Data Visualization with kdb+ using ODBC: A Tableau case study

Author: Michaela Woods

Contents

[Kdb+ and 3rd-Party Visualization Tools 3](#_Toc519094520)

[Connecting to kdb+ using ODBC 4](#_Toc519094521)

[Connecting to Tableau Desktop 5](#_Toc519094522)

[Connecting to Tableau Server 5](#_Toc519094523)

[Other Considerations 6](#_Toc519094524)

[Tableau functionality for kdb+ 7](#_Toc519094525)

[Calling q from Tableau 7](#_Toc519094526)

[Datatypes 9](#_Toc519094527)

[Simple Parameters 10](#_Toc519094528)

[Tableau Filters and Caching 12](#_Toc519094529)

[Caching 12](#_Toc519094530)

[Dynamic Parameters 16](#_Toc519094531)

[Predefining parameter options in a q function 16](#_Toc519094532)

[Using parameters with calculated fields 17](#_Toc519094533)

[Multiple Data Sources 18](#_Toc519094534)

[Actions 18](#_Toc519094535)

[Publishing to Tableau Server 21](#_Toc519094536)

[Conclusion 23](#_Toc519094537)

[Appendix A 24](#_Toc519094538)

# Kdb+ and 3rd-Party Visualization Tools

Business intelligence (BI) tools are widely used across many industries for their interactive nature that enables users to easily create and customize dynamic data visualizations. While Kx provides its own visualization tool, Dashboards for Kx, clients may have incumbent solutions they wish to retain and independently connect to kdb+. Alternatively, many organizations may wish to migrate their backend database to kdb+ for increased efficiency and scalability, while at the same time retaining their existing visualization front end. This paper aims to provide guidance on how such integration may be achieved.

One such example of a widely used BI tool is Tableau and this paper outlines how it can be used to access kdb+ via ODBC. ODBC stands for Open Database Connectivity which is a standard application programming interface that can be used to connect to different database management systems as it has been specifically designed to be independent of databases and operating systems. Version 3.4 of Kx included an updated version of its Windows ODBC driver (ODBCv3) to support wider access to kdb+ data.

This paper illustrates the flexibility by which kdb+ data can be accessed by Tableau using ODBC. It explains further how kdb+’s caching feature may be used in this instance to improve performance by optimizing repeated queries. It should be noted, however, that there will always be limitations on third-party solutions that were not designed from the outset for processing real-time streaming data. For example, Kx’s visualization tool Dashboards for Kx is optimized for streaming queries and inherits functionality like user management, load balancing, access control, caching and queuing from the underlying platform as well as direct access to q for comprehensive querying capabilities. Such features and their ability to support high volume, low latency access to streaming data cannot be assumed in third-party products.

Guidelines on connection, setup, queries and advice on how to maximize performance will be discussed throughout this paper. For both new and existing users, this paper aims to reduce the learning curve, boost efficiency and increase usability when combining these two technologies.

All tests were run using kdb+ version 3.5 and Tableau 10.3.

# Connecting to kdb+ using ODBC

Instructions on how to connect kdb+ from Tableau Desktop for both Windows and Linux can be found on the Kx webpage <http://code.kx.com/q/interfaces/q-server-for-odbc3/>.

For an ODBC driver to connect to an application, it needs to have a DSN. This stands for Data Source Name which is a data structure that contains the information about a specific database. Included in the DSN are the name, directory and driver of the database, and depending on the type of DSN, the username and password of the user.

With Administrator rights, adding a new DSN is relatively straightforward, as detailed in the instructions in the webpage linked to above. It should be noted that there is also a second way to add a DSN which does not require Administrator access which might be useful for some users. This second method involves defining the DSN connection details in a registry file as opposed to adding new DSNs directly in the ODBC Data Source Administrator. This method is an alternative to steps 3, 4 and 5 provided in the webpage:

1. Copy qodbc.dll to the correct location.
2. Define the registry file and save it to C:\Users\<username> with a .reg extension. Here is an example of what the file might look like.

Windows Registry Editor Version 5.00

[HKEY\_CURRENT\_USER\SOFTWARE\ODBC\ODBC.INI\DEV]

"Driver"="P:\\....\\qodbc3\\w64\\qodbc3.dll"

"Description"="KDB ODBC3 Driver"

"HOST"="hostname:port"

"UID"="username"

"PWD"="password"

[HKEY\_CURRENT\_USER\SOFTWARE\ODBC\ODBC.INI\ODBC Data Sources]

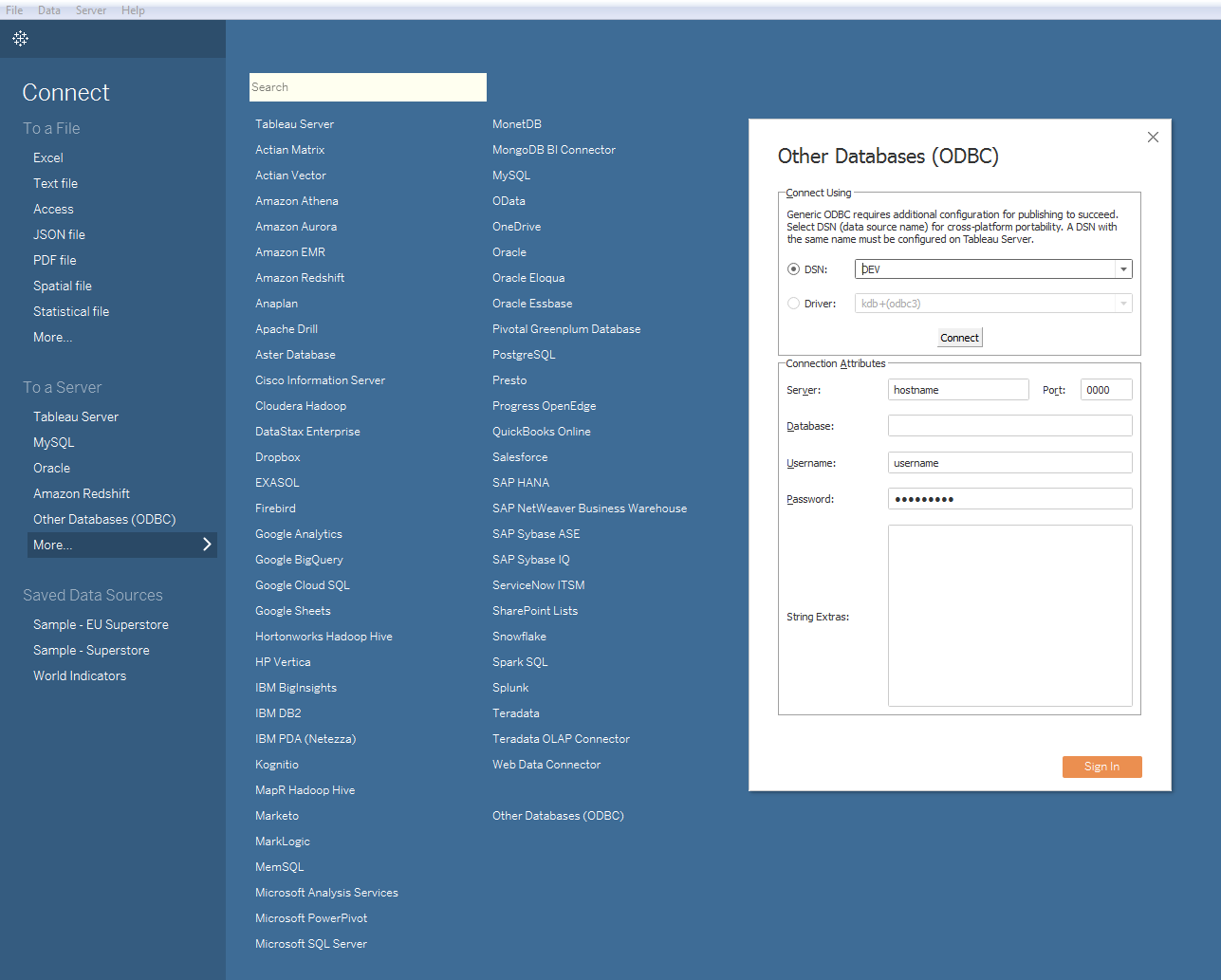
"DEV"="KDB ODBC3 Driver"

1. Double click on the file when saved. This will create the correct driver entries, which for this example will be a new kdb+ DSN called "DEV".

This second method has an additional benefit of making it easier to maintain and share connection details with multiple users as the DSN details reside in a separate text file rather than in the Windows registry.

## Connecting to Tableau Desktop

Once a kdb+ DSN has been added, and the rest of the setup instructions on the webpage are followed, you are ready to connect to kdb+ from Tableau. On opening Tableau, you will be prompted to select the type of database you wish to connect to, select the option "Other Databases (ODBC)".



Next, select the correct DSN from the dropdown list and click "Connect". This will automatically populate the Connection Attributes in the bottom half of the windows using the details defined earlier in the registry file. The final step is to click the "Sign In" button which creates a connection to the kdb+ process, enabling the database to be queried via Tableau's Custom SQL, as demonstrated in the following section.

## Connecting to Tableau Server

The setup instructions defined on the Kx webpage, and above, are specifically for a user connecting from Tableau Desktop. This is the local version of Tableau installed on a desktop or laptop. Tableau Server, on the other hand, is installed on a Windows server and is accessible to users via a browser. Tableau Server brings additional collaboration, security and scalability capabilities not available using only Tableau Desktop.

Tableau Workbooks can be shared between both by publishing from Tableau Desktop to Tableau Server. This procedure will be detailed later, in the section “Publishing to Tableau Server”.

To connect via Tableau Server, the registry file that was presented in the previous section needs to be configured. This process may be handled by an organization’s support team, depending on the installation setup. The driver also needs to be installed, and then the connection can be initialized in a similar way as described for Tableau Desktop.

## Other Considerations

Since a release on 2017.09.11[[1]](#footnote-1), qodbc3 allows specification of connection details without a DSN. This means all details, except the password, will be saved by Tableau in a workbook or saved data source. However, this change only affects desktop users. Because the password is not embedded, the DSN is still required to be defined on the server as this is the only way the password will be picked up for published reports.

It is also important to note that connection details are embedded in both the Tableau workbook and the DSN definition. For version management, when sharing workbooks between developers or when publishing them to Tableau Server, this can become problematic. One workaround solution to manage this is to wipe these details from the workbook with a script before sharing or publishing workbooks. This concept will be explored in more detail in the later section "Publishing to Tableau Server".

# Tableau functionality for kdb+

## Calling q from Tableau

Once a successful connection has been made, the next step is to begin by running some sample queries. Tableau's Custom SQL is the method by which q queries can be run from Tableau. In particular, the q() function can be used to send synchronous queries to kdb+, as shown below.

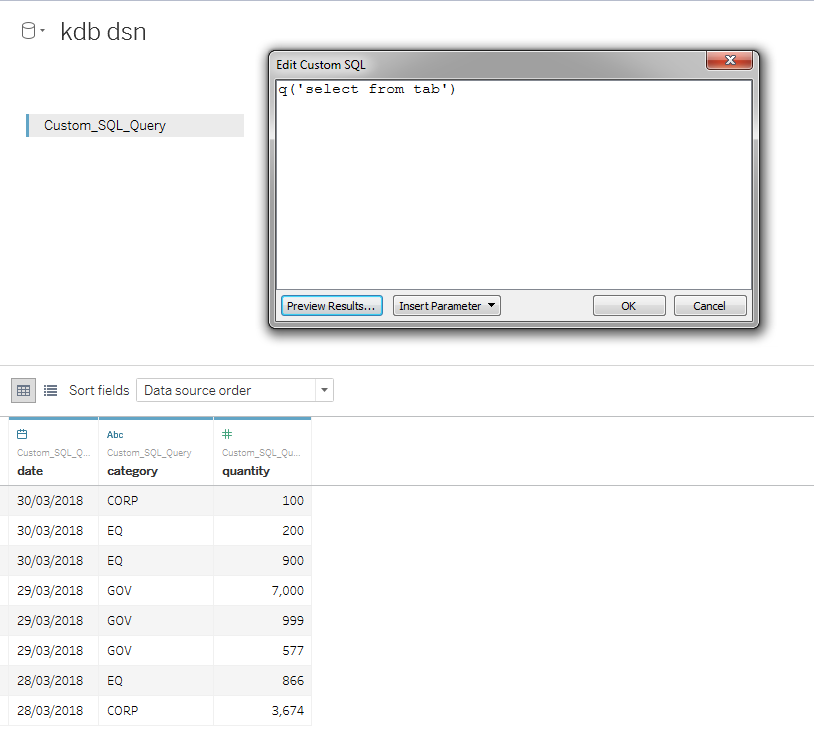
To demonstrate this, define a table 'tab' in the q process that you are connecting to.

q)N:8

q)dates:2018.03.28 + til 3

q) tab: ([] date:N?dates;category:N?`CORP`EQ`GOV;volume:N?til 10000)

Then, in Tableau run the following in the Custom SQL.



Now the data in the table 'tab' is available for use in Tableau. Note that if tab is a not a partitioned table (and is small enough to be handled via sql), you can just type its name into the table selector, there is no need to use q('select from tab'). Other acceptable syntaxes are :

* q('tablename')
* q('select from table where date in 2018.07.02')
* q('function',<Parameters.Date>)
* q('{[mydate] func[..]',<Parameters.Date>)

Queries can be a simple select statement or can become much more complex and flexible using inbuilt Parameters supplied by Tableau, which will be demonstrated in the next section.

As documented on code.kx.com, it should be noted that there are known SQL compatibility issues including:

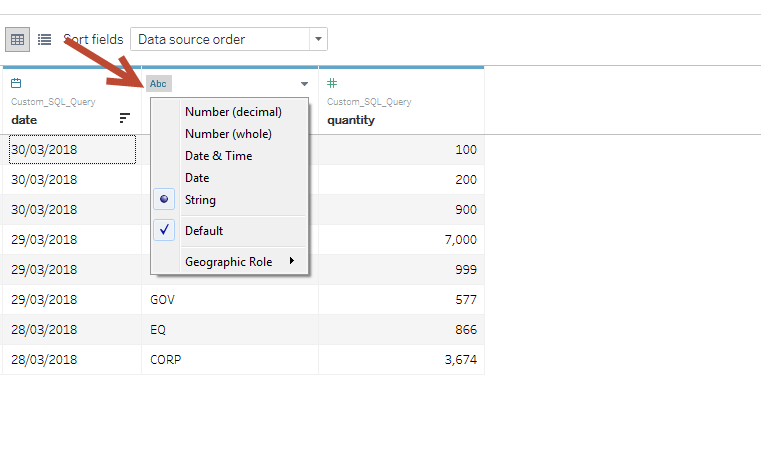
* SQL string literals are trimmed like q symbols
* MIN() and MAX() don't work on strings
* q strings and bools lack nulls, therefore SQL operations on null data resulting in these types 'erase' nulls
* COUNT and COUNT DISTINCT don't ignore nulls
* SQL selects from partitioned tables are not supported -- one should pre-select from a partitioned table using the q() function instead.

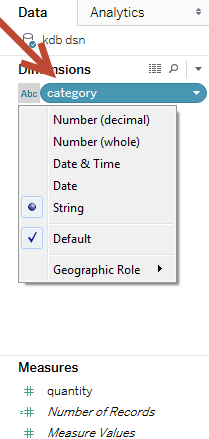
## Datatypes

Tableau caters for multiple q datatypes.

|  |  |
| --- | --- |
| **Data Type Tableau** | **Datatype q** |
| String | Symbol, String |
| Date | Date |
| Date & Time | Timestamp |
| Numerical | Int, float |
| Boolean | Boolean |

Upon loading of data Tableau automatically interprets the datatype of a field. It is recommended that the user checks these have been interpreted correctly after the data is loaded. If it is incorrect, the datatype can then be easily changed on the Data Source page or in the Data pane as shown below.





## Simple Parameters

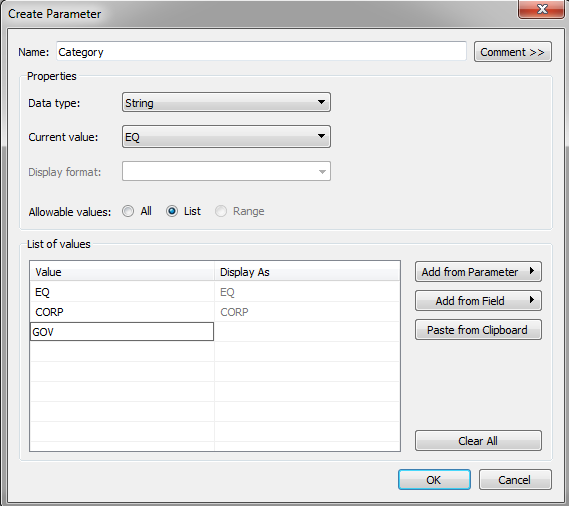
Tableau parameters provide further flexibility when working with q functions. To demonstrate, define a function 'func' that selects from the table 'tab' defined in the previous section. This function can be called from Tableau using Tableau defined parameters.

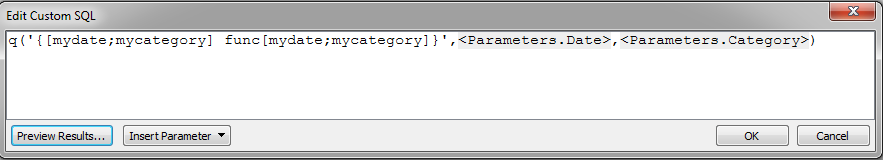
q) func:{[mydate;mycategory]

select from tab where date in mydate, category in mycategory

};

Take the parameter 'mycategory' in this example, a list of allowable symbols that are acceptable for the parameter 'mycategory' can be defined in Tableau.





This can be done in the Custom SQL stage when you are writing your query. These parameters can then be shown and made available for users as a dropdown list on Worksheets and Dashboards as can be seen below.

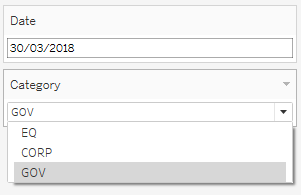


Tableau parameters are limited to static values, and a single select option when placed in a view. However, there are ways to make them more dynamic and flexible. This will be explored in the later section "Dynamic Parameters".

## Tableau Filters and Caching

As shown above, parameters are a useful tool for creating user-defined inputs to visualizations. However, there are cases where the user may want to return the entire data set first and only afterward reduce the data set. This can be achieved using Tableau's filters.

Tableau Category Parameter as defined in the previous section:

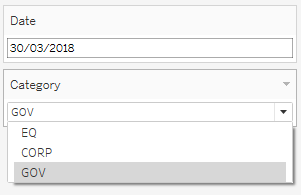
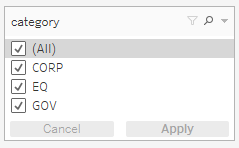


Tableau Category Filter:



Filters are the standard way to reduce the set of data that is displayed on a worksheet. Note from the above screenshots that filters are not limited to a single select option like parameters are.

Filters are most effective with fast queries on small datasets. For longer queries and/or larger datasets, filters become challenging from a performance point of view. This is because every time a filter selection is changed, the Custom SQL query runs the same query multiple times per view to build dimensions. Therefore the more filters and dimensions you add to a view, the slower performance becomes.

Caching

One suggested way to get around this inefficiency is to introduce caching in kdb+. Caching is when results from previous queries or calculations are stored in an internal lookup table (or cache) which can then be used for faster data retrieval on subsequent queries. Caching here is being used to address the problem of filters causing queries to be re-run.



The following example demonstrates the performance improvement of caching when incorporated into a simple q function, **getTotalVolume** (below), which extracts the total volume by symbol from a table, t.

The table t contains mock data of randomly generated symbol and volume values for demonstrative purposes.

q) N:100000000

q) t:([] sym:N?`3;volume:N?10.0)

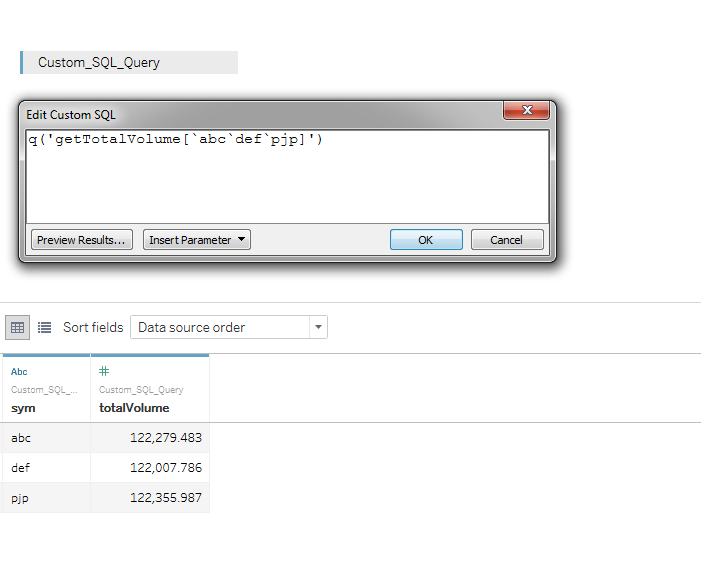
// Function used to compute the total volume by symbol from the table t

q) getTotalVolume:{[syms]

  select totalVolume:sum volume by sym from t where sym in syms

  };

Below is a sample output of this function when called from Tableau. Query response times for an increasing number of symbols runs from hundreds of milliseconds to seconds, see Table 1.



|  |  |
| --- | --- |
| **Number of Symbols** | **Time** |
| 1,000,000 | 13 ms |
| 10,000,000 | 120 ms |
| 100,000,000 | 1038 ms |

**Table 1 – No Caching**

To incorporate caching, the existing function can be modified to store the total volume result for each queried symbol in a keyed table, called **volumeCache**. Whenever the function is called from Tableau, an internal lookup is performed on the **volumeCache** table to determine if the calculation for the requested symbol has already been performed. If so, the result can be immediately returned, otherwise a calculation against the table t is performed.

q) volumeCache:([sym:`u#`symbol$()];totalVolume:`float$())

q) getTotalVolume:{[syms]

  if[-11h~type syms;syms:enlist syms];

// Get the list of syms which contain entries in the volumeCache

  // Extract the totalVolume values for those symbols

  if[count preCalculated:([]sym:syms) inter key[volumeCache];

    result:select from volumeCache where ([]sym) in preCalculated

];

  // If all syms are contained in the volumeCache then return result

  if[not count notPreCalculated:([]sym:syms) except key[volumeCache];

    :result

  ];

  // For syms not present in volumeCache, perform lookup

  result,:newEntries:select totalVolume:sum volume by sym from t where ([]sym) in notPreCalculated;

  // upsert new results to volumeCache

  upsert[`volumeCache;newEntries];

  result

 };

Tableau queries against this modified function are significantly faster and become sub-millisecond when symbols are already present within the volumeCache, see Table 2. This approach greatly reduces the effect of filtering previously highlighted.

|  |  |  |
| --- | --- | --- |
| **Number of Symbols** | **Time(1st Query)** | **Time(2nd Query)** |
| 1,000,000 | 3 ms | <0ms |
| 10,000,000 | 96 ms | <0ms |
| 100,000,000 | 1021 ms | <0ms |

**Table 2 – With Caching**

## Dynamic Parameters

As previously mentioned in the section "Simple Parameters", Tableau parameters are limited to static values, and a single select option when placed in a view. However, there are a number of ways to make parameters smarter which can increase their usability and flexibility. Below two such methods are described, including "Predefined parameters" and "Parameters in Calculated Fields".

### Predefining parameter options in a q function

From the previous example, the input parameter, Category, is limited to selecting only one value at a time. This can be made more flexible by defining a range of acceptable values in the function. For example, if the user wanted the option of returning all categories, they could input the value "all" in the Category field. In the example below, when the input value equals all, the lookup searches for all categories including ÈQ`CORP`GOV".

q) func:{[mydate;mycategory]

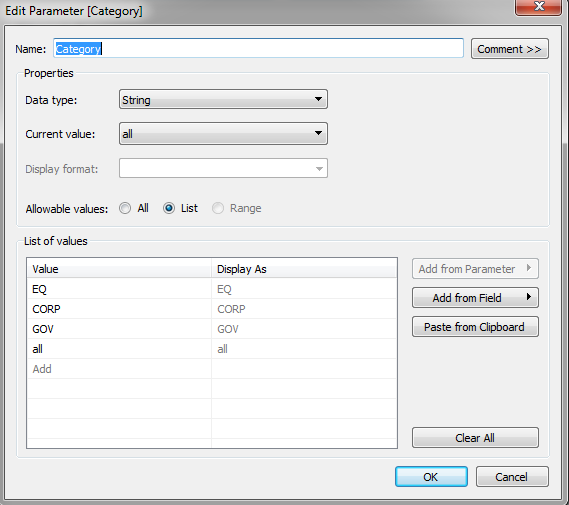
$[mycategory=`all;

         select from tab where date in mydate;

         select from tab where date in mydate, category in mycategory]

  };

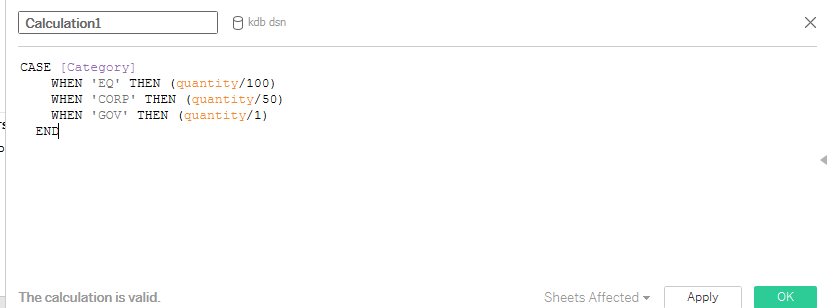
This can then be added to the list of predefined values in the definition of Category in Tableau.



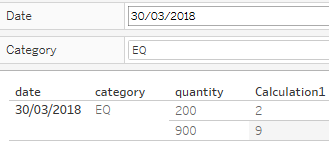
### Using parameters with calculated fields

Using Parameters in conjunction with Tableau's calculated field functionality can be a convenient and flexible tool in calculations as well as graphical representation. This is useful when the output the user wants to see is dependent on an input parameter and a field needs to be adjusted accordingly.

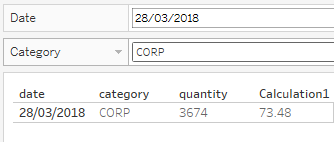
For example, in the user-defined Calculation1 logic below, the quantity field will be divided by a different amount depending on the input Category value.



Below is illustrated the sample output in the case where the user enters a Category value of EQ.



In contrast, when the user selects CORP the calculated field will be divided by 50.



## Multiple Data Sources

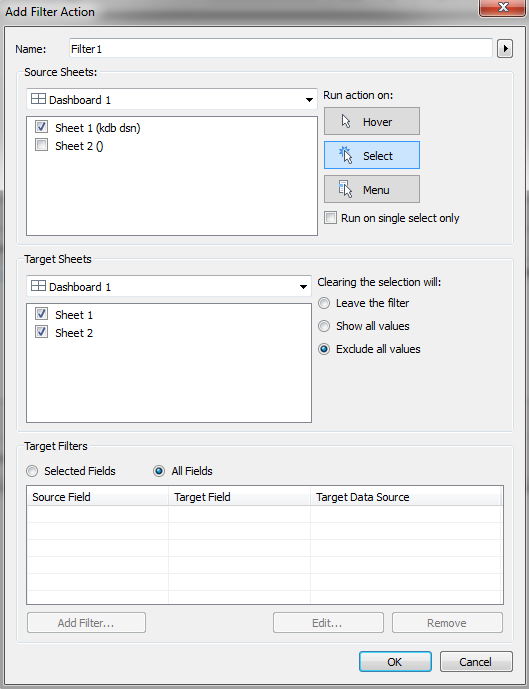
Kdb+ is very performant at joining data sets using the various join functions and can easily do so in memory at the gateway level. However, it is also worth noting that it is possible to join two or more different datasets In Tableau if they share a common dimension or key. This can be useful when it is desirable to join certain datasets for reporting purposes only.

Tableau maintains connections to multiple data sources via a number of open live connections to a q instance. This functionality makes it possible to use the results from one data source to subsequently filter another. So far, in this paper, the examples have described functionality using only one data source. For the rest of this section working with multiple data sources and joining them in Tableau will be explored.

One of the first things to note is that fields from different data sources can be included on the same Worksheet, provided the sources are mapped to each other. In Tableau, fields from different data sources can be mapped to each other even if they have a different name, so long as they are the same datatype. This can be controlled and edited in Data –> Edit Relationships.

### Actions

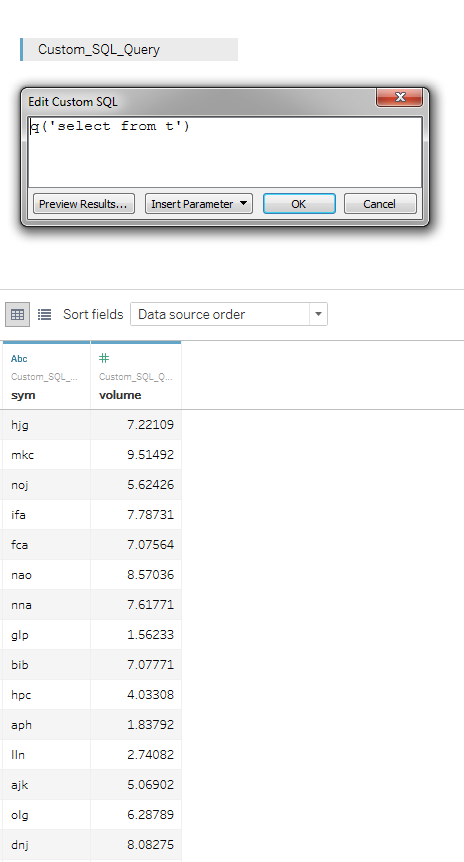
Once a dashboard is created, the Filters are controlled in Dashboard -> Actions. When setting up Actions for kdb+ data sources, it is important to note how the selection is cleared. For large datasets, it is recommended that you select the action 'Exclude all values'. This feature prevents data from being displayed in Sheet 2 until data is first selected in Sheet 1. This has a very significant effect on performance as it means Tableau only builds dimensions for views within the dataset that has been filtered.

`

The following example demonstrates how much of an improvement on performance this feature can have. Once a table 't' is defined and subsequently called from Tableau, the next step is to create a Dashboard.

q) N:10000000

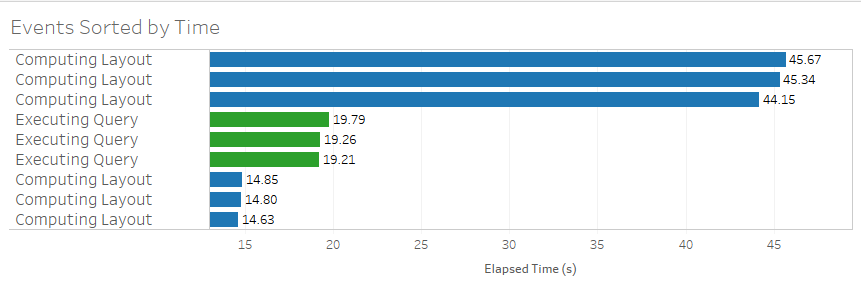
q) t:([] sym:N?`3;volume:N?10.0)



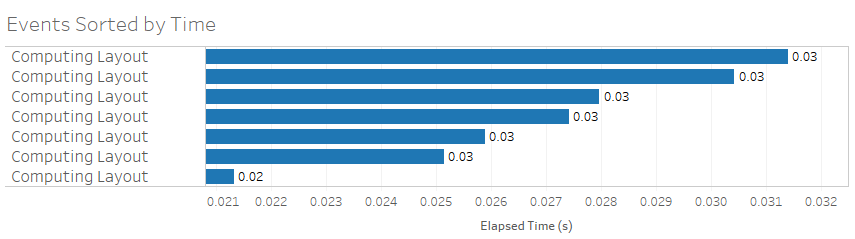
Step by step instructions on how to build the dashboard shown below and performance tests can be found in Appendix A.



**Action Selection = Show all values'**



**Action Selection = 'Exclude all values'**



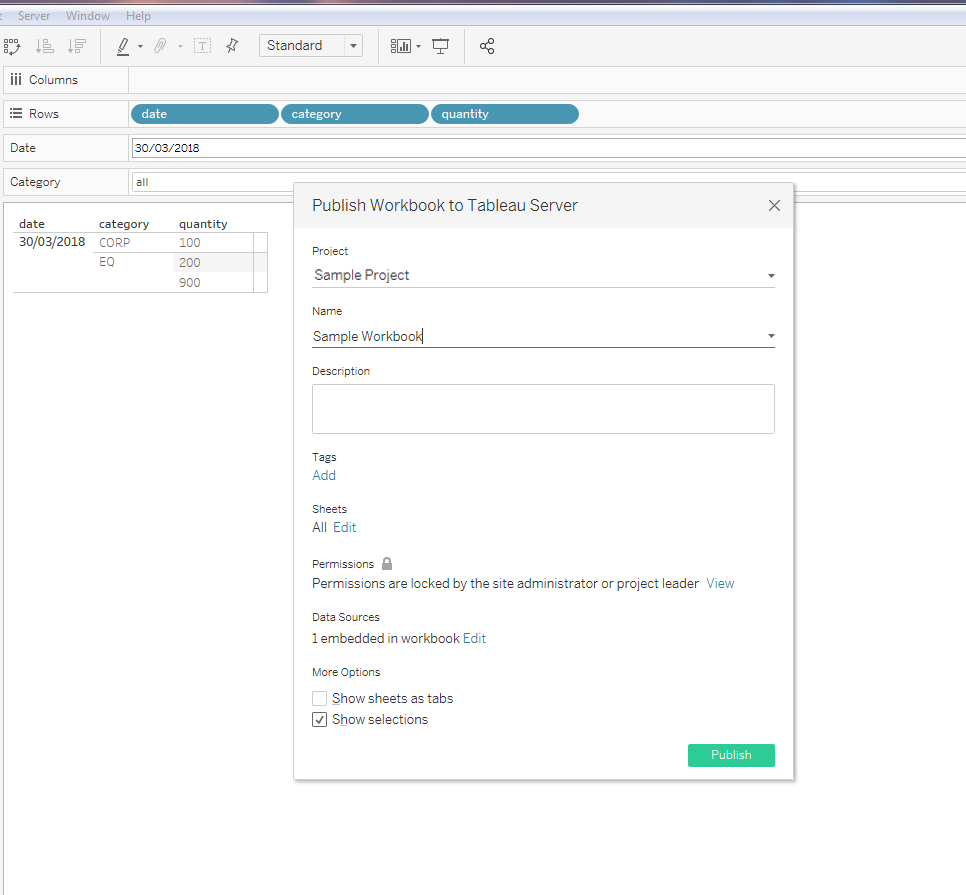
It is clear that by using the 'Exclude all values' option there is a clear performance improvement. Computing time reduces from ~45secs per select/deselect down to ~0.3ms. Also, when using 'Exclude all values' there is no Executing Query time.

Leveraging this feature can be hugely useful when working with kdb+ and Tableau where the volume of datasets can be very large.

I don’t think this section is needed – its just a promotion of what they have and not about ODBC integration

# Publishing to Tableau Server

As mentioned in an earlier section of this paper, to share workbooks between Tableau Desktop and Tableau Server you can publish the former to the latter. There is detailed documentation and instructions provided by Tableau on the general publishing procedure which involves publishing from within an already open workbook <https://onlinehelp.tableau.com/current/pro/desktop/en-us/publish_workbooks_howto.html>.



This is not an ideal way to publish Workbooks that are connected to a kdb+ database because of the fact that connection details are stored within the Workbook itself. Take the following scenario:

* A Workbook has been developed in Tableau Desktop and is ready to share to the Testing partition in Tableau Server.
* Throughout development, a Development DSN has been used. But the Workbook needs to be published to a UAT DSN.
* So the DSN details need to be changed to the UAT DSN before publication.
* The Workbook again needs to be promoted, this time to the Production partition.
* The Workbook must be reopened, and the DSN details changed to the Production DSN before finally promoting to Production.

This is a manual process, and as such can be error-prone. For kdb+ connections, it is recommended to use the tabcmd command line utility which, among other things, enables the user to publish to Tableau Server from the command line. This utility allows the user to deploy sheets programmatically, streamlining the process hugely. It also means that as part of the deploy procedure, the workbook can be edited by a script before publishing via tabcmd. This means you can do some efficient things like:

* Wipe out the connection details that are automatically embedded in the workbook
* Edit which DSN to point to e.g. DEV, UAT, QA, Prod
* Edit which Tableau Server to publish e.g. tableau.net or tableau-uat.net
* Edit which Tableau Environment to publish to e.g. Development, Testing or Production
* Edit the Tableau Project name

Using tabcmd and a script to edit the workbook can be a hugely effective way to make the publishing process much smoother when connecting to kdb+, especially when scaling use cases and looking to publish across multiple environments and DSN's.

# Conclusion

Kdb+ is well known as the world’s fastest time series database and Kx's own visualization tool, Dashboards for Kx, is indeed optimized for this technology. However, kdb+ is still flexible enough to connect with a range of different visualization tools from third-parties. Demonstrated in this whitepaper is how to connect to one such visualization tool, Tableau, by means of the ODBC driver provided by Kx.

You will have seen how to set up a connection between kdb+ and Tableau using ODBC with detailed instructions on how best to use Tableau's query functionality, datatypes, parameters and filters with the q language.

The key takeaways include how kdb+’s caching feature may be used to improve performance by optimizing repeated queries from Tableau. Further to this, improving performance when using filters with multiple data sources was explored. The methods provided in this paper demonstrated that kdb+ and Tableau can be combined in an efficient way despite limitations that occur when combing the two technologies.

If the question is “Is Tableau the best, most performant visualization tool when combining with kdb+?”, the answer is “No. There are other visualization tools out there that are better tailored to the kdb+ technology, for example Kxs’ visualization solution “Dashboards for Kx.” But if the question is “Can you combine these two technologies in an effective way?” the answer is “Yes”, and by applying the functionality described in this paper to work around many of the limitations described, it is clear it is possible to have an effective Tableau visualization front-end connected to a kdb+ backend.

# Appendix A

1. Create 'Sheet 1'
   1. Drag and drop 'sym' to Columns.
   2. Drag and drop 'Number of Records' to Rows.
   3. Drag and drop 'volume' to the Marks pane on colour. Right click and make it Discrete.
2. Create 'Sheet 2'
   1. Drag and drop 'sym' to Rows.
   2. Drag and drop 'volume' to Rows. Right click and make this both a Dimension and Discrete. This means every row will be displayed and not just the summed value.
3. Create 'Dashboard 1'
   1. Drag 'Sheet 1' onto the top of the Dashboard.
   2. Drag 'Sheet 2' onto the bottom of the Dashboard.
4. Make 'Sheet 1' a filter for 'Sheet 2' on the Dashboard.
   1. Hover over 'Sheet 1' and on the top right hand side select the middle icon that looks like a filter.
5. Testing Performance with default filter selection
   1. Select Help > Settings and Performance > Start Performance Recording
   2. Select and deselect some of the bars in the top graph. You should notice much slower performance on deselect.
   3. Select Help > Settings and Performance > Stop Performance Recording

A performance workbook will then popup and you can analyse the performance.

1. Testing Performance with selection='Exclude all values'
   1. Select Dashboard > Actions > Edit > Select 'Exclude all values'.
   2. Repeat step 5

A second performance workbook will popup which can be compared with the previous one to analyse performance.

1. The latest version of the ODBC can be found at <https://github.com/KxSystems/kdb/blob/master/c/qodbc3.zip> [↑](#footnote-ref-1)