QlikView and Qlik Sense – Big Data Methodologies

Published: June, 2016

Version: 3.0

Inventors: Michael Robertshaw, Loic Fromont, Anthony Alteirac, Joe Bickley, Bill Kehoe

Scribe: Ian Crosland

Qlik | QlikView | Sense

Contents

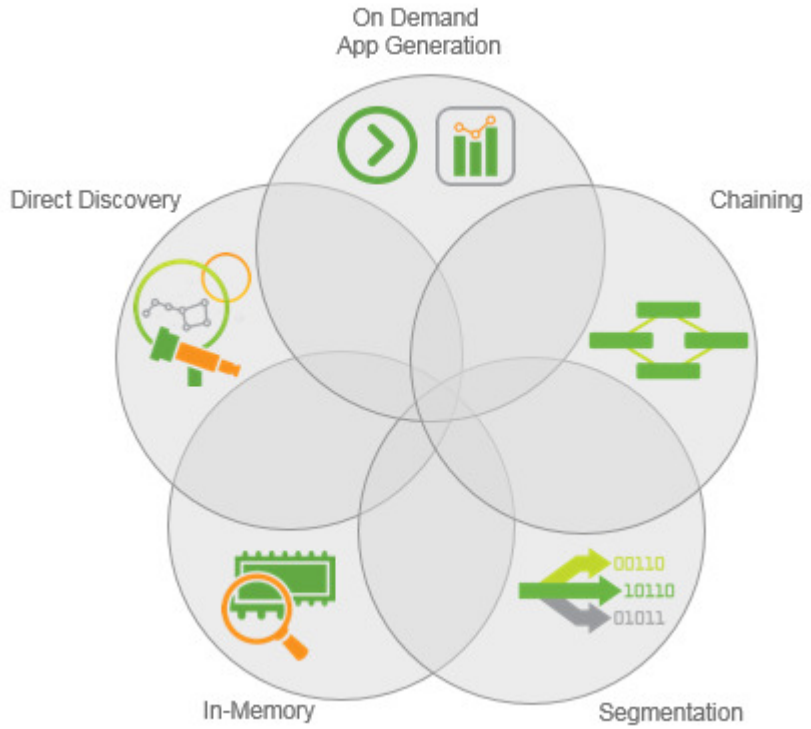
1 Overview

This document provides a technical overview of the various options we have deploying QlikView and/or Qlik Sense in a “Big Data” environment. The technical focus of the document will discuss one approach in detail namely “on demand app generation”, details of other options namely Direct Discovery and QVD segmentation have been covered extensively in other documents.

The “on demand app generation” approach expands the potential use cases for Business Discovery, enabling business users to conduct associative analysis on larger data sources. It provides a shopping list experience to allow users to first select data they are interested in discovering insights on which interactively an analysis app with full Qlik in memory capabilities.
The following part of the paper provides a technical overview of implementing and using “on demand app generation” in both QlikView and Qlik Sense.

2 Use Cases

A determination of which method or indeed methods to apply (as they are not mutually exclusive) can be thought of at a high level on a few factors, namely data volume and complexity of analysis required.



Below is a pro/con analysis of the various approaches:

Method	Details	Data Volumes	Pro	Con
In Memory	Highly compresses data into memory. Methods for data load can extend this even further.	100s millions to billions	- No limit on Qlik functionality - Customizable SQL/script generation - Can be used for non-sql sources	Limited to memory on single server
Segmentation & Chaining	Users move between multiple related segmented apps (e.g. by region).	100s millions to billions of rows per segmented app	- No limit on Qlik functionality - Customizable SQL/script generation - Can be used for non-sql sources	Overt data replication
Direct Discovery	Combines the associative capabilities of the Qlik in-memory dataset with a SQL query model.	Billions of rows although very limited data model	Measure data resides at source until execution	- Non optimized SQL generation - No Set analysis - No section access with high cardinality joins - Can only be used with SQL sources - Source code in engine and cannot be modified - Does not support PLACEHOLDER function for SAP HANA
On Demand App Generation	User selections spawn generation of a filtered data set for analysis via extension in Sense or custom EDX in View	Massive scalability Including high cardinality, dimensions and rows	- No limit on Qlik functionality - Customizable SQL/script generation - Can be used for non-sql sources e.g. SAP Bex and PLACEHOLDER function in HANA - Section access possible in all cases - Source code using native API's and shared	- Higher implementation time

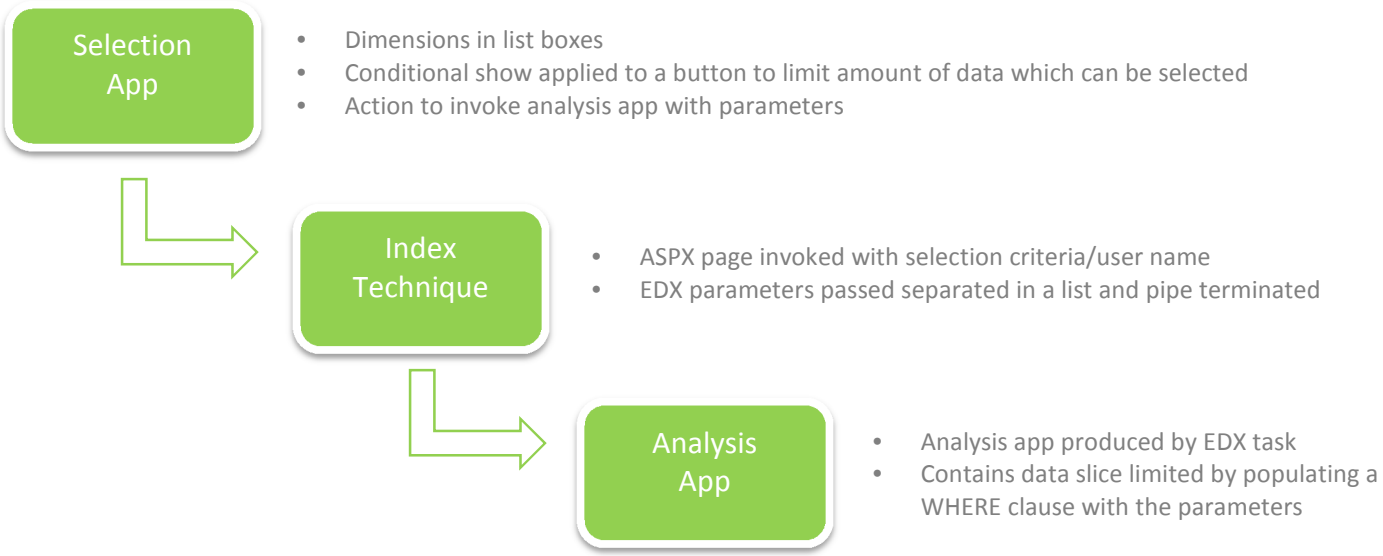
Below is a high level summary of the On Demand approach and how this helps the business user and IT to deliver value from a Big Data Environment:

- Provides users with a “shopping list” experience allowing a particular subset of data such as a Time Period, Customer Segment or Geography from the “index” app to interactively populate the “analysis” app.
- Provides full QlikView/Qlik Sense functionality on a latent subset that is hosted in memory.
- Influencing a non-SQL query such as a Teradata Aster or MapR that cannot be performed using Direct Discovery.
- Governed by IT as to how large and when a detail app can be invoked based on data volume or dimensional selections.

3 On Demand App Generation - QlikView

An example of a QlikView deployment for this methodology is described in the following section along with an example in this pack. QlikView relies on construction of the script and EDX tasks for a server based deployment. Qlik Sense makes use of the API's in the platform to achieve the same result.

Deployment in a desktop and server environment will use the same methodology but a different indexing technique, one for the server is described in the flow below:



The complexity for the server environment is when the user interacts with the apps via a browser. In a server deployment, only the QlikView Distribution Service "Publisher" can execute the QlikView Script to materialise data into the subsequent QlikView analysis app.

The Web server and QlikView server services deliver content from the QlikView app but cannot modify the data in it. A combination of techniques are required for the user to be able to influence the resultant analysis app content.

3.1 QlikView Architecture Overview

The following diagram describes the architecture of an on demand app generation process which has been deployed live at one of our customers:

1. The selection app is populated with dimensional data on schedule.
2. User selects dimensional criteria. After the governed limit is reached an ASPX page is invoked.
3. QMS API and EDX indexes the analysis app with the most recent data from the Teradata database with only the data slice relevant to the user.
4. The analysis app deployed to access point with user security.

3.2 Selection App

The concept behind the selection app is to provide the user with a shopping list of dimensions from which they can select a range of data values (It is recommended to limit the amount of data with a conditional show calculation on the button which will invoke the reload task).

The procedure should follow the steps below:

1. Create a selection app that offers a selection of Dimensions but no Facts.
2. Place a button on a sheet. This should be enabled only when sufficient selection criteria have been met essentially to minimise the reload time based on the data volumes.

   an example of a conditional show condition:

   ```
   GetPossibleCount( [OrderDate]) * GetPossibleCount( [ProductID]) > 0
   AND
   GetPossibleCount( [OrderDate]) * GetPossibleCount( [ProductID]) < 20
   ```

   This would force the user to select at least one value in each of these dimensions before the button was enabled. They can then control the breadth (Segment Name) and depth (Dates) of data to be loaded into the analysis app.

3. Assign an action on this button which would invoke an ASPX page, passing it the selection criteria and the current Username. The selection criteria must be labelled and delimited.

   The example below constructs the EDXparms variable which consists of repeated instances of the various field names (ProductID, OrderDate and TerritoryID) combined with a comma-delimited value list with each set delimited by a pipe.

```
='http://WIN-006FPHODF1N/OnDemand/Default.aspx'

      & '&user=' & replace( OSuser(), '\' , '#')

      & '&EDXparms='
```

```
                        & if( GetSelectedCount( [Product Name]) >0,
                                        'ProductID=' & concat( [ProductID], ',')
& '~')

                        & if( GetPossibleCount( [OrderDate]) >0,
                                    'OrderDate=' & concat(distinct
date([OrderDate],'YYYY-MM-DD'), ',')  & '~')

                        & if( GetSelectedCount( [Territory Name]) >0,
                                        'TerritoryID=' & concat( [TerritoryID],
',')& '~' )
```

3.3 Analysis App

The analysis app will contain the relevant chart and objects the user requires for analysis along with the script elements which will consume the variables produced from the Index app.

The reload task will begin with an initial value for the variable EDXparms. The script in the app must parse the fields [delimited by pipe] and value lists [delimited by comma] from EDXparms then use this to modify the data source query that will be performed by the script.

The modification of the query may be as simple as string replacement (using concatenation) or could be construction of a WHERE clause for an SQL query.

The value lists could be in different formats for example distinct values or values containing wildcards, the variable manipulation will cater for that.

For example a combination of ProductID (with wild cards) and distinct TerritoryID's could produce the following string:

EDXparms=ProductID=prod1,prod2*|TerritoryID=1,2,3*

which would need to resolve to a where clause as follows:

WHERE (ProductID LIKE 'prod1' OR ProductID like 'prod2*) AND (TerritoryID IN (1, 2, 3))*

(with appropriate delimiters pending on datatype numeric/string).

An example script is shown below which will produce a list of ProductID's to pass into a where clause based on some Product Name selections.

Firstly the delimiters are set for the fields and values

```
SET FieldDelimiter = '|';
SET ValueDelimiter = ',';
```

Set the conditions parameter to 0 and populate the field name and field values variables

```
LET nConditions = 0; // number of conditions in WHERE clause
LET nEDXparms = SubStringCount( '$(EDXparms)', '$(FieldDelimiter)' );
FOR i = 1 to nEDXparms +1;
    LET sEDXparm = SubField('$(EDXparms)', '$(FieldDelimiter)', i);
    LET sField = SubField( '$(sEDXparm)', '=', 1);
    LET sValues = SubField( '$(sEDXparm)', '=', 2);
```

Enter the logic to parse the Product Name selections

```
    SWITCH '$(sField)'

      // Derived Table query upon Product table
      CASE 'ProductName', 'ProductID'

       IF Len('$(sValues)') >0 THEN
```

Identify any wild cards to build a LIKE statement to be passed into the where clause if required

```
     // there are Values to search for
        IF Index('$(sValues)', '*') > 0 OR Index('$(sValues)', '%') > 0 OR
        Index('$(sValues)', '?') > 0 OR Index('$(sValues)', '_') > 0
        THEN

          // Build a LIKE clause with OR between each term
          LET sCondition = '';
          LET nValues = SubStringCount( '$(sEDXparm)', '$(ValueDelimiter)' )+1;
          FOR j = 1 to nValues;
              LET sValue = SubField( '$(sValues)', '$(ValueDelimiter)', j);
              IF j > 1 THEN
                LET sCondition = '$(sCondition) OR $(sField) LIKE "$(sValue)"';
              ELSE
                LET sCondition = '$(sField) LIKE "$(sValue)"';
            ENDIF;
          NEXT j; // value
```

For multiple values an OR clause is added to the variable

```
          LET sCondition = '($(sCondition))'; // put the OR clauses in brackets
          LET sCondition = replace('$(sCondition)', '*', '%');
          LET sCondition = replace('$(sCondition)', '?', '_');
```

If wildcards are not detected a distinct value IN clause will be generated to pass into the resultant WHERE clause variable

```
   ELSE
          // Build an IN clause
          LET sCondition = '';
          LET nValues = SubStringCount( '$(sEDXparm)', '$(ValueDelimiter)' )+1;
          LET sValueList = '';
          FOR j = 1 to nValues;
              LET sValue = SubField( '$(sValues)', '$(ValueDelimiter)', j);
              IF j > 1 THEN
                LET sValueList = '$(sValueList), "$(sValue)"';
              ELSE
                LET sValueList = '"$(sValue)"';
              ENDIF;
          NEXT j; // value
          LET sCondition = '$(sField) IN ($(sValueList))';
      ENDIF
```

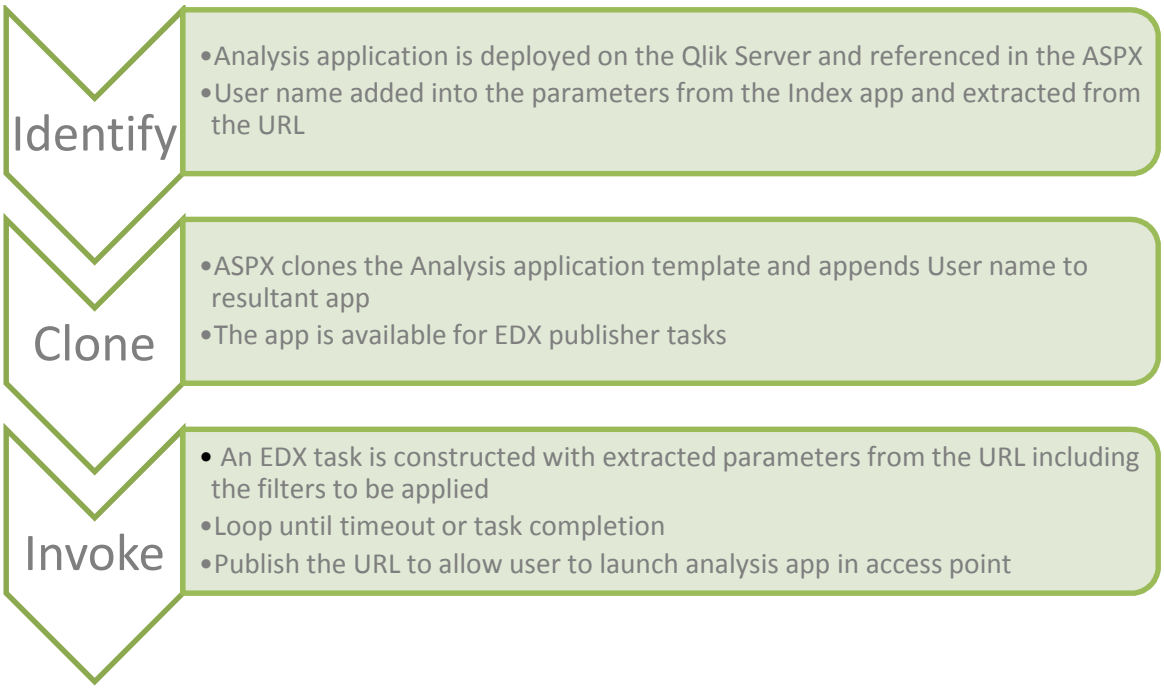
Populate the variable which will go to make up the first elements of a subsequent where clause

```
// replace stringDelimiter with SingleQuote

  LET sCondition = replace('$(sCondition)', '"', chr(39));


 IF nConditions > 0 THEN // multiple clauses separated by AND condition
     LET sWhere = '$(sWhere) AND $(sCondition)';
 ELSE
     LET sWhere = '$(sCondition)';
 ENDIF;
 LET nConditions = nConditions + 1;
 ELSE
// No search terms
TRACE # No values in field $(sField);
ENDIF;
```

After the EDXparms have been parsed the modified query can be executed in the analysis application using the populated WHERE clause variable.

3.4 Custom ASPX using QMS API

A button in the Index app will cause a custom page to be invoked, passing it the user identity and the selection criteria. The following procedure is followed:

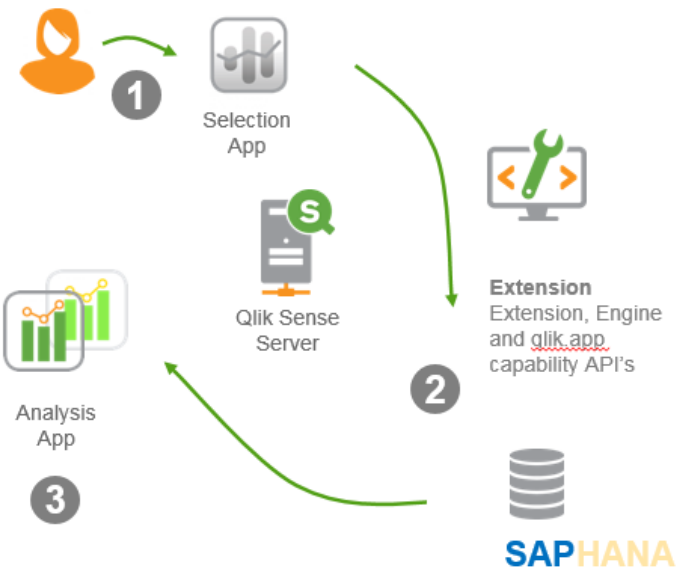

Identify
•Analysis application is deployed on the Qlik Server and referenced in the ASPX
•User name added into the parameters from the Index app and extracted from the URL
Clone
•ASPX clones the Analysis application template and appends User name to resultant app
•The app is available for EDX publisher tasks
Invoke
• An EDX task is constructed with extracted parameters from the URL including the filters to be applied
•Loop until timeout or task completion
•Publish the URL to allow user to launch analysis app in access point

4 On Demand App Generation - Qlik Sense

An example of a Qlik Sense deployment for this methodology is delivered as an example with the installation of Qlik Sense 3.0 and beyond. The sample files include the following:

- Extension Object to invoke the on demand process
- Selection application containing sample flight data showing high level aggregated metrics and dimensions for selections
- A detail level app which gets populated with data driven from the selection app

Below is a high level architecture of the process flow:



1. Selection App is populated with dimensional data on schedule.
2. User selects dimensional criteria in a selection app containing the extension. After the governed limit is reached an extension object button appears. The extension object invokes the Engine, Extension and qlik.app capability API's.
3. Analysis app updated with the relevant data from the user generated selections.

The On Demand App Generation extension makes use of the following APIs:

- Extension API
  - to implement the “paint” behaviour for drawing the buttons and status messages
  - to establish the extension’s property panel configuration using the initialProperties object
  - to make use of the built-in support for AngularJS by returning the HTML template in the “template” property
- qlik.app capability API
  - to identify the names of all the fields of the source app
  - to determine the selection state of the fields that are bound to expressions in the script of the target app
  - to count the number of selected (or optional and/or excluded) values of bound fields that have selection quantity constraints
- Engine API
  - to find and open the on-demand template app
  - to access the script of the on-demand app template and identify the bind expressions
  - to create a copy of the template app (i.e. ‘the generated app’)
  - to set the script of the generated app so that all bind expressions from the template app’s script have been updated to contain the desired selection state harvested from the data selection app
  - to save the generated app
  - to reload the generated app