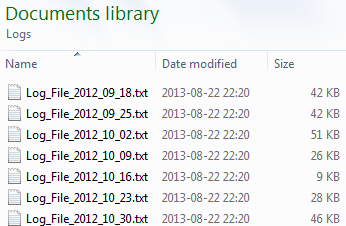
**QlikView Advance Scripting**

# Loops in the script

## For - Next Loops:

"For…Next" to create a loop. An enclosed Load will then be executed several times, once for each value of the loop counter or until the exit condition is met. Let’s take an example.

The most common case is that you have several files with the same structure, e.g. log files, and you want to load all of them:



   For each vFileName in Filelist ('C:\Path\\*.txt')

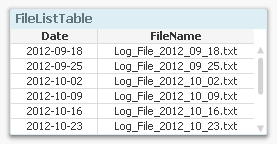
      Load \*,

         '$(vFileName)' as FileName

      From [$(vFileName)];

   Next vFileName

Another common case is that you already have loaded a separate table listing the files you want to load. Then you need to loop over the rows in this table, fetch the file name using the Peek() function, and load the listed file:



   For vFileNo = 1 to NoOfRows('FileListTable')

      Let vFileName = Peek('FileName',vFileNo-1,'FileListTable');

      Load \*,

         '$(vFileName)' as FileName

      From [$(vFileName)];

   Next vFileNo

## Looping over the same record

You can also have iterations inside the Load statement. I.e. during the execution of a Load statement the same input record is read several times. This will result in an output table that potentially has more records than the input table. There are two ways to do this: Either by using a While clause or by calling the Subfield() function.

Using a While Clause:

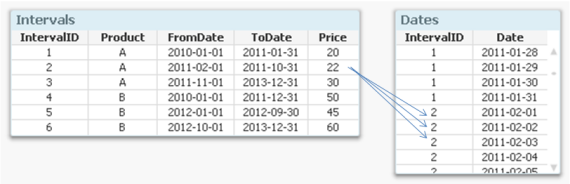
Dates:

   Load

      IntervalID, Date( FromDate + IterNo() - 1 ) as Date

      Resident Intervals

      While IterNo() <= ToDate - FromDate + 1 ;

[](https://community.qlik.com/servlet/JiveServlet/showImage/38-3274-45891/Intervals+with+Arrows.png)

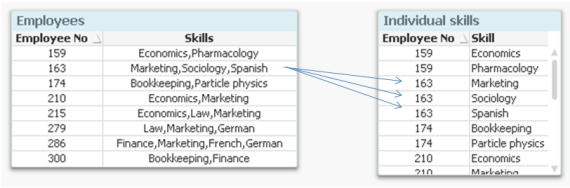
Using Subfield:

[Individual Skills]:

   Load

      [Employee No], SubField(Skills, ',') as Skill

      Resident Employees;

[](https://community.qlik.com/servlet/JiveServlet/showImage/38-3274-45892/Skills+w+Arrows.png)

## Loop through Folders and sub folders

The below script gets and stored all the Qlikview filenames in folder and subfolders.

Set vConcatenate = ;

FileList:  
LOAD  
'' AS SourceFile  
AUTOGENERATE 0;

sub ScanFolder(Root)

          for each FileExtension in 'qvw'

                    for each FoundFile in filelist( Root & '\\*.' & FileExtension)

                              FileList:  
                              LOAD '$(FoundFile)' as SourceFile  
       AUTOGENERATE 1;

                              Set vConcatenate = Concatenate;

                    next FoundFile

          next FileExtension

          for each SubDirectory in dirlist( Root & '\\*' )

                    call ScanFolder(SubDirectory)

          next SubDirectory

end sub

Call ScanFolder('C:\Jagan\QV Dashboards\QV Production Dashboards') ;

# Include and Must\_Include Variable:

The include and must\_include variables specify a file that contains text that should be included in the script. The entire script can thus be put in a file. This is a a user-defined variable.

The difference between include and must\_include is that **include will fail silently if the file is not found during script reload, while must\_include will throw an error if the file is not found**.

Examples:

$(Include=abc.txt);

$(Must\_Include=def.txt);

Let env = '<A absolute or relative file path, for example ...\Config\Test\>';  
**$(Include=$(env)\ConnectionString.txt)**

# Reload, Open and Open QlikView file without data using command

You can Reload, Open and Open QlikView file without data using Bat file or even in Qlikview

## Using Batch file (Reload, Open, Open QlikView file without data)

1. Create batch file with below code for reload only

**C:\Program Files\QlikView\qv.exe /R C:\Users\Icon\Desktop\Test.qvw**

1. Create batch file with below code for open QlikView file

**C:\Program Files\QlikView\qv.exe /O C:\Users\Icon\Desktop\Test.qvw**

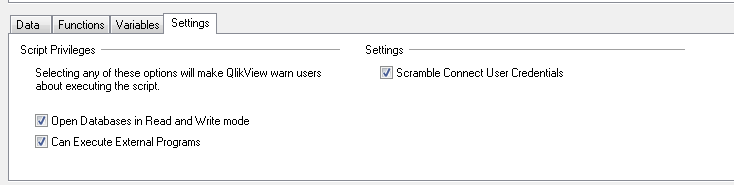
1. Create batch file with below code for to Open the Qlikview File without data

**C:\Program Files\QlikView\qv.exe /Nodata C:\Users\Icon\Desktop\Test.qvw**

## Within QlikView file (Reload, Open, Open QlikView file without data)

You can open other document and reload it within Qlikview, before run below code in Qlikview, you must enable some feature or option in Qlikview

Open Qlikview->Edit Script->Setting Tabs-> Enable **Can Execute External Programs**



1. Execute below code for reload only

**Execute C:\Program Files\QlikView\qv.exe /R C:\Users\User123\Desktop\Test.qvw**

1. Execute below code for open QlikView file

**Execute C:\Program Files\QlikView\qv.exe /O C:\Users\User123\Desktop\Test.qvw**

1. Execute below code for to Open the Qlikview File without data

**Execute C:\Program Files\QlikView\qv.exe /Nodata C:\User123\ Govind \Desktop\Test.qvw**

1. Execute below code for Create folder

**Execute cmd.exe /c mkdir C:\Users\User123\Desktop\Test**

1. **Create a Batch file in the Qlikview Script**

The below create a batch file within Qlikview Script:

BatFile:

Load

  'REM This is a batch file test' As [@echo off]

AutoGenerate(1);

For Each vFile in FileList('c:\temp\Folder1\\*.txt')

  BatFile:

  Load

  'MOVE $(vFile) C:\Temp\Folder2' As [@echo off]

  AutoGenerate(1);

Next

Store BatFile into c:\temp\test.bat (txt);

Drop Table BatFile;

EXECUTE c:\temp\test.bat;

# Trace

The trace statement writes a string to the Script Execution Progress window and to the script log file, when used. It is very useful for debugging purposes. Using variables one can customize the message.

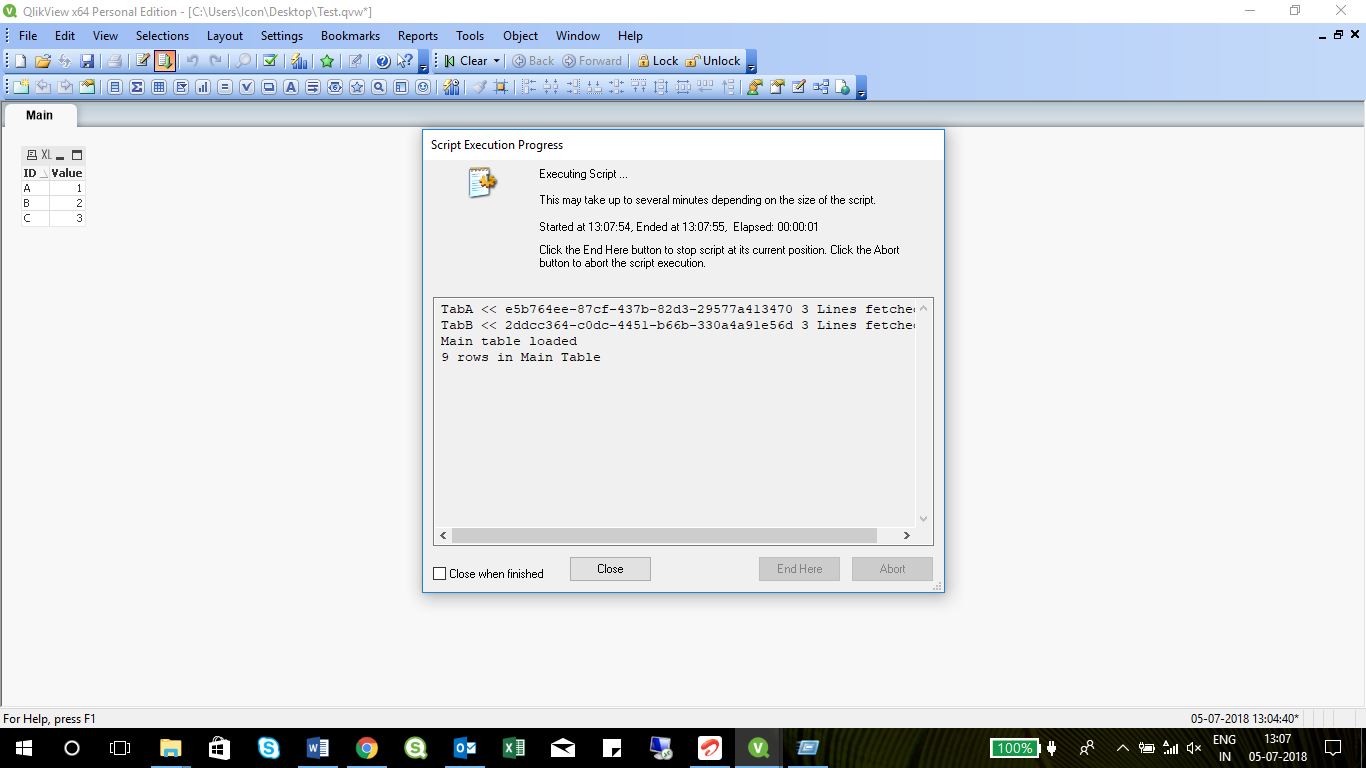
**Examples**:

trace Main table loaded;

Let MyMessage = NoOfRows('MainTable') & ' rows in Main Table';

trace $(MyMessage);

Output:



# Qvd Load Runtime

The Sub Runtime calculates the time and count of rows from data source and stores it in a QVD. So you can watch the change of runtime and volume over time.

a. Insert the following Sub in a new tab in your generator

**LET vStart = gmt();  
sub** **RunTime**(**vFact**,**vQVDName**)

// The sub needs to be called after the extraction is finished!  
// vFact holds the name of the extraction e.g. Sales data  
// vQvDName holds the name of the Runtime QVD to be created   
// Prequirement: Variable vStart must be defined and set before calling the sub!

// needed for first set of runtime  
TmpRunTime:  
**Load** \* INLINE [  
Tmp  
$(vFact)  
];   
  
// now calculate runtime and store in qvd

**LET** ***vDiff*** = time(gmt() -vStart); // calculates the difference  
   **LET** ***vDuration*** = second(vDiff)+minute(vDiff)\*60+hour(vDiff)\*3600;  // calculate time in seconds  
  
    ***$(vFact)***:  
    **load** \* from ***$(vQVDName)***(qvd);  // load previous data  
    **load**

***Tmp***  as Fact,                                 // append new data  
       ***$(vDuration)***  as Duration,   
       ***$(vNrOfRows)*** as NoRecords,   
       today() as CreateDate  
     resident TmpRunTime;  
     **store** ***$(vFact)*** into ***$(vQVDName)***;          
  
 **END** Sub

# ErrorHandling in Qlikview

Error handling is one of the most important—and overlooked—topics for programmers, regardless of the language used

There are variables in Qlikview to determine what action is to be taken by QlikView when an error is encountered during script execution. Let’s explain this concept with an example.

1. ErrorMode
2. ScriptError
3. ScriptErrorDetails
4. ScriptErrorCount
5. ScriptErrorList
6. ErrorMode**:**

Values assigning to this variable will decide what action to be taken by qlikview when an error is encountered.

**Set ErrorMode=0 🡪** Will ignore the failure and continue script execution.

**Set ErrorMode= 1 🡪**Will halt the script execution when an error is encountered and prompt user for action.

**Set ErrorMode= 2 🡪**Will display an "Execution of script failed..." error message immediately on failure, without prompting the user for action

1. ScriptError:

This variable will return the error code of the last executed script statement. Each error code has corresponding meaning.

**Error code:**

1🡪No Error. 4🡪General ODBC Error. 7🡪General HTML Error 10🡪 Table Not Found.

2🡪General Error. 5🡪General OLEDB Error. 8🡪 File Not Found. 11🡪Field Not Found.

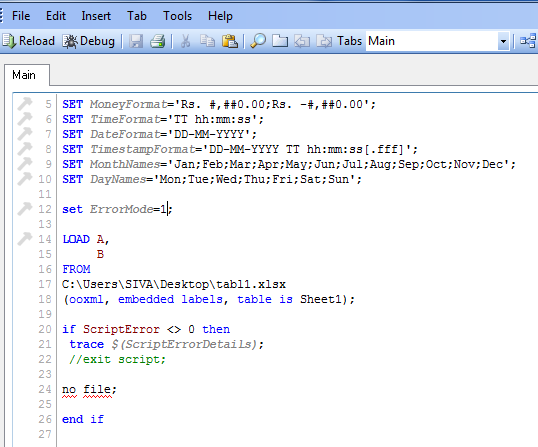
3🡪 Syntax Error. 6🡪General XML Error. 9🡪 Database Not Found 12🡪File Has Wrong Format

13→BIFF Error. 14→BIFF Error Encrypted. 15→BIFF error unsupported version. 16→Semantic Error

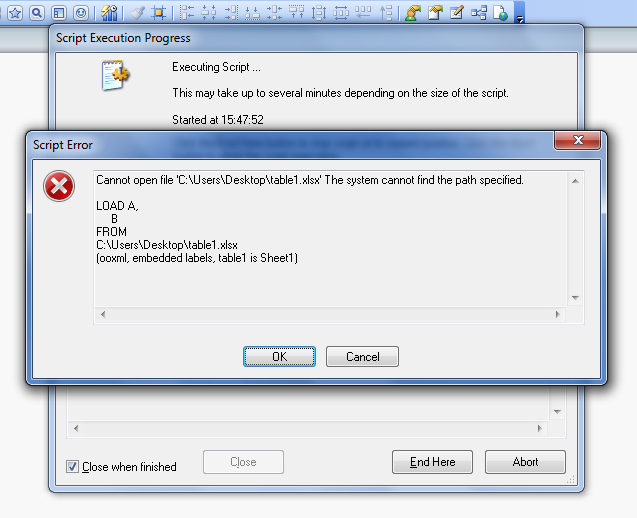
## **ScriptErrorDetails**: Returns detailed description of the error.

1. ScriptErrorCount:Returns total number of statements that have caused error.
2. ScriptErrorList: It will contain concatenated list of all script errors that have occurred during script execution.

Example: Writing an Qlikview script to show how this error variable works. Lets load an file that doesn’t exist in the mentioned location.



Now, run the QVW file to test this process. It will throw the error as shown below.



# Incremental Load

## Incremental Load concepts in Qlikivew using SQL

**Set** ***vNAV\_Database*** = 'QLIKVIEW\_NAV\_DB.dbo.';  
**SET** ***vQVDFileDir*** = 'Source QVDs NAV\'; // Path to QVD file(s)  
**SET** ***vIncrementalDate*** = '[Posting Date]'; // Enter DateTime Stamp field from DB  
**SET** ***vDropTable*** = "'True'"; // 'True' or 'False' on whether to keep data in memory  
**LET** ***vThisExecTime*** = today(); // Set Data Time Stamp of Load  
**LET** ***vPrevQtrStart*** = QuarterStart(today(),-1,4); // Set QuarterStart of Previous Quarter wrt Reload Date  
**LET** ***vCurrQtrStart*** = QuarterStart(today(),0,4); // Set QuarterStart of Current Quarter wrt Reload Date

**--------------------------------------------------------------------------------------------------------------------------------------**

**SET** ***vTableName\_SRT*** = 'FACT\_QV\_SalesReturn\_Transaction'; // Database table name  
**SET** ***vFinalQVDName\_SRT*** = 'FACT\_SalesReturn\_Transaction'; // Final QVD Name  
**SET** ***vInitQVDName\_SRT*** = 'FACT\_SalesReturn\_Transaction\_init'; // Initial QVD Name  
**SET** ***vIncrementalDate\_SRT*** = '[SRNDET\_DOCDATE]'; // Enter DateTime Stamp field from DB  
  
**SET** ***vSQL\_Columns\_SRT***= // SQL Load Block  
"   
 [SRNDET\_RID],  
 [SRNDET\_LOCID],  
 [SRNDET\_FINYEAR],  
 [SRNDET\_SGID]"  
;  
**SET** ***vQVD\_Columns\_SRT***= // QVD Load Block  
"  
 [SRNDET\_RID],  
 [SRNDET\_LOCID],  
 [SRNDET\_FINYEAR],  
 [SRNDET\_SGID]"  
;  
  
//Inital Load OR load on start of every new Quarter  
  
**IF** ISNULL(QvdCreateTime('$(vQVDFileDir)$(vInitQVDName\_SRT).qvd')) OR ***$(vThisExecTime)*** = ***$(vCurrQtrStart)***   
 OR ***$(vReloadTypeFlag)*** = 1 then   
  
 //Create QVD with data till previous quarter  
 [$(vFinalQVDName\_SRT)]:  
 SQL SELECT   
 ***$(vSQL\_Columns\_SRT)***  
 FROM ***$(vSIM\_Database)$(vTableName\_SRT)***  
 WHERE ***$(vIncrementalDate\_SRT)*** < '$(vPrevQtrStart)'  
 ;  
   
 **If** ScriptErrorCount = 0 then  
 //Store Table into QVD  
 **STORE** [$(vFinalQVDName\_SRT)] INTO [$(vQVDFileDir)$(vInitQVDName\_SRT).qvd];  
 **If** ***$(vDropTable)*** = 'True' then  
 **DROP** TABLE [$(vFinalQVDName\_SRT)];  
 **End** If  
 **End** If  
   
 //Merge above QVD with data from SQL from previous quarter till date   
 [$(vFinalQVDName\_SRT)]:  
 **SQL** SELECT   
 ***$(vSQL\_Columns\_SRT)***  
 FROM ***$(vSIM\_Database)$(vTableName\_SRT)***  
 WHERE ***$(vIncrementalDate\_SRT)*** >= '$(vPrevQtrStart)'  
 AND ***$(vIncrementalDate\_SRT)*** <= '$(vThisExecTime)'  
 ;  
   
   
  
 Concatenate  
 **LOAD**  
 ***$(vQVD\_Columns\_SRT)***  
 FROM [$(vQVDFileDir)$(vInitQVDName\_SRT).qvd] (qvd)  
 ;  
   
   
 **If** ScriptErrorCount = 0 then  
 //Store Table into QVD  
 **STORE** [$(vFinalQVDName\_SRT)] INTO [$(vQVDFileDir)$(vFinalQVDName\_SRT).qvd];  
 //Set Last Execution Time  
 **LET** ***vLastExecTime*** = vThisExecTime;  
 //Drop Table?  
 **If** ***$(vDropTable)*** = 'True' then  
 **DROP** TABLE [$(vFinalQVDName\_SRT)];  
 **End** If  
 **End** If   
  
//Incremental Load  
**ELSE**   
  
 [$(vFinalQVDName\_SRT)]:  
 **SQL** SELECT   
 ***$(vSQL\_Columns\_SRT)***  
 FROM ***$(vSIM\_Database)$(vTableName\_SRT)***  
 WHERE ***$(vIncrementalDate\_SRT)*** >= '$(vPrevQtrStart)'  
 AND ***$(vIncrementalDate\_SRT)*** <= '$(vThisExecTime)'  
 ;  
   
   
 Concatenate  
 **LOAD**  
 ***$(vQVD\_Columns\_SRT)***  
 FROM [$(vQVDFileDir)$(vInitQVDName\_SRT).qvd] (qvd)  
 ;  
   
   
   
 **If** ScriptErrorCount = 0 then  
 //Store Table into QVD  
 **STORE** [$(vFinalQVDName\_SRT)] INTO [$(vQVDFileDir)$(vFinalQVDName\_SRT).qvd];  
 //Set Last Execution Time  
 **LET** ***vLastExecTime*** = vThisExecTime;  
 //Drop Table?  
 **If** ***$(vDropTable)*** = 'True' then  
 **DROP** TABLE [$(vFinalQVDName\_SRT)];  
 **End** If  
 **End** If   
   
**ENDIF**

## Incremental Loading of Multiple QVDs from a List of Table Names

Below is a script that was developed to quickly pull the fields from a list of tables into QVDs and perform incremental loads on the QVDs going forward.

**Set** ***vQVDPath*** = QVDs/; //Enter Location to save QVDs in the path should end it back slash.For example: C:\MyQVDs\

//Set the variables so the last time of script execution is known  
**SET** ***vLastExecTime***          = 0; // resetting vLastExecTime  
**LET** ***vLastExecTime***          = timestamp(if(isnull(QvdCreateTime('$(vQVDPath)ReloadHistory.QVD')), 0,ConvertToLocalTime(QvdCreateTime('$(vQVDPath)ReloadHistory.QVD'), 'GMT', 1)), 'YYYY-MM-DD hh:mm:ss');  
**LET** ***vExecTime***                  = timestamp(UTC(), 'YYYY-MM-DD hh:mm:ss');  
**LET** ***vTodaysDate***     = today();  
  
  
//For the 1st reload, this section will be skipped.  
**LET** ***FirstReload*** = isnull(QvdCreateTime('$(vQVDPath)ReloadHistory.QVD'));  
  
**if** Not ***$(FirstReload)*** then  
// Read Reload History Data  
ReloadHistory:  
**Load**   
No,   
[Last Reload Ended],  
[Reload Started]   
FROM ***$(vQVDPath)***ReloadHistory.qvd (qvd);  
  
**end** if  
  
ReloadHistory:  
**LOAD**   
RowNo() as No,  
'$(vLastExecTime)' as [Last Reload Ended],  
'$(vExecTime)' as [Reload Started]  
Autogenerate(1);  
  
//ReloadHistory.qvd will keep track of when the last reload occurred for incremental loading purposes. If you need to reload all the data then delete ReloadHistory.qvd from your file structure.  
 **STORE** \* FROM ReloadHistory INTO ***$(vQVDPath)***ReloadHistory.qvd;  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Add a list of tables to be loaded from your data source below.  Keep the first row "TableName".  This is the name of the field.  
TablesList:  
**Load** \* Inline [  
TableName  
Account  
Opportunity  
];  
  
**For** ***i*** = 0 to (NoOfRows('TablesList')-1);  
**Let** ***varTableName*** = Peek('TableName',***$(i)***, 'TablesList');  
  
// Load Tables  
                ***$(varTableName)***\_SQL:  
**SQL** Select \*  
FROM ***$(varTableName)***  
WHERE LastModifiedDate >=***$(vLastExecTime)*** and LastModifiedDate < ***$(vExecTime)***;   //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Change the "LastModifiedDate" to the name of the field for the last modified data time stamp in your tables.  
                  
// For the 1st reload, this section will be skipped.  
**if** Not ***$(FirstReload)*** and not isnull(QvdCreateTime('$(vQVDPath)$(varTableName)\_SQL.qvd')) then  
  
//CONCATENATE ($(varTableName))   
***$(varTableName)***\_SQL:  
**Load** \*   
FROM ***$(vQVDPath)$(varTableName)***\_SQL.qvd (qvd)  
WHERE NOT EXISTS(Id);   // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Change "Id" to the primary key for your tables (note all your tables must use the same name for the primary key for this script to work.  
                  
**end** if  
  
// If table exists then proceed to the next step  
**if** NoOfRows('$(varTableName)\_SQL') > 0 then   
  
**STORE** ***$(varTableName)***\_SQL INTO ***$(vQVDPath)$(varTableName)***\_SQL.qvd;  
  
**DROP** TABLE ***$(varTableName)***\_SQL;  
  
**End** if  
**Next**;  
  
**Drop** Table TablesList;

# Partial Reload

Partial Reload is used whenever you just want to add some new data without reloading all other tables.

I am having an application with 3 CSV files with large set of Data

[airports.csv](http://www.ourairports.com/data/airports.csv) (6,824,049 bytes) large file, containing information on all airports on this site.in

[airport-frequencies.csv](http://www.ourairports.com/data/airport-frequencies.csv) (1,167,404 bytes)Large file, listing communication frequencies for the airports in airports.csv.

[runways.csv](http://www.ourairports.com/data/runways.csv) (2,844,907 )Large file, listing runways for the airports in airports.csv.

 The initial load will take around 5 seconds in my system and now I am adding the word ADD and REPLACE before [airports.csv](http://www.ourairports.com/data/airports.csv) and  [airport-frequencies.csv](http://www.ourairports.com/data/airport-frequencies.csv" \t "_blank)

LOAD id,

     airport\_ref,

     airport\_ident,

     length\_ft,

     width\_ft,

     surface,

     lighted,

     closed

FROM

C:\Users\user123\Desktop\runways.csv

(txt, codepage is 1252, embedded labels, delimiter is ',', msq);

ADD LOAD id,

     ident,

     type,

     name,

     latitude\_deg,

     longitude\_deg,

     elevation\_ft,

     continent

FROM

C:\Users\user123\Desktop\airports.csv

(txt, codepage is 1252, embedded labels, delimiter is ',', msq);

REPLACE LOAD id,

     airport\_ref,

     airport\_ident,

     type,

     description,

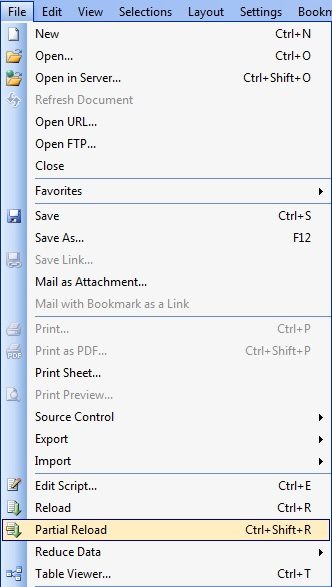
     frequency\_mhz

FROM

[C:\Users\user123\Desktop\airport-frequencies.csv]

(txt, utf8, embedded labels, delimiter is ',', msq);

while doing partial reload from file menu,  
  
When executing a partial script reload, all lines are executed except for LOAD and SELECT not preceded by ADD or REPLACE.

[](https://community.qlik.com/servlet/JiveServlet/showImage/102-5427-1-49890/Partial-Reloads.jpg)

it will load only the  [airports.csv](http://www.ourairports.com/data/airports.csv) and [airport-frequencies.csv](http://www.ourairports.com/data/airport-frequencies.csv)(Based on the ADD and REPLACE) and the reload time is less than 2 seconds.

# Turning Unoptimized Loads into Optimized Loads

1. **Test 1**

When concatenating two tables that don't have the same number of fields, if the second table has the same fields than the first one and then some extra fields the load will still be optimized, if done the other way around it will not be optimized.

e.g. All Loads are optimized

1. Table:
2. LOAD               // Optimized
3. A,
4. B,
5. C
6. FROM Table1.qvd (qvd);
8. CONCATENATE(Table) // Optimized
9. LOAD
10. A,
11. B,
12. C,
13. D,
14. E
15. FROM Table2.qvd (qvd);

e.g. 2nd Load isn't optimized

1. Table:
2. LOAD               // Optimized
3. A,
4. B,
5. C,
6. D,
7. E
8. FROM Table2.qvd (qvd);
10. CONCATENATE(Table) // Not optimized
11. LOAD
12. A,
13. B,
14. C
15. FROM Table1.qvd (qvd);

1. **Test 2**

Second table has some fields in common with the first but is missing some, each table has 50 Million rows, 2nd load will not be optimized but loading the table in optimized mode, adding the missing fields, storing it and loading it again optimized will be faster than just concatenating the tables straight up.

e.g. 2nd load isnt optimized (in this example it took about 1 min to load. PC is Core i5 x64 4 GB running Windows 7).

1. R00:
2. LOAD ShipperID,       // Optimized
3. OrderDate,
4. CustomerID,
5. UnitPrice,
6. sales,
7. COS
8. FROM
9. R00\_1.QVD
10. (qvd);
12. Concatenate(R00)      // Not Optimized
13. LOAD ShipperID,
14. CustomerID,
15. Discount,
16. ProductID,
17. Quantity,
18. UnitPrice
19. FROM
20. R00\_2.QVD
21. (qvd);

If you load the 2nd table without concatenating it, add the missing fields store it and load it again to concatenate it while reading it optimized it will be faster (In my example took 50% of the time).

1. R00:
2. LOAD ShipperID,                // Optimized
3. OrderDate,
4. CustomerID,
5. UnitPrice,
6. sales,
7. COS
8. FROM
9. R00\_1.QVD
10. (qvd);
12. R000\_Aux:
13. LOAD ShipperID,                // Optimized
14. CustomerID,
15. Discount,
16. ProductID,
17. Quantity,
18. UnitPrice
19. FROM
20. R00\_2.QVD
21. (qvd);
23. concatenate(R000\_Aux)          // Not Optimized. 0 records are added
24. LOAD null() as ShipperID,
25. null() as OrderDate,
26. null() as CustomerID,
27. null() as UnitPrice,
28. null() as sales,
29. null() as COS
30. autogenerate(0);
32. store R000\_Aux into R000\_Aux.QVD;
33. drop table R000\_Aux;
35. concatenate(R00)                //This load will now be optimized!
36. LOAD ShipperID,
37. OrderDate,
38. CustomerID,
39. UnitPrice,
40. sales,
41. Discount,
42. ProductID,
43. Quantity,
44. COS
45. FROM
46. R000\_Aux.QVD
47. (qvd);

# Optimized Loads using Exits()

1. **Where exits:**

Whenever we are using to QVD or any source as data source, we need to keep the Optimized load better reload times. But in some cases we didn't keep optimized loads especially if we are using filters with where command.

**LOAD \* FROM QVDANAME.qvd (qvd) Where Match(REGION\_NAME,'US','CANADA');**

In the above case, we are loading the data only from US and CANADA region and this is unoptimized load. But we can load as Optimized load by using **EXISTS** key word with temp table.

First, create the Temp table with filters and use EXISTS key word in main QVD loading statement like below:

TEMP:

LOAD \* INLINE [

REGION\_NAME

US

CANADA ];

LOAD \* FROM QVDANAME.qvd (qvd) Where EXISTS (REGION\_NAME) ;

 DROP Table TEMP ;

1. **The exists issue**

 The exists function works as expected ONLY if the field it is applied to is a unique key. If it is not, only the first occurrence of the key is taken into account by the function.

test1:  
**LOAD** \* INLINE [  
F1, F2  
a, 21  
b, 422  
b, 412  
c, 424  
c, 48  
c, 42  
d, 24  
e, 56  
];

And we need to append table test2, but only the records where F1 does not exist already. An obvious way would be as follows (field names are the same in both tables, so auto concatenate will occur):

test2:  
**LOAD** \* INLINE [  
F1, F2  
a, 210  
a, 212  
a, 223  
b, 422  
f, 424  
h, 24  
h, 242  
g, 561  
g, 562  
g, 563  
g, 564  
g, 565  
]  
Where  
not Exists(F1);

The expected result would be a table containing all rows from test1 and all rows from test2 where F1 does not exist:

F1, F2  
a, 21  
b, 422  
b, 412  
c, 424  
c, 48  
c, 42  
d, 24  
e, 56  
f, 424  
h, 24  
h, 242  
g, 561  
g, 562  
g, 563  
g, 564  
g, 565

Well, it is not. F1 is not a unique key. We have several rows with the same value for F1 in the second table (same in the first table, but exists is applied to the second one, so the first is not affected). So what QlikView does is, it adds the first non-existing F1 value it encounters, for example row (h, 24). The next row, (h, 242) is NOT added as QlikView now considers value h to exist. The exists function even checks the current load for F1, not only the data loaded so far by previous reload statements.  
So the final result is 5 rows less than expected:

F1, F2  
a, 21  
b, 422  
b, 412  
c, 424  
c, 48  
c, 42  
d, 24  
e, 56  
f, 424  
h, 24  
g, 561

This issue is with us like since always, so obviously this is not a bug, it is a feature. So extra caution is needed when applying exists. A common case where this may cause unexpected behavior is in incremental reloads.

An easy workaround would be to create a 3rd field, F1 as F1\_lookup, and then apply the exists function as follows :

Where  
not Exists(F1\_lookup, F1)

1. **Optimized QVD Load With a Where Clause and Dates**

If you are working with large quantities of data and especially if you are working with large data sets of historical data where you add a portion of new data to the historical data each week (an incremental load), then you will want to be working with qvd files in your loadscript. And if you are working with qvd files then you certainly want to load data from the qvd files with anoptimized load. The optimized load is many times faster than loading the same data with an unoptimized load.

Imagine that you have a date like 01/01/2014 in a variable named start\_dtand another date like 12/31/2015 in a variable named end\_dt and you want to load invoice data where the invoice date, inv\_date, is between start\_dt and end\_dt. First, build a little table of all possible dates between start\_dt and end\_dt. Use this in the loadscript before loading any other data containing a INV\_DATE field:  
  
/\* Build table of dates just for the "where exists" clause \*/  
**TMP\_DATE\_TABLE:  
Load date(date('$(start\_dt)')+IterNo()-1) as INV\_DATE   
AutoGenerate(1)  
While date(date('$(start\_dt)')+IterNo()-1) <= date('$(end\_dt)');**   
  
That code can build a large table of dates in your loadscript in just a few seconds. Now, you can code the load from qvd like this:  
  
**INVOICE\_DATA:   //this is an optimized load  
Load \* from invoice\_data.qvd (qvd)   
where exists(INV\_DATE);  
Drop table TMP\_DATE\_TABLE; //no longer needed**