



Bardess Bytes of Wisdom: 50 Qlik Tips to Accelerate Your Development

“Bardess Bytes of Wisdom: 50 Qlik Tips to Accelerate Your Development is a great resource for any Qlik user at any level. I learned so much and I recommend this book to anyone looking to get more from their data.”

— Mike Muglia, Qlik Luminary, 2018-Present

Foreword

“I have had the pleasure of working on numerous projects with Bardess Group over the years and they bring a level of expertise that is truly unmatched. Bardess Bytes of Wisdom: 50 Qlik Tips to Accelerate Your Development is a great resource for any Qlik user at any level. I learned so much and I recommend this book to anyone looking to get more from their data.”

— **Mike Muglia**, Qlik Luminary, 2018-Present

“I am excited about introducing this first eBook in our knowledge-sharing series, Bardess Bytes of Wisdom, which is all about people helping people with data. This eBook presents innovative ways to use Qlik technology and accelerate user adoption. It brings Bardess’ thought leadership to the forefront, and any user will benefit from our collected knowledge and hands on experience.”

— **Barbara Pound**, CEO Bardess Group, Ltd.

“Bardess has many talented and passionate Qlik developers, and they represent the best and brightest advocates of this premier data analytics platform. This book will allow you to take advantage of their expertise and commitment.”

— **Joe DeSiena**, President Bardess Group, Ltd.

Table of Contents

- 01 - Call to Action: Subroutines
- 02 - Generate a Table Without a Source
- 03 - Change Inline Load Delimiter Setting to Copy from Excel
- 04 - Create a Change Log to Track Your Work
- 05 - Naming Conventions are Crucial to Consistency
- 06 - Clear Your Variables to Save Time Debugging
- 07 - Use Variable Parameters to Create Your Own Functions
- 08 - Setup A Lab Environment
- 09 - Develop Faster Using Test Cases
- 10 - An Overlooked Gem
- 11 - Understanding Join Distinct
- 12 - Dual Data Type Caveat
- 13 - Configuring the Rules for Qlik Sense® Ports
- 14 - Quickly Rename Fields with One Statement
- 15 - Use Preceding Load to Enhance Readability
- 16 - Load the Entire Contents of a File as a Value
- 17 - Running Dynamically Created Code
- 18 - Avoid the Rabbit Hole
- 19 - Use Placeholder Tables to Establish Field Formats
- 20 - Use ApplyMap for Fast VLookup Functionality
- 21 - Do While: How to Loop Until a Condition is Met
- 22 - Cardinality and How it Impacts Application Size
- 23 - Use AutoNumber to Optimize Your Data Model
- 24 - Effectively Work with Qlik Key Fields
- 25 - Ignore Potential Future Use Cases and Remove Unused Fields
- 26 - Using Arrays for Iteration

27 - Use SubField to Expand Records

28 - Use Wildcard to Load All Similar Files

29 - Dual Behavior in Set Analysis

30 - Use Copy/Paste to Save Time and Avoid Typos

31 - Pick Match, an If Statement Equivalent

32 - How to Associate Mixed Granularity Data

33 - When Should You Upgrade Qlik Sense®?

34 - Create A Codebase

35 - Using Advanced Search to Filter in Set Analysis

36 - Changing Column Width

37 - Making Date Formats Functional

38 - Create Your Own Concatenated Keys

39 - Looping Over Records Using While

40 - Troubleshooting Associations: Subset Ratio

41 - Hidden Default Apps

42 - Using FieldValueList for Loops

43 - Did You Try Turning It Off and On Again?

44 - FileList Mask Order Test

45 - How to Create a Quick Month Map

46 - Get QVD Metadata from XML Headers

47 - Where do Reload Logs Reside?

48 - Natively Loading JSON

49 - Section Access Tips and Tricks

50 - Understanding QVDs and Optimized Loads

Conclusion

Call to Action: Subroutines

To start it off we are going to talk about Subroutines. A subroutine is a user defined program within your script that you can call at a later time.

For example:

```
Sub RowCount(pRowTable)

    Let RC.Rows = Num(NoOfRows('$ (pRowTable)'), '#,##0');

    Trace ----- $(pRowTable): $(RC.Rows) rows;

    RC.Rows=;

End Sub;

Fact:
Load
    Rand() as Random
AutoGenerate
    (Ceil(Rand()*100));

CALL RowCount('Fact');
```

Which returns:

Data load progress

Data load is complete.

Elapsed time 00:00:00

Started loading data

Fact << AUTOGENERATE(1000)
Lines fetched: 1,000
Temp << AUTOGENERATE(1)
Lines fetched: 2
Temp
Lines fetched: 2
----- Fact: 2,000 rows

App saved

Finished successfully
0 forced error(s)
0 synthetic key(s)

☐ Close when successfully finished

Close

This is a very simple example, but you can start to see the possibilities.

Generate a Table Without a Source

You might have noticed we used this in the last example. AutoGenerate is so useful and has so many use cases. We probably use this in every app we create: from empty stub tables, generating a new field using another's unique values, or as a way to log different steps within the load.

What it does is generate a table with the number of rows specified, allowing you to create records with the output of functions, variables or strings without needing to specify a source.

Here is an example, which includes another tip if you catch it:

```
// Create empty table and setting field types
```

```
Log:
```

```
Load
```

```
    Null()                as Log.Table,
    Num(Null(), '#,##0')  as Log.Rows,
    Timestamp(Null())     as Log.Timestamp
```

```
AutoGenerate
```

```
    (0);
```

```
Fact:
```

```
LOAD
```

```
    TransLineID,
    TransID,
    "Num",
    Dim1,
    Dim2,
    Expression1
```

```
FROM
```

```
[lib://QVD/SampleTransactions.qvd](qvd);
```

```
// Add Log record
Concatenate(Log)
Load
    'Fact'                as Log.Table,
    NoOfRows('Fact')     as Log.Rows,
    Now()                 as Log.Timestamp
AutoGenerate
    (1);
```

Which returns:

Log
Log.Table
Log.Rows
Log.Timestamp

Preview of data

Log.Table	Log.Rows	Log.Timestamp
Fact	2,057	4/20/2019 6:45:41 PM

Changing the Inline Load Delimiter

One thing which would have been useful years ago was knowing we could copy data directly from Excel into the load script for an Inline Load if we changed the delimiter setting in the load script.

What this means is you can copy a data set from Excel:

	A	B	C	D
1	Region	Year	Sales	
2	North	2018	100	
3	North	2019	75	
4	West	2019	54	
5	East	2017	560	
6	East	2018	234	
7	East	2019	345	
8	South	2018	995	
9	South	2019	939	
10				
11				
12				

Paste into Qlik and add '(delimiter is \t)' at the end of the Inline Load statement.

** \t is for tab, our Excel default delimiter. You can change this to whatever you want. See the [documentation](#) for more info.*

1	
2	Fact:
3	Load * Inline [
4	Region Year Sales
5	North 2018 100
6	North 2019 75
7	West 2019 54
8	East 2017 560
9	East 2018 234
10	East 2019 345
11	South 2018 995
12	South 2019 939
13] (delimiter is \t);

Reload and there you have it!

Fact
Region
Year
Sales

▼ Preview

Fact	
Rows	8
Fields	3
Tags	\$ascii \$text \$numeric \$integer

Fact		
Region	Year	Sales
North	2018	100
North	2019	75
West	2019	54
East	2017	560
East	2018	234
East	2019	345
South	2018	995

Create a Change Log to Track Your Work

This is a process we wish was implemented sooner. Currently, in every app we build, we create a script section named Change Log. This little piece of documentation gives a bit of background on the app, and then a list of changes over time with tags to be able to find where they were done easily.

Example

```
///$tab Subroutine
Sub RowCount(pRowTable)

    Let RC.Rows = Num(NoOfRows('$ (pRowTable)'), '#,##0');
    Trace ----- $(pRowTable): $(RC.Rows) rows;
    RC.Rows=;

End Sub;

Fact:
Load
    Rand() as Random
AutoGenerate
    (100000);                // [v1.1.01]

CALL RowCount('Fact');      // [v1.1.02]

Temp:
Load
    Chr(IterNo()) as Char
AutoGenerate
    (1)
While
    IterNo() <= 2;

Outer Join(Fact)
Load
    Char
Resident
    Temp;
```

```
Drop Table Temp;
```

```
CALL RowCount('Fact');
```

```
///$tab Exit  
exit script;
```

```
///$tab Change Log
```

```
/*****
```

```
Application: 101 - Subroutine
```

```
Description:
```

```
Example application to provide a simple example of how a subroutine  
works.
```

```
*****  
*****
```

```
Version: 1.1
```

```
Date: 2019-05-01
```

```
Notes: None
```

```
Changes:
```

- Upped the fact records from 1000 to 100000 [v.1.1.01]
- Added additional RowCount call. [v1.1.02]

```
*****  
*****
```

```
Version: 1.0
```

```
Date: 2019-041-12
```

```
Notes: None
```

```
Changes:
```

- Initial Build

```
*****  
*****
```

```
Version: X.X (Change Log Definition)
```

```
Date: %Date of Change%
```

Author: %Full Name% (%Email Address%)

Notes:

Used for change log template.

[Tags] can be used to take advantage of the search box to navigate directly to change specified.

Changes:

- [vX.X.x1]
- [vX.X.x2]
- [vX.X.x3]

***** /

Naming Conventions are Crucial to Consistency

Like most other programming languages, it is important to decide and follow some programming standards. One key standard is how to name things. We do not propose that you should name things the way we do, that isn't the point. What is important is to keep the way you name items consistent.

As an example, here are a few ways we name things.

Type	Standard	Example
Regular Variable	'v' Prefix	vStartDate
Expression Variable	'x' Prefix	xNetSales
Set Analysis Variable	'set' Prefix	setWTD
Field Name	Snake Case	order_amount
Key Field	'%' Prefix	%product
Flag Field	'_flag' Suffix	ytd_flag

As consultants, we interact with a lot of our client's apps. In these scenarios, we will take on their way of doing things to keep the app consistent. We do this so that we are not merging two different styles. Keep in mind that consistency is the key, and that consistency may require using different standards.

Clear Your Variables to Save Time Debugging

This is a very short and sweet one. Clear your variables. Trust us, it is worth the effort. It's easy and will save you time debugging.

To clear a variable just do the following:

```
RC.Rows=;
```

That's it.

Use Variable Parameters to Create Your Own Functions

In other words, user defined functions. By utilizing variable parameters, you can create your own functions to use later.

For example, a function to calculate the number of weeks between two date fields:

```
SET fWeekDiff = '(((Year($2)*52)+Week($2))) - (((Year($1)*52)+Week($1)))';
```

```

Data:
Load
    StartDate,
    EndDate,
    $(fWeekDiff(StartDate,EndDate)) as NoOfWeeks
;
Load
    Date#(StartDate, 'M/D/YYYY') as StartDate,
    Date#(EndDate, 'M/D/YYYY') as EndDate
Inline [
StartDate, EndDate
1/3/2017, 2/14/2017
2/14/2017, 6/1/2017
6/1/2017, 8/14/2017
];

```

Setup A Lab Environment

While it is great that Qlik® offers a Qlik Sense® desktop version, we usually suggest trying to set up a server environment for local development. We often run a Windows Server VM with Qlik Sense Enterprise installed using a unified license, which can be accessed in a browser.

We do it this way for three main reasons.

1. It is the closest local environment to what the majority of customers have installed.
2. It keeps it contained and we can easily take snapshots and do rollbacks.
3. When using MacOS it is cheaper than paying for a hosting service.

While we often choose to work in a VM, it is worth knowing that you can install Qlik Sense® Enterprise locally on Windows 8+. We have found that this has made the situation exponentially better for development and testing compared to using the desktop versions. By using a unified license, applying the same signed key to multiple deployments lets you share the same users and access types. Users can access all connected deployments using the same Professional or Analyzer access allocation.

Develop Faster Using Test Cases

We are firm believers that boiling down a problem to its simplest form is the fastest path to a solution. One of the fundamentals of programming is the ability to break a problem into several smaller problems. If you combine these concepts together, you can create a large number of test cases.

We find that we are able to develop faster and more accurately by separating the particular problems out of the application and testing it on the simplest scenario possible. Another benefit is you can keep these examples for later reference. Here is an example we made on [NullAsValue](#). We created this test case in a separate app in a couple of minutes, versus trying it on a client's app over hundreds of millions of rows with a reload taking hours.

Test Case: NullAsValue Execution Test

Description: This Test is to determine whether the NullAsValue gets applied on the data being loaded in or on the resulting data.

Result: On the resulting data

```
status_map:
Mapping Load * Inline [
input, output
1, one
2, two
3, three
];
```

```
Raw:
Load
    'A'      as dim,
    1        as num
AutoGenerate
    (1);
```



```
Concatenate(Raw)
Load
    'B'          as dim,
    Null()       as num
AutoGenerate
    (1);
```

```
Concatenate(Raw)
Load
    'C'          as dim,
    3            as num
AutoGenerate
    (1);
```

```
NullAsValue num;
NullValue = 2;
```

```
Check:
NoConcatenate Load
    dim,
    ApplyMap('status_map', num) as num
resident
    Raw;
```

```
Drop table Raw;
```

An Overlooked Gem - Controlling Data with Set Analysis

You would probably be surprised that this is one of our most used functions within a dashboard. It is just such a cool function, though it is almost exclusively used for UI/UX functionality.

Only()

Only is an aggregation function that can be used on a string field. If a single result comes back it will return the result, if more than one value occurs it returns null.

Example

FieldA	FieldB
Apple	Fuji
Apple	Honey Crip

=Only(FieldA) would return **Apple**

=Only(FieldB) would return NULL

Documentation

One of the best features is that you can use set analysis with it, which gives you a lot of control. It can be used to pull out expression definitions stored in a field, used to create dynamic labels, and many other things.

Understanding Join Distinct

This is something that we discovered the hard way. The Qlik® Associative Engine applies the DISTINCT keyword on the resulting dataset. Therefore, when you perform a join distinct, you are not joining the distinct table to the other table. You are performing the join and then returning the distinct records from the resulting join.

Take this example:

`original:`

```
Load * Inline [  
id, dim, sales  
1, A, 100  
2, A, 50  
3, B, 75  
3, B, 75  
4, C, 1000  
];
```

`Left Join(original)`

```
LOAD Distinct * Inline [  
dim, desc  
A, Big Fish  
B, Reoccurring  
B, Reoccurring  
C, High Roller  
];
```

You might expect:

id	dim	sales	desc
1	A	100	Big Fish
2	A	50	Big Fish
3	B	75	Reoccurring
3	B	75	Reoccurring
4	C	1000	High Roller

However, what you actually get is:

id	dim	sales	desc
1	A	100	Big Fish
2	A	50	Big Fish
3	B	75	Reoccurring
4	C	1000	High Roller

If you want to achieve the first result, you will need to do something like the following.

original:

```
Load * Inline [  
id, dim, sales  
1, A, 100  
2, A, 50  
3, B, 75  
3, B, 75  
4, C, 1000  
];
```

tmp:

```
LOAD Distinct * Inline [  
dim, desc  
A, Big Fish  
B, Reoccurring  
B, Reoccurring  
C, High Roller  
];
```

```
Left Join(original)
```

```
Load
```

```
    *
```

```
Resident
```

```
    tmp;
```

```
Drop Table tmp;
```

Dual Data Type Caveat

Dual combines a number and a string into a single record, such that the number representation of the record can be used for sorting and calculation purposes, while the string value can be used for display purposes.

Syntax:

```
Dual(text, number)
```

Caveat

When we think of the data type combining both text and a number value, we tend to think that this means a value can be the unique combination of the two. However, this is not the case.

The number in Dual is the core value, while the text is just the display value.

What does this mean?

This means a number value can only have **one** display value. While a text value can be the display value of multiple numbers

Example

Data:

Load

```
RowNo()          as Id,
Text(String)     as String,
Num(Number)      as Number,
Dual(String,Number) as Dual
```

Inline [

String, Number

A, 1

B, 2

C, 3

A, 4

Z, 1

];

Output

	Id	Q	String	Q	Number	Q	Dual	Q
		1	A			1	A	
		2	B			2	B	
		3	C			3	C	
		4	A			4	A	
		5	Z			1	A	

Troubleshooting

We have made this mistake when trying to combine mixed granularity within dates. Say at the end of the year a client's finance team closes the books and closes their general ledger. They then report this at a year level. However, in the current year, they report at the month level.

To incorporate both sets of data in a chart we came up with the concept of doing Month-Year, with PY being a full previous year.

Month	MonthNum	Year	MonthYear Dual
PY	0	2018	PY-2018
Jan	1	2019	Jan-19
Feb	2	2019	Feb-19
Mar	3	2019	Mar-19
Apr	4	2019	Apr-19
May	5	2019	May-19
Jun	6	2019	Jun-19
Jul	7	2019	Jul-19
Aug	8	2019	Aug-19
Sep	9	2019	Sep-19
Oct	10	2019	Oct-19
Nov	11	2019	Nov-19
Dec	12	2019	Dec-19

Then a new requirement came and a particular budget was only at the year level. So we stuck with a similar concept, but this time CY.

Month	MonthNum	Year	MonthYear Dual
PY	0	2018	PY-2018
CY	0	2019	CY-2019
Jan	1	2019	Jan-19
Feb	2	2019	Feb-19
Mar	3	2019	Mar-19
Apr	4	2019	Apr-19
May	5	2019	May-19
Jun	6	2019	Jun-19
Jul	7	2019	Jul-19
Aug	8	2019	Aug-19
Sep	9	2019	Sep-19
Oct	10	2019	Oct-19
Nov	11	2019	Nov-19
Dec	12	2019	Dec-19

Now this worked out as is. However, we made the Month field a **Dual** and it took us far too long to realize that `Dual('PY', 0)` and `Dual('CY', 0)` would not provide the expected results. It would always default to PY since there can only be one display value per number.

Hopefully knowing this can save you some time in the future.

Configuring the Rules for Qlik Sense® Ports

Qlik® has great documentation, however it is a bit cumbersome when it talks about networking. Below are the ports needed.

Also, you need to create an inbound and outbound rule. [Here](#) is a tutorial. It is much less complicated than it sounds.

*P.S. We are not covering HTTP because you should **never** do it.*

Single Node

Port	Purpose	Comment
443	HTTPS Traffic	Absolutely necessary
4244	Windows Authentication	*Only versions prior to April 2018
4242	Qlik Repository	*Only if you want to talk to the QRS API

Multi Node

We suggest looking at the [documentation](#) since there are so many variations depending on the use case.

Quickly Rename Fields with One Statement

This is a cool trick that we use a lot because of preference. The idea is you can rename all the fields within your Qlik Sense® application using one statement, two if you count the prep work.

```
data:
Load
    'Test'          as data_name,
    Rand()          as random_number
AutoGenerate
    (100);

field_rename_map:
Mapping Load
    FieldName(IterNo(), 'data')          as old_name,
    Capitalize(Replace(FieldName(IterNo(), 'data'), '_', ' ')) as new_name
AutoGenerate
    (1)
While
    IterNo() < NoOfFields('data')+1;

Rename Fields Using field_rename_map;
```

Documentation

We tend to use this so that the fields in the data model are the same as the front end. This was something we started doing because previously in Qlik Sense®, even if you named a Master Item differently, the field name showed up in the breadcrumb trail. Now it is not as important.

Additionally, we do it at the end of the load script so the field names can stay consistent up to that point. This makes troubleshooting a lot easier on both sides.

Use Preceding Load to Enhance Readability

Preceding Load allows you to perform additional load steps before initializing a table. Each step uses the output of the preceding load statement as input.

Original:

```
Load
    Pick(Floor((Rand()*10)/2)+1, 'A', 'B', 'C', 'D', 'E')      as Customer,
    'Q' & (Mod(RowNo(), 4)+1)                                   as Quarter,
    Ceil((Rand() * 100) * (Rand()*10))                         as Sales
AutoGenerate
    (100);
```

New:

```
Load
    CustomerName,
    Quarter,
    TotalSales,
    If(Customer = Previous(Customer),
        TotalSales - Previous(TotalSales), 0) as QoQChange
;
Load
    Customer as CustomerName,
    Quarter,
    Sum(Sales) as TotalSales
Resident
    Original
Group By
    Customer,
    Quarter
Order by
    Customer,
    Quarter;
```

```
Drop Table Original;
```

In our team's opinion, the biggest benefit of using this is code readability. There may be times when putting all the transformations in a single Load statement is faster, however, in most cases it is easier to understand code that pieces out the transformations into multiple steps versus having many nested functions.

With that said, if performance is a high priority, it is worth testing various alternatives. All in one load, preceding load, exporting to QVDs and performing the transformation on a new ingestion, and more. There will almost always be multiple ways to solve a problem. A priority in any type of development is that it is readable, whether it is you 6 months in the future or a colleague who takes over support.

Load the Entire Contents of a File as a Value

Have you ever wanted to consume a whole file's content into Qlik Sense® as a single value? Perhaps, a series of README files or code files which can be shown on the front end? This is a little tricky since Qlik Sense was really built to consume tabular data. Therefore, when connecting to files it is trying to determine the columns and rows. Here is a way to get around that:

```
test:
Load
    "@1:n" as Source
FROM
    [lib://data/readme.md](fix, utf8, record is 100000 lines);
```

This will load the entire file contents as one field up to **100,000** lines. This can be changed of course.

A more complicated example: We want to load a series of HTML files, which we will use as a source for an extension to render.

```
html:
Load
    Num(Null()) as Counter
AutoGenerate
    (0);

for each file in FileList('lib://QlikShare/Dev/data/html-test-cases/*.html')

    Concatenate(html)
    Load
        'Test Case '&Subfield(FileBaseName(), '.', 1) as Id,
        Subfield(FileBaseName(), '.', 2) as Name,
        "@1:n" as Example,
        Ceil(Subfield(FileBaseName(), '.', 1)/3) as WowFactor,
        1 as Counter
    From
        [$(file)](fix, utf8, record is 100000 lines);

Next file;
```

Which results in:

Preview of data			
Counter	Id	Name	Example
1	Test Case 1	link	Go to Google
1	Test Case 10	js_btn	<!DOCTYPE html> <html> <body> <h1>Basic Javascript Test</h1> <button type="button" onclick="docur
1	Test Case 11	geo_location	<body> <p>Click the button to get your coordinates.</p> <button onclick="getLocation()">Try It</button>
1	Test Case 2	tbl	<table> <tr> <th>Month</th> <th>Savings</th> </tr> <tr> <td>January</td> <td>\$100</td> </tr> <tr>
1	Test Case 3	tbl_2	<table> <tr> <th>Month</th> <th>Savings</th> </tr> <tr> <td>January</td> <td>\$100</td> </tr> <tr> <td>
1	Test Case 4	inline_css	 <h2 style="font-family: Arial, Helvetica, sans-serif;text-align:center;color:indianred;">Inline CSS: Helve
1	Test Case 5	img	<img src="https://trey.bardesscloud.com/content/Default/Qlik_default_plant.png" style="width:150px; hei

Running Dynamically Created Code

This concept can be used in so many ways and can come in handy for complex situations. Essentially, by using the dollar sign expansion, you can execute a script you have programmatically built inside the script or pulled in from source files.

Notice `$(vAnswer)` in the example below. It is going to be expanded by the dollar sign expansion and be executed like any other script.

`Answer:`

`Load * Inline [`

`Answer`

`Yes`

`No`

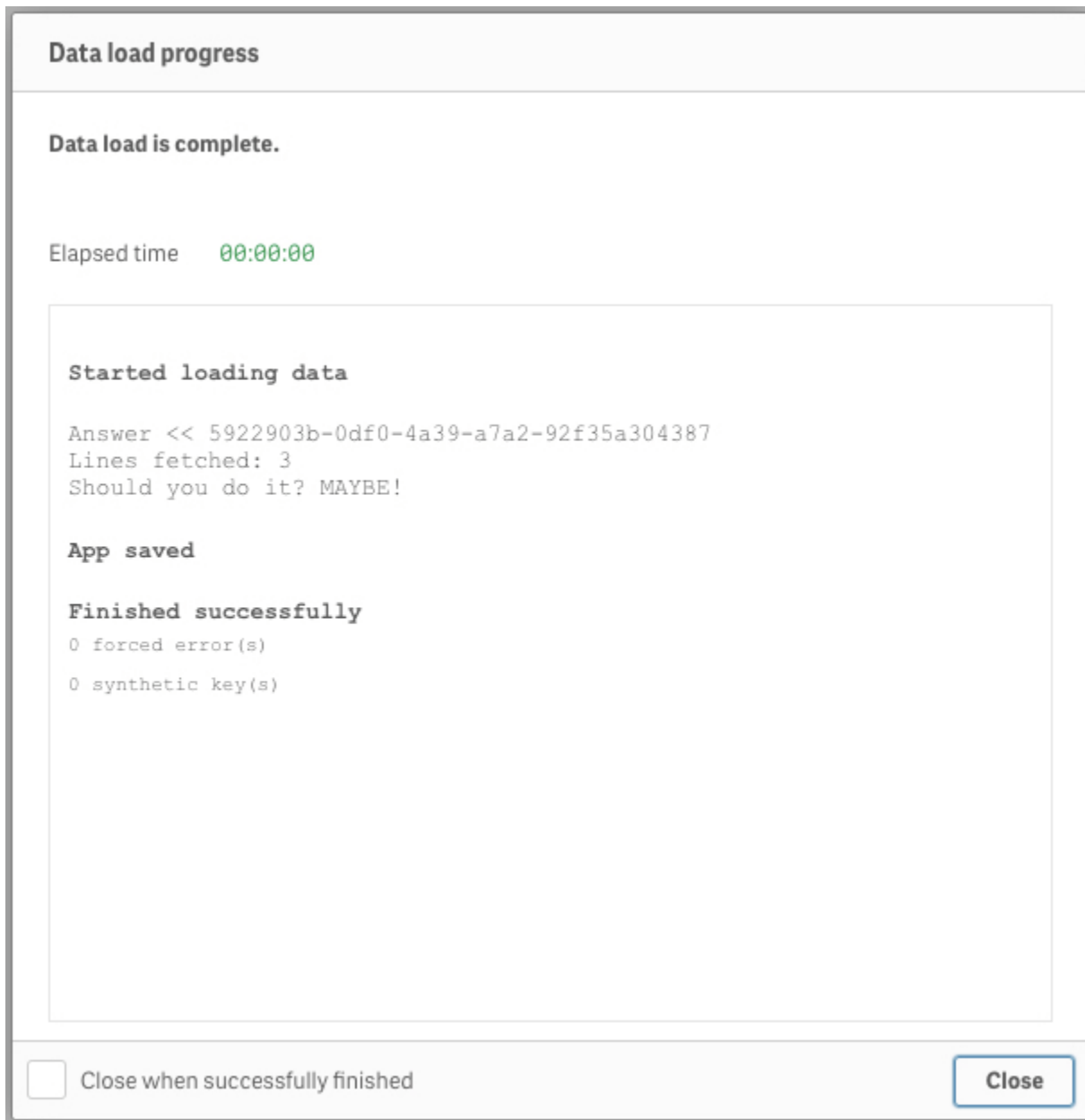
`Maybe`

`];`

`Let vAnswer = 'Trace Should you do it? ' & Upper(Peek('Answer',Floor(Rand()*3),'Answer')) &'! ';;`

`$(vAnswer)`

Which returns:



Now this is a fun and simple example, but not really practical. A more complex and practical example is loading script from external files and executing them on the fly.

Let's say that the DBA team needs you to run a series of audit queries each day. However, this list of SQL queries will always be changing and the solution should be flexible. Is this something that Qlik Sense® can handle?

Of course! All we need to do is combine the tip covered above with this concept.

```
Audit:
```

```
Load
```

```
    Text(Null()) as AuditName,
```

```
    Text(Null()) as Query
```

```
AutoGenerate
```

```
    (0);
```

```
for each file in FileList('lib://QlikShare/audit/queries/*.sql')
```

```
    Concatenate(Audit)
```

```
    Load
```

```
        Subfield(FileBaseName(), '.', 2) as AuditName,
```

```
        "@1:n" as Query
```

```
    From
```

```
        [$(file)](fix, utf8, record is 10000 lines);
```

```
Next file;
```

```
AuditRecords:
```

```
NoConcatenate Load
```

```
    Text(Null()) as AuditName
```

```
AutoGenerate
```

```
    (0);
```

```
For q = 0 to NoOfRows('Audit')-1
```

```
    Let vQuery = Peek('Query', $(q), 'Audit');
```

```
    Let vAuditName = Peek('AuditName', $(q), 'Audit');
```

```
    tmp:
```

```
    $(vQuery);
```

```

Concatenate(AuditRecords)
Load
    '$(vAuditName)' as AuditName,
    NoOfRows('tmp') as AuditRecords
AutoGenerate
    (1);

Store tmp into [lib://QlikShare/audit/results/$(vAuditName).csv](txt);
Drop Table tmp;

Next q;

Left Join(Audit)
Load
    AuditName,
    AuditRecords
Resident
    AuditRecords;

Drop Table AuditRecords;

```

This will load the script from all of the SQL files in a folder, then execute them, track the number of records returned by the query, and then store the individual results to a CSV file for the DBA team to analyze. Impressive!

Avoid the Rabbit Hole

Keep in mind that just because you can do something, it doesn't necessarily mean you should. You need to be aware of the actual value something will provide. If it takes you 40 hours to automate something that is done manually for 5 minutes every month, it will take you **40 YEARS** to see the return on your investment. That is what we consider to be over-engineering. Usually it isn't something that obvious, but just keep in mind there are rabbit holes at every step in the programming process.

This is something many programmers struggle with. For example, if an application is loading from a database using `SELECT *`, with no transformation, based on habit we would want to export it to a QVD first and then load it. However, in reality, all we are doing is creating additional overhead.

This is just a reminder that while we can do amazing things, sometimes the actual business value is increased by handling it in a much simpler and less exciting way.

Use Placeholder Tables to Establish Field Formats

You may have noticed that we used these in a few of the previous examples. The idea is that you create an empty table that you can concatenate to. The number one reason we do this is to be explicit. We know Qlik Sense® can and will auto-concatenate, but we want to be extremely clear in what we are doing, so that if there are any bugs, we don't have to troubleshoot code that has some behind the scenes *magic*.

Another reason to do this is to establish the format of a field. This initialization of a table with the formats will override later formatting attributes.

Placeholder :

Load

```
Text(Null())           as TextTest,
Num(Null(), '0000')    as NumTest,
Timestamp(Null(), 'YYYY-MM-DD hh:mm:ss') as TimestampTest
```

AutoGenerate

```
(0);
```

```
Concatenate(Placeholder)
Load
    1                as TextTest,
    Num(1, '#,##0')  as NumTest,
    Now()            as TimestampTest
AutoGenerate
    (1);
```

```
Concatenate(Placeholder)
Load
    'Two'                as TextTest,
    2                    as NumTest,
    Date(Floor(Now()), 'MM/DD/YYYY') as TimestampTest
AutoGenerate
    (1);
```

Which returns:

Preview of data		
TextTest	NumTest	TimestampTest
1	0001	2019-09-14 18:11:51
Two	0002	2019-09-14 00:00:00

Use ApplyMap for Fast VLookup Functionality

This is a post that might be common knowledge to you, but if not, it can be a crucial addition to your workflow.

[ApplyMap](#) is a function that is very similar to VLookup in Excel. Essentially you can use a mapping table that you can bulk find and replace; provide an input and receive and return an output that matches.

Example

```
String_map:
Mapping Load * Inline [
input, output
1, One
2, Two
3, Three
];

Data:
Load
  RowNo() as Num,
  ApplyMap('String_map', RowNo(), 'N/A') as String
AutoGenerate
  (4);
```

Which returns:

Data

Num	String
1	One
2	Two
3	Three
4	N/A

**Note the third parameter is what should be provided if there isn't a match. By default it is the input value.*

While it is a great feature, which can also be accomplished by the [LookUp](#) function, it is extremely fast. If you are working with large datasets, it is usually faster to utilize multiple ApplyMaps than to do a join.

Now we have no evidence to support this claim, but our hypothesis is that ApplyMap is so fast because it happens at the bit-stuffed pointer level. We would love to have a conversation with Qlik® R&D around this to better understand the inner workings of the function.

One important thing to remember, the mapping happens against the first field **no matter what**. Field names do not matter. So always make sure your input field is the first and your output field is the second. Due to the potential for forgetful moments and repetitive troubleshooting, we suggest that you always create the mapping tables with the field names 'input' and 'output.' And since the mapping tables are treated differently, it does not matter if you have several mapping tables with the same name fields.

Another bonus is that the mapping tables are dropped at the end of the script, which alleviates some cleanup.

Do While: How to Loop Until a Condition is Met

Most loops you will see are [For](#) loops. These are great if you can programmatically determine the number of loops you need to do. For example, the number of rows of a table. However, what if you need to loop as long as it takes to finish? You do not know how long it will take. What if you want to extract from an unreliable database that fails for unspecified reasons and you have to keep trying until you succeed? These types of examples are where Do While loops come in.

[Do..Loop](#), or as we like to call it, Do While, allows you to loop until a condition is met.

In this example, we want to loop for 5 seconds give or take.

```
// Determine numerical value of a second
Let vSecond = Num(Timestamp#('00:00:01','hh:mm:ss') - Time#('00:00:00','hh:mm:ss'));
Let vEnd = Num(Now()) + ($vSecond*3);
```

```
Set vIter = 0;
```

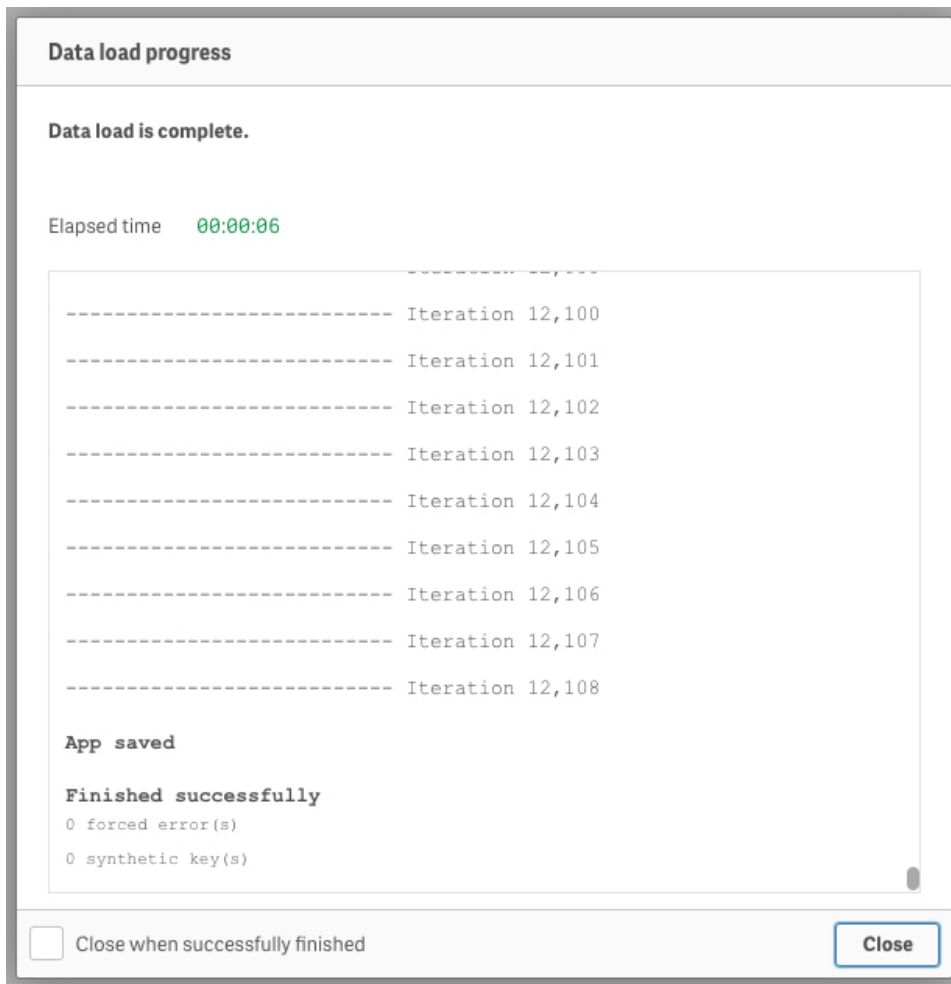
```
Do While Num(Now()) < $(vEnd)
```

```
Trace ----- Iteration $(vIter);
```

```
Let vIter = Num(vIter + 1, '#,##0');
```

Loop

Which returns:



12,108 iterations in 5 seconds, so 2,481 a second. That's pretty fast! Granted, we weren't doing any heavy lifting, but it is still impressive. While this example is not practical, it does prove the general point.

Cardinality and how it Impacts Application Size

One of the key things to understand when trying to tune a Qlik Sense® application for performance is understanding the inner-workings. If you haven't done so already, we strongly suggest you read Henric's blog [Symbol Tables and Bit-Stuffed Pointers](#) which goes into detail on how Qlik Sense stores data.

After grasping that, it starts to make sense why cardinality has such an impact on application size. Keep in mind, before trying to minimize data size, try and minimize the number of distinct values. It can play a huge role in cutting down size, without limiting functionality, which is always an easier sell to your users.

**P.S. Take some time and read through all of Henric's blogs. He is one of the best Qlik bloggers out there.*

Use AutoNumber to Optimize Your Data Model

To follow up on cardinality, a very useful function is [AutoNumber](#). AutoNumber will create an integer for each distinct value in your defined namespace, which is usually a field. The second parameter allows you to define the namespace, which means you can use it across multiple fields while keeping data association integrity.

The best use case for AutoNumber is for keys. This will create the smallest footprint for your key fields and maintain the relationship. This is because the amount of space needed for an integer is much smaller than a string, or dual value field like a timestamp.

One tip, apply AutoNumber at the end of your data model build. It is much easier to troubleshoot data model issues when you can see the actual data. After you know those linkages are good, then use the function.

Effectively Work with Qlik Key Fields

Here are a few things we strongly suggest when working with Qlik key fields.

- Make separate key fields*
- Prefix the field with '%'
- Set HidePrefix variable to '%'
- Use AutoNumber

*Because of how the association engine works, you should never use a key field in an expression. It can return some very unexpected results. To handle this, you should make a duplicate field to handle the association and then use the other field in your expressions.

A visual of what we mean:

```
Let HidePrefix = '%';
```

Sales:

Load

```
    RowNo()                                as Id,
    AutoNumber(Pick(Ceil(Rand()*4), 'A', 'B', 'C', 'D'), 'Product') as %ProductId,
    Ceil(Rand()*100)                        as Qty
```

AutoGenerate

```
(1000);
```

Product:

Load

```
    AutoNumber(Chr(RowNo()+64), 'Product') as %ProductId,
    Chr(RowNo()+64)                        as ProductId
```

AutoGenerate

```
(5);
```

Ignore Potential Future Use Cases and Remove Unused Fields

We can't tell you how many times a client has asked us to leave a field in the data model in case they want to use it later.

Just don't do it. If they want to use it later, you can bring it in later. You can easily set up the app so that it is a very simple task down the road.

- Comment it out
- Use a drop field statement which they can remove later (suggested)

If you are tuning an app that is already slow and having issues, you should not be thinking about the future state. You should be thinking about getting what you currently have working to an acceptable level. You won't be adding functionality to a slow app if no one is using it because it takes 3 minutes to load.

Using Arrays for Iteration

Arrays and/or Lists are very common in programming languages. In Qlik scripting there are two ways to go about creating an array of sorts.

The first way is storing a list in a variable:

```
Set vCount = 'One', 'Two', 'Three';

for each i in $(vCount)

    Trace ----- Variable Array $(i);

Next i
```

The second way is by using a field:

```
Cnt:
Load * Inline [
Cnt
One
Two
Three
];

For each x in FieldValueList('Cnt');

    Trace ----- Field Array: $(x);

Next x;
```

The main use for these will be iterating over their values. There are not any native list/array operations, however you can get around that utilizing other Qlik Sense® functions. For example: concatenate, where not exists, reinitializing the variable, etc.

Use SubField to Expand Records

If you are like us, you probably use [SubField](#) a lot. We used it for years before coming across one of its most powerful features . If you do not provide the position parameter, it will expand a record for each value.

We have worked with a few older databases that liked to store arrays within a single column. This is a good example of where this tip is at its most helpful.

Data:

Load

```
RowNo()                                as ID,
Replace(Replace(Qty, '(', '['), ')', ']') as Qty
```

Inline [

Qty

(10,12,15,20)

(25,25,25,25)

(1080,720,2,3)

](delimiter is \t);

Expanded:

Load

```
RowNo()                                as LineID,
ID,
Qty,
SubField(PurgeChar(Qty, '['), ',')     as Quantity
```

Resident

Data;

Drop Table Data;

Which results in:

ID	Q	LineID	Q	Qty	Q	Quantity	Q
	1		1	[10,12,15,20]		10	
	1		2	[10,12,15,20]		12	
	1		3	[10,12,15,20]		15	
	1		4	[10,12,15,20]		20	
	2		5	[25,25,25,25]		25	
	2		6	[25,25,25,25]		25	
	2		7	[25,25,25,25]		25	
	2		8	[25,25,25,25]		25	
	3		9	[1080,720,2,3]		1080	
	3		10	[1080,720,2,3]		720	
	3		11	[1080,720,2,3]		2	
	3		12	[1080,720,2,3]		3	

Wildcard Load for Bulk Loading

One good thing to know is that you can use an asterisk (*) within a file name. Qlik Sense® will load all the files that fit the criteria.

If you have a directory named QVD, which has all of your QVDs prepped and ready for your data model, you can do the following to load them all.

Load

*

From

```
[lib://QVD/*.qvd](qvd);
```

If you have extracts with a date or notation to them, you can load them in a similar fashion.

Fact:

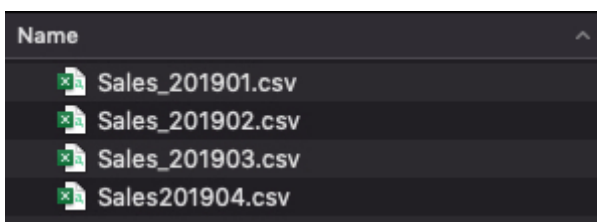
Load

*

From

```
[lib://Data/Sales_*.csv](txt, utf8, embedded labels, delimiter is ',');
```

Using these files as an example:



All the files would be loaded except for Sales201904.csv, because it doesn't match the Sales_*.csv criteria since it is missing the underscore.

One thing to note is that you are relying on the Qlik® Associative Engine's brain on what to do with the tables. You aren't being explicit with what you want. In the above case, tables with the same fields will be auto-concatenated, while others with a different structure will be created as different tables.

If you want to be explicit, which you know we do, you can use a loop.

Sales:

Load

```
Null() as Tmp
```

AutoGenerate

```
(0);
```

```

For each File in FileList('lib://Data/Sales_*.csv')

    Concatenate(Sales)
    Load
        *
    From
        [$(File)](txt, utf8, embedded labels, delimiter is ',',');

Next File

Drop Field Tmp;

```

Dual Behavior in Set Analysis

By default you need to reference the display (read: string) value of a dual field in set analysis versus the numerical value. We personally think this is a strange behavior of Qlik Sense®. The downside to this is that string operations are much slower than numerical ones. To overcome this you need to leverage advanced search within set analysis.

Example: Use set analysis to filter down to dual value with a numerical value of 1*

**Remember in the Dual Data Type Caveat section how two numbers can represent the same string?*

See the different outputs:

Id	Q	String	Q	Number	Q	Dual	Q	Only({<Dual={A}>} Dual)	Only({<Dual={1}>} Dual)	Only({<Dual={'1'}>} Dual)	Only({<Dual={'=1'}>} Dual)	Only({<Dual={'>=1<=1'}>} Dual)
Totals								-	-	-	-	A
1	A			1		A		A	-	-	A	A
2	B			2		B	-	-	-	-	B	-
3	C			3		C	-	-	-	-	C	-
4	A			4		A		A	-	-	A	-
5	Z			1		A		A	-	-	A	A

You'll notice that you need to use greater than or equal to and less than or equal to for it to work.

```
=Only({<Dual={'>=1<=1'}>} Dual)
```


Use Copy/Paste to Save Time and Avoid Typos

There's nothing worse than waiting for a long reload only to discover that you made a typo. One thing we have made a habit of is copy and pasting script items: table names, field names, variables, etc. This is a simple very simple tip, but it can save you a lot of time down the road.

If you don't use hot keys, we strongly suggest learning this one.

- Win: Ctrl + C (Copy) Ctrl + V (Paste)
- Mac: ⌘+C (Copy) ⌘ + V (Paste)

Pick Match, an If Statement Equivalent

One of the most resource intensive functions in a chart is the **If** statement. In many scenarios, what you can do is leverage the combination of **Pick** and **Match** to replicate the functionality while increasing performance.

If Statement

```
If([Inventory Type]='Individual', Sum(Qty), If([Inventory Type] = 'Bulk',  
Sum(BulkQty)))
```

Pick Match Equivalent

```
Pick(Match([Inventory Type], 'Individual', 'Bulk'),  
Sum(Qty),  
Sum(BulkQty))
```

If you need an Else condition, there is a simple trick you can do by adding a '+1' and making the first option the Else result:

```
Pick(Match([Inventory Type], 'Individual', 'Bulk')+1,  
'Non-Physical Item',  
Sum(Qty),  
Sum(BulkQty))
```

How to Associate Mixed Granularity Data

There will always be situations where the data you get isn't at the same granularity as the other data you have. The benefit of Qlik® is that you can leverage the Associative Engine model to account for these scenarios. Some instances will call for a link table, others a concatenated fact, but in the end you will be able to associate the data so you can visualize and compare them effectively.

Here is a simple example of sales data that is at the Product level, while the budget is at the Product Category level. There are many ways to handle this scenario given the limited amount of fields, however, we think it is worth showing in this way so you can see how to associate across a field.

Data

Sales:

```
Load
    RowNo() as Id,
    *
Inline [
ProductId, Amount
1001, 100
1002, 50
1001, 25
1003, 75
1004, 50
1002, 15
1002, 75
];
```

Product:

```
Load * Inline [
ProductId, Product, Category
1001, One, Alpha
1002, Two, Alpha
1003, Three, Beta
1004, Four, Beta
];
```

```
Concatenate(Sales)
Load
    'Budget-' & Category as ProductId,
    Budget
Inline [
Category, Budget
Alpha, 250
Beta, 175
];
```

```
Concatenate(Product)
Load
    'Budget-' & Category as ProductId,
    Category
Inline [
Category, Budget
Alpha, 250
Beta, 175
];
```

If you look closely, we are leveraging the ProductId field to tie the Budget in the Sales table to the Category field in the Product table. This will let us do the following:

	Id	Product	Sum(Amount)	Sum(Budget)
Totals			390	425
	1	One	100	0
	2	Two	50	0
	3	One	25	0
	4	Three	75	0
	5	Four	50	0
	6	Two	15	0
	7	Two	75	0
-	-	-	0	425

Category	Sum(Amount)	Sum(Budget)	% Diff
Totals	390	425	-8%
Alpha	265	250	6%
Beta	125	175	-29%

* We included the Budget column in the top chart to show you that you won't be able to correctly visualize it at the granularity beneath it.

When Should You Upgrade Qlik Sense®?

With Qlik Sense® issuing multiple releases a year, it can become overwhelming trying to stay up to date on the latest release. And even then, is it worthwhile to be on the latest version?

Here is what we recommend as a rule of thumb: when upgrading or installing for the first time, install the latest release with a patch out. We always suggest staying one release behind so that you aren't the production testers. With most software, there will be bugs. If you have mission critical items, it's better safe than sorry.

Now, when to upgrade? We suggest staying no more than 3 releases behind current. So if you are already one behind, that means you would upgrade every 2 releases: ~ 6 months. Note, that each release is only supported for 2 years. At the minimum, you should stay within the support window.

The caveat to these recommendations is that if there is a feature or bug fix you need. Those are good reasons to upgrade outside of these recommendations.

Create A Codebase

One of the reasons we started writing these blogs was to have a place to store our random Qlik info. But even before this, over the years we have collected a number of code snippets that we can refer to whenever a similar problem comes up. You don't need a blog or ebook, you just need a folder.

To be honest, there are a lot of repetitive problems within the Qlik® ecosphere. We think that is true for programming languages in general. However with Qlik scripting being a relatively niche area, it is even more apparent.

With that said, save your work! Create a folder and start adding .qvs files and give them some comments.

We recommend downloading [VS Code](#). There is a [Qlik](#) extension made by [Xavier Han](#), which will give you syntax highlighting among other things. Both great and free. Also, we highly recommend using the Chrome extension [Add-Sense](#). Credit to [Erik Wetterberg](#), it makes pulling load scripts from Qlik Sense apps much easier.

If you want to up your game and potentially learn a new skill, create a [Git](#) repo. Many offer at least one free private repo, We use [GitHub](#), but there are many options out there. This is something we think every developer should have, even if it's just the basics.

**Disclaimer: If you are working for a client, make sure it is okay for you to have a copy of it.*

Using Advanced Search to Filter in Set Analysis

[Set Analysis](#) is extremely powerful, however it can seem overwhelming at times. One thing we felt helped us further understand the mechanics is recognizing that what works within a filter search, will also work within a Set Analysis condition between double quotes.

For example, take this dataset:

	id	name	=Sum(exp)
Totals			198
	1	Joe	23
	1	Sara	54
	2	Emily	14
	2	Mike	3
	2	Tim	24
	3	Christina	80

Using advanced search functionality, you can filter down the data based on an expression:

...

✖

✔

Q id

Q =Sum(exp)>50|

×

1

3

Apply the same expression in Set Analysis and you will receive the same filtering capability

Expression: =Sum({<id={"=Sum(exp)>50"}>} exp)

	id	Q	name	Q	=Sum(exp)	=Sum({<id={"=Sum(exp)>50"}>} exp)
Totals					198	157
	1		Joe		23	23
	1		Sara		54	54
	2		Emily		14	0
	2		Mike		3	0
	2		Tim		24	0
	3		Christina		80	80

You can not only use expressions, but take advantage of the other search features, such as the ? wildcard. In this example, ???? would filter the name down to values with 4 characters.

...

✕

✓

🔍 name

🔍 ???? ✕

Mike

Sara

Expression: =Sum({<name={"????"}>} exp)

	id	name	=Sum(exp)	=Sum({<id={"=Sum(exp)>50"}>} exp)	=Sum({<name={"????"}>} exp)
Totals			198	157	57
	1	Joe	23	23	0
	1	Sara	54	54	54
	2	Emily	14	0	0
	2	Mike	3	0	3
	2	Tim	24	0	0
	3	Christina	80	80	0

You can think of Set Analysis as a where clause for an expression or a set of hard coded filters. There is a series of documentation, cheat sheets, and blogs out there that go into more detail of all the potential ways you can leverage the search capability.

Lastly, there are some crazy things you can do using Dollar-sign expansion as well. Although we won't go into it in this section, it's worth noting that Qlik Sense's feature, the [Dollar-sign expansion preview](#), has made it much easier to implement these types of expressions.

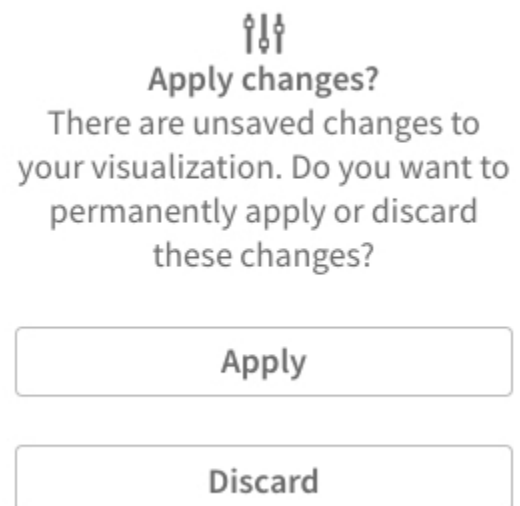


Changing Column Width

We've heard a few people mention that you can't change the column width within a [table](#), but it can be done. If you haven't stumbled on the solution yet, follow the steps below:

1. In read mode, use your mouse to change the column to your desired width by dragging the vertical border on the header.
2. Open Edit mode
3. Click on the table
4. You should see a prompt to "Apply Changes", click "Apply" to save your new layout.

We admit the UX is unintuitive and could be better, but once you know it, you won't forget it.



Making Date Formats Functional

Troubleshooting issues that are the result of the date format can take extraordinary amounts of time. After a while, we started to make a habit of always using dates in a numerical format when used for conditions, whether it be in Set Analysis or a Where clause.

Here are a few examples.

Load Script

Sales:

Load

```
    RowNo() as Id,  
    Num(Date) as DateNum,  
    *
```

Inline [

Date, Amt

10/8/2019, 100

10/9/2019, 125

10/10/2019, 75

11/1/2019, 150

11/10/2019, 200

];

Left Join(Sales)

Load

```
    DateNum,  
    1 as FutureFlag
```

Resident

Sales

Where

```
    Num(DateNum) > Num(Today());
```

Set Analysis



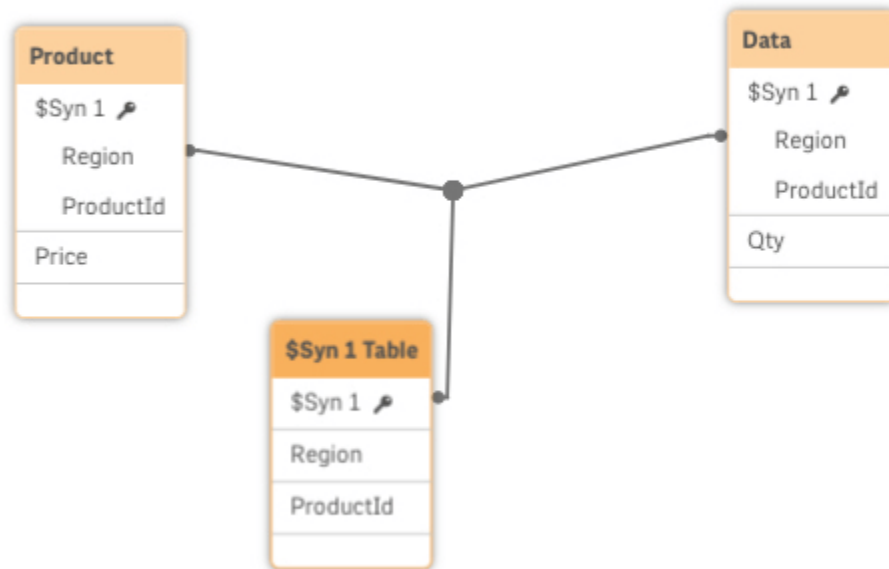
With this, you don't have to worry about setting the correct date format.

Create Your Own Concatenated Keys

What do you do when you need to associate multiple fields between two tables? The Qlik® Associative Engine creates a synthetic key and everyone says that is bad. First, it is good to understand what a [synthetic key](#) is. In reality, it is basically just a concatenated key generated by the Associative Engine. The reason it is taboo is because it is automatic. In theory, you do not know how it was made or what the results are going to be because it happened in a black box.

The proper procedure is to create your own concatenated key. As always, it is a best practice to **be explicit**. Even if the output is the same, it is better to program with thoughtfulness, rather than laziness.

Synthetic Key



Concatenated Key

```
Data:
Load
    Region & '|' & ProductId as %Region_Product,
    Region,
    Qty
Inline [
Region, ProductId, Qty
North, A, 12
East, B, 7
West, C, 19
North, B, 5
East, C, 8
West, D, 10
];
```

```

Product:
Load
    Region & '|' & ProductId as %Region_Product,
    ProductId,
    Price
Inline [
Region, ProductId, Price
North, A, 5
East, A, 5
West, A, 6
North, B, 13
East, B, 8
West, B, 9
North, C, 1
East, C, 1
West, C, 1
North, D, 21
East, D, 12
West, D, 7
];

```

**We almost always use a pipe | as the separator.*

By doing the above, it will remove the synthetic key.



In this case, the data results are the same. However, in complex data models synthetic keys can produce unexpected results. Also, in some scenarios you may not even want the fields associating. In that case the answer would not be a concatenated key at all. You would either want to remove the field or rename it, leaving the one correct field to define the association.

Looping Over Records Using While

The **While** clause is an extremely powerful feature that you probably won't use frequently, but when you do it's a godsend. Note that this is a part of the Do..Loop scripting convention. There is not a specific help page on the keyword.

Using While allows you to loop on a record within a load statement. It will basically keep loading that record until its condition is met. We like to think of them as mini-loops.

So take this super simple example. We have an array which is separated by pipes. We want to pull in the first three values of the array, but not any past that.

Data:

```
Load * Inline [  
Array  
12|13|AAA|1  
12|7|BBB|2  
];
```

newData:

```
Load  
    SubField(Array,'|', IterNo()) as ArrayValue,  
    IterNo() as ArrayIteration  
Resident  
    Data  
While  
    IterNo() <= 3;  
  
Drop Table Data;
```

IterNo is a function that keeps track of the iteration you are on. You can leverage this in your While clause as well as your field expressions.

This script example uses IterNo as the **SubField** parameter to pull out the first three values separated by the delimiter, producing our desired results.

NewData
ArrayValue
ArrayIteration

▼ Preview

NewData		Preview of data	
Rows	6	ArrayValue	ArrayIteration
Fields	2	12	1
Keys	0	13	2
Tags	\$numeric \$integer	AAA	3
		12	1
		7	2
		BBB	3

One place you may have seen this used is in Master Calendar scripts floating around on the [Community](#). We've seen it used to get a distinct list of dates quickly and/or to generate a new list of dates.

Example:

```
Let vDateField = 'Date';
```

```
AllDates:
```

```
Load
```

```
    FieldValue('$(vDateFieldName)', IterNo()) AS EveryDate
```

```
AutoGenerate
```

```
    (1)
```

```
While
```

```
    Not IsNull(FieldValue('$(vDateField)', IterNo()));
```

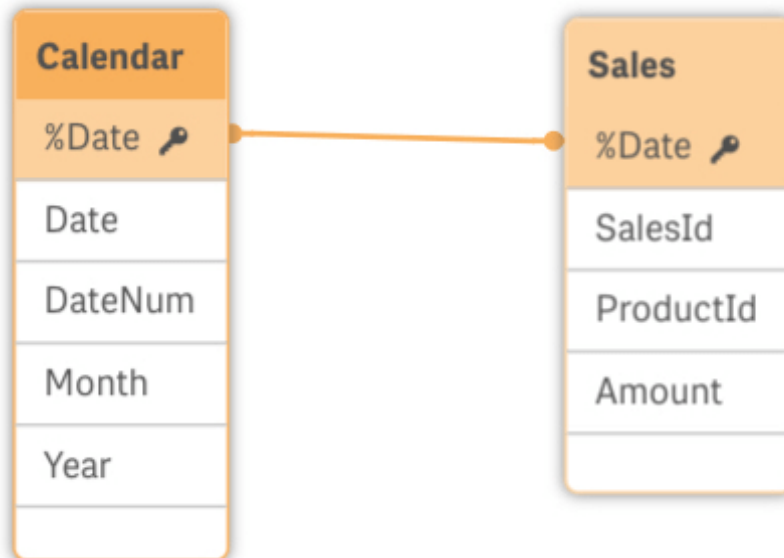
Troubleshooting Associations: Subset Ratio

One quick tip about data modeling. If you find things in your model are not associating right, take a look at your key fields. If you add up the [subset ratio](#) percentages of a key in each table that it is in and it equals 100%, this means there are no matching values. So even though the data model viewer shows them as linked because the fields exist in each table, the values within them are not linked. This could be due to formatting or misunderstood key fields.

If you take a look at the example below, %Date in the Calendar table has 46.5%. While %Date in the Sales table has 53.4%. This equates to ~100%. So each table has values for %Date, but none of them exist in the other table.

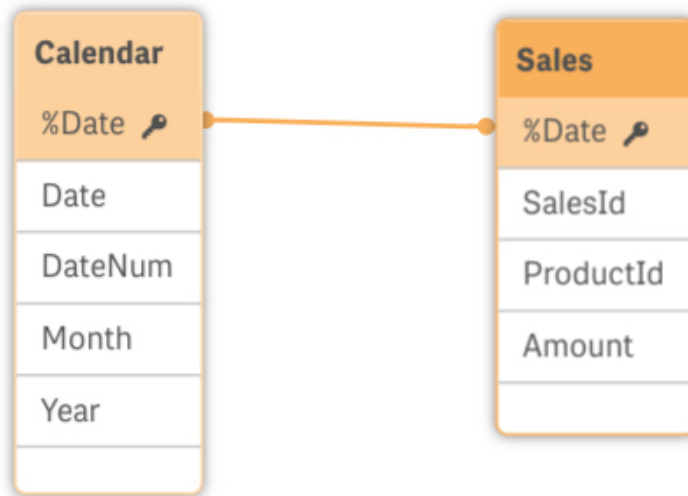
If you take a closer look, you'll notice that the date formats are different. This is a simple mistake that can throw off your model entirely.

Calendar



%Date		Preview of data				
Density	100%	%Date	Date	DateNum	Month	Year
Subset ratio	46.5%	11/3/2019	11/3/2019	43772	Nov	2019
Has duplicates	false	11/2/2019	11/2/2019	43771	Nov	2019
Total distinct values	58	11/1/2019	11/1/2019	43770	Nov	2019
Present distinct values	27	10/31/2019	10/31/2019	43769	Oct	2019
Non-null values	27	10/30/2019	10/30/2019	43768	Oct	2019
Tags	\$key	10/29/2019	10/29/2019	43767	Oct	2019
		10/28/2019	10/28/2019	43766	Oct	2019

Sales



%Date		Preview of data			
		%Date	SalesId	ProductId	Amount
Density	100%	11/03/2019	1	1	35
Subset ratio	53.4%	11/02/2019	2	2	3
Has duplicates	false	11/01/2019	3	3	70
Total distinct values	58	10/31/2019	4	4	291
Present distinct values	31	10/30/2019	5	0	79
Non-null values	31	10/29/2019	6	1	492
Tags	\$key	10/28/2019	7	2	409

Hidden Default Apps

Most of us are familiar with the License Monitor and Operations Monitor dashboards. These two logging apps are automatically installed and placed in the 'Monitoring apps' stream.

Monitoring apps



License Monitor

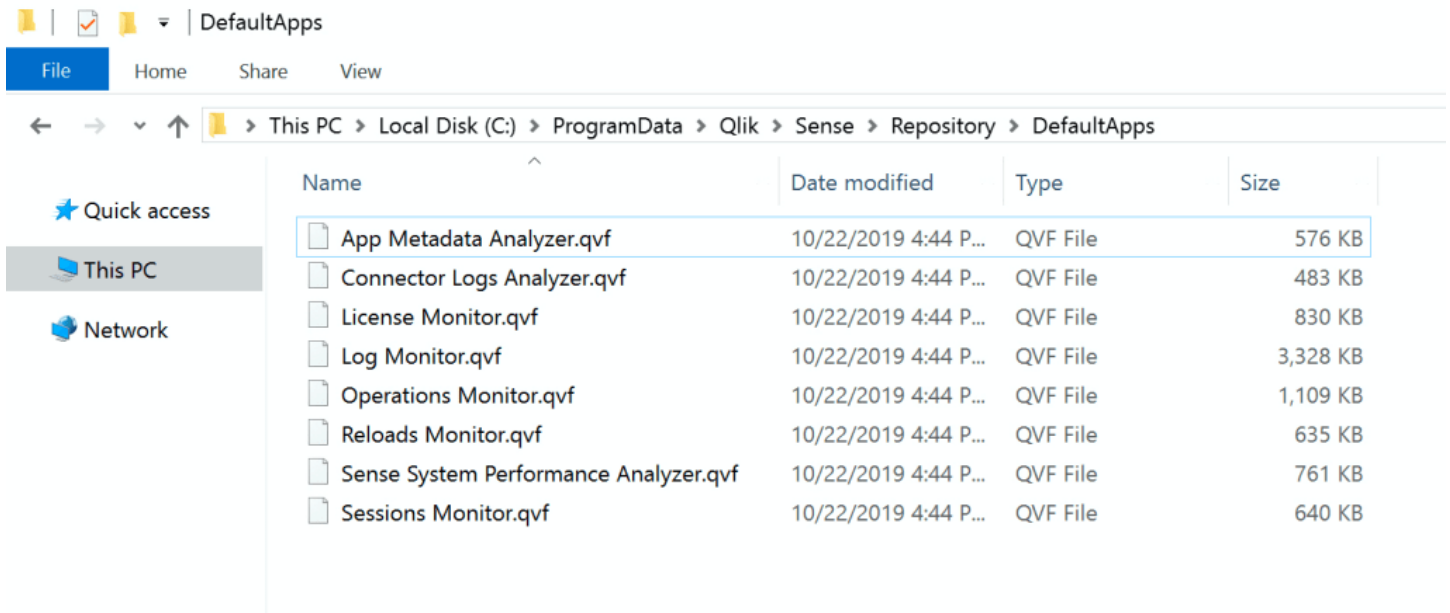


Operations Monitor

Did you know that these are only a couple of dashboards that come with the Qlik Sense Enterprise installation?

If you take a look at the following folder location, you will find that there are several additional applications which can be imported through the QMC. These are especially useful when tuning applications and your environment.

Folder: C:\ProgramData\Qlik\Sense\Repository\DefaultApps



We suggest taking some time to import these apps and reviewing their results. They are extremely handy. Check the [documentation](#) for more info on how to configure and use them.

Using FieldValueList for Loops

This is a great option for loops. If you need to loop through a list of field values you can use the [FieldValueList](#) mask. This could replace using the combination of [NoOfRows](#) and [Peek](#), assuming you only need to loop through the distinct list of values.

```
Numbers:
Load * Inline [
Num
Zero
One
Two
Two
Three
];
```

```
Trace ----- FieldValueList;;
```

```

Let vIter = 0;

For each n in FieldValueList('Num')

    Trace ----- Iteration: $(vIter) | Number: $(n);

    Let vIter = $(vIter) + 1;

Next n;

vIter=;n=;

Trace ----- NoOfRows + Peek;;

For i = 0 to NoOfRows('Numbers')-1

    Let vNum = Peek('Num',$(i),'Numbers');

    Trace ----- Iteration: $(i) | Number: $(vNum);

Next i;

vNum=;i=;

```

Result

Data load progress

Data load is complete.

Elapsed time 00:00:00

```
Numbers << 7dad076a-0428-4181-8243-e742021198d6
Lines fetched: 5
----- FieldValueList:
----- Iteration: 0 | Number: Zero
----- Iteration: 1 | Number: One
----- Iteration: 2 | Number: Two
----- Iteration: 3 | Number: Three
----- NoOfRows + Peek:
----- Iteration: 0 | Number: Zero
----- Iteration: 1 | Number: One
----- Iteration: 2 | Number: Two
----- Iteration: 3 | Number: Two
----- Iteration: 4 | Number: Three
```

App saved

☐ Close when successfully finished

Close

Notice the difference in how the two options handle the value Two. FieldValueList (Blue) uses the symbol table, so it loops over the unique values, whereas NoOfRows + Peek (Red) loops over the rows of the internal table.

Did You Try Turning It Off and On Again?

After working with Qlik Sense® for a long time, you'll soon find out that restarting the services can solve many unexpected behaviors and is a good starting point for environment bugs.

Learning the proper services restart order can take time, however. See below for the full list in the correct order.

Single Node Environment

1. Stop Qlik Sense Repository Service (Which stops the following)
 1. Qlik Sense Scheduler Service
 2. Qlik Sense Proxy Service
 3. Qlik Sense Printing Service
 4. Qlik Sense Engine Service
2. Stop Qlik Sense Service Dispatcher Service
3. **Optional: Restart Qlik Logging Service*
4. Restart Qlik Sense Repository Database
5. Start Qlik Sense Service Dispatcher
6. Start Qlik Sense Repository Service
7. Start remaining Qlik Sense services in any order
 1. Qlik Sense Scheduler Service
 2. Qlik Sense Proxy Service
 3. Qlik Sense Printing Service
 4. Qlik Sense Engine Service

Multi-Node Environment

If you have a multi-node environment, you should first stop the services on all RIM nodes. Perform the above on the Central node. Once complete, start the services on each of the RIM nodes, starting at step 5 from above.

**If your system has crashed and the Engine Service does not want to stop, you can end the process using the Task Manager.*

FileList Mask Order Test

There is a feature available when doing a for each loop called [FileList](#). It is a way to get a list of files in a directory that matches a wildcard path.

Since this is a parameter of the for each statement and not a function, there is not a lot of information on it. You might have noticed that we like doing test cases. We built a quick test to have as a reference for the future. One of the things we wanted to determine was what determines the returned list order. So we did a quick test.

Our guess was that there were three options:

1. File Name
2. File Time
3. File Size

Result: File Name ascending by alphabetical order.

Test 1

The test is pretty simple. We are going to create three files then read them back in using the FileList mask and capture the order.

The first test is determined as such:

- File 1: z_Table
 - Highest Alpha
 - Lowest File Time
 - Middle Size
- File 2: m_Table
 - Middle Alpha
 - Middle File Time
 - Highest Size
- File 3: a_Table
 - Lowest Alpha
 - Highest File Time
 - Lowest Size

```

z_Table:
Load
    1 as FieldA
AutoGenerate
    (1000);

m_Table:
Load
    1 as FieldB
AutoGenerate
    (100000);

a_Table:
Load
    1 as FieldC
AutoGenerate
    (10);

result:
Load
    Num(Null())      as Order,
    Null()           as File
AutoGenerate
    (0);

For each t in 'z_Table', 'm_Table', 'a_Table'

    Trace ----- $(t);
    Store $(t) into [lib://QVD/test/for-each-order/$(t).qvd](qvd);

Next t;

For each file in FileList('lib://QVD/test/for-each-order/*.qvd')

    Concatenate(result)
    Load
        RowNo() as Order,
        '$(file)' as File
    AutoGenerate
        (1);

Next file;

```

Test 2

In this test we reversed the table store order, which reversed the File Time order.

- File 1: z_Table
 - Highest Alpha
 - **Highest File Time**
 - Middle Size
- File 2: m_Table
 - Middle Alpha
 - Middle File Time
 - Highest Size
- File 3: a_Table
 - Lowest Alpha
 - **Lowest File Time**
 - Lowest Size

```
z_Table:
Load
  1 as FieldA
AutoGenerate
  (1000);

m_Table:
Load
  1 as FieldB
AutoGenerate
  (100000);

a_Table:
Load
  1 as FieldC
AutoGenerate
  (10);

result:
Load
  Num(Null())      as Order,
  Null()           as File
AutoGenerate
  (0);
```

```

For each t in 'a_Table', 'm_Table', 'z_Table'

    Trace ----- $(t);
    Store $(t) into [lib://QVD/test/for-each-order/$(t).qvd](qvd);

Next t;

For each file in FileList('lib://QVD/test/for-each-order/*.qvd')

    Concatenate(result)
    Load
        RowNo()      as Order,
        '$(file)'    as File
    AutoGenerate
        (1);

Next file;

```

Test 3

In this test we reversed the File Size for z_Table and a_Table.

- File 1: z_Table
 - Highest Alpha
 - Highest File Time
 - **Lowest Size**
- File 2: m_Table
 - Middle Alpha
 - Middle File Time
 - Highest Size
- File 3: a_Table
 - Lowest Alpha
 - Lowest File Time
 - **Middle Size**

```

z_Table:
Load
    1 as FieldA
AutoGenerate
    (10);

m_Table:
Load
    1 as FieldB
AutoGenerate
    (100000);

a_Table:
Load
    1 as FieldC
AutoGenerate
    (1000);

result:
Load
    Num(Null())      as Order,
    Null()           as File
AutoGenerate
    (0);

For each t in 'a_Table', 'm_Table', 'z_Table'

    Trace ----- $(t);
    Store $(t) into [lib://QVD/test/for-each-order/$(t).qvd](qvd);

Next t;

For each file in FileList('lib://QVD/test/for-each-order/*.qvd')

    Concatenate(result)
    Load
        RowNo()      as Order,
        '$(file)'    as File
    AutoGenerate
        (1);

Next file;

```

Result

Every test returned the same thing: a_Table was the first within the list. This means that the FileList mask returns an alphabetical file list.

▼ Preview

result		result	
Rows	3	Order	File
Fields	2	1	lib://QVD/test/for-each-order/a_Table.qvd
Tags	\$numeric \$integer \$ascii \$text	2	lib://QVD/test/for-each-order/m_Table.qvd
		3	lib://QVD/test/for-each-order/z_Table.qvd

How to Create a Quick Month Map

Here is a super quick tip. If you ever need to convert month names or prefixes to numbers or dates you can leverage the default variables and Subfield to quickly create a mapping table to later use with an ApplyMap.

```
SET ThousandSep=',';
SET DecimalSep='.';
SET MoneyThousandSep=',';
SET MoneyDecimalSep='.';
SET MoneyFormat='$#,##0.00;-$#,##0.00';
SET TimeFormat='h:mm:ss TT';
SET DateFormat='M/D/YYYY';
SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';
SET FirstWeekDay=6;
SET BrokenWeeks=1;
SET ReferenceDay=0;
SET FirstMonthOfYear=1;
SET CollationLocale='en-US';
SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';
SET LongMonthNames='January;February;March;April;May;June;July;August;September;October;November;December';
SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
SET LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';
SET NumericalAbbreviation='3:k;6:M;9:G;12:T;15:P;18:E;21:Z;24:Y;-3:m;-6:μ;-9:n;-12:p;-15:f;-18:a;-21:z;-24:y';
```

```

month_map:
Mapping Load
    SubField('${MonthNames}', ';') as month,
    RowNo() as month_num
AutoGenerate
    (1);

```

```

example:
Load
    MakeDate(year, ApplyMap('month_map', month, Null())) as date,
    ApplyMap('month_map', month, Null()) as month_number,
    *
Inline [
month, year, qty
Jan, 2020, 10
Feb, 2020, 15
Mar, 2020, 5
Apr, 2020, 20
];

```

and this returns:

year	Q	month	Q	month_number	Q	date	Q	qty	Q
2020		Jan		1		1/1/2020		10	
2020		Feb		2		2/1/2020		15	
2020		Mar		3		3/1/2020		5	
2020		Apr		4		4/1/2020		20	

Get QVD Metadata from XML Headers

Each QVD contains an XML header which contains metadata about that QVD. It is very common to leverage this to make a QVD Catalog dashboard to provide some overarching insight into all of the QVDs in your environment.

To see, just select a QVD in a folder connection and change the File format from QVD to XML.

Select data from Dev/QVD/sample_data.qvd

Tables → File format
XML

Filter tables

Fields
<input checked="" type="checkbox"/> QvdTableHeader/Fie... 2
<input checked="" type="checkbox"/> QvdTableHeader/Fie... 15
<input checked="" type="checkbox"/> QvdTableHeader/Lin... 2
<input checked="" type="checkbox"/> QvdTableHeader 11

<input checked="" type="checkbox"/> QvBuild...	<input checked="" type="checkbox"/> CreatorDoc	<input checked="" type="checkbox"/> CreateUtcTi...	<input checked="" type="checkbox"/> SourceFile...	<input checked="" type="checkbox"/> TableNa...
50622	74ccaa38-231b-4cac-9d12-3b09c1	2020-01-20 19:13:36	-1	sample_data

One thing to note is that XML is semi-structured. Fields will appear and disappear based on the data available. For example, a QVD header will contain a Tag table if any of the fields are tagged. If none are tagged, it will not exist for that QVD. Make sure you handle these different edge cases in your load script.

Where do Reload Logs Reside?

If you do not know, reload logs reside in the following folders:

- C:\ProgramData\Qlik\Sense\Log\Script
 - *On the Qlik Engine that the application reloaded on.*
- ~\QlikShare\ArchivedLogs\%Server_Name%\Script

Log files have the following naming convention:

```
%app_guid%.%reload_time%.%engine_session_id%
```

Example

75070c2c-375d-421a-a1db-accfe900b934.20200610T142244.030-0400.1B25998C34DCEB4ABD95.log

- app_guid: 75070c2c-375d-421a-a1db-accfe900b934
- reload_time (YYYY-MM-DDTHH.mm.ss-Z): 20200610T142244.030-0400
- engine_session_id: 1B25998C34DCEB4ABD95

Do you ever wonder why some reload logs stay in the first, while others get moved to the archive folder?

The answer is tasks. If an application gets reloaded by a task, that reload log will eventually get archived.

If you reload the application through the data load editor it never gets archived.

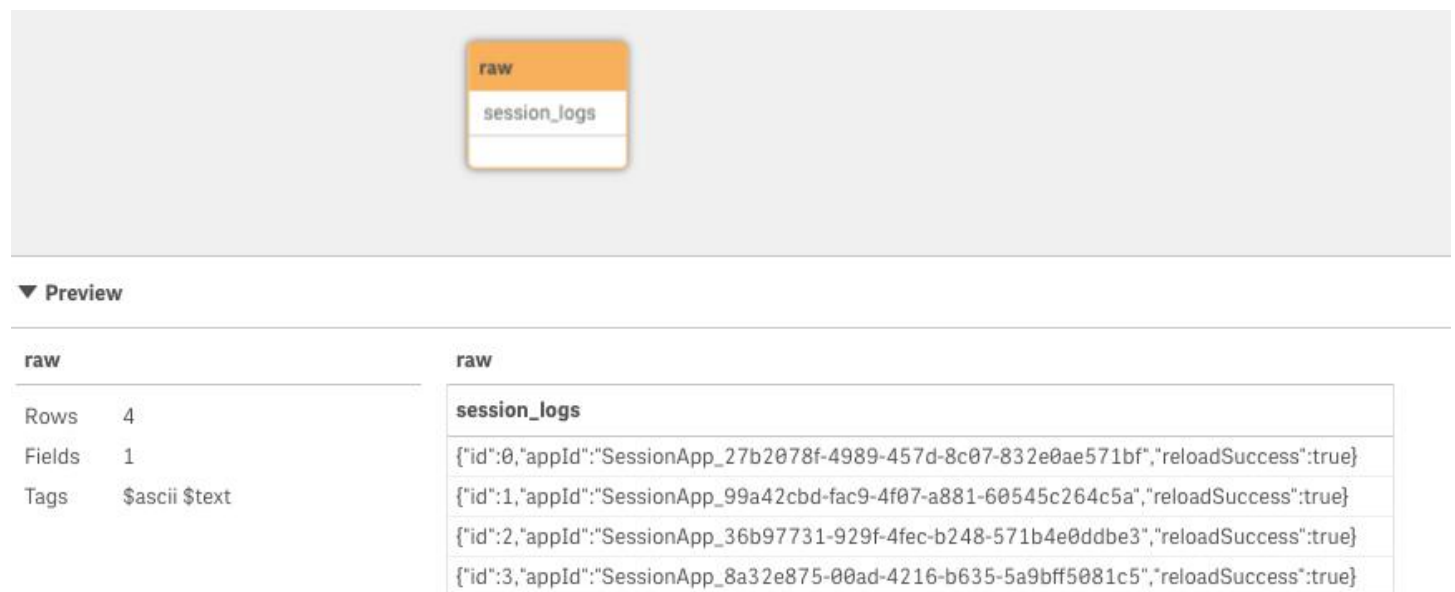
Natively Loading JSON

We want to begin by mentioning that this only works for flat json, meaning non-nested json hierarchies.

It is possible to leverage this for nested json but takes more work and knowledge of the schema.

Since our intention is to keep these things brief, we will save that for another time.

Getting into it, imagine you have a field with json values and we want to convert the json to a table.



The screenshot shows a data tool interface. At the top, there is a field named 'raw' with a sub-field 'session_logs'. Below this, there is a 'Preview' section. The preview shows a table with 4 rows and 1 field. The field is named 'session_logs' and contains JSON data. The data is as follows:

raw
session_logs
{ "id": 0, "appId": "SessionApp_27b2078f-4989-457d-8c07-832e0ae571bf", "reloadSuccess": true }
{ "id": 1, "appId": "SessionApp_99a42cbd-fac9-4f07-a881-60545c264c5a", "reloadSuccess": true }
{ "id": 2, "appId": "SessionApp_36b97731-929f-4fec-b248-571b4e0ddbe3", "reloadSuccess": true }
{ "id": 3, "appId": "SessionApp_8a32e875-00ad-4216-b635-5a9bff5081c5", "reloadSuccess": true }

What we can do is leverage [From_Field](#) and put json for our table format.

Script

```
raw:
LOAD
    "@1:n" as session_logs
FROM
    [lib://QlikShare/Dev/json/session-reload.log](fix, codepage is 28591, embedded labels);

json:
Load
    *
From_Field
    (raw, session_logs)(json, utf8, no labels);
```


Result

json
id
appId
reloadSuccess

▼ Preview

json		json		
Rows	4	id	appId	reloadSuccess
Fields	3	0	SessionApp_27b2078f-4989-457d-8c07-832e0ae571bf	-1
Tags	\$numeric \$integer \$ascii \$text	1	SessionApp_99a42cbd-fac9-4f07-a881-60545c264c5a	-1
		2	SessionApp_36b97731-929f-4fec-b248-571b4e0ddbe3	-1
		3	SessionApp_8a32e875-00ad-4216-b635-5a9bff5081c5	-1

We can also directly load a .json file as well. Although, for it to work it can only be a single flat json structure with no labels or headers.

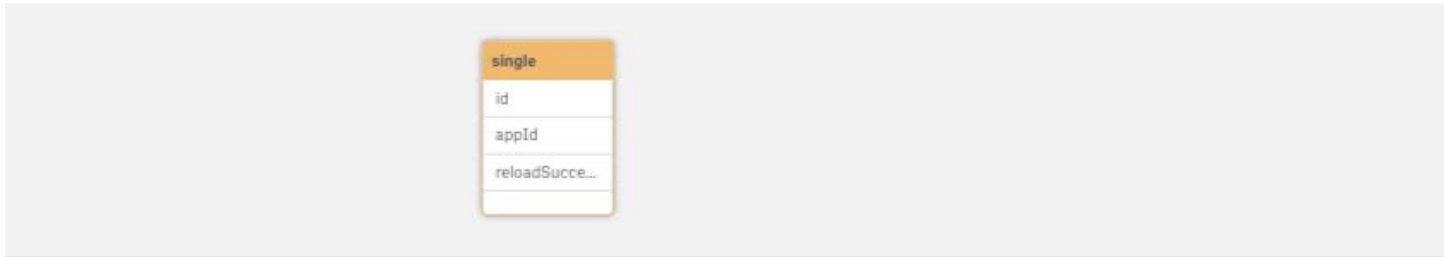
Example

```
QlikShare > Dev > json > {...} session-single.json > ...  
1 {  
2   "id": 0,  
3   "appId": "SessionApp_27b2078f-4989-457d-8c07-832e0ae571bf",  
4   "reloadSuccess": true  
5 }  
6
```

Script

```
single:
Load
    *
FROM
    [lib://QlikShare/Dev/json/session-single.json](json, utf8, no labels);
```

Result



▼ Preview

single

Rows	1
Fields	3
Tags	\$numeric \$integer \$ascii \$text

single

id	appId	reloadSuccess
0	SessionApp_27b2078f-4989-457d-8c07-832e0ae571bf	-1

We find this tip particularly interesting because it isn't mentioned in the Qlik Sense® documentation. You would expect json to be one of the options [here](#), but it's not.

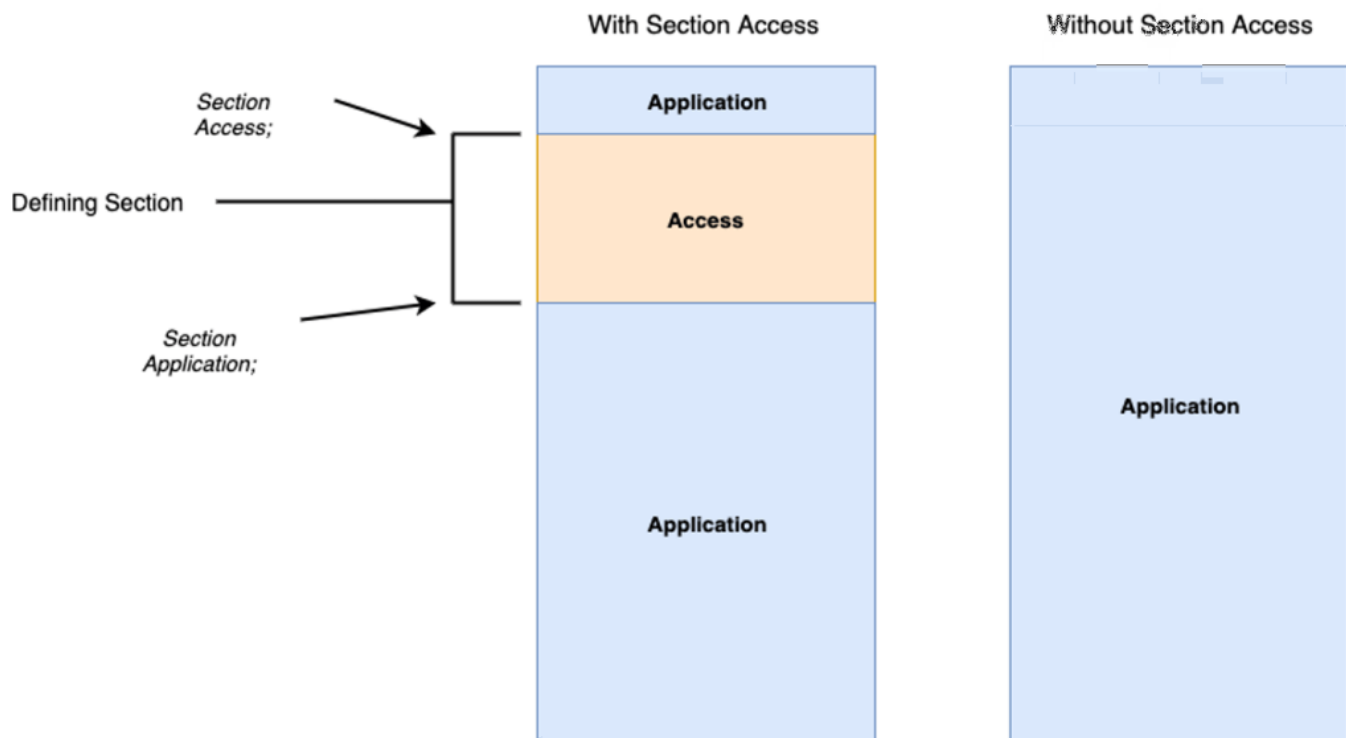
Section Access Tips and Tricks

This post isn't necessarily made to teach you what [Section Access](#) is, but rather to help you better understand it and provide you with tools to help troubleshoot it.

Keywords

Without tips like these, it can take 6-7 years of using Section Access to fully understand what the keywords `Section Access` and `Section Application` do. Essentially, you can think of a Qlik Sense® (or QlikView®) load script to be defined in two parts. First your application part, which defines everything on your Qlik Sense® application. Second is your security part, which defines the security portion, meaning it isn't defining your application but the security on it.

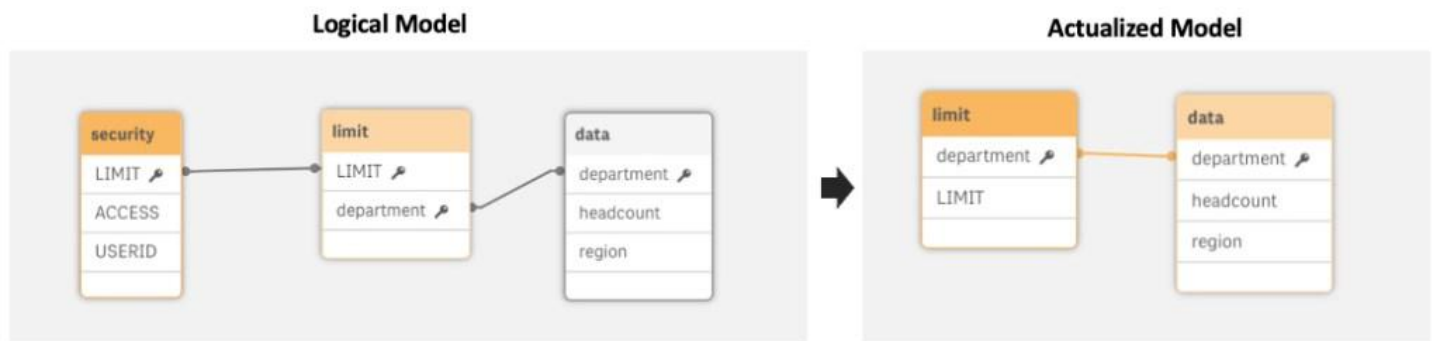
`Section Access` basically signifies to the interpreter that we are now defining our access portion. **Who** can access **What**? `Section Application` then tells us that we are resuming defining our application.



This helped us better understand what each of the keywords do and when to use them.

Security Table

The security table is there to define the security, obviously. The easiest way to think about it is that the table defines a pre-selection for the user. The data associated with that selection is what they have access to.



Fields

- **ACCESS**
 - ADMIN – Does not apply Section Access to user
 - USER – Applies Section Access to user
- **USERID**
 - DOMAIN\USER of user access Qlik dashboard

Tip: Use “=OSUser()” in a KPI chart or the Users page in QMC to see.

- **GROUP**
 - User directory Group attribute to use to define rule.
- **OMIT**
 - Field to omit from user’s model.
- **%FIELD%**
 - Name of field to link to a data table to apply row level security, or key to omit table, which contains OMIT field.

Testing

There’s nothing worse than releasing an application and immediately receiving multiple emails saying that users can’t access the dashboard, or worse, that their colleagues are seeing data they aren’t allowed to see.

Since Section Access acts as a selection it is very simple to test. What we suggest you do is comment out the keywords, create a simple sheet and test the user’s access. You can filter on USERID to limit the data to what they will see when Section Access is applied.

Example

User – Test

Security Test

Q USERID

QSDEVTEST

INTERNAL\sa_scheduler

QSDEV\BARDESSGROUP

Data

department	department	Sum(headcount)	Sum(target)
Totals		29346	31528
Hip Hop	Hip Hop	29346	31528

User – BardessGroup

Security Test

Q USERID

QSDEV\BARDESSGROUP

INTERNAL\sa_scheduler

QSDEVTEST

Data

department	department	Sum(headcount)	Sum(target)
Totals		1752304	2119863
Country	Country	1250761	1424817
Hip Hop	Hip Hop	29346	31528
Lo-Fi	Lo-Fi	72118	78045
Pop	Pop	143662	222763
Rock	Rock	256417	362710

Limit Table

Over the years, we have learned to always create a bridge table between the security table and the data table. This is because we want the security table to be one row per user. The limit table will then define the association and therefore security. Similarly, if you are using OMIT, we will create an omit table which contains the fields to omit and the group which is tied to those. This is to keep the relationship one to many and the security table to one row per user.

Script

Section Access;

security:

```
Load * inline [  
ACCESS, USERID, DEPT, OMITGROUP  
ADMIN, INTERNAL\sa_scheduler  
USER, QSDEV\BARDESSGROUP, ADMIN  
USER, QSDEV\TEST, HIP HOP, A  
];
```

column_omit:

```
Load * Inline [  
OMITGROUP, OMIT  
A, target  
A, pii  
];
```

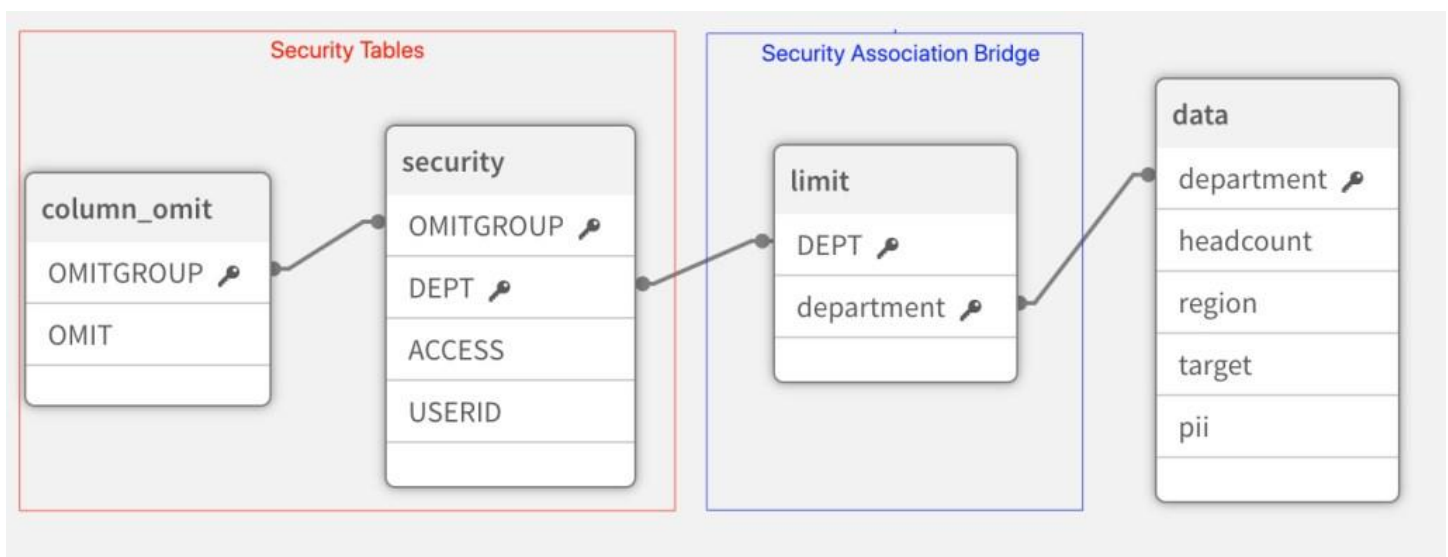
Section Application;

limit:

```
Load distinct  
    'ADMIN' as DEPT,  
    department  
Resident  
    data  
Where  
    department <> 'Army';
```

```
Concatenate(limit)
Load distinct
    Upper(department) as DEPT,
    department
Resident
    data;
```

Model



Troubleshooting

UPPER CASE!

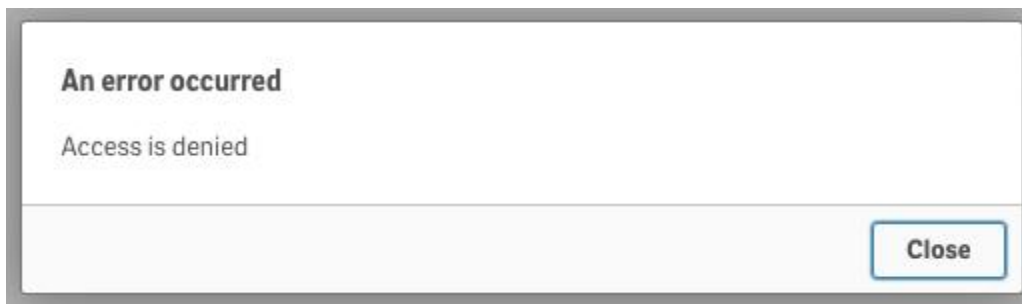
We wonder how much time collectively the Qlik Community has spent troubleshooting Section Access and the culprit was that the data wasn't upper case. Make your field names and value **UPPER CASE**. This will solve many headaches with Section Access if you remember this.

Tasks Fail

If your reload succeeds when run by you in the hub but fails as a task, this is because the internal service account doesn't have access. In Qlik Sense this is INTERNAL\SA_SCHEDULER. Make sure to add it to your Section Access security table as an admin.

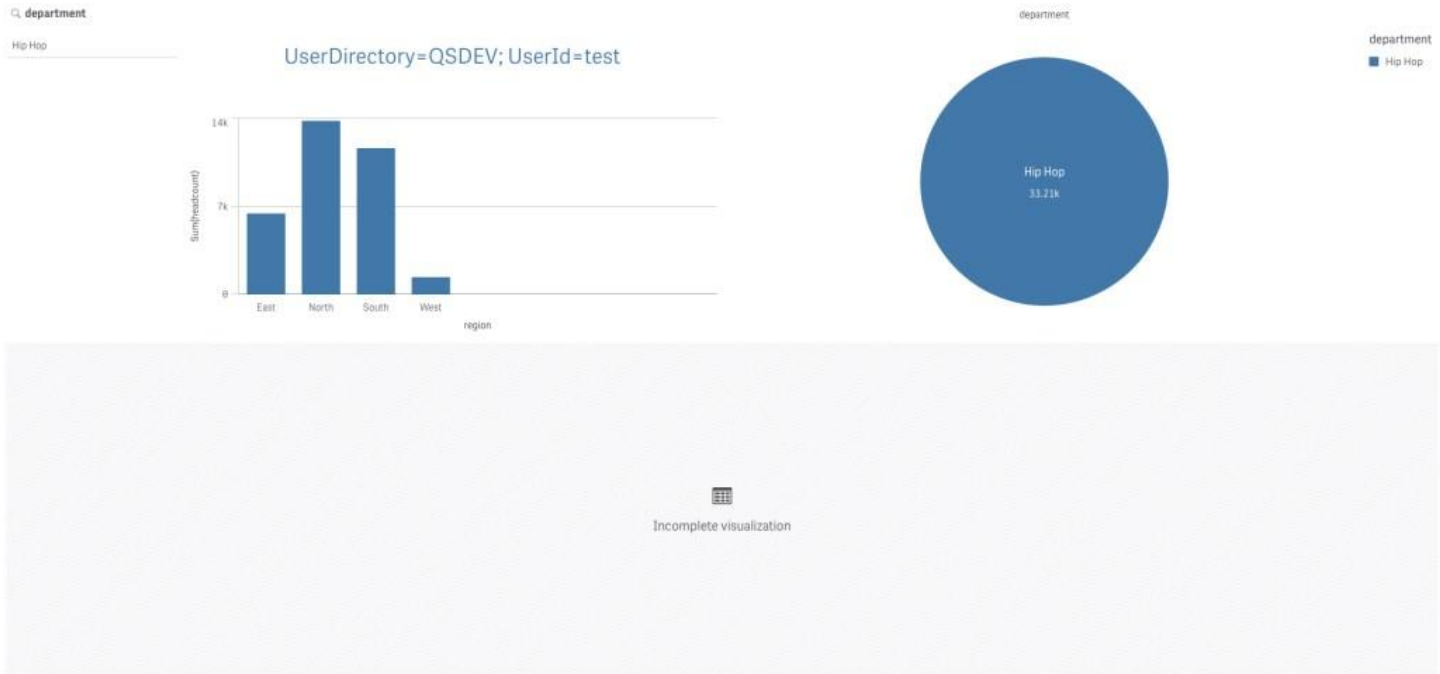
Access Denied

If a user cannot open the application from the hub, they are not in the Section Access security table.



OMIT Experience

If you omit a field for a user and a chart uses that field as a dimension, it will cause an error. To account for this you will need to make sure there is a calculation condition on the dimension/measure so that it is hidden for the users that do not have access to it. That or leverage a Hide/Show mechanic in a container to replace the chart with something else that would be useful.



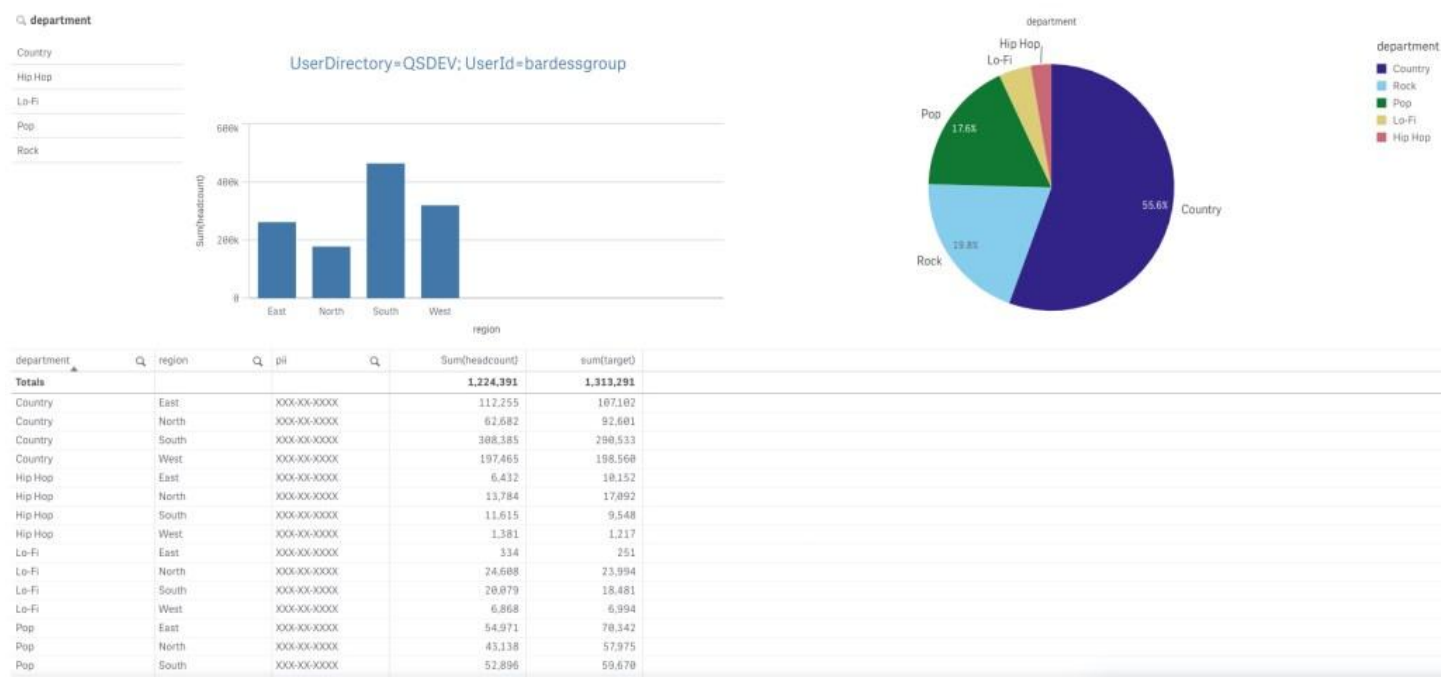
What we typically do is add an expression in the calculation condition:

Example – Omit field pii

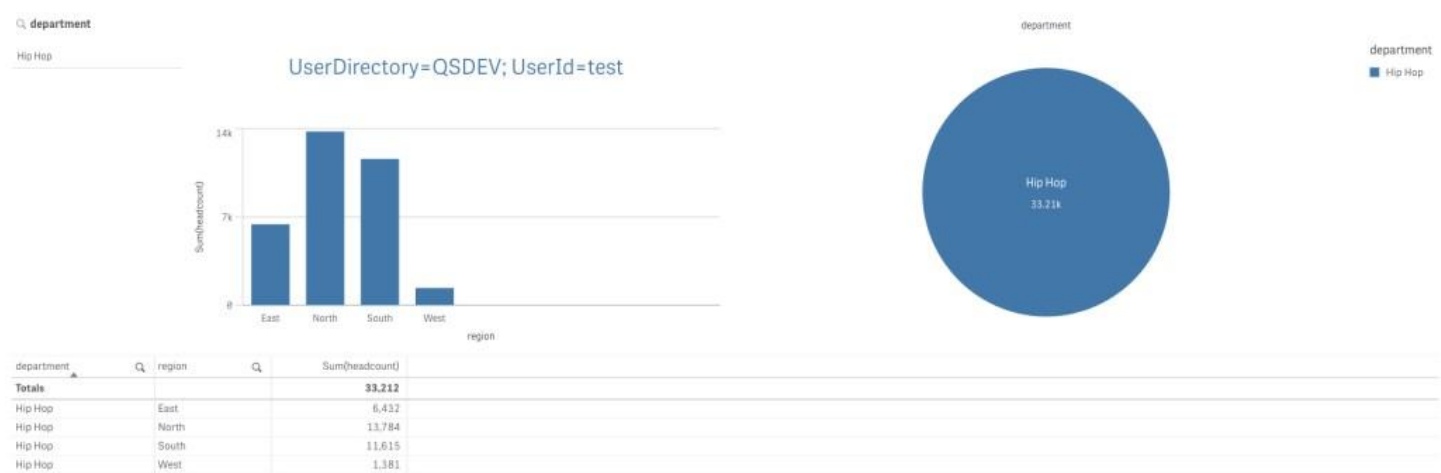
```
=If(Len( '$(=MaxString({1}[pii]))' )>1,1,0)
```

We make sure we are using the set of 1 {1} so the users that do have access to the field are not affected by the condition because of their filters. Additionally, we are checking the MaxString value so it works for both text and numeric fields. If the field doesn't exist, it will return NULL which will return -, hence the condition of Len() > 1.

User With Access



User Without Access



Hopefully this helps you troubleshoot quickly rather than spending time and getting frustrated from working with Section Access.

Understanding QVDs and Optimized Loads

If you have been working with Qlik Sense® or QlikView® for some time, we are sure you have heard of QVDs. If you have been working with Qlik for some time hopefully you have heard of **optimized** QVD loads and **non-optimized** QVD loads. Either way, hopefully we can reveal some of the mysteries around QVDs and these two loads in our final section.

What is a QVD?

A QVD is a proprietary data file format made by Qlik® which is made up of two parts:

1. XML metadata header
 1. This contains the data around the data, such as the number of records, fields, tags, etc..
2. Binary data
 1. This is the actual data which has been compressed in a way unique to Qlik. One of Qlik's key differentiators.

How is data stored in a Qlik application?

This is a crucial point when optimizing in the Qlik world. There is a wonderful blog post by **Henric Cronström** who explains how the Qlik® Associative Engine stores data internally which you can read [here](#). We advise any Qlik developer to read this, and read it again. This will drive so many decisions for optimization.

To summarize, data is stored in a columnar format in the sense that each field has its own structure. There is a distinct list of values for a field which contain a mapping of the value and a reference point. This reference point (bit-stuffed pointer) is what is populating the data tables. So ultimately there is only one single value stored for each field value. All duplicate values for the rows in the table are references. This allows the data to be compressed immensely. This is also why field cardinality is such a large driver of the Qlik Sense® application size.

QVD Data Storage

So if Qlik® has this special data structure / storage mechanism, how does this affect QVDs? Well, QVD binary data is stored in a very similar format as it exists in memory.

Optimized QVD Loads

Optimized QVD Loads, in a simple explanation, take the data from the QVD and push it directly into memory. There is no processing or time necessary to read or interpret the data, it is already in the form it needs to be in. The time it takes is mainly moving the file from disk to memory. It really is a feat that Qlik can load hundreds of millions of rows in seconds.

Unoptimized QVD Loads

Knowing what an optimized QVD load does, it makes more sense of what causes an unoptimized QVD load. Once you perform a field transformation or where condition, it becomes unoptimized; *with a few exceptions*. This is because Qlik Sense® can't just directly push the data into memory anymore. Now the data has to be *unwrapped* and modified before it is stored in memory. This is still typically faster than other data formats such as flat files, databases, APIs, etc., but it is nowhere near as fast as an optimized load. This is why the Qlik community holds it in such a high regard.

What breaks a QVD Load?

It makes more sense to describe what you can do and still **keep** a QVD load optimized.

- Renaming fields.
- You can load a field twice, with a different name of course.
- *Simple* Where Exists.
 - Simple: Where Exists([Field])
 - **Not** Simple: Where Exists([Field], [FieldA]&[FieldB])
 - The field used in the Exists clause must be in the load script.
 - Ex: Tbl: Load **Period**, Dim, Amt From [Sales.qvd] (qvd)
Where Exists(**Period**);
- Concatenate to a table, only if it contains all of the fields of the table it is concatenating to.
 - QVD can contain additional fields.

To summarize what would break an optimization:

- Field transformation
- New field: expression, number, string, etc.
- Where & While conditions
- Joins
- Concatenating a QVD to a table which has fields not in the QVD

Strategy

We will admit this can be a bit limiting. There are going to be use cases where the data needs to be modified in some way. Our suggestion is to frame out your load to best leverage QVD loads to optimize getting data into your application. Sometimes this can be more roundabout, but will likely be faster.

Example

Sales QVD:

- 400 Million Rows
- 15 Years of Data
- 3 Company Ledgers

Requirement:

- Rolling Three Years
- OL Ledger Only

Traditional Load Script (*Unoptimized*)

```
sales:
Load
    Id,
    Year,
    Period,
    CostCenter,
    Account,
    Region,
    Company,
    Ledger,
    Qty,
    Amount
From
    [lib://QVD/Sales.qvd](qvd)
Where
    Year >= Year(Today())-2
    AND Ledger = '0L';
```

Logic

How can we leverage a QVD optimized load to get the smallest data set we need in the quickest way?

Assuming all things are equal:

- 400M rows / 15 years = ~27M per year
 - 3 years = ~100M rows
- 400M rows / 3 ledgers = ~133M rows per ledger
 - 1 ledger = 133M rows.

3 years of data seems to be the smaller cut of data.

We would want to leverage the simple where exists condition to filter down to the three years of data and then further filter down to a single ledger.

Optimized Load Script

```
// Get rolling Three Years
year_exists:
Load
    Year(Today())-(IterNo()-1) as Year
AutoGenerate
    (1)
While
    IterNo() <= 3;

sales:
Load
    Id,
    Year,
    Period,
    CostCenter,
    Account,
    Region,
    Company,
    Ledger,
    Qty,
    Amount
From
    [lib://QVD/Sales.qvd](qvd)
Where
    Exists(Year); // Keeping QVD Load optimized

// Remaining Logic
Inner Join(sales)
Load * Inline [
    Ledger
    0L
];

Drop Table year_exists;
```

Even though there are more steps, it will cut down the reload time greatly. This is another example of how you should test different scenarios and strategies to determine which is faster. In our example, it could be that the **0L** ledger only accounts for 10% of the data and is much smaller than three years of the total dataset.

Conclusion

With this, we will come to a close. We hope this information has helped you as much as it has helped us. Our goal is to innovate and create cool and interesting solutions for complex problems. This has led us to realize that understanding the fundamentals is crucial to accelerate our journey. We have found it very rewarding taking these ideas and topics and boiling them down into a simple form to help teach others, while better understanding them ourselves.

Thank you for reading!

