Traffic cost and revenue monitoring

Emin Gabrielyan  
2013-09-14

# Table of contents

1. Table of contents 1

2. Introduction 1

3. Retrieval and saving in HTML 2

4. Mailing of charts 8

5. Linking Excel to the HTML files 12

6. Count analysis 16

7. Computing the period of the entire traffic 18

8. Equations of the exponentially increasing intervals 19

9. Building the increasing intervals 23

10. Building the chart input 29

11. The horizontal axis labels 34

12. The chart 36

13. Code data 38

14. VBA scripts 41

15. Evolution demo 46

16. Installation 48

17. References 49

18. Acronyms 49

19. Legal 50

# Introduction

This document presents a module for real time visual monitoring of the voice traffic. This module consists of an Excel file and of a bash script. It is programmed to periodically mail the traffic charts to a list of recipients. The chart is constructed in Excel.

This monitoring method uses bash and Excel VBA scripts. It retrieves data via bash MySQL command, processes the query results in the Excel, creates a chart, and mails the resulting images to recipients via a bash script. Recent CDR chunks are periodically collected into a local HTML file with MySQL and a bash script. The script keeps a log of requests so as to request only chunks coming after the last retrieval. The HTML files of call records (alimented by the MySQL flow) are being periodically loaded into an Excel file and a visual chart of the traffic pattern is constructed. To avoid a huge HTML file, the flow is split across three input html files, which are being rotated, such that, once in a while, a new file enters into the rotation and the oldest file quits the rotation. An Excel VBA script periodically saves the chart into PNG files, and a bash script emails the PNG files to the recipients.

The chart shows six key traffic parameters, the overall traffic’s hourly cost and the hourly margin (forming together the revenue), the average PDD, the rates of successful and failed conversations per minute, and the total amount of traffic minutes per hour. Monitoring of the cost permits to trigger fraud alerts if the hourly cost exceeds a critical limit.

The entire period of the available pattern is broken into a number of intervals. Users are usually interested in greater details for the recent traffic and in averaged indicative values for the older data. Often two or more simultaneous charts are needed to meet the user needs, one with large intervals (e.g. monthly, weekly, or daily) and another with short intervals (such as hourly or minutely). To meet both aims in the single chart we suggested exponentially increasing intervals breaking the entire period of the traffic into a constant number of intervals, such as the most recent interval is the shortest and each preceding interval is longer than its next by a constant factor, so as the oldest interval is the largest one. In such a way, on the same chart, the user obtains greater details for the fresh data, for example with 20-minute long intervals, and averaged reference values for old data, for example with 3 or 4-hour long intervals. As an input, user needs to provide only the factor of the largest (the earliest) interval size over the shortest (the most recent) one. For example, such a factor can be equal to 10. In order to find the breakout, we solved the following equation.

For that purpose we solved the following equation:

Such that:

Where *n*+1 is the number of intervals, *a* is the length of the interval and depends on its index ranging from 0, for the most recent, to *n* for the earliest, *P* is the entire period of call records, *f* is the increase factor between two adjacent intervals, and *k* is the user input defining the factor between the largest and smallest intervals.

With a given *n*, *P*, *k*, we find the values of all *a* and *f*.

The following sections describe our method in more details.

# Retrieval and saving in HTML

In the following table I describe the code which is in charge of retrieving the chunks of recent call records into an HTML file. The same script is also responsible for mailing the PNG files generated by the Excel file after processing the data collected in the HTML files.

|  |  |
| --- | --- |
| #Code  #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_# | Comments  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| #!/bin/bash  # Copyright (c) 2013 by Emin Gabrielyan and switzernet.com  to=""  to="$to,emin.gabrielyan@switzernet.com"  to="$to,emin.gabrielyan@gmail.com"  to="$to,d9a.monitor@switzernet.com"  to="$to,d9a.monitor@yahoo.com"  to="$to,elen.virabyan@intarnet.com"  to="$to,nicolas.bondier@switzernet.com"  tag="[1'd9a'1 monitor]" | Defining the list of the recipients and the subject tag |
| server=$(head -1 connect.txt | tail -1 | tr -d "\r\n")  user=$(head -2 connect.txt | tail -1 | tr -d "\r\n")  password=$(head -3 connect.txt | tail -1 | tr -d "\r\n")  database="porta-billing" | Loading the MySQL connection parameters |
| fi1=''  fi1="$fi1"'1 as "Table", concat("\_",CLI) as "CLI", concat("\_",CLD) as "CLD"'  fi1="$fi1"', setup\_time, connect\_time, disconnect\_time, disconnect\_cause'  fi1="$fi1"', revenue, charged\_amount'  fi1="$fi1"', concat("\_",account\_id) as "account\_id", charged\_quantity' | Constructing the The set of fields of a MySQL request for retrieving the answered call records |
| fi2=''  fi2="$fi2"'0 as "Table", concat("\_",CLI) as "CLI", concat("\_",CLD) as "CLD"'  fi2="$fi2"', setup\_time, connect\_time, "\_" as "disconnect\_time", disconnect\_cause'  fi2="$fi2"', "\_" as "revenue", "\_" as "charged\_amount"'  fi2="$fi2"', concat("\_",account\_id) as "account\_id", "\_" as "charged\_quantity"' | The set of fields of a MySQL request for retrieving the failed call records |
|  | Continued… |

You will notice that we add a column [Table] to distinguish from each other the two tables of answered and failed calls. For the table of answered calls the value of this column is equal to 1 and for the table of failed calls the value of the column is equal to 0. While in the data base the records of answered and failed calls are stored in separate tables, we merge them into one flow when saving into the HTML file. The fields which are present only in the table of answered calls are added with an empty value “\_” in the table of failed calls. These fields are [disconnect time], [revenue], [charged amount], and [charged quantity]. We also add a prefix “\_” to all text values that risk to be interpreted as numerical values by the Excel file. These are the phone numbers stored in [CLI], [CLD], and [account id] fields.

The HTML outputs of queries using these fields result into the following tables. Here is the output of the query running on the table of answered calls.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table** | **CLI** | **CLD** | **setup\_time** | **connect\_time** | **disconnect\_time** | **disconnect\_cause** | **revenue** | **charged\_amount** | **account\_id** | **charged\_quantity** |
| 1 | \_412255\* | \_414381\* | 15000 | 2013-09-13 17:35:12 | 2013-09-13 17:36:33 | 16 | 0.03000 | 0.00905 | \_412255\* | 81 |
| 1 | \_412155\* | \_417990\* | 14000 | 2013-09-13 17:36:25 | 2013-09-13 17:36:34 | 16 | 0.01650 | 0.01340 | \_412155\* | 9 |
| 1 | \_412255\* | \_2125233\* | 16000 | 2013-09-13 17:21:31 | 2013-09-13 17:36:36 | 16 | 0.00000 | 0.19608 | \_412255\* | 905 |
| 1 | \_412255\* | \_334500\* | 27000 | 2013-09-13 17:36:32 | 2013-09-13 17:36:36 | 16 | 0.01000 | 0.00019 | \_412255\* | 4 |
| 1 | \_412155\* | \_413242\* | 20000 | 2013-09-13 17:26:05 | 2013-09-13 17:36:36 | 16 | 0.00000 | 0.07046 | \_412155\* | 631 |
| 1 | \_412155\* | \_417644\* | 35000 | 2013-09-13 17:36:29 | 2013-09-13 17:36:37 | 16 | 0.01467 | 0.01467 | \_412155\* | 8 |
| 1 | \_412450\* | \_4424768\* | 13000 | 2013-09-13 17:13:15 | 2013-09-13 17:36:49 | 16 | 0.48000 | 0.13433 | \_412450\* | 1414 |
| 1 | \_412255\* | \_171926\* | 11000 | 2013-09-13 17:31:58 | 2013-09-13 17:36:49 | 16 | 0.00000 | 0.02910 | \_412255\* | 291 |
| 1 | \_413250\* | \_413273\* | 17000 | 2013-09-13 17:22:49 | 2013-09-13 17:36:52 | 16 | 0.00000 | 0.09414 | \_413250\* | 843 |

Here is the output of the second query carried out on the table of the failed calls.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table** | **CLI** | **CLD** | **setup\_time** | **connect\_time** | **disconnect\_time** | **disconnect\_cause** | **revenue** | **charged\_amount** | **account\_id** | **charged\_quantity** |
| 0 | \_181825\* | \_374237\* | 0 | 2013-09-13 17:36:35 | \_ | 16 | \_ | \_ | \_181872\* | \_ |
| 0 | \_00333895\* | \_412155\* | 0 | 2013-09-13 17:36:41 | \_ | 16 | \_ | \_ | \_212.249.\* | \_ |
| 0 | \_00141786\* | \_412155\* | 0 | 2013-09-13 17:36:49 | \_ | 16 | \_ | \_ | \_212.249.\* | \_ |
| 0 | \_412255\* | \_334500\* | 0 | 2013-09-13 17:36:51 | \_ | 40 | \_ | \_ | \_412255\* | \_ |
| 0 | \_412255\* | \_334500\* | 0 | 2013-09-13 17:36:51 | \_ | 16 | \_ | \_ | \_412255\* | \_ |
| 0 | \_00493032\* | \_412255\* | 0 | 2013-09-13 17:36:52 | \_ | 16 | \_ | \_ | \_212.249.\* | \_ |
| 0 | \_002135571\* | \_412255\* | 0 | 2013-09-13 17:36:53 | \_ | 16 | \_ | \_ | \_212.249.\* | \_ |

In the above examples, a few digits are removed from CLI, CLD, and account id and are replaced by asterisk for the sake of the privacy.

|  |  |
| --- | --- |
| #Continuing  #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_# | Comments  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| tlog="stop.log" | This file contains the stop time of the last successfully retrieved period. The times are stored in YYMMDD HH:MM:SS format. |
| tmargin=300 | When computing the start and stop times of the period to be retrieved from the database, the start is based on the previous stop, and the stop is based on the current time. This is the backward margin in seconds between the current time and the stop time. You must leave a few minutes for replication and delays in the billing. The value of 300 seconds is proved to be enough. |
| tmax=$((48\*3600)) | If last stop is too far in the past or the script is launched the first time, this parameter defines the maximal size of the chunk. |
| trotate=$((48\*3600))  files=2 | Output HTML files are rotated. According this parameter the rotation occurs at the midnight every 2nd day. The current file is 0.html, and there are two more files 1.html and 2.html alimented by the rotation. |
| utc=""  utc="-u" | If the server’s time is in current time zone, the value of [utc] is empty. If the server is running in UTC, the value must be “-u” |
| empty="empty.html"  if [ ! -f "$empty" ]  then  (  echo -n "<table border=1><tr>"  for h in Table CLI CLD PDD Start Stop Code Revenue Cost Account Duration  do  echo -n "<th>$h</th>"  done  echo -n "</tr></table>"  ) > "$empty"  fi | The empty.html file contains only the headers of a table. This file is necessary to give to the Excel file empty input files when no data is available yet. |
| for ((i=files+1;i<10;i++))  do  if [ -f $i.html ]  then  rm -f $i.html  fi  done | The rotation files are deleted if they are after the limit of rotation. This cleanup might be necessary when the value of [files] is decreased manually and the script is re-launched. |
| function chunk  { | Beginning of the function chunk. It retrieves the records of the successive period of time and saves the results in the HTML file. |
| local now  local stop  local istop  local last  local ilast  local idiff  local re  local start  local range  local i  local next  local limit  local qu1  local qu2 | Declaring all variables as local |
| now=$(date $utc +%Y-%m-%d\ %H:%M:%S)  stop=$(date -u -d "$now +0000 -$tmargin seconds" +%Y-%m-%d\ %H:%M:%S)  istop=$(date -d "$stop +0000" +%s) | Computing the date now to determine the stop time of the period to be retrieved. The current time is computed in UTC if the server is running in UTC. In time arithmetic, for the simplicity, we consider as if all times are in UTC; the result will remain correct. [istop] contains the stop time in number of seconds from the beginning of epoch. |
| if [ -f "$tlog" ]  then  last=$(tail -1 "$tlog" | tr -d "\r\n")  ilast=$(date -d "$last +0000" +%s)  idiff=$((istop-ilast))  if [ $idiff -gt 0 ]  then  if [ $idiff -le $tmax ]  then  re=0  else  re=1  echo last at $last is too old so shift and skip $((idiff-tmax)) seconds  fi  else  re=-1  echo last at $last is $((-idiff)) seconds ahead so wait  fi  else  re=2  fi | If file stop.log does not exist, returns 2 in variable [re]. If the last retrieval time is not too far in past, returns 0. If the last retrieval time is too far in past returns 1. If the last retrieval time is ahead the stop time, returns -1. It is possible if you manually increase the margin of stop time with respect to the current time. In such a case the next stop can be before the previous stop. |
| if [ $re -eq -1 ]  then  return  fi | If the current stop is before the previous, do nothing and return from the function. Wait until the next time the function will be called. |
| if [ $re -eq 0 ]  then  start=$(date -u -d "$last +0000 +1 second" +%Y-%m-%d\ %H:%M:%S)  fi | In case the previous stop is not too far in the past, the start time of the current chunk is set to the next second after the previous stop. Note again that for the time arithmetic purposes only we consider as if the times are in UTC zone (the result will be correct in any other zone, but input zone (see +0000) must correspond to the output zone (see “-u” key). |
| if [ $re -eq 1 -o $re -eq 2 ]  then  start=$(date -u -d "$stop +0000 -$tmax seconds +1 second" +%Y-%m-%d\ %H:%M:%S)  fi  range='"'"$start"'" and "'"$stop"'"' | If the previous stop is too far in the past or simply does not exists, the start time of the current request is computed based on the [tmax] value. The [range] string now contains the start and stop times of the current request to be sent to the server. |
| [ $re -eq 2 ] && re=$((100+re))  if [ $re -eq 0 -o $re -eq 1 ]  then  if [ $((ilast/trotate)) -ne $((istop/trotate)) ]  then  re=$((100+re))  fi  fi | Determining whether the rotation of files must take place. If the stop.log did not exist, then yes. If the previous stop time exists, and the new stop time falls into a different rotation interval (daily interval if [trotate] is equal to 24\*3600, or hourly interval if [trotate] is 3600), then prepare for a rotation. |
| if [ $re -ge 100 -a $re -le 102 ]  then  for ((i=files;i>=0;i--))  do  if [ -f $i.html ]  then  if [ $i -eq $files ]  then  next=$(date $utc +%y%m%d\_%H%M%S)  next=$next\ $(date $utc -d "$(stat -c %y $i.html)" +%y%m%d\_%H%M%S)  next=$next\ CDR  else  next=$((i+1))  fi  echo rotate $i.html to "$next.html"  cp -f -p $i.html "$next.html"  fi  cp -f "$empty" $i.html  done  fi | Now rotate the files if requested so. Start a loop from the last file and go down to the current 0.html file. Copy the file to the next and replace the current index with an empty file. Note that when copying the file, we preserve its modification time with option “-p”. The modification time is used to construct the name of the archive (when the file quits the rotation cycle). Whether the local time is in UTC or not, the archives are named in UTC format if the server provides the times in UTC and therefore the contents of files are in UTC as well. |
| echo collect between $range at $now +0000  limit=9999  qu1=''  qu1="$qu1"'select '"$fi1"' from CDR\_Vendors'  qu1="$qu1"' where disconnect\_time between '"$range"  qu1="$qu1"' order by disconnect\_time limit '$limit';'  qu2=''  qu2="$qu2"'select '"$fi2"' from CDR\_Vendors\_Failed'  qu2="$qu2"' where connect\_time between '"$range"  qu2="$qu2"' order by connect\_time limit '$limit';' | Displaying a log message. Building the query for retrieving the records of the range from the table of answered calls and for retrieving the records from the same range from the table of failed calls. As a key time value for the answered calls is the disconnect time. |
| mysql -h"$server" -u"$user" -p"$password" "$database" -e "$qu1 $qu2" -H >> 0.html  if [ $? -eq 0 ]  then  echo -ne "$stop\r\n" >> "$tlog"  fi | Sending the two queries to the database server and adding the output in HTML format into the 0.html file. |
| } | End of the function chunk |
|  | Continued… |

The HTML tables generated by function chunk are added to the HTML file one after the other. As all tables have the same number of matching columns, they are loaded into the Excel file worksheet as a single continuous table.

The next section describes the rest of the script.

# Mailing of charts

The same script, started in the previous section, is responsible also for the detection of the charts generated by the Excel file, and their mailing to the recipients.

|  |  |
| --- | --- |
| #Continuing  #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_# | Comments  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| function chart  { | Beginning of the function chart, responsible for mailing of PNG images |
| local fo  local im  local ways  local attach  local files  local subject  local body  local i  local way  local account  local options  local err | Declaring all variables created in this function as local. |
| fo="charts"  if [ ! -d "$fo" ]  then  return  fi  if [ ! -d "smtp" ]  then  return  fi  if [ ! -d "emailed" ]  then  mkdir emailed  fi | If folder “charts” does not exists, the place where the Excel file stores the output PNG files, then return from the function. Interrupt the function also if the folder smtp is empty. Folder smtp contains the SMTP connection parameters. More than one SMTP connection can be used under this script. SMTP connections are used randomly and if failed, another try, with eventually another connection is attempted. Folder emailed stores the successfully emailed image files. |
| im=$(ls -1 $fo | head -3)  if [ ! -z "$im" ]  then  ways=$(ls -1 smtp | wc -l)  if [ $ways -gt 0 ]  then | If folder of images contains charts and if there are at least one connection file in the smtp folder then start processing the transmission of images. Do not send more than 3 charts per function call. |
| attach=$(echo "$im" | while read f; do if [ ! -z "$f" ]; then echo "-attach $fo/$f"; fi; done)  attach=$(echo "$attach" | tr "\r\n" " " | sed -e "s/ +/ /g;s/^ //;s/ $//")  files=$(echo "$im" | tr "\r\n" " " | sed -e "s/ +/ /g;s/^ //;s/ $//")  subject="$tag $files" | Construct the attachment options in the variable [attach]. If more than one chart is generated, all files, but at most 3, will be attached to a single email. Add the list of files in the variable [subject]. It also contains the tag for the routing to the corresponding IMAP folder of our projects system. |
| if [ -f body.err ]  then  body=$(cat body.err)  else  body=""  fi | If the file body.err exists, load it into the variable body. It contains the list of the previous email transmission attempts with their corresponding errors. If no error occurred at the last call of this function, no such file must exist. |
| for((i=1;i<=8;i++))  do  way=$((RANDOM % ways + 1))  account=smtp/$(ls -1 smtp | head -$way | tail -1)  echo "$(date +%Y-%m-%d\_%H:%M:%S) $subject $account"  options=$(cat $account | tr "\r\n" " ")  body="$(echo -n "$body" | tr "\r\n" "==") ...Via $account"  err=$(echo "$subject:$body" | email $options -subject "$subject" $attach "$to" 2>&1)  if [ -z "$err" ]  then  break  fi  body="$body {$err}"  done | Give 8 tries to transmission of the charts via SMTP. Select randomly a transmission file number in variable [way]. The transmission file name is in variable [account]. Read the SMTP transmission options (login, password, server, encryption, etc) from the account file. Update the body of the email by indicating the account via which the current transmission will be attempted. Give a try to the transmission. The error output is redirected to the standard output by “2>&1”. This ensures that errors will be collected into the variable [err]. If this variable turns to be empty, break the transmission attempts loop, otherwise add the error output into the body and continue the loop. |
| if [ -z "$err" ]  then  echo "$im" | while read f  do  if [ ! -z "$f" ]  then  mv $fo/$f emailed  fi  done  if [ -f body.err ]  then  rm body.err  fi  else  echo -n "$body" > body.err  fi | If by the end of the loop, there are no errors, then move the attached files from the folder charts to the folder emailed. Remove the body.err file if any. If the loop ended without success, then save the error messages in the body.err file for reloading the error messages at the next call of the chart function. |
| fi  fi | If there were images to send and if there where smtp ways to transmit the emails |
| } | End of the chart function |
| while true  do  chunk  chart  sleep 60  done | Here is the main loop of the bash script. Download a new period of call records with the chunk function and aliment the input files of the Excel with the html output of MySQL. Check whether Excel generated any chart, and email to recipients if yes. |

The smtp folder must contain one text file per smtp connection account.

$ ls -1 smtp

d9a.monitor.gmail.com.txt

d9a.monitor.switzernet.com.txt

d9a.monitor.yahoo.com.txt

$

The text files contain the options of the email bash program. See the two examples below, where the passwords are replaced by asterisks.

$ cat smtp/d9a.monitor.yahoo.com.txt

-from-addr d9a.monitor@yahoo.com

-from-name d9a\_Monitor\_Yahoo.com

-smtp-server smtp.mail.yahoo.com

-tls

-smtp-auth=login

-smtp-user=d9a.monitor@yahoo.com

-smtp-pass=\*\*\*\*

$ cat smtp/d9a.monitor.switzernet.com.txt

-from-addr d9a.monitor@switzernet.com

-from-name d9a\_Monitor\_Switzernet.com

-smtp-server smtp.switzernet.com

-smtp-port=587

-smtp-auth=plain

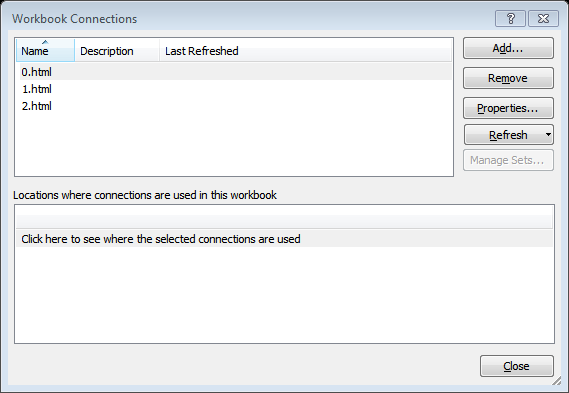
-smtp-user d9a.monitor@smtp.switzernet.com

-smtp-pass \*\*\*\*

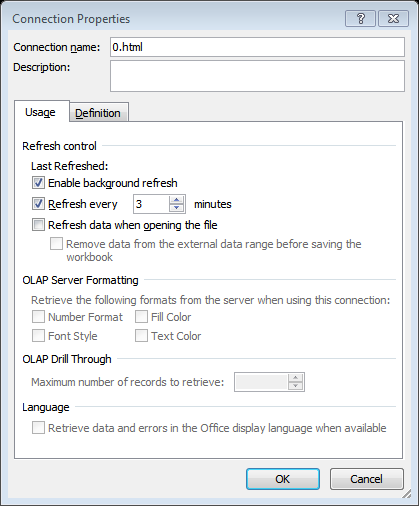
$

# Linking Excel to the HTML files

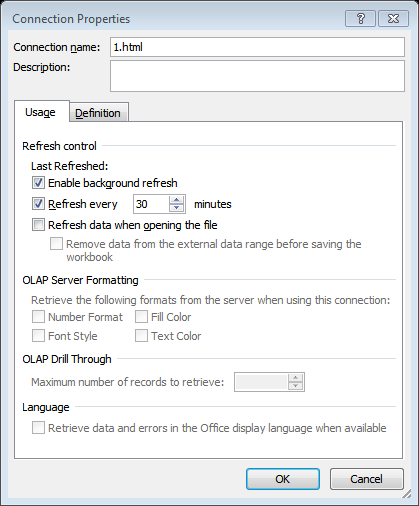
The three html files 0.html, 1.html, and 2.html are imported into corresponding three worksheets 0, 1, and 2, of an Excel workbook. Use [From Web] button of the [Data] tab of the Excel file in order to import 0.html file into the 0 worksheet starting from the upmost left position A1. Similarly are imported 1.html and 2.html. In data connection properties we see the following 3 connections.



The first connection is configured to refresh every 3 minutes.

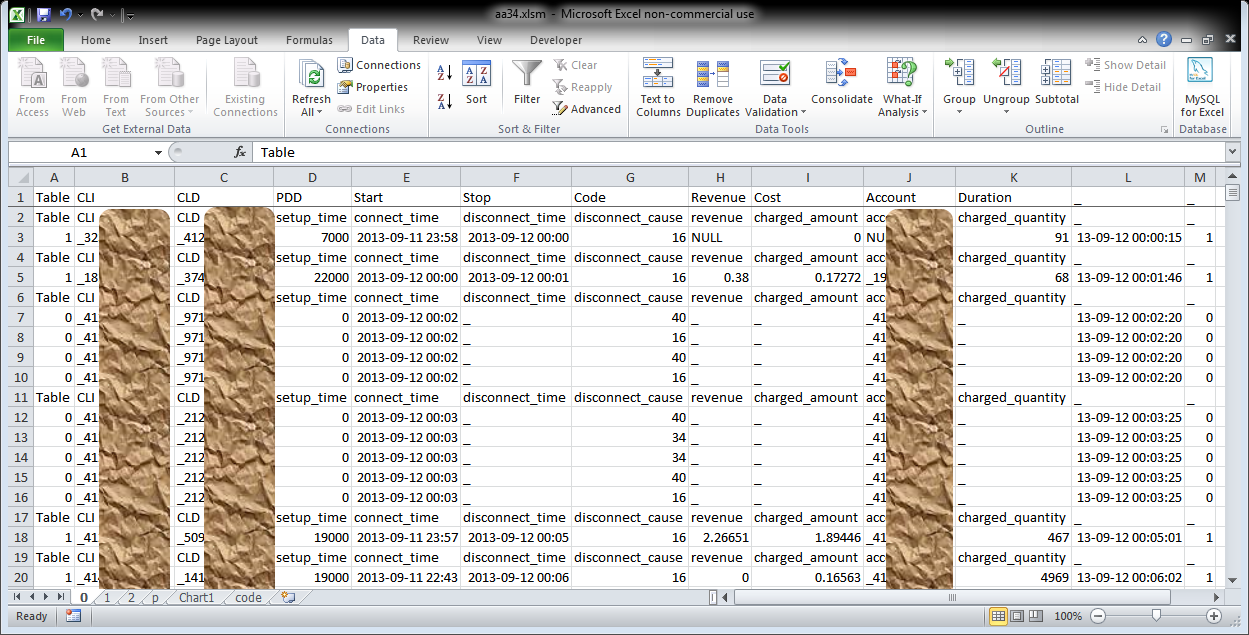


The other two connections refresh every 30 minutes.

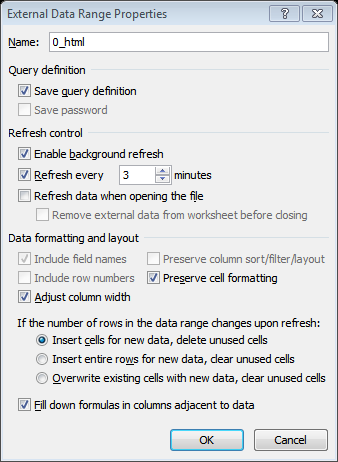


The first connection is the one reading the 0.html file which is updated by a bash script every minute. The other two files are changed only at the log rotation point and no need to refresh them more frequently than every half an hour.

Our data sheets look as follows.



When in the worksheet alimented by an html file, the properties panel provides you additional control information.



The refresh frequency appears also in this panel. In this panel, for each of the worksheets, we defined the so called query names as 0\_html, 1\_html, and 2\_html. You will see later that these names allow us to access the query parameters from within a VBA script. We also check the box for filling down formulas in columns adjacent to data. Thanks to this option the formulas that we provide L and M columns of each of the input worksheets are automatically propagated or cut along with the data read from the input html files. The formula must be present only in the first row.

The column L contains the times of call records (answered or failed). The column M contains 1 if the call is answered, is longer than the minimal duration p!$B$2, and can be considered as a successful conversation. The value of a cell in column M is 0, if the call is too short or is failed. The value of p!$B$2 is 10 seconds by default but it is under the control of the user to change it. Here are the formulas of the input worksheets 0, 1, and 2.

L2 =IF(A2=1,F2,IF(ISNUMBER(A2)\*(A2=0),E2,"\_"))

M2 =IF(A2=1,IF(K2>p!$B$2,1,0),IF(ISNUMBER(A2)\*(A2=0),0,"\_"))

The remaining calculations are carried out in the ‘p’ worksheet.

# Count analysis

The data read in 0, 1, and 2 worksheets is processed in the ‘p’ worksheet.



The cells B14 to B18 contain information on the data loaded into the 0 worksheet. The following cells contain similar information on the other input worksheets.

B14 =COUNT('0'!L:L)

B15 =COUNTIF('0'!M:M,0)

B16 =COUNTIF('0'!M:M,1)

B17 =MIN('0'!L:L)

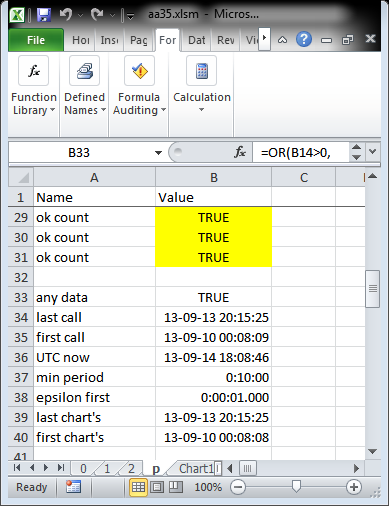
B18 =MAX('0'!L:L)

And for validation we also have this check.

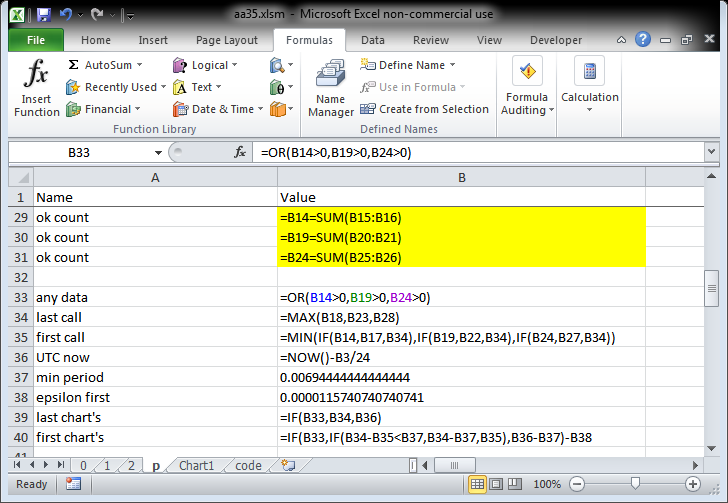
B29 =B14=SUM(B15:B16)

# Computing the period of the entire traffic

Here we compute the period of the entire traffic, i.e. the values required for the construction of the horizontal time axis.



Below are the formulas behind the same cells.



B33 contains true if any input call record from any worksheet is available, otherwise its value is false. B34 contains the last call and B35 contains the first call across all three input files 0.html, 1.html, and 2.html. If no records are loaded the chart will temporarily show on the horizontal axis a 10 min interval in the current time. If records are available but the start is too close to stop, the start time will be moved backward to have at least a 10 minute long interval.

# Equations of the exponentially increasing intervals

To solve the problem of the exponentially increasing intervals for a given *n* (the number of intervals minus 1) and *k* (the factor of the largest over smallest interval), the following equation must be solved, and the intervals *a*, and the factor *f* between two adjacent intervals must be found.

The factor between the intervals at the both ends of the period gives us.

Therefore

Further, the first equation of this section can be rewritten.

We already know *f*. In order to find *a*, we need to find the result of the following formula.

In the above example *f* is more than 1.

We can rewrite the same sum starting from the other end and by reducing the successive interval instead of increasing.

Therefore *a* can be found also by computing the following sum.

The answer of the following infinite sum is known for *x* less than 1.

Therefore

But we need the answer of a limited sum.

The limited sum can be written by subtracting from the infinite sums its infinite tail.

The infinite tail can be rewritten as follows.

The last piece of the above sum is our infinite sum.

Therefore:

Or

Finally

Therefore, coming back to the largest interval

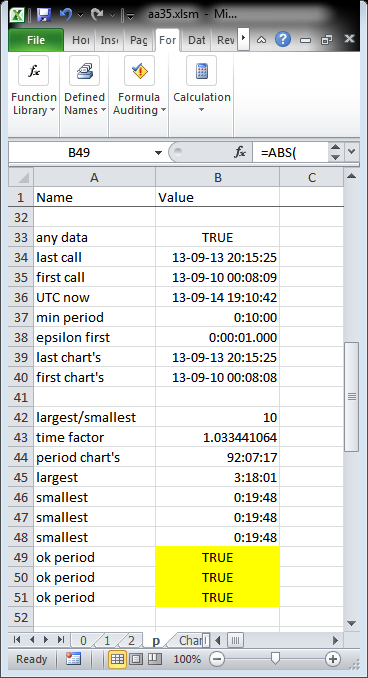
We can write

And therefore the smallest interval is equal to

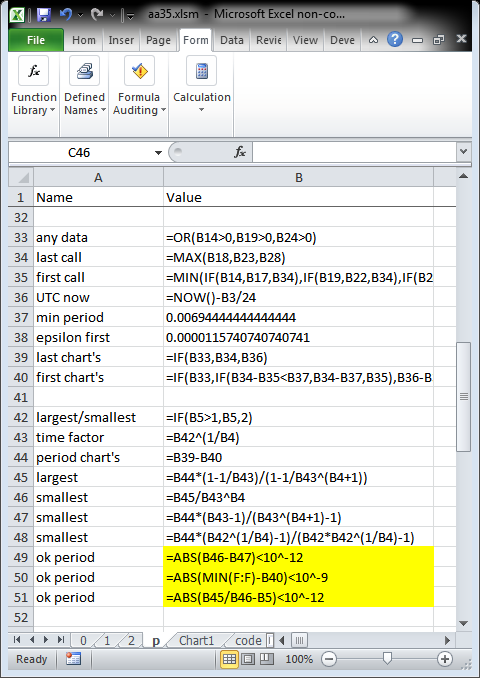
Finally in terms of the factor between the largest and smallest intervals the formula looks as follows.

Or finally:

The Excel version is shown below.



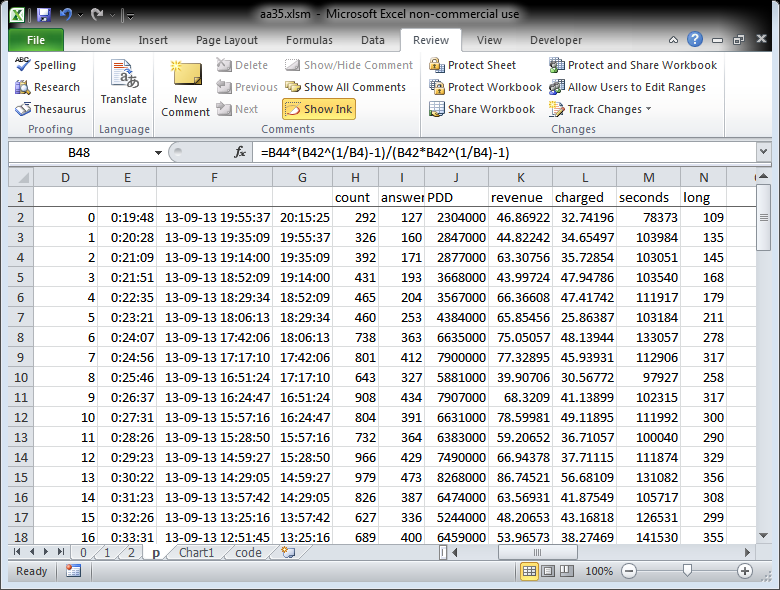
The following capture shows the formulas corresponding to the previous capture. It’s the implementation of the previous equations in Excel.



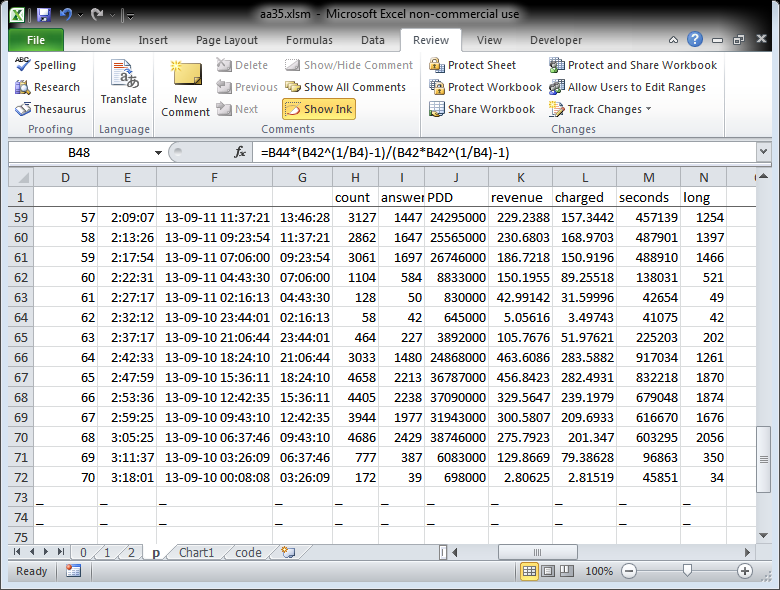
# Building the increasing intervals

The following two captures show the interval building and the data selection for each interval.

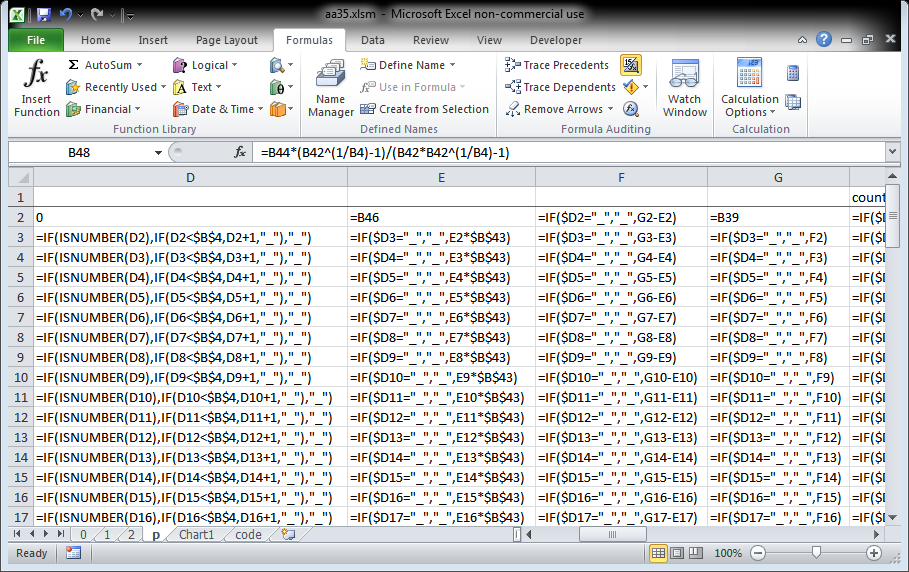
Here are the recent short intervals.



And here are the early long intervals.



Now the formulas are shown via the following captures. The column E represents the length of the interval, the column F the beginning of the interval, and the column G, the end of the interval. The formula shows that each successive interval length (next in excel line but preceding in time) is larger the previous one by a factor $B$43.



Then the values are computed as follows.

For [count]

H2=IF($D2="\_","\_",COUNTIF('0'!$L:$L,"<="&p!$G2)-COUNTIF('0'!$L:$L,"<="&p!$F2)+COUNTIF('1'!$L:$L,"<="&p!$G2)-COUNTIF('1'!$L:$L,"<="&p!$F2)+COUNTIF('2'!$L:$L,"<="&p!$G2)-COUNTIF('2'!$L:$L,"<="&p!$F2))

For [answered]

I2=IF($D2="\_","\_",SUMIF('0'!$L:$L,"<="&p!$G2,'0'!A:A)-SUMIF('0'!$L:$L,"<="&p!$F2,'0'!A:A)+SUMIF('1'!$L:$L,"<="&p!$G2,'1'!A:A)-SUMIF('1'!$L:$L,"<="&p!$F2,'1'!A:A)+SUMIF('2'!$L:$L,"<="&p!$G2,'2'!A:A)-SUMIF('2'!$L:$L,"<="&p!$F2,'2'!A:A))

For [PDD]

J2=IF($D2="\_","\_",SUMIF('0'!$L:$L,"<="&p!$G2,'0'!D:D)-SUMIF('0'!$L:$L,"<="&p!$F2,'0'!D:D)+SUMIF('1'!$L:$L,"<="&p!$G2,'1'!D:D)-SUMIF('1'!$L:$L,"<="&p!$F2,'1'!D:D)+SUMIF('2'!$L:$L,"<="&p!$G2,'2'!D:D)-SUMIF('2'!$L:$L,"<="&p!$F2,'2'!D:D))

For [revenue]

K2=IF($D2="\_","\_",SUMIF('0'!$L:$L,"<="&p!$G2,'0'!H:H)-SUMIF('0'!$L:$L,"<="&p!$F2,'0'!H:H)+SUMIF('1'!$L:$L,"<="&p!$G2,'1'!H:H)-SUMIF('1'!$L:$L,"<="&p!$F2,'1'!H:H)+SUMIF('2'!$L:$L,"<="&p!$G2,'2'!H:H)-SUMIF('2'!$L:$L,"<="&p!$F2,'2'!H:H))

For [charged]

L2=IF($D2="\_","\_",SUMIF('0'!$L:$L,"<="&p!$G2,'0'!I:I)-SUMIF('0'!$L:$L,"<="&p!$F2,'0'!I:I)+SUMIF('1'!$L:$L,"<="&p!$G2,'1'!I:I)-SUMIF('1'!$L:$L,"<="&p!$F2,'1'!I:I)+SUMIF('2'!$L:$L,"<="&p!$G2,'2'!I:I)-SUMIF('2'!$L:$L,"<="&p!$F2,'2'!I:I))

For [seconds]

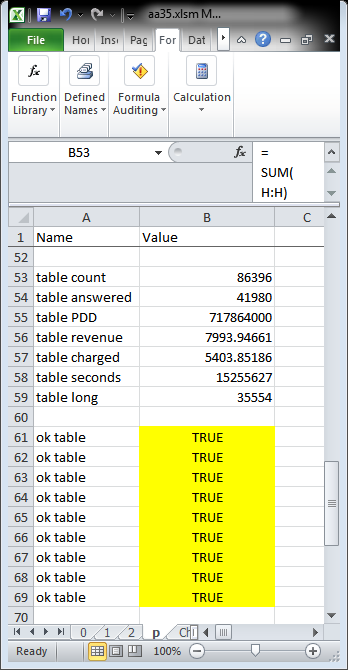
M2=IF($D2="\_","\_",SUMIF('0'!$L:$L,"<="&p!$G2,'0'!K:K)-SUMIF('0'!$L:$L,"<="&p!$F2,'0'!K:K)+SUMIF('1'!$L:$L,"<="&p!$G2,'1'!K:K)-SUMIF('1'!$L:$L,"<="&p!$F2,'1'!K:K)+SUMIF('2'!$L:$L,"<="&p!$G2,'2'!K:K)-SUMIF('2'!$L:$L,"<="&p!$F2,'2'!K:K))

For [long]

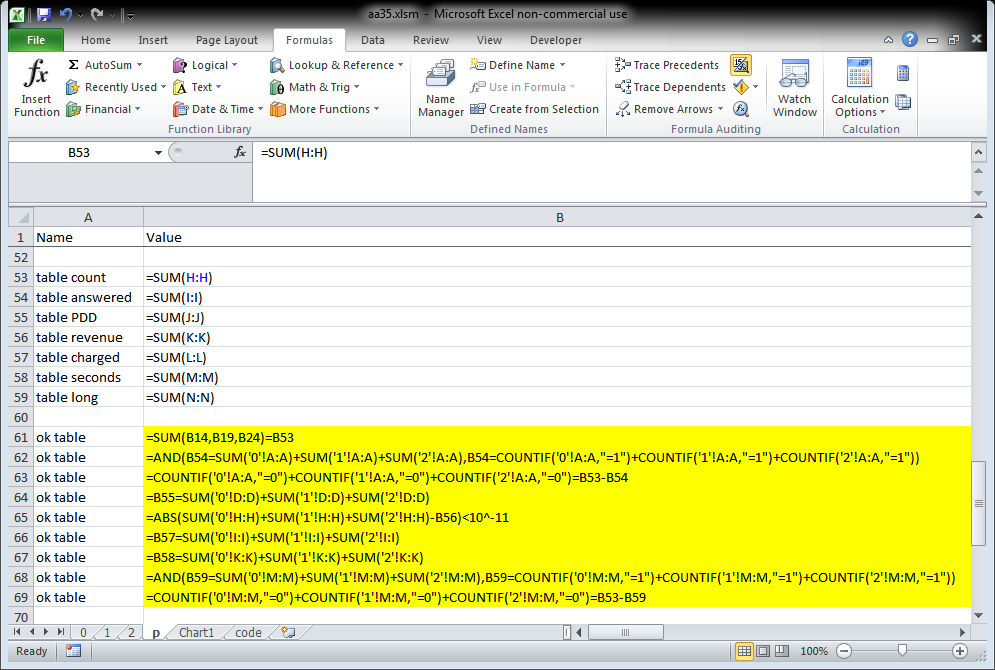
N2=IF($D2="\_","\_",SUMIF('0'!$L:$L,"<="&p!$G2,'0'!M:M)-SUMIF('0'!$L:$L,"<="&p!$F2,'0'!M:M)+SUMIF('1'!$L:$L,"<="&p!$G2,'1'!M:M)-SUMIF('1'!$L:$L,"<="&p!$F2,'1'!M:M)+SUMIF('2'!$L:$L,"<="&p!$G2,'2'!M:M)-SUMIF('2'!$L:$L,"<="&p!$F2,'2'!M:M))

For validating the data constructed in the table, we compare the sums of columns H:H through N:N with the sums of the input excel sheets corresponding to 0.html, 1.html, and 2.html input files. If the intervals breakout is wrong the sums will not correspond.

The cells from B53 through B59 shown in the capture below correspond to the 7 columns of the data (from H through N).



Here are the formulas behind the above cells.

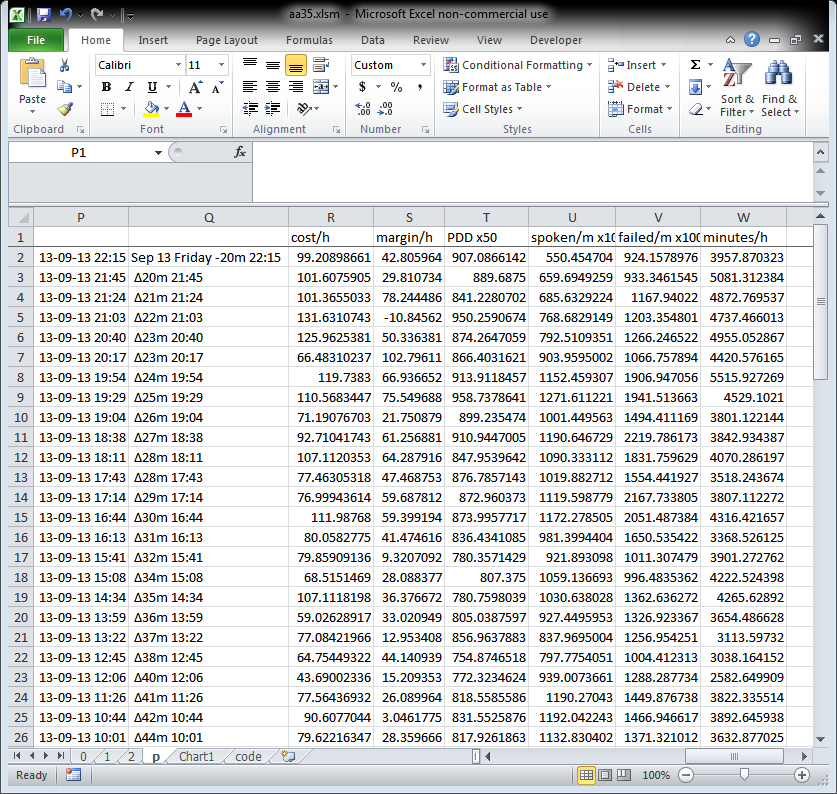


For example in the cell B64 above, we compare the sums of columns D:D in all three input sheets 0, 1, and 2 with the sum of the column J:J. Similarly are successfully validated all other columns of the interval breakout.

# Building the chart input

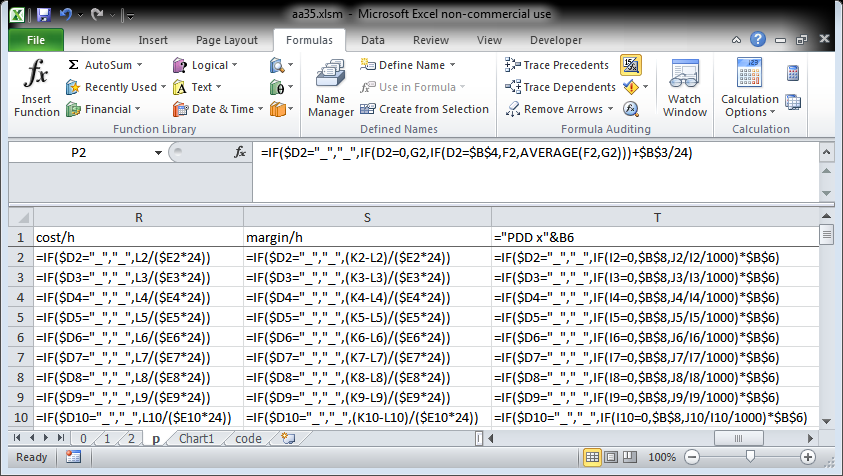
While the data columns of the breakout into intervals contain the totals of count, answered, PDD, revenue, charged, seconds, and long values, the visualization needs the hourly or minutely rates of these values. For instance the direct visualization of the totals of minutes and costs (without converting them into rates) would result into a disproportional chart with its values at the oldest intervals higher (with respect to the same values at the recent interval) by the same factor as the oldest interval itself, is larger than the most recent shortest interval. As for PDD, its average value is needed, and not the total.

For the sake of the proportionality we convert the values before visualization.

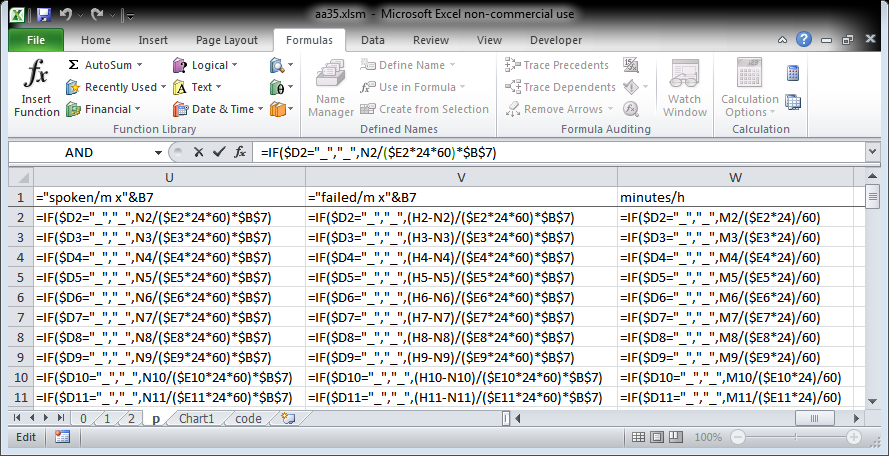


The first columns P and Q in the above capture are responsible for the labeling of the horizontal axis.

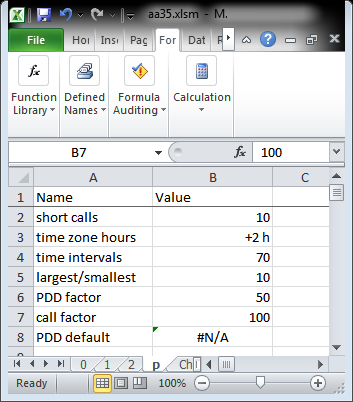
The formulas corresponding to the cost per hour, margin per hour, and PDD are shown below.



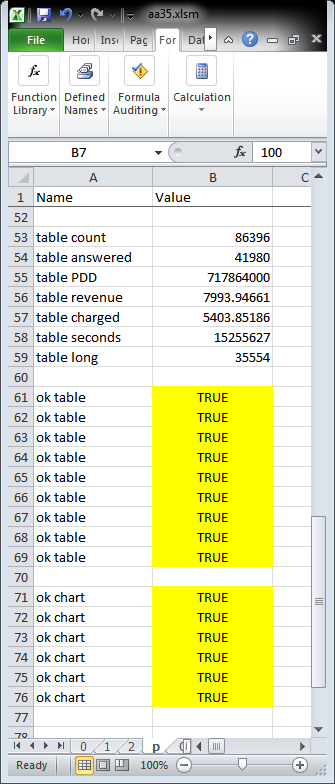
The formulas corresponding to spoken per minute, failed per minute, and minutes per hour are shown below.



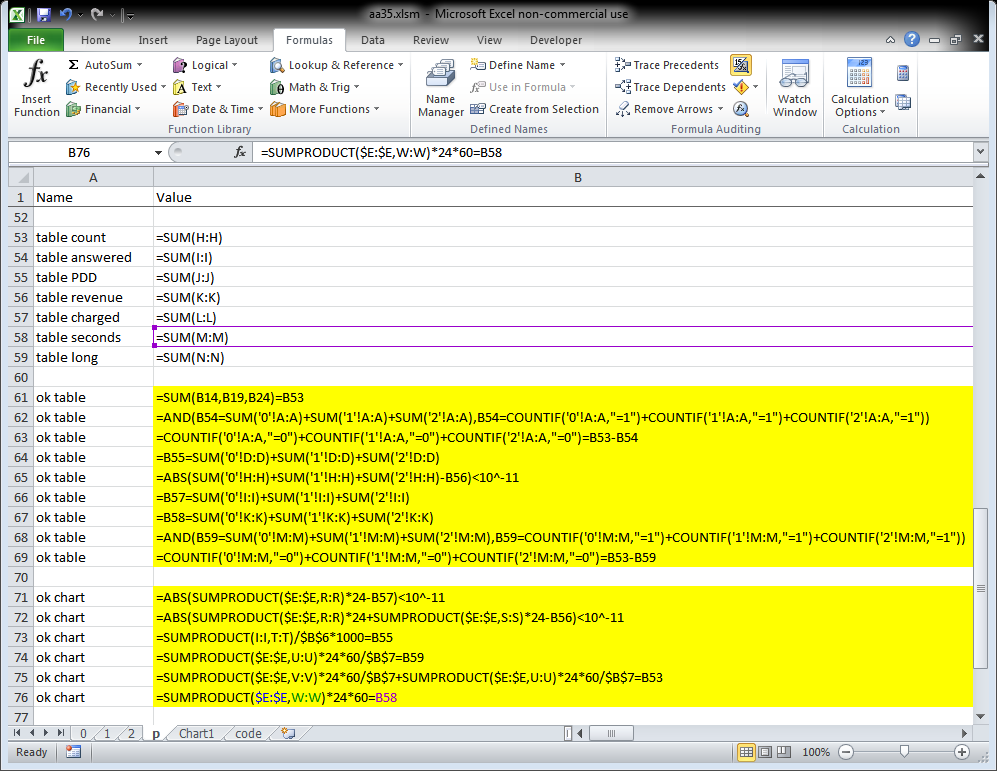
The cost per hour and margin per hour are shown along one vertical axis in forms of histograms. The other values, the PDD, the call attempts per hour, and minutes per hour are shown as curves on the secondary vertical axis. As PDD and call rates are scaling differently with respect to the values of minutes per hour, two adjustment factors are used, PDD factor and call factor. They are stored in the parameters’ cells in the same datasheet.



Control formulas are used to validate the construction of the chart input data.



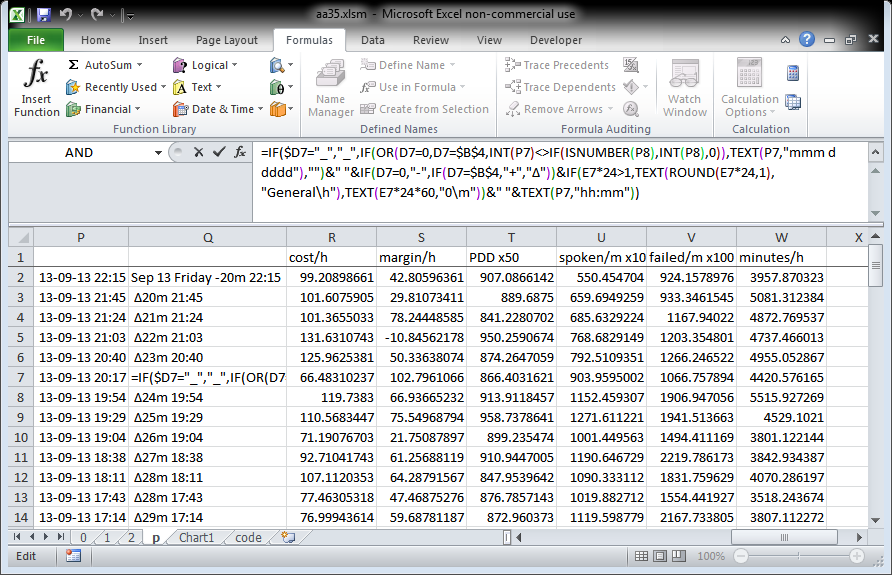
The formulas behind the chart data validation cells (from B71 through B76) are shown in the next capture.



In all these validation formulas sum of products of the rate values with the lengths of the intervals is used to obtain the totals and compare with the original check sum.

# The horizontal axis labels

The horizontal axis is labeled so as to not overwhelm the graph with dense label set.



Here are the formulas used for the values of the intermediary P column and the column Q containing the labels to show.

P2 =IF($D2="\_","\_",IF(D2=0,G2,IF(D2=$B$4,F2,AVERAGE(F2,G2)))+$B$3/24)

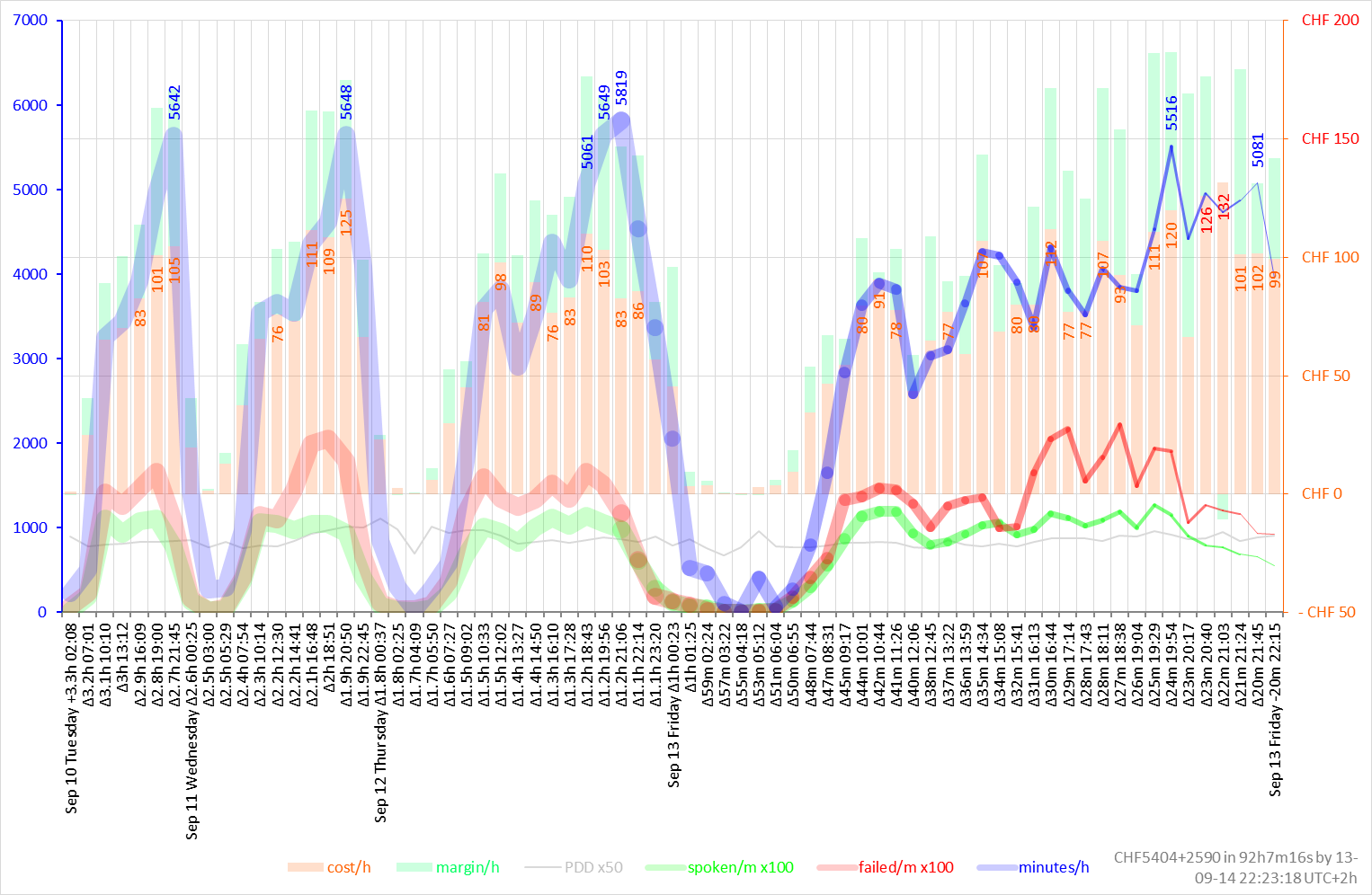
Q2 =IF($D2="\_","\_",IF(OR(D2=0,D2=$B$4,INT(P2)<>IF(ISNUMBER(P3),INT(P3),0)),TEXT(P2,"mmm d dddd"),"")&" "&IF(D2=0,"-",IF(D2=$B$4,"+","Δ"))&IF(E2\*24>1,TEXT(ROUND(E2\*24,1),"General\h"),TEXT(E2\*24\*60,"0\m"))&" "&TEXT(P2,"hh:mm"))

If we are dealing with the very first (recent) interval, the P column contains the most recent edge of the interval (the rightmost time). If we are dealing with the very last (the oldest) interval, the P column contains the oldest edge of the interval (the leftmost time). In all other, middle cases, the P column contains the middle of the interval. The P column is also responsible to convert the UTC time into the local time zone using the time shift value stored in $B$3 cell.

The Q columns shows the month, day of the month, and the day of the week, if it is about the leftmost or the rightmost interval, or if it is an interval that compared to the previous one in time changes the day. Then the label shows the width of the interval in hours or minutes (if the width is less than one hour). The width is preceded by the delta sign, except for the rightmost interval, where the width is preceded by the minus sign indicating the fact that the time shown is the rightmost edge of the rightmost interval and not its middle, and except also for the leftmost interval, where the width is preceded by the plus sign indicating the fact that the time shown on the axis is the leftmost edge of the leftmost interval and not its middle (as for the all remaining intervals between these two exceptions).

# The chart

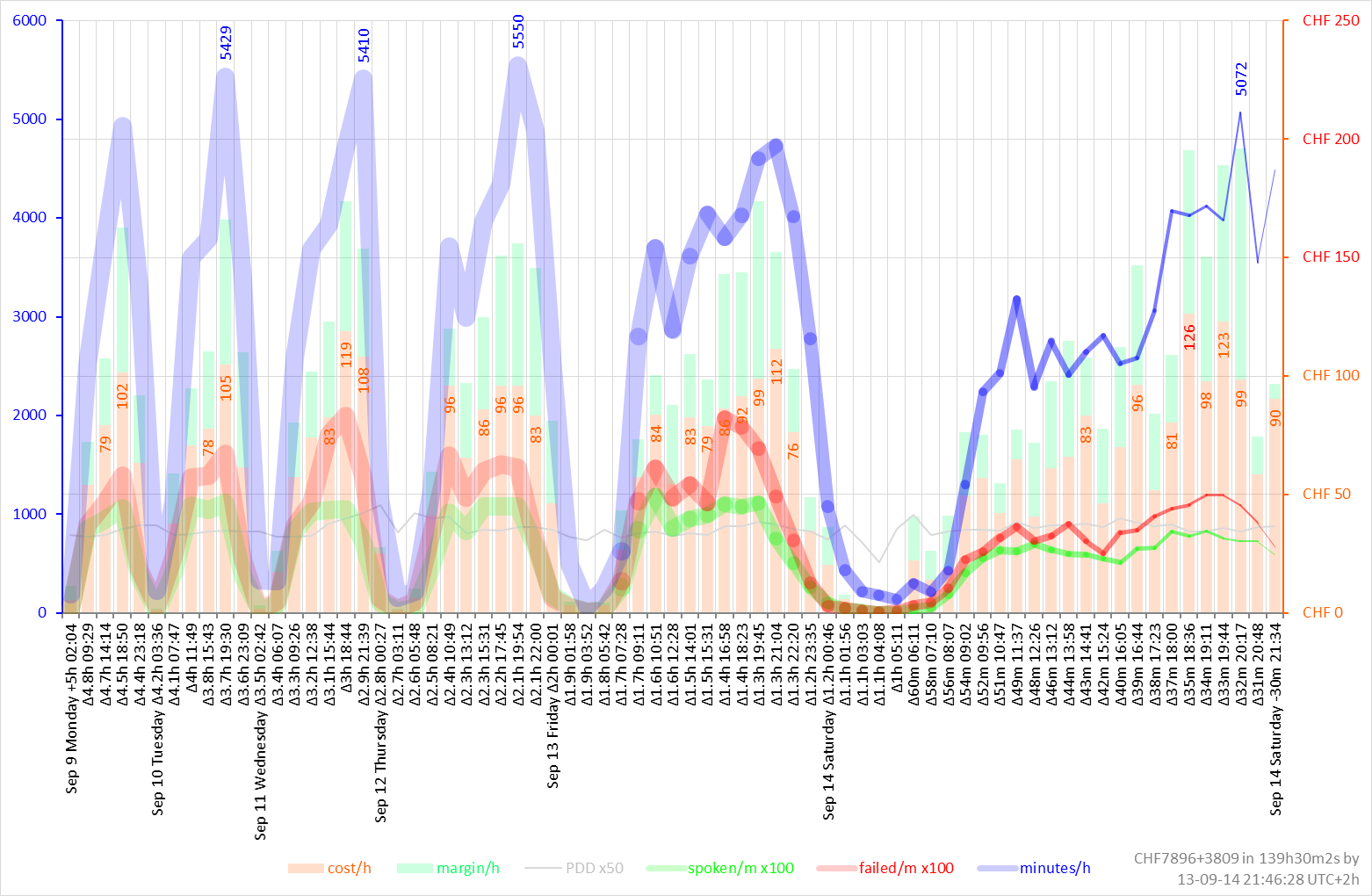
Here is the final visual output obtained from the chart columns.



As you see, the left side of the chart is more dense as the intervals there cover longer periods (3.3 hours for the leftmost one) while the right side of the chart is more detailed with its shorter intervals (20 minutes for the rightmost one).

The values of minutes are displayed only if they exceed the level of 5000 minutes per hour. The cost values are displayed if they exceed the level of CHF 75 per hour and are in red if they exceed the level of CHF 125 per hour.

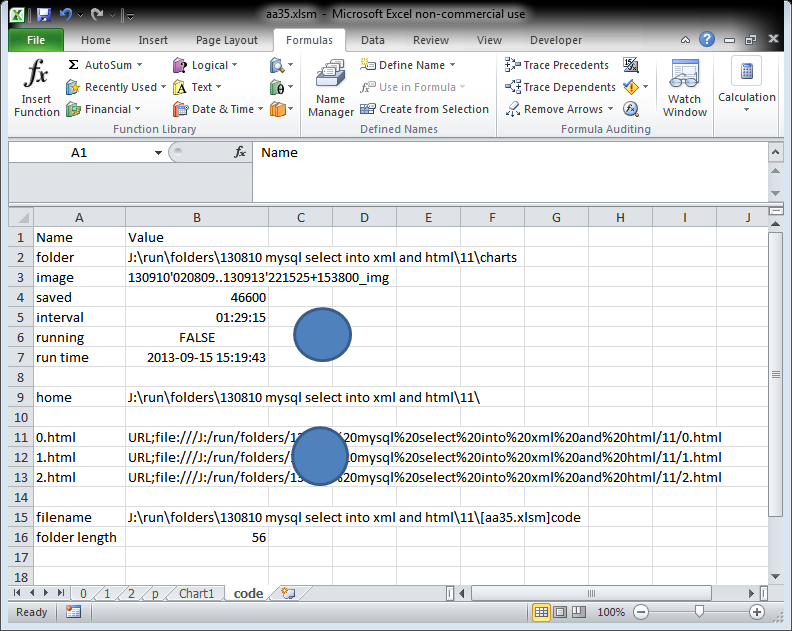
After a collection of more data the chart looks as follows. We rotated due to the lack of space.



You see that multiple days are squeezed into a dozen intervals on the left hand side of the chart (with the leftmost interval width equal to 5 hours) while the right side still shows the current activity in great details with the rightmost interval width equal to 30 minutes only. In both examples shown above, the factor of the largest over the smallest interval is equal to 10. This is a parameter that can be modified on the fly in the Excel worksheet.

# Code data

A worksheet “code” is created for managing macros and the associated data.



The B2 cell of this worksheet contains the folder name where the VBA script must save the charts. The API exporting the images needs a filename with an absolute path. In the capture containing the formulas you will see that the absolute pathname is constructed by retrieving the full filename of the Excel file.

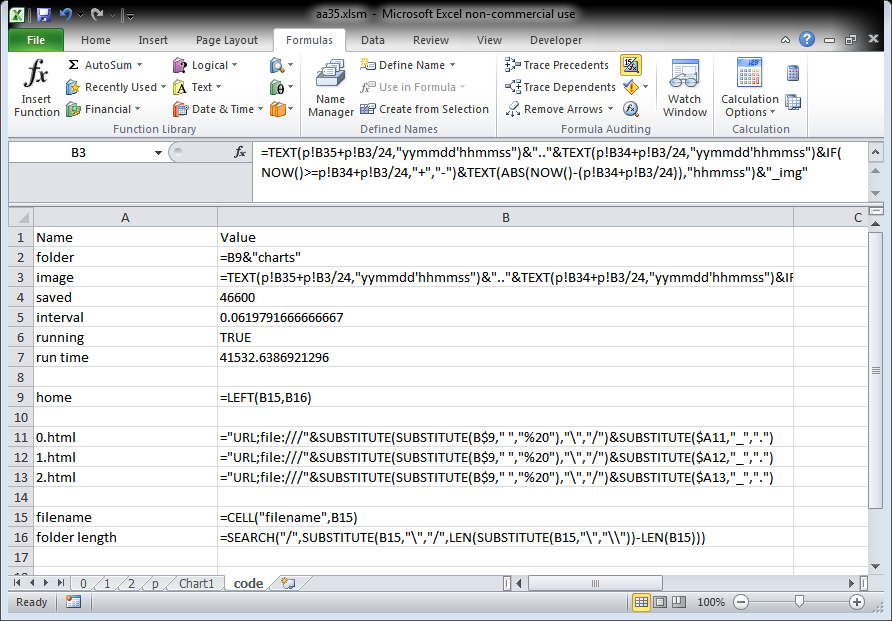
The B3 cell contains the image file name prefix. The formula in the formula bar of the next capture shows the construction of the text in the B3 cell. The filename begins with two date and time values in YYMMDD’HHMMSS format separated by double dots. The first date-and-time value is the time of the first available call record (in 0.html, 1.html, and 2.html input files) and the second one is the time of the most recent call record. Then, in the file name, the first two date-and-time values are followed by a time in HHMMSS format preceded by plus or minus sign. This is the gap between the stop time and the time of the creation of the chart.

The B4 cell contains an index of images. The index is incremented by 100 at each save.

The B5 cell contains the interval in time defining the frequency at which the charts must be generated. Here the charts are generated every one-and-half hours.

The B6 cell contains a Boolean value indicating whether the chart generation subroutine is running or not. By clicking on the first circle you can launch or stop the background periodic chart generation script. Launching of the background script is carried out manually. You must click one on the circle when you open the Excel file the first time.

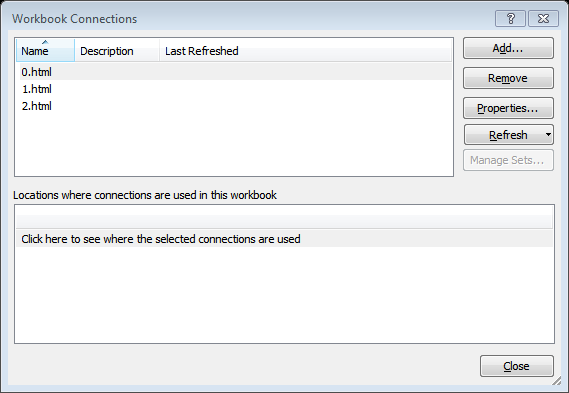
The B7 cell shows the run time of the background subroutine waiting its execution. When the subroutine is executed it generates the chart and schedules its execution for the next run. The value of the B7 cell is changed at that moment.



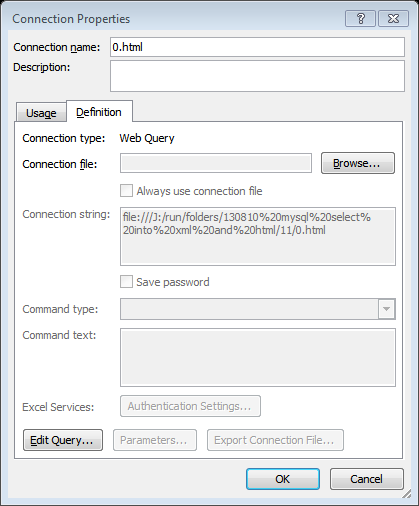
In cell B9 we compute the home location of the Excel file.

We use the full path Excel file name in cell B15 for that purpose.

In cells B11 through B13 we define the values of the web page queries to be used for loading the 0.html, 1.html, and 2.html files. Whenever you move the script and the Excel file to another folder or computer, you must update the connection settings.



The file location can be found in the Definition tab of the properties panel of the connection.



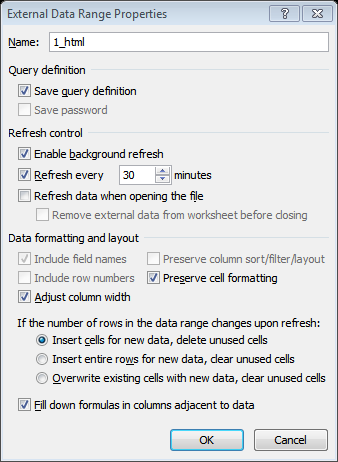
In order to avoid the manual modification (via Edit Query button) taking too much time when dealing with large HTML files, we use VBA scripts to atomize the update.

# VBA scripts

In this section we describe the VBA subroutines used.

|  |  |
| --- | --- |
| 'Code  '\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_# | Comments  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Sub Oval1\_Click()  Dim code As String  code\_data = "code" | Beginning of the subroutine associated to the click event of the first circle |
| Dim run As Boolean  run = Sheets(code\_data).Range("B6")  If run Then  run = False  Else  run = True  End If  Sheets(code\_data).Range("B6") = run | Reading the Boolean value from cell B6, inversing its value and saving back into cell B6. |
| If run Then  Periodic  Else  Dim runtime As Date  runtime = Sheets(code\_data).Range("B7")  On Error GoTo ErrHandler1:  Application.OnTime runtime, "Periodic", , False  On Error GoTo 0  End If  Exit Sub | If the new value of [run] is true, then launch the “Periodic” subroutine. Otherwise try to stop it. The date-and-time of the scheduled subroutine is supposed to be in the cell B7. Read this time value and stop the scheduled. Before attempting to stop, the default error handler is changed. The error handler is reset to the default “GoTo 0” after the call of the system function. |
| ErrHandler1:  MsgBox "nothing to stop"  On Error GoTo 0 | If stopping fails display a message and reset the error handler to the default. If you attempt to stop the periodic subroutine that is not there. It happens if you reopen an Excel file closed while the periodic subroutine still scheduled. |
| End Sub | End of the subroutine associated to the first circle |
| Sub Periodic()  Dim code As String  code\_data = "code"    Dim run As Boolean  run = Sheets(code\_data).Range("B6")  If run Then  Export2  Dim interval As Date  Dim runtime As Date  interval = Sheets(code\_data).Range("B5")  runtime = Now + interval  Application.OnTime runtime, "Periodic"  Sheets(code\_data).Range("B7") = runtime  End If  End Sub | Execute the body only if the B6 cell value is True. Export the chart into a file. Compute the next time this subroutine must be called. Schedule this subroutine’s call. Save the time into B7 cell and exit. The value in B7 cell is needed for stopping the scheduled procedure if eventually requested by a clicking on the first circle, i.e. by Oval1\_Click(). |
| Sub Export2()  Dim code As String  code\_data = "code" | Beginning of the subroutine for exporting the charts of the Excel workbook into image files. |
| Dim saved As Long  saved = Sheets(code\_data).Range("B4") | Read the counter of saved chart sets. We call it counter of chart sets as this subroutine does not save a particular chart, but eventually all charts available in the Excel workbook. |
| Dim i As Integer, exported As Integer  exported = 0 | Counter exported is the index of the chart in the workbook. |
| Dim folder As String, image As String  folder = Sheets(code\_data).Range("B2").Value  image = Sheets(code\_data).Range("B3").Value | Reading the current folder name and the image name prefix from the code data worksheet. The Excel formulas in this worksheet ensure that the value of cell B2 points to the “charts” subfolder at the current location of the Excel file. |
| For Each chartObj In ActiveWorkbook.Charts  chartObj.Export folder & "\" & image & (saved + exported) & ".png"  'chartObj.Export folder & "\" & image & (saved + exported) & ".jpg"  exported = exported + 1  Next | Go through all chart sheets, the sheets dedicated to individual Excel charts. Save into PNG files and increase the exported counter. |
| For Each sheetObj In ActiveWorkbook.Worksheets  For i = 1 To sheetObj.ChartObjects.Count  sheetObj.ChartObjects(i).Activate  ActiveChart.Export folder & "\" & image & (saved + exported) & ".png"  exported = exported + 1  Next i  Next | Go through all worksheets, and in each, go through all charts available in the worksheet. Export each chart into a PNG file and increase the exported counter each time. |
| saved = saved + 100  Sheets(code\_data).Range("B4") = saved | Increment the counter of saved chart sets and write its new value into cell B4 of the code data worksheet. |
| End Sub | End of the subroutine exporting the Excel charts into PNG files. |
| Sub Oval2\_Click()  Connections  End Sub  Sub Auto\_Open()  Connections  MsgBox "Click on the 1st circle of the 'code' worksheet to start or stop the generation of charts"  End Sub | Subroutine Connections inspects the coherence of the links to input HTML files 0.html, 1.html, and 2.html. As the absolute paths are stored in the excel file, moving the folder of the Excel file with the input HTML files will cause connection problems. Therefore at each opening of the file we examine and reset the connection properties so as to always link to the input HTML files located in the current folder. At the opening of the Excel workbook we also remind the user that the automatic generation of charts, if needed, must be started manually. |
| Sub Connections()  Dim code As String  code\_data = "code" | Beginning of the connection inspection subroutine. |
| Dim ws As Worksheet  Dim qt As QueryTable    Dim msg As String    msg = ""    For Each ws In ThisWorkbook.Worksheets  For Each qt In ws.QueryTables | Scan all worksheets and in each worksheet all query tables of eventual connections. In our example only three query tables are associated to the worksheets 0, 1, and 2 respectively. |
| If ws.Name = "0" And qt.Name = "0\_html" Then  With qt  .Connection = Sheets(code\_data).Range("B11").Value  '.WebSelectionType = xlEntirePage  '.WebFormatting = xlWebFormattingNone  '.WebPreFormattedTextToColumns = True  '.WebConsecutiveDelimitersAsOne = True  '.WebSingleBlockTextImport = False  '.WebDisableDateRecognition = False  '.WebDisableRedirections = False  '.Refresh BackgroundQuery:=False  End With  End If | If a query table named “0\_html” is found in worksheet 0 then set its connection string to the value corresponding to the 0.html file located in the same folder as the Excel file. Note that we do not refresh the connection. The new data will be loaded at the next scheduled refresh. We only prepare the correct link string, which is already computed in the code data worksheet by Excel formulas. |
| If ws.Name = "1" And qt.Name = "1\_html" Then  qt.Connection = Sheets(code\_data).Range("B12").Value  End If    If ws.Name = "2" And qt.Name = "2\_html" Then  qt.Connection = Sheets(code\_data).Range("B13").Value  End If | Similarly, set the correct link string for the files 1.html and 2.html. |
| If msg <> "" Then  msg = msg & ", "  End If  msg = msg & qt.Name | Collect the names of the query tables met in the [msg] string variable. |
| Next qt  Next ws | End of the loop going through all eventual query tables in each available worksheet. |
| MsgBox "Reset " & msg & " Query Tables"  End Sub | Display the list of query tables encountered and do end the subroutine. |

To find out or set the query table name, go to the worksheet read from the remote source, select any cell in the area loaded from the remote source, in the data tab the properties button will highlight. Click on the properties button to get the following panel, where you can set the name of the query table (used in the above VBA macro) and consult or change some of the connection properties (for example the refresh time).



# Evolution demo

The following GIF animation shows 27 snapshots of the chart corresponding to the following charts taken with the intervals of one-and-half hour. The most recent call on the first chart is dated 2013-09-14 00:36:06 CET and the most recent call of the last chart is dated 2013-09-15 16:08:16 CET.

Below is the list of input PNG files used for the construction of the animated GIF file.

$ ls -1 \*.png

130909'020416..130914'003606+000734\_img46400.png

130909'020416..130914'020249+001109\_img46500.png

130909'020416..130914'033418+000958\_img46600.png

130909'020416..130914'050149+000946\_img46700.png

130909'020416..130914'063427+000725\_img46800.png

130909'020416..130914'080431+000740\_img46900.png

130909'020416..130914'093212+000721\_img47000.png

130909'020416..130914'110202+000755\_img47100.png

130909'020416..130914'123301+000719\_img47200.png

130909'020416..130914'135938+001111\_img47300.png

130909'020416..130914'152950+000820\_img47400.png

130909'020416..130914'170028+000805\_img47500.png

130909'020416..130914'182701+001104\_img47600.png

130909'020416..130914'200218+000807\_img47700.png

130909'020416..130914'213418+001138\_img47800.png

130909'020416..130914'230941+000831\_img47900.png

130909'020416..130915'004155+001143\_img48000.png

130909'020416..130915'021105+001442\_img48100.png

130909'020416..130915'034828+001242\_img48200.png

130909'020416..130915'052344+000935\_img48300.png

130909'020416..130915'065637+001204\_img48400.png

130909'020416..130915'083058+000955\_img48500.png

130909'020416..130915'100405+001213\_img48600.png

130909'020416..130915'113934+000859\_img48700.png

130909'020416..130915'130809+000854\_img48800.png

130909'020416..130915'143529+000859\_img48900.png

130909'020416..130915'160816+000955\_img49000.png

$

The GIF animation is created with image magic as follows.

$ cd gif

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

$ cd ..

Zoom out your browser if the animation does not fit in your screen.

You can observe the squeezing of the old stats on the left hand of the chart as the time evolves. In this animation cycle no html file rotation occurs. In case of the input files rotation, the days will disappear from the left, most squeezed edge of the chart, once in a while.

The [GIF file](gif/out.gif) presented above is 4.6 MB in size.

$ wc -c gif/out.gif

4582511 gif/out.gif

$

# Installation

Download the ZIP file of the last version from the <code> depository.

Unzip the content.

Edit the connect.txt file so as the first line contains the database MySQL server, the second line your username, and the third line your password.

Remove the sample SMTP connection files from the smtp subfolder. They do not contain valid passwords. The files are provided as examples. Add your own SMTP connection files in the smtp subfolder. We strongly recommend you to add more than one file.

Eventually, you may need to change the list of the recipients on the top of the bash script.

Launch the Cygwin window. Change the current directory to the folder of the downloaded and unzipped package. Run the bash script. It will create all files, if launched the first time, and will keep alimenting and rotating the html files. This script is also responsible for mailing.

$ ./aa41.sh.txt

collect between "2013-09-15 11:26:02" and "2013-09-15 16:16:28" at 2013-09-15 16:21:28 +0000

collect between "2013-09-15 16:16:29" and "2013-09-15 16:17:32" at 2013-09-15 16:22:32 +0000

collect between "2013-09-15 16:17:33" and "2013-09-15 16:18:36" at 2013-09-15 16:23:36 +0000

collect between "2013-09-15 16:18:37" and "2013-09-15 16:19:41" at 2013-09-15 16:24:41 +0000

collect between "2013-09-15 16:19:42" and "2013-09-15 16:20:46" at 2013-09-15 16:25:46 +0000

Open the Excel file. The chart sheet contains the current chart. Click on data refresh if you wish to force the loading from the html files. Otherwise the html files are loaded according to the frequency defined in the connections. In order to activate (or deactivate) the automatic generation of charts, click on the blue circle in the [code] worksheet.

# References

In this section you will find our previous publications concerning the interaction of Excel with a remote MySQL database.

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/>

# Acronyms

MySQL, My Structured Query Language

CDR, Call Data Records

HTML, Hyper Text Markup Language

BASH, Bourne Again Shell

VBA, Visual BASIC for Applications

BASIC, Beginner's All-Purpose Symbolic Instruction Code

PNG, Portable Network Graphics

PDD, Post Dial Delay

CLI, Caller Line Identification

CLD, Called line

SMTP, Simple Mail Transfer Protocol

IMAP, Internet Message Access Protocol

API, Application Programming Interface

YYMMDD, Year Year Month Month Day Day

HHMMSS, Hours Hours Minutes Minutes Seconds Seconds

# Legal

Copyright © by Emin Gabrielyan and switzernet.com

Linked is the MS-Word version of [this document](index4.docx).

END OF THIS DOCUMENT