!I110)] the list of the recipients separated by semicolons

[[smtp\_subject](#RANGE!I111)] the subject prefix, the filename of the attachment will be appended to the subject. The subject prefix contains the routing tag of our project management system

[[smtp\_error](#RANGE!I112)] this string collects all SMTP error reports. Upon a successful email transmission all past errors will be sent in the body of the email and this field will be cleaned.

[smtp way] is generated randomly so as to pick up one of the three available servers for transmission of the email. The three SMTP accounts are independent, and even in case of a failure of our email server the emails will go through and the employees will be alerted. Private email addresses of key employees must be used as well if the hypothesis of a failure of the company email server is assumed. If you use this Excel file with less SMTP accounts, copy the parameters a valid account over the yellow input fields of more than one account. You must replace the asterisks in yellow [[smtp\_password](#RANGE_I108)] input fields with the true passwords. The Excel file must be stored securely as it contains authentication information.

In the next section we show the VBA script relying on the parameters of this section for generating and emailing the Excel charts to the recipients provided in [[smtp\_to](#RANGE!I110)] field.

# Chart emailing VBA script

|  |  |
| --- | --- |
| Code\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Comments |
| Sub Oval\_SMTP\_Click() Chart\_Source Chart\_EmailEnd Sub | Chart generation and the emailing can be triggered by clicking on a circle labeled “SMTP”. Otherwise the mailing is carried out by a periodic script which will be presented in following sections. |
| Sub Chart\_Source()  range("[chart\_range](#RANGE!I35)").Calculate Charts(1).SetSourceData Source:=range(range("[chart\_range](#RANGE!I35)").Value) End Sub | The Excel [chart range](#RANGE!I35) is updated in case user has changed the number of intervals. The value of [[chart\_range](#RANGE!I35)] is computed by Excel formulas as a function of the number of intervals. |
| Sub Chart\_Email()  Dim [chart\_path](#RANGE!I99) As String Dim [chart\_name](#RANGE!I99) As range [chart\_path](#RANGE!I99) = range("[chart\_path](#RANGE!I99)").Value Set [chart\_name](#RANGE!I99) = range("[chart\_name](#RANGE!I99)")  If Len(Dir([chart\_path](#RANGE!I99), vbDirectory)) = 0 Then MkDir [chart\_path](#RANGE!I99) End If [chart\_name](#RANGE!I99).Calculate Charts(1).Export [chart\_path](#RANGE!I99) & "\" & [chart\_name](#RANGE!I99).Value  SMTP ([chart\_path](#RANGE!I99) & "\" & [chart\_name](#RANGE!I99).Value) End Sub | The folder name for stocking the charts is computed in the Excel worksheet’s [[chart\_path](#RANGE!I99)] cell. This is a folder in the same location as the Excel file with the same name as the Excel file but suffixed by “\_Chart” text. We verify if the folder exists. If it does not, we create it. We refresh the value of the filename of the chart (as it contains the current time) by recalculating cell [[chart\_name](#RANGE!I99)]. The PNG file is created by the Export method and it is then sent with SMTP subroutine shown below. |
| Sub SMTP(filename As String)  Worksheets(1).Unprotect  Dim iMsg As Object Dim body As String  Set iMsg = CreateObject("CDO.Message")  iMsg.Configuration.Load -1 | Beginning of the subroutine emailing the image file of a chart. The filename is provided as a parameter. The worksheet is unprotected as the subroutine will write to the cells locked for the user. Object “CDO.Message” holds all SMTP connection and media values and is responsible for sending the email. The Load method of .Configuration object resets all values to defaults.  |
|  With iMsg.Configuration.Fields .Item("http://schemas.microsoft.com/cdo/configuration/sendusing") = 2  .Item("http://schemas.microsoft.com/cdo/configuration/smtpserver") \_ = range("[smtp\_server](#RANGE!I103)").Value  .Item("http://schemas.microsoft.com/cdo/configuration/smtpserverport") \_ = range("[smtp\_port](#RANGE!I104)").Value  .Item("http://schemas.microsoft.com/cdo/configuration/smtpauthenticate") \_ = range("[smtp\_authenticate](#RANGE!I105)").Value  .Item("http://schemas.microsoft.com/cdo/configuration/smtpusessl") \_ = range("[smtp\_ssl](#RANGE!I106)").Value  .Item("http://schemas.microsoft.com/cdo/configuration/sendusername") \_ = range("[smtp\_user](#RANGE!I107)").Value  .Item("http://schemas.microsoft.com/cdo/configuration/sendpassword") \_ = range("[smtp\_password](#RANGE_I108)").Value  .Update End With | The fields of the Configuration object are self-explanatory. Here we load the current values of cells [[smtp\_server](#RANGE!I103)], [[smtp\_port](#RANGE!I104)], [[smtp\_authenticate](#RANGE!I105)], [[smtp\_ssl](#RANGE!I106)], [[smtp\_user](#RANGE!I107)], and [[smtp\_password](#RANGE_I108)] from the Excel worksheet into the Configuration of the CDO.Message object. |
|  body = ""  If Len(range("[smtp\_error](#RANGE!I112)").Value) > 0 Then body = body & "\*SMTP Errors\*:" \_ & range("[smtp\_error](#RANGE!I112)").Value & vbNewLine & vbNewLine End If  If Len(range("[db\_error](#RANGE!I8)").Value) > 0 Then body = body & "\*MySQL Errors\*:" \_ & range("[db\_error](#RANGE!I8)").Value & vbNewLine & vbNewLine End If If Len(range("[chunk\_report](#RANGE!I91)").Value) > 0 Then body = body & "\*Chunk Synchronizations\*:" \_ & range("[chunk\_report](#RANGE!I91)").Value & vbNewLine & vbNewLine End If | Preparing the body of email. If SMTP errors occurred in past, add header “SMTP Error” followed by the errors. If MySQL errors occurred add them into the body. Similarly add the [[chunk\_report](#RANGE!I91)] messages into the body. |
|  With iMsg .To = range("[smtp\_to](#RANGE!I110)").Value .CC = "" .BCC = "" .From = range("[smtp\_from](#RANGE!I109)").Value .Subject = range("[smtp\_subject](#RANGE!I111)").Value & " " & Dir(filename) .TextBody = body .AddAttachment filename End With | Compose the email. Subject is the prefix provided in [[smtp\_subject](#RANGE!I111)] worksheet cell followed by the base name of the file (without the path). |
|  On Error GoTo Error\_Handler\_1 iMsg.Send On Error GoTo 0 | Send the email, but if there are errors, process them under Error Handler 1. |
|   range("[smtp\_error](#RANGE!I112)").Value = "" range("[db\_error](#RANGE!I8)").Value = "" range("[chunk\_report](#RANGE!I91)").Value = "" Worksheets(1).Protect Password:="" Exit Sub | As the errors accumulated in cells [[smtp\_error](#RANGE!I112)], [[db\_error](#RANGE!I8)], and [[chunk\_report](#RANGE!I91)] are successfully communicated to the users, the contents of these cells are cleared. The worksheet is protected and the subroutine ends. |
| Error\_Handler\_1: range("[smtp\_error](#RANGE!I112)").Value = range("[smtp\_error](#RANGE!I112)").Value \_ & " On " & Format(Now, "yy-mm-dd hh:mm:ss") \_ & " error " & Err.Number & " occured" \_ & " while sending email via " & range("[smtp\_server](#RANGE!I103)").Value \_ & " {" & Replace(Err.Description, vbCrLf, "\_") & "}." On Error GoTo 0 Worksheets(1).Protect Password:="" | We land under this label if error occurs during the transmission of the email. The error time, error number, and its description is appended into the field [[smtp\_error](#RANGE!I112)]. The worksheet is protected and the subroutine ends. The errors accumulated in cells [[smtp\_error](#RANGE!I112)], [[db\_error](#RANGE!I8)], and [[chunk\_report](#RANGE!I91)] are not cleared. |
| End Sub | End of the subroutine emailing the image file of a chart. |

# The scheduler

Here are the parameters used by the scheduler.

|  |  |  |  |
| --- | --- | --- | --- |
|  | H | I | Formula |
| $I$139 | schedule\_period | 0:15:00 |   |
| $I$140 | schedule\_running | TRUE |   |
| $I$141 | schedule\_next | 2013-10-01 14:28:46 |   |
| $I$142 | schedule\_reporting | 1:00:00 |   |
| $I$143 | schedule\_report | 2013-10-01 14:43:01 |   |
| $I$144 |   |   |   |
| $I$145 | error window |   | {$I$145} =IF([db\_error](#RANGE!I8)&[chunk\_report](#RANGE!I91)&[smtp\_error](#RANGE!I112)<>"","..."&RIGHT(IF([db\_error](#RANGE!I8)<>""," >>MySQL Errors:"&[db\_error](#RANGE!I8),"")&IF([chunk\_report](#RANGE!I91)<>""," >>Sync Errors:"&[chunk\_report](#RANGE!I91),"")&IF([smtp\_error](#RANGE!I112)<>""," >>Sync Errors:"&[smtp\_error](#RANGE!I112),""),251)&"!","") |

[[schedule\_period](#RANGE!I139)] defines the frequency at which the application must download the data from the remote MySQL server.

[schedule\_running] indicates whether the scheduler is active or not. The scheduler is activated and deactivated by clicking on the circle labeled “Scheduled” or “Suspended” depending on the value of this cell.

[schedule\_next] is the time the next retrieval is going to take place.

[schedule\_reporting] defines the frequency at which the reports with charts are emailed to the recipients.

[schedule\_report] is the time after which the next report will be sent.

[error window] is a concatenation of messages that will appear on the screen of the worksheet.

Here is the code of the scheduler.

|  |  |
| --- | --- |
| Code\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Comments |
| Sub Oval\_Schedule\_Click() Schedule  Dim label As String If range("schedule\_running").Value Then label = "Scheduled" Else label = "Suspended" End If Worksheets(1).Unprotect Worksheets(1).Shapes("Oval 10").TextFrame.Characters.Text = label Worksheets(1).Protect Password:="" End Sub | When clicking on the circle button of the scheduler, run the scheduler and change the label of the circle according to the value of cell [schedule\_running]. |
| Sub Schedule() Worksheets(1).Unprotect Dim schedule\_running As range Set schedule\_running = range("schedule\_running")  Worksheets(1).Unprotect schedule\_running.Value = Not (schedule\_running.Value) | Beginning of Schedule subroutine. This subroutine is called each time the schedule button is clicked. At the beginning of the subroutine we inverse the value of [schedule\_running] cell. |
|  If (schedule\_running.Value) Then Periodic | If the schedule must be running, run the Periodic subroutine. This subroutine does some stuff and calls itself for a re-execution after a while. |
|  Else On Error GoTo Error\_Handler\_1 Application.OnTime range("schedule\_next").Value, "Periodic", , False On Error GoTo 0 End If | If the status of the scheduler is changed to False, find out in [schedule\_next] cell the next execution time of the scheduled “Periodic” subroutine and remove it. If errors occur while removing, go to the error handler. |
|   Worksheets(1).Protect Password:=""Exit Sub | Protecting the worksheet and quitting the subroutine . |
| Error\_Handler\_1: MsgBox "nothing to stop" On Error GoTo 0 Worksheets(1).Protect Password:="" | We land here if the scheduled procedure “Periodic” is not found. Such a situation will occur if you exit the Excel file without stopping the scheduler and then try to stop the procedure after reopening the Excel file. A message will be displayed that there is nothing to stop, the worksheet will be protected, and the subroutine will quit. |
| End Sub | End of the schedule subroutine |
| Sub Periodic() Worksheets(1).Unprotect  Dim schedule\_next As range Dim schedlue\_report As range Set schedule\_next = range("schedule\_next") Set schedule\_report = range("schedule\_report") | Beginning of the subroutine periodic. The subroutine does the actual stuff and also, upon the completion, schedules itself for the next execution. We begin by creating two references to cells [schedule\_next] and [schedule\_report] of the worksheet. These are the next schedule and next emailing times. |
|  Sync\_Calls Refresh\_Calls | Do synchronization of calls, i.e. look at the old chunks and compare the number of calls on the remote server with the number of calls collected in the worksheet, download the missing calls if needed, and refresh the calls, i.e. download the calls from the last download until the current time (minus small margin). |
|  Sheets(Replace(range("[worksheet\_name](#RANGE!I24)").Value, "'", "")).Unprotect range(range("[worksheet\_name](#RANGE!I24)").Value & "!A:F").NumberFormat = "General" range(range("[worksheet\_name](#RANGE!I24)").Value & "!C:C").NumberFormat = "yy-mm-dd hh:mm:ss" | After copying the remote record sets to input columns of the worksheet, the number formatting of the worksheet columns change unpredictably. We correct the formatting here, for esthetic purposes only.  |
|  If IsEmpty(schedule\_report) Or Now() > schedule\_report.Value Then Reporting Worksheets(1).Unprotect schedule\_report.Value = Now() + range("schedule\_reporting").Value End If | Launch the reporting (i.e. chart emailing) if it is already the time. Schedule the next reporting time in the [schedule\_reporting] cell as current time plus the [schedule\_reporting] interval. Note that [schedule\_reporting] cell does not define the exact interval between the reports but the minimal period between two consecutive reports. |
|     Worksheets(1).Unprotect schedule\_next.Value = Now() + range("schedule\_period").Value Application.OnTime schedule\_next.Value, "Periodic"  Worksheets(1).Protect Password:="" | Schedule the next execution of this subroutine. Save the time of the next execution in [schedule\_next] cell. This value in the worksheet is needed in case we wish to cancel the scheduled execution (by clicking on the schedule button). Before exiting the subroutine protect the worksheet. |
| End Sub | End of the subroutine Periodic |
| Sub Reporting()  Chart\_Source Chart\_Email End Sub | Reporting subroutine refreshes the chart source range (in case the user changed the number of intervals) and proceeds with the email sending. |

# Installation

Make sure that you have MySQL ODBC driver installed.

In your computer’s Start Menu do search for ODBC. Open the ODBC Data Source Administrator. Click on [Drivers] tab:



If MySQL connector is not installed, download and install it from the MySQL web site <http://dev.mysql.com/downloads/connector/odbc/>. Make sure that the driver appears in the ODBC Data Source Administrator.

[Download](code) and open the last version of Excel file. This excel file contains sample data. The data represents only 6 hours and the chart is noisy. The chart will be less noisy when a few days of stats are collected.

In the yellow input cells of SMTP connections enter the right parameters and credentials for each of the three email accounts. All three input data sets must be complete. Therefore copy the same dataset more than once if you use less than three email accounts. Edit [[smtp\_to](#RANGE!I110)] field and replace it with the list of the recipients of the charts of this application. Click on the SMTP button to ensure that the emails are sent successfully. Proceed with this test several times until at least one email via each of three accounts is sent successfully in order to validate all three accounts. Edit [[smtp\_subject](#RANGE!I111)] field to refer to a valid project tag name.

Refer to the next section if you have “User-defined type not defined” compile error.

Fulfill the input yellow cells corresponding to your remote database connection. Click on the Re-sync button. If no error occurred you can activate the scheduler by clicking on the main blue button.

You can reinitialize the resynchronization of the old blocks by deleting the content of cell [[chunk\_last](#RANGE!I70)] highlighted in green. You can completely delete the data in the columns from A to F starting from the 2nd row and launch the procedure from the scratch. You have to delete also the content of the [[stop\_did](#RANGE!I14)] cell if you delete the input data. In order to delete the input data you need to unprotect the worksheet.

# ADODB Reference

For ADODB and CDO objects our Excel file’s VBA uses Microsoft ActiveX Data Objects in the references. Normally you do not need to do anything special. You can see the library under Tools>References of the VBA project window (you must unprotect the worksheet first).



If the reference is not added you may have the following message “Compile error: User-defined type not defined”.



Stop debugging and add the reference via Tools>References window. Note that the reference menu will be unavailable if you do not stop the debug mode. The Microsoft ActiveX Data Objects Library reference must be in the list by default, you only need to activate it by clicking on the checkbox. If you do not have it in the list of available references, then try to install Microsoft Data Access Components Service Pack from <http://www.microsoft.com/en-us/download/details.aspx?id=5793>

# Further work

Attempt the retransmission of the email with a random account immediately if the transmission is failed. In the current version we wait until the next reporting time for the retransmission.

Automatically save the Excel file. Handle the issues of read-only zipped or attached files.

Automatically save as archives upon the change of the month.

# Issues with range calculation in Excel 2013

When moving from Excel 2010 to Excel 2013, run-rime errors occur with the range calculation method. Using the range calculation when the automatic calculation is disabled is apparently not reliable at all. The activations of the automatic calculation must be the overhead to accept.

Below you will find discussions related to the range calculation method used under the recent version of Excel.

Calculate Method of Range Class Failed
<http://www.mrexcel.com/forum/excel-questions/93906-calculate-method-range-class-failed.html>

Run-Time error when you use the Range.Calculate method in Excel
<http://support.microsoft.com/?kbid=292476>

Circular reference does not calculate during Range.Calculate execution in Excel 2002
<http://support.microsoft.com/default.aspx?scid=kb;en-us;827990>

Error message when you use the Range.Calculate method to calculate formulas in an Excel workbook: "Run-time error '1004'"
<http://support.microsoft.com/default.aspx?scid=kb;en-us;825011>

How formula calculations are performed in Excel
<http://support.microsoft.com/default.aspx?scid=kb;en-us;825012>

We report also that the manual calculation raises issues also with range’s copy from record set method. The workaround was unfortunately to remain in the automatic calculation mode.

 Application.Calculation = xlCalculationAutomatic

 range("data\_lines,data\_land").Calculate

 'Application.Calculation = xlCalculationManual

 range(range("data\_land").Value).CopyFromRecordset rs1

# References

MySQL Connector/ODBC Installation
<http://dev.mysql.com/downloads/connector/odbc/>

Microsoft Data Access Components (MDAC) 2.8 SP1
<http://www.microsoft.com/en-us/download/details.aspx?id=5793>

VBA ADODB not found
<http://forums.codeguru.com/showthread.php?372750-VBA-ADODB-not-found>

Excel VBA program stopped executing MySQL query
<http://stackoverflow.com/questions/17500077/excel-vba-program-stopped-executing-mysql-query>

How To Open ADO Connection and Recordset Objects
<http://support.microsoft.com/kb/168336>

Using Excel VBA to Copy an ADODB Recordset to Multiple Worksheets
<http://deepinthecode.com/2013/03/19/using-excel-vba-to-copy-an-adodb-recordset-to-multiple-worksheets/>

Connector/ODBC Programming
<http://dev.mysql.com/doc/refman/5.5/en/connector-odbc-examples-programming.html>

How Do I Use the Connection Object in ADO
<http://msdn.microsoft.com/en-us/library/ms807027.aspx>

Using ADO with Microsoft Visual Basic
[http://msdn.microsoft.com/en-us/library/windows/desktop/ms677497(v=vs.85).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms677497%28v%3Dvs.85%29.aspx)

UNION Syntax
<http://dev.mysql.com/doc/refman/5.6/en/union.html>

How to Send Emails from an Excel Spreadsheet Using VBA Scripts
<http://www.makeuseof.com/tag/send-emails-excel-vba/>

Sending mail from Excel with CDO
<http://www.rondebruin.nl/win/s1/cdo.htm>

VBScript To Send Email Using CDO
<http://www.paulsadowski.com/wsh/cdo.htm>

Exchange Server > Exchange Server 2003 > Exchange Server 2003 SDK June 2007 > Reference > CDOEX > COM Classes > Configuration CoClass
[http://msdn.microsoft.com/en-us/library/exchange/ms870485(v=exchg.65).aspx](http://msdn.microsoft.com/en-us/library/exchange/ms870485%28v%3Dexchg.65%29.aspx)

Dictionary, Encyclopedia and Thesaurus
<http://acronyms.thefreedictionary.com/>

Connecting Excel to a remote MySQL server
<http://www.switzernet.com/3/public/130715-excel-to-remote-mysql/>

Properties of Excel to remote MySQL connection
<http://www.switzernet.com/3/public/130715-excel-mysql-connections/>

Creating a vendor cost on-line monitoring chart
<http://switzernet.com/3/public/130716-vendor-cost-monitor/>

Retrieval of hourly cost revenue and traffic
<http://switzernet.com/3/public/130723_cost_revenue_and_traffic_excel_mysql/>

Incremental retrieval and visualization with Excel MySQL connector
<http://www.unappel.ch/2/public/130807-excel-vba-mysql-CDR_Vendors/>

Traffic cost and revenue monitoring
<http://switzernet.com/3/public/130915-mysql2html2excel2chart2smtp/>

VBA code to save workbook with no prompt
<http://www.mrexcel.com/forum/excel-questions/67564-visual-basic-applications-code-save-workbook-no-prompt.html>

Convert, Edit, and Compose Images
<http://www.imagemagick.org/>

# Acronyms

ACD, Average Call Duration

ADO, ActiveX Data Objects

ADODB, ADO Database

ALOC, Average Length of Conversation

ASR, Answer Seizure Ration

BASH, Bourne Again Shell

CDO, Collaboration Data Objects

CDOEXM, Collaboration Data Objects for Exchange Management

COM, Component Object Model

DSN, Data Source Name

GIF, Graphic Interchange Format

HTML, Hyper Text Markup Language

MySQL, My SQL

ODBC, Open Database Connectivity

PC, Personal Computer

PDD, Post Dial Delay

PNG, Portable Network Graphics

SDK, Software Development Kit

SIP, Service Initiation Protocol

SMTP, Simple Mail Transfer Protocol

SP, Service Pack

SQL, Structured Query Language

SSL, Secure Sockets Layer

VBA, Visual BASIC for Applications

VoIP, Voice over IP

# Legal

Copyright © 2013 by Emin Gabrielyan and switzernet.com

MS Word version of [this document](index7.docx)

END of This Document