Best Practices for Designing Efficient Tableau Workbooks

Third Edition

Alan Eldridge

Tableau Software

12 July 2015

Foreword

Once again, I would like to acknowledge that this document is a distillation of materials written by many authors. All I have done is to bring it together into a single document and try to apply some structure. Many people reading this will recognise their fingerprints across sections (in fact some will recognise entire swathes of text). To all of you I give thanks because without your excellent work and lack of copyright infringement claims I'd still be researching and writing.

This document has been updated to reflect the capabilities of Tableau 9.0. Future releases of Tableau will provide new features and capabilities that may change some of these guidelines.

Contents

What is efficiency?

There are several factors that make a workbook "efficient". Some of these factors are technical and some more user-focused but in general an efficient workbook is:

- A workbook that takes advantage of the "principles of visual analysis" to effectively communicate the message of the author and the data, possibly by engaging the user in an interactive experience.
- A workbook that responds in a timely fashion. This can be a somewhat subjective measure, but in general we would want the workbook to provide an initial display of information and to respond to user interactions within a couple of seconds.

The first section of this document focuses on the first point and is mostly about workbook design. The second section then focuses on the factors that affect workbook performance. As you will discover there are many factors that contribute to workbook performance, including:

- the data connection and underlying data source;
- the query;
- the calculations;
- the visualisation design;
- some differences between Tableau Desktop and Server;
- other environmental factors such as hardware configuration and capacity.

Why do we care?

In our extensive experience, most performance problems that customers encounter are workbook related. If we can fix these – or better yet, through education prevent them in the first place – then we can fix the problems.

If you are working with small data volumes then many of these recommendations are not critical. You can just brute-force your way through the problem. However, when you are dealing with hundreds of millions of records the effect of poor workbook design is amplified and you must give more thought to the guidance in this whitepaper.

Of course, practice makes perfect and following these guidelines for all workbooks is recommended. Remember, your design is not complete until you have tested your workbook over the expected production data volumes.

The laws of physics

Before we dive into the technical details of how various features affect the performance of workbooks, there are three basic tenets that will help you author efficient dashboards and views:

If it isn't fast in the data source, it won't be fast in Tableau.

If your Tableau workbook is based on a slow running query then your workbook will also be slow. In the following sections we will identify tuning tips for your databases to help improve the time it takes for queries to run. Additionally, we'll discuss how Tableau's fast data engine can be used to improve query performance.

If it isn't fast in Tableau Desktop, it won't be fast in Tableau Server.

A workbook that performs slowly in Tableau Desktop won't get any faster by publishing it to Tableau Server. In general, workbooks will perform slightly slower on Tableau Server because

- there are multiple users all sharing the server resources to generate workbooks simultaneously; and
- the server has to do the work to render the dashboards and charts rather than this being done on the client workstation.

You should invest your initial efforts tuning your workbook in Tableau Desktop before you start looking to tune the performance in Tableau Server.

The exception to this rule is if Tableau Desktop is encountering resource limits that aren't present on the server – e.g. your PC does not have enough RAM to support the data volume you are analysing. Some users encounter slow performance or even "out of memory" errors when working with a data set on their low-spec, 2GB RAM workstation, but find performance of the published workbook to be acceptably fast because the server has far more memory and processing power.

> *A quick note at this point – throughout this document we refer to Tableau Server but in most places the guidance is also appropriate for Tableau Online if you prefer to use our hosted solution over an on-premises deployment. The obvious exceptions are points on tweaking/tuning server configuration parameters and on installing/updating software on the server tier – in the SaaS world these are being looked after for you!*

Everything in moderation!

As with all things in life, too much of a good thing can be bad. Don't try to put absolutely everything into a single, monolithic workbook. While a Tableau workbook *can* have 50 dashboards, each with 20 chart objects, talking to 50 different data sources, it will almost certainly perform slowly.

If you find yourself with a workbook like this, consider breaking it into several separate files. This is easy to do – since Tableau 8.1 you can simply copy dashboards between workbooks and Tableau will bring all the associated worksheets and data sources. If your dashboards are overly complex, consider simplifying them and using interactions to guide the end users from view to view. Remember, we don't price our software by the document so feel free to spread the data out a little.

What is Tableau good for?

At Tableau Software, we seek to change how people view, interact with, and understand data. As a result, we do not attempt to deliver the same kind of experience as traditional enterprise BI platforms. Tableau is at its best when used to create workbooks that are:

- **Visual** – there is a mountain of evidence that shows the most effective way for humans to understand large, complex sets of data is through visual representation. Tableau's default behaviour is to present data using charts, diagrams and dashboards. Tables and crosstabs have their place (and are supported) and we will talk more on how to best use them later.

- **Interactive** – Tableau documents are primarily designed for interactive delivery to users, either on their desktops, over the web or on a mobile device. Unlike other BI tools that primarily produce print-focused output (either to actual paper or to a document such as a PDF), the focus is on creating rich, interactive experiences that allow users to explore data and be guided through business questions.
- **Iterative** – discovery is an inherently cyclical process. Tableau is designed to speed the cycle from question to insight to question so that users can quickly develop a hypothesis, test it with available data, revise that hypothesis, test it again, and so on.
- **Fast** – historically the BI process has been slow. Slow to install and configure software, slow to make data available for analysis and slow to design and implement documents, reports, dashboards, etc. Tableau allows users to install, connect and develop documents faster than ever before – in many cases reducing the time to produce an answer from months or weeks to hours or minutes.
- **Simple** – traditional enterprise BI tools are often beyond the capability of most business users, either through cost or complexity. In many cases, users need the assistance of IT or a power user to help create the queries and documents they want. Tableau provides an intuitive interface for non-technical users to query and analyse complex data without needing them to become database or spreadsheet experts.
- **Beautiful** – they say beauty is in the eye of the beholder, but when it comes to visual communication there are best practices to be followed. Through features such as “Show Me”, Tableau guides non-technical users to create effective, understandable charts based on the data being used.
- **Ubiquitous** – increasingly, users are no longer creating documents for a single delivery platform. Users need to view and interact with data on their desktops, over the web, on mobile devices, embedded in other applications and documents, and more. Tableau allows a single document to be published and then used across all these platforms without any porting or redesign.

What is Tableau not really good for?

Tableau is a rich and powerful tool but it’s important to understand at the start that there are some problems for which it is probably not the best solution. This doesn’t mean it can’t do these things – Tableau can be coaxed to perform many tasks that were not in its original design specification. What we mean is that these are not the types of problems Tableau was developed to solve and therefore if you pursue them the effort/reward ratio will likely be unfavourable and the resulting solution may perform poorly or inflexibly.

We suggest you consider revisiting your requirements or consider another approach if:

- You need a document that has been designed for paper, not the screen. By this, we mean if you have a need to control complex page layouts, need features such as page, section and group headers/footers, or need precise WYSIWYG formatting. Tableau can produce multi-page reports but they lack the level of format control that is available in dedicated, banded-style reporting tools.

- You need a complex push-delivery mechanism for documents with personalisation (also called “bursting”) sent via multiple delivery modes. Tableau can be used to create push-delivery systems but this is not a native feature of Tableau. It requires development of a custom solution built around the TABCMD utility, or the introduction of 3rd party solutions such as *Push Intelligence from Metric Insights*[1]. Tableau Server includes the concept of report subscriptions but this is a per-user pull model vs. report bursting.
- The <u>primary</u> use case for the reader is to export the data to another format (often a CSV or Excel file). This often means a tabular report with many rows of detailed data. To be clear, Tableau does allow users to export data from a view or dashboard to Excel – either at a summary or detail level. However, when the <u>primary</u> use case is to export it means this is an ersatz extract-transform-load (ETL) process. There are much more efficient solutions than a reporting tool to achieve this.
- You need highly complex, crosstab-style documents that perhaps mirror existing spreadsheet reports with complex sub-totalling, cross-referencing, etc. Common examples here are financial reports such as P&L, balance sheet, etc. Additionally, there may be the need for scenario modelling, what-if analysis and even write-back of assumption data. If the underlying granular data is not available or if the report logic is based on “cell references” rather than rolling up records to totals then it might be appropriate to continue using a spreadsheet for this style of report.

[1] Push Intelligence for Tableau - http://bit.ly/1HACxul

Is it the data connection/data source?

One of the powerful features of Tableau is its ability to connect to data across many different platforms. Broadly speaking these platforms can be characterised as one of the following:

- File-based data sources – such as Excel and CSV;
- Relational database data sources – such as Oracle, Teradata and SQL Server, as well as specialised analytic appliances such as HP Vertica, IBM Netezza, etc;
- OLAP data sources – such as Microsoft Analysis Services and Oracle Essbase;
- “Big data” data sources – such as Hadoop;
- Cloud-based data sources – such as Salesforce, Google, etc.

Each type of data source has its own set of advantages and disadvantages, and is treated uniquely.

Note that Tableau Desktop is supported on both Windows and Mac OS X and the set of supported data sources on Mac is not the same as on Windows. Minimising the differences between the platforms is something Tableau will work towards but today there are some data sources that are only supported on one platform.

General advice

Use native drivers

As of version 9, Tableau supports native connectivity to over 40 different data sources. This means Tableau has implemented techniques, capabilities and optimizations specific to these data sources. Engineering and testing activities for these connections ensure they are the most robust Tableau has to offer.

Tableau also supports general-purpose ODBC for accessing data sources beyond the list of native connectors. As a publicly defined standard, many database vendors make ODBC drivers available for their databases and Tableau can also use these ODBC drivers to connect to data. There can be differences in how each database vendor interprets or implements capabilities of the ODBC standard. In some cases Tableau will recommend or require you to create a data extract to continue working with a particular driver. There will also be some ODBC drivers and databases that Tableau is unable to connect to.

If there is a native driver for the data source you are querying you should use this over the ODBC connections as it will generally provide better performance.

Test as close to the data as possible

As stated earlier, a general principal is that if a data source performs queries slowly then the experience in Tableau will be slow. A good way to test the raw performance of the data source is to (if possible) install Tableau Desktop on the machine where the data source resides and to run some queries. This will eliminate factors such as network bandwidth and latency from the performance and allow you to better understand the raw performance of the query in the data source. Additionally, using the *localhost* name for the data source instead of the DNS name can help determine if environmental factors such as slow name resolution or proxy servers are adding to the poor performance.

Files

This category covers all file-based data formats – text files such as CSV, Excel spreadsheets and MS Access being the most common, however this also includes data files from statistical platforms SPSS, SAS and R. Business users often use data in this format because it is a common way to get data out of "governed" data sets – either by running reports or performing a query extract.

In general it is best practice to import file-based data sources into the Tableau fast data engine. This will make queries perform much faster and also results in a much smaller file to store the data values. However, if the file is small or if you need a live connection to the file to reflect changing data you can connect live.

Shadow extracts

In Tableau 9 and later when you connect to non-legacy Excel/text or statistical files, Tableau transparently creates an extract file as part of the connection process. This is called a shadow extract and it makes working with the data much faster than if you were to directly query the file.

You may notice that the first time you use a large file it can take several seconds to load the data preview pane. This is because Tableau is extracting the data from the file and writing it to a shadow extract file. By default these files are created in `C:\Users\<username>\AppData\Local\Tableau\Caching\TemporaryExtracts` with a hashed name based on the pathname and date last modified of the data file. Tableau keeps shadow extracts for the five most-recently used file data sources in this directory, deleting the least recently used file when a new one is created. If you subsequently reuse a file that has a shadow extract Tableau simply opens the extract file and the data preview appears almost instantly.

Although shadow extract files contain underlying data and other information similar to the standard Tableau extract, shadow extract files are saved in a different format (with a .ttde extension), which means that they cannot be used the same way Tableau extracts are.

Legacy connectors for Excel and text files

Prior to Tableau 8.2 connections to Excel and text files made use of Microsoft's JET data engine driver. In Tableau 8.2 and later we have introduced a native driver for these files that provides better performance and works with larger, more complex files. However there are some situations in which you might prefer to use the legacy drivers – e.g. if you want to use custom SQL to union data together from multiple text files. In these scenarios users have the option to revert back to the legacy JET driver. Access to MS Access files still uses the JET driver.

A detailed listing of the differences between the two drivers can be found here:

http://tabsoft.co/1HACvmj

Note that the JET drivers are not available on Mac OS and therefore Tableau Desktop for Mac does not support reading MS Access files, nor does it provide the legacy connection option for Excel and text files.

Relational

Relational data sources are the most common form of data source for Tableau users and Tableau provides native drivers for a wide selection of platforms. These can be row or column based, personal or enterprise, and accessed via native drivers or generic ODBC. This category technically also includes Map-Reduce data sources as they are accessed through SQL access layers like Hive or Impala, however we will discuss them in more detail in the "big data" section below.

There are many internal factors that impact query speed in a relational database systems (RDBMS). Changing or tuning these will usually require assistance from your DBA, but can yield significant performance improvements.

Row-based vs. column-based

RDBMS systems come in two main flavours – row-based or columns-based. Row-based storage layouts are well-suited for OLTP-like workloads which are more heavily loaded with interactive transactions. Column-based storage layouts are well-suited for analytic workloads (e.g., data warehouses) which typically involve highly complex queries over large sets of data.

Today, many high performing analytic solutions are based on column-based RDBMS and you may find your queries perform faster if you use such a solution. Examples of column-based databases supported by Tableau are Actian Vector, Amazon Redshift, HP Vertica, IBM Netezza, ParAccel, Pivotal Greenplum, SAP HANA and SAP Sybase IQ.

Indexes

Correct indexing on your database is essential for good query performance:

- Make certain you have indexes on columns that are part of table joins.
- Make certain you have indexes on columns used in filters.
- Be aware that using discrete date filters in some databases can cause queries to not use indexes on date and datetime columns. We'll discuss this further in the filter section, but using a range date filter will ensure the date index is used. For example, instead of using `YEAR([DateDim])=2010` express the filter as `[DateDim] >= #2010-01-01# and [DateDim] <= #2010-12-31#).`
- Ensure you have statistics enabled on your data to allow the query optimiser to create high-quality query plans.

Many DBMS environments have management tools that will look at a query and recommend indexes that would help.

NULLS

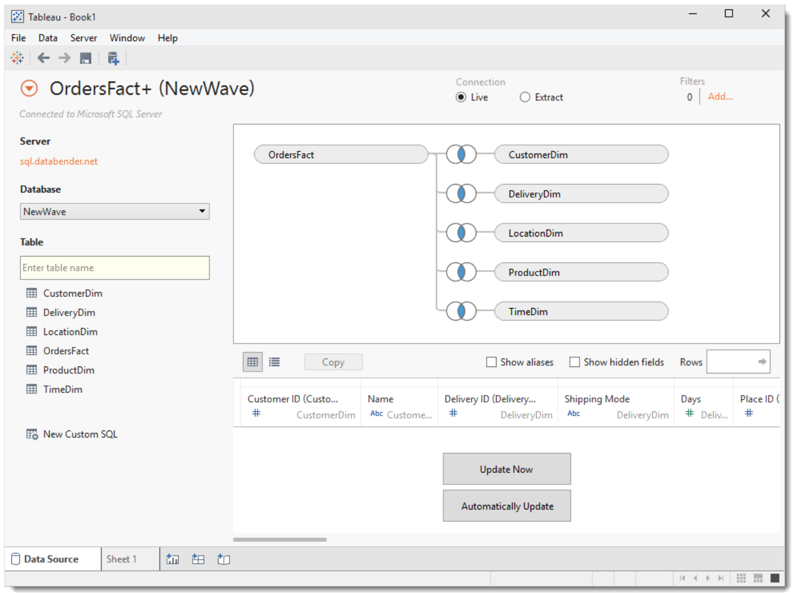
Having NULL values in dimension columns can reduce the effectiveness of indexes in some databases. Consider an example where you have a dimension with incomplete data and some records contain NULL values. If you want to exclude those NULL records the NOT ISNULL filter can (in some cases) cause a full table scan which is much slower than an index scan.

Where possible, define your dimension columns as NOT NULL.

Referential Integrity

When you join multiple tables in a data source Tableau has a nifty (and generally invisible to the user) feature called “join culling”. Since joins cost time and resources to process on the database server, we really don’t want to enumerate every join that we declared in our data source all the time. Join culling allows us to query only the relevant tables instead of all tables defined in your join.

Consider the following scenario where we have joined multiple tables in a small star schema:



With join culling, double-clicking on the Sales measure generates the following query:

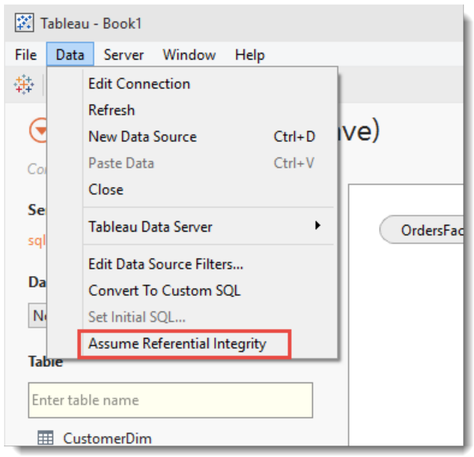
```
SELECT SUM([OrdersFact].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[OrdersFact] [OrdersFact]
GROUP BY ()
```

Without it, a far less efficient query is generated:

```
SELECT SUM([OrdersFact].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[OrdersFact] [OrdersFact]
  INNER JOIN [dbo].[CustomerDim] [CustomerDim]
    ON ([OrdersFact].[Customer ID] = [CustomerDim].[Customer ID])
  INNER JOIN [dbo].[DeliveryDim] [DeliveryDim]
    ON ([OrdersFact].[Delivery ID] = [DeliveryDim].[Delivery ID])
  INNER JOIN [dbo].[LocationDim] [LocationDim]
    ON ([OrdersFact].[Place ID] = [LocationDim].[Place ID])
  INNER JOIN [dbo].[ProductDim] [ProductDim]
    ON ([OrdersFact].[Product ID] = [ProductDim].[Product ID])
  INNER JOIN [dbo].[TimeDim] [TimeDim]
    ON ([OrdersFact].[Date ID] = [TimeDim].[Date ID])
GROUP BY ()
```

All the dimension tables must be joined in order to ensure that correct measure sums are calculated from the start. For example, if our fact table contained data for 2008-2012 but the time dimension table only had values for 2010-2012, the result SUM([Sales]) would potentially change depending on whether the time table is included.

Prior to Tableau 8.1, join culling only occurs if referential integrity rules are enforced in the source DBMS - sometimes referred to as “hard” referential integrity. However many customers have data sources where referential integrity is enforced either at the application layer or through an ETL process - this is referred to as “soft” referential integrity. In Tableau 8.1 and later, users can tell Tableau that soft referential integrity is in place and that join culling can be safely used.



Note that while Tableau can use either hard or soft referential integrity it is often much better to use hard referential integrity because the database can do join culling too. For more information, see the following series of articles by Russell Christopher on his Tableau Love blog:

http://bit.ly/1HACmPV
http://bit.ly/1HACqPn

Partitioning

Partitioning a database improves performance by splitting a large table into smaller, individual tables (called partitions or shards). This means queries can run faster because there is less data to scan and/or there are more drives to service the IO. Partitioning is a recommended strategy for large data volumes and is transparent to Tableau.

Partitioning works well for Tableau if it is done across a dimension – e.g. time, region, category, etc. – that is commonly filtered so that individual queries only have to read records within a single partition.

Be aware that for some databases, “range of date” filters (not discrete filters) are necessary to ensure the partition indexes are correctly used – otherwise a full table scan can result with extremely poor performance.

Temp Tables

There are many operations in Tableau that can result in the use of temp tables – e.g. creating ad-hoc groups and sets, applying context filters, and performing data blending. It is recommended that you grant your users permission to create and drop temp tables, and ensure your environment has sufficient spool space for their queries they are running.

OLAP

Tableau supports several OLAP data sources:

- Microsoft Analysis Services
- Microsoft PowerPivot (both PowerPivot for Excel and PowerPivot for SharePoint)
- Oracle Essbase
- SAP BW
- Teradata OLAP

There are functional differences when connecting to OLAP versus relational due to the underlying language differences between MDX/DAX and SQL. The key points to keep in mind are that both have the same user interface in Tableau, the same visualisations, and the same expression language for calculated measures. The differences are mostly to do with metadata (how and where it is defined), filtering, how totals and aggregations work and how the data source can be used in data blending.

More details on the differences using Tableau over relational data sources vs. OLAP data sources can be found in the following knowledge base article:

http://tabsoft.co/1HAF47P

SAP BW data extracts

Since Tableau 8.1 you can extract data from SAP BW cubes into Tableau's data engine (note – this requires a special keycode that can be obtained from Tableau). Tableau retrieves leaf level nodes (not drill-through level data) and makes them into a relational data source. Since multidimensional to relational transformation does not preserve all cube structures, switching back and forth between extract and live connection freely without impacting the state of your visualisation is not supported for cube extracts. You will need to make your choice before you start building up your viz. However you don't have to decide on everything upfront. You can switch between alias options (key, long name etc.) after the extraction.

More detail on creating SAP BW extracts can be found here:

http://tabsoft.co/1SuYf9d

Big Data

Big data is a heavily overloaded term in the world of data analysis, however in this document we are particularly using it to refer to platforms that are based on Hadoop. In Tableau 9.0, there are four supported Hadoop distributions that support Hive and/or Impala connections:

- Amazon EMR
  - HiveServer
  - HiveServer2
  - Impala
- Cloudera Hadoop
  - HiveServer
  - HiveServer2

    - Impala
- Hortonworks Hadoop Hive
    - HiveServer
    - HiveServer2
    - Hortonworks Hadoop Hive
- MaPR Hadoop Hive
    - HiveServer
    - HiveServer2
- Spark SQL
    - SharkServer
    - SharkServer2
    - SparkThriftServer

Hive acts as a SQL-Hadoop translation layer, translating the query into MapReduce which is then run over HDFS data. Impala executes the SQL statement directly on HDFS data (bypassing the need for MapReduce). Tableau also supports Spark SQL, an open source processing engine for big data that can perform up to 100x faster than MapReduce by running in-memory rather than on-disk.

Impala is generally much faster than Hive and Spark is proving to be faster still.

Even with these additional components, Hadoop is often not sufficiently responsive for analytical queries like Tableau creates. Tableau data extracts are often used to improve query response times - more information on extracts and how they can be used with “big data” is discussed later.

Further details for improving performance against Hadoop data sources can be found here:

> http://tabsoft.co/1BkFo62

Cloud

Tableau currently supports the following cloud data sources:

- Salesforce.com
- Google Analytics
- oData (including Windows Azure Marketplace DataMarket)

This first group of sources read a set of data records from a web service and load them into a Tableau data extract file. “Connect live” is not an option for these data sources, however the extract file can be refreshed to update the data it contains. Using Tableau Server, this update process can be automated and scheduled.

- Amazon Redshift
- Amazon RDS
- Google BigQuery
- Microsoft Azure SQL Data Warehouse

While these are also cloud data sources they operate like a relational data source and allow both live connections and extracts. We will not cover them further here (refer to the relational data source section above) other than to point out you will generally want to keep these as live connections to avoid transferring large volumes of data from the cloud.

Salesforce

When connecting to Salesforce the following limitations of the connector need to be considered:

There is no “connect live” option

There are several reasons we chose to go with extract-only instead of a live connection, including:

- Performance – live analytic queries against Salesforce are (in general) slow.
- API quotas - Hitting Salesforce live too often can cause an account to be suspended if the daily quota is hit. By extracting, we make efficient use of APIs to minimize the number of API calls required and avoid hitting limits. To maintain optimum performance and ensure that the Force.com API is available to all of their customers, Salesforce.com balances transaction loads by imposing two types of limits: Concurrent API Request Limits (http://sforce.co/1f19cQa) and Total API Request Limits (http://sforce.co/1f19kiH).

The initial extract can be very slow

The first time you extract the data out of Salesforce, it could take a while based on table size, force.com load, etc. This is because objects are downloaded in their entirety.

Join options are limited

When choosing the multiple table option, be aware that you can only join objects on their PK/FK keys (left and inner join only).

You cannot pre-filter data

There is no ability to pre-filter the data through the Salesforce connector. If this requirement is critical for you, you could use a third party Salesforce ODBC driver (for example from Simba, or DataDirect) that supports live connections. You could then take an extract against this connection.

DBAmp also provide a solution which allows you to dump Salesforce data into SQL Server database. You could then connect to Tableau via the SQL Server connector.

Formula columns cannot be migrated over

If you have calculated fields, you will need to recreate them in Tableau after you extract the data.

There is a 10k character limit on queries

The Force.com API restricts queries to 10,000 total characters. If you are connecting to one or more tables that are very wide (lots of columns with potentially long column names), you may hit that limit when trying to create an extract. In these cases, you should select fewer columns to reduce the size of the query. In some cases, Salesforce.com may be able to increase this query limit for your company. Contact your Salesforce administrator to learn more.

Google Analytics

Google Analytics (GA) will sample your data when a report includes a large number of dimensions or a large amount of data. If your data for a particular web property within a given date range exceeds (for a regular GA account) 50,000 visits, GA will aggregate the results and return a sample set of that data. When GA has returned a sample set of that data to Tableau, Tableau will display the following message on the bottom right-corner of a view:

“Google Analytics returned sampled data. Sampling occurs when the connection includes a large number of dimensions or a large amount of data. Refer to the Google Analytics documentation to learn more about how sampling affects your report results.”

It is important to know when your data is being sampled because aggregating certain sample sets of data can cause highly skewed and inaccurate inferences. For example, suppose you aggregate a sample set of data that describes an unusual category of your data. Inferences on the aggregated sample set may be skewed because there are an insufficient number of samples contained in that category. To build GA views that allow you to make accurate inferences about your data, ensure that you have a large enough sample within the category that you are making inferences about. The recommended minimum sample size is 30.

For information about adjusting the GA sample size and more information about GA sampling, refer to the GA documentation:

http://bit.ly/1BkFoTG

To avoid sampling, there are two approaches to take:

- Run multiple GA reports at the session or hit level to break the data into unsampled chunks. You would then download the data to an Excel file and use Tableau’s extract engine to “add data from data source…” to reassemble the data into a single dataset.
- Upgrade your GA to a Premium account – this increases the number of records that can be included in a report. This will make it much easier to chunk the data down for analysis. Looking forward, Google has announced that they will be enabling GA Premium customers to export their session and hit level data to Google BigQuery for further analysis. This would be a much simpler approach as Tableau can connect directly to BigQuery.

Finally, note that the API that Tableau uses to query GA limits the query to a maximum of 7 dimensions and 10 measures.

Data Server

While not a data source in itself, another way to connect to data sources is via Tableau Server’s data server. The data server supports both live connections as well as data extracts and provides several advantages over standalone data connections:

- As the metadata is stored centrally on the Tableau Server it can be shared across multiple workbooks and across multiple authors/analysts. The workbooks retain a pointer to the centralised metadata definition and each time they are opened they check to see if there have been any changes made. If so, the user is prompted to update the copy embedded in the workbook. This means that changes to business logic only need to be made in one place and they can then be propagated across all dependent workbooks.

- If the data source is a data extract, it can be used across multiple workbooks. Without the data server, each workbook will contain its own local copy of the extract. This reduces the number of redundant copies which in turn reduces the required storage space on the server as well as any duplicate refresh processes.
- If the data source is a live connection, the drivers for the data source do not need to be installed on every analyst's PC, only the Tableau Server. The data server acts as a proxy for queries from Tableau Desktop.

One significant improvement in the data server in V9 is that it now supports the creation of temp tables. This capability will significantly improve the performance of queries where there are filters, sets and ad-hoc groups with large numbers of members. For example, the following query which is based on an ad-hoc group of many points on a scatter plot will now be generated on the data server, just like it is in Tableau Desktop:

```
SELECT [DimProduct].[ProductName] AS [ProductName],
  SUM([FactSales].[DiscountAmount]) AS [sum:DiscountAmount:ok],
  SUM([FactSales].[SalesAmount]) AS [sum:SalesAmount:ok],
  [t0].[Product Name (group)] AS [Product Name (group)]
FROM [dbo].[FactSales] [FactSales]
  INNER JOIN [dbo].[DimProduct] [DimProduct] ON ([FactSales].[ProductKey] =
[DimProduct].[ProductKey])
  INNER JOIN [#Tableau_5_3_Group] [t0] ON ([DimProduct].[ProductName] =
[t0].[ProductName])
GROUP BY [t0].[Product Name (group)],
  [DimProduct].[ProductName]
```

Extracts

So far we have discussed techniques for improving the performance of data connections where the data remains in the original format. We call these live data connections and in these cases we are dependent on the source data platform for both performance and functionality. In order to improve performance with live connections, it's often necessary to make changes to the data source and for many customers this is simply not possible.

An alternative available to all users is to leverage Tableau's fast data engine and to extract data from the source data system into a Tableau Data Extract. This is for most users the quickest and easiest way to significantly improve the performance of a workbook over any data source.

An extract is:

- A persistent cache of data that is written to disk and reproducible;
- A columnar data store – a format where the data has been optimised for analytic querying;
- Completely disconnected from the database during querying. In effect, the extract is a replacement for the live data connection;
- Refreshable, either by completely regenerating the extract or by incrementally adding rows of data to an existing extract;
- Architecture-aware – unlike most in-memory technologies it is not constrained by the amount of physical RAM available;

- Portable – extracts are stored as files so can be copied to a local hard drive and used when the user is not connected to the corporate network. They can also be used to embed data into packaged workbooks that are distributed for use with Tableau Reader;
- Often much faster than the underlying live data connection.

Tom Brown at The Information Lab has written an excellent article explaining several use cases where extracts provide benefit (make sure you also read the comments for additional examples from other users):

http://bit.ly/1F2iDnT

There is one important point to make about data extracts – they are not a replacement for a data warehouse, rather a complement. While they can be used to collect and aggregate data over time (i.e. incrementally add data according to a periodic cycle) this should be used as a tactical, rather than long term, solution. Incremental updates do not support update or delete actions to records that have already been processed – changing these requires a full reload of the extract.

Finally, extracts cannot be created over OLAP data sources such as SQL Server Analysis Services, or Oracle Essbase. The exception to this rule is that you can create extracts from SAP BW (see the relevant section above).

When to use extracts? When to use live connections?

Like everything, there's a time and a place for data extracts. The following are some scenarios where extracts may be beneficial:

- Slow query execution - if your source data system is slow to process the queries being generated by Tableau Desktop, creating an extract may be a simple way to improve performance. The extract data format is inherently designed to provide fast response to analytic queries so in this case you can think of the extract as a query acceleration cache. For some connection types this is a recommended best practice (e.g. large text files, custom SQL connections) and some sources will only work in this model (see the section on cloud data sources).
- Offline analysis - if you need to work with data while the original data source is not available (e.g. you are disconnected from the network while travelling or working from home). Data extracts are persisted as a file which can easily be copied to a portable device such as a laptop. It is a simple matter to switch back and forth between an extract and a live connection if you move on and off the network.
- Packaged workbooks for Tableau Reader/Online/Public - if you are planning to share your workbooks for other users to open them in Tableau Reader or if you are planning to publish them to Tableau Online or Public, you will need to embed the data into a packaged workbook file. Even if the workbook uses data sources that can also be embedded (i.e. file data sources) data extracts inherently provide a high level of data compression so the resulting packaged workbook is significantly smaller.
- Additional functionality - for some data sources (e.g. file data sources via the legacy JET driver) there are functions in Tableau Desktop that are not supported (e.g. median/count distinct/rank/percentile aggregations, set IN/OUT operations, etc.). Extracting the data is a simple way to enable these functions.

- Data security - if you wish to share a subset of the data from the source data system, you can create an extract and make that available to other users. You can limit the fields/columns you include as well as share aggregated data where you want users to see summary values but not the individual record-level data.

Extracts are very powerful, but they are not a silver bullet for all problems. There are some scenarios where using extracts might not be appropriate:

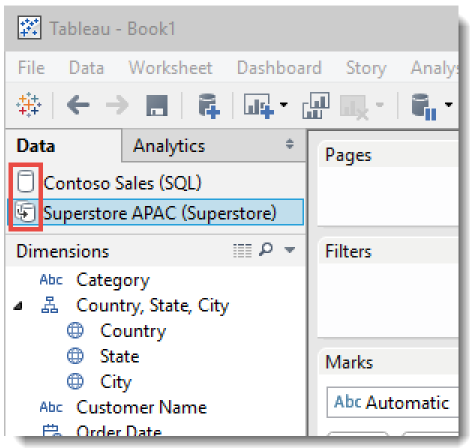
- Real-time data - because extracts are a point of time snapshot of data they would not be appropriate if you need real-time data in your analysis. It is possible to automatically refresh extracts using Tableau Server and many customers do this at intra-day frequencies but true real-time data access would require a live connection.
- Massive data - if the volume of data you need to work with is massive (the definition of "massive" will vary from user to user but generally it will be millions to billions of records) then extracting this may not be practical. The resulting extract file may be excessively large or the extract process may take many, many hours to complete. Note that there are a couple of exceptions to this guideline. If you have a massive source data set but you are going to work over a filtered, sampled and/or aggregated subset of this data, then using an extracts may actually be a great idea. Generally speaking the Tableau extract engine was designed to work well for up to a few hundred million records but this will be influenced by the shape and cardinality of your data.
- Pass-through RAWSQL functions - if your workbook uses pass-through functions these will not work with a data extract.
- Robust user-level security - if you have a requirement for robustly-enforced, user-level security then this needs to be implemented in the data source. If you have user-level filters applied at the workbook level then these can always be removed by a user allowing them access to all data in the extract. The exception to this guideline is if the extract has been published to Tableau Server with data source filters defined and other users are accessing this extract via the data server. Note - you will need to ensure that download permissions are revoked from users to ensure they cannot bypass the enforced filters.

Creating extracts in Desktop

In most cases the initial creation of an extract is done in Tableau Desktop and is very simple. After you have connected to your data, go to the Data menu and click "Extract Data" – then accept the defaults on the dialog box (although more on this later). Tableau will ask where you want to save the extract – choose any location to save the file, although Tableau will probably direct you towards 'My Tableau Repository | Datasources' which is just fine too!

Now wait for the extract to be created, how long you'll wait depends on the database technology being used, network speed, data volumes, etc. It is also dependent on the speed and capacity of your workstation as creating an extract is a memory and processor intensive activity.

You'll know it's done when the data source icon changes – it will have another database icon behind it, representing a 'copy', which is exactly what an extract is.



When you create an extract this way (via Tableau Desktop) the processing occurs on your workstation and so you will need to ensure it has sufficient capacity to complete the task. Extract creation uses all resource types – CPU, RAM, disk storage, network I/O – and processing large data volumes on a small PC can result in errors if any are exhausted. It is recommended that large extracts be done on a suitable workstation – fast CPU with multiple cores, lots of RAM, fast I/O, etc.

The extract creation process requires temp disk space to write working files – in Tableau 8.1 and earlier it could require up to the square of the resulting extract file (e.g. a 100MB extract may require several GB of temp space) but in Tableau 8.2 and later this has been reduced significantly and the process now only needs up to 2x the final extract file size. This working space is allocated in the directory specified by the TEMP environment variable (usually `C:\WINDOWS\TEMP` or `C:\Users\USERNAME\AppData\Local\Temp`). If this drive has insufficient space, point the environment variable to a larger location.

If it is impossible (or just impractical) to do an initial extract process on a workstation, the following workaround can be done to create an empty extract that is then published to Tableau Server. Create a calculated field that has `DateTrunc("minute", now())` in it. Then add it to the extract filters and exclude the single value it shows – be quick because you have a minute before this filter is no longer valid. If you need longer just make the publishing interval wider (e.g. round to 5 mins or 10 mins or an hour if you need). This will build an empty extract on your desktop. When you publish to server and trigger the refresh schedule, it will populate the full extract since the timestamp we excluded is not the same anymore.

Creating extracts using the data extract API

Tableau also provides an application programming interface (API) to enable developers to directly create a Tableau Data Extract (TDE) file. Developers can use this API to generate extracts from on-premises software and software-as-a-service. This API lets you systematically get data into Tableau when there is not a native connector to the data source you are using.

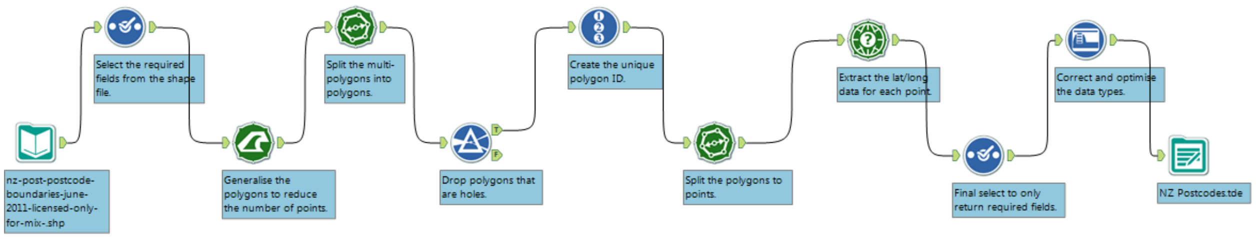
This API is available for developers in Python and C/C++/Java on both Windows and Linux. You can find more information on the API here:

http://tabsoft.co/1Sv1n55

Creating extracts using 3rd party tools

Many 3rd party tool developers have used the data extract API to add native TDE output to their applications. These applications include analytic platforms like Adobe Marketing Cloud as well as ETL tools like Alteryx and Informatica.

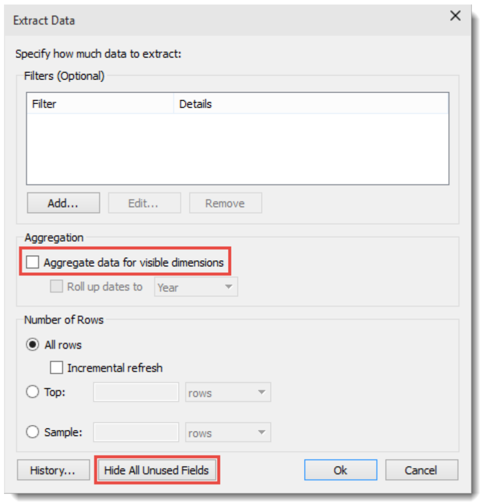
If you have complex data preparation requirements, tools like Alteryx and Informatica can be used to efficiently perform the ETL stages and then directly output the prepared data into a TDE file for use in Tableau Desktop.


nz-post-postcode-boundaries-june-2011-licensed-only-for-mix-.shp
Select the required fields from the shape file.
Generalise the polygons to reduce the number of points.
Split the multi-polygons into polygons.
Drop polygons that are holes.
Create the unique polygon ID.
Split the polygons to points.
Extract the lat/long data for each point.
Final select to only return required fields.
Correct and optimise the data types.
NZ Postcodes.tde

Aggregate extracts

Using an aggregate extract can always improve performance. Even if you're on Teradata or Vertica with huge amounts of data, extracting data can provide an improvement, as long as you aggregate and filter the data appropriately. For example, you can filter the data if you are concerned with only the most recent data.

You can define the extract ahead of time by choosing which fields you want and selecting the "Aggregate data for all visible dimensions" check box in Tableau Desktop's Extract Data dialog box. Alternatively, after doing your analysis and building your dashboard, when you are ready to publish, you can go back into the Extract Data dialog box and click the button for Hide All Unused Fields. Then when you extract the data, it will be the absolute minimum required to create the view.



Creating aggregate extracts is a very powerful technique when you have a large amount of base data but you need to create summary views that query across the whole data set. For example, you might have a billion records of detailed transaction data representing 10 years of sales and you want to start by showing the overall 10 year sales trend. The query for this initial view would potentially be slow as it needs to query across all billion rows. By creating an extract that is aggregated at the yearly level we can reduce the query effort required at view time as we will only have 10 numbers in the extract. Of course this is an oversimplified example – in reality you would have more dimensions than just time but the effect of significantly reducing the number of records that need to be queried at view time.

We can create quite sophisticated workbooks that have multiple levels of detail by creating multiple aggregated extracts, each tuned to support a specific level of detail, or by combining aggregated extracts with live connections. For example, you might have a set of initial summary views that use a highly aggregated extract but when you drill to detail you use action filters from the summary views to another sheet that connects via a live connection. This means that the summary views will be fast as they don't need to trawl over the full base data set, but also we don't need to extract all the base data to support the drill down actions. Also, the live connection will perform quickly because at the drill down level we are only accessing a small set of records.

In this way, you can mix and match, and aggregate at different levels to resolve nearly any performance issues so that get the results as fast as necessary. Since Tableau is efficient with memory, improving performance this way is usually relatively easy and you can have multiple extracts running at the same time.

Optimising extracts

Tableau Server not only optimises the physical columns that are in the database, but the additional columns that are created in Tableau. These columns include the results of deterministic calculations, such as string manipulations and concatenations, where the result is never going to change, as well as groups and sets. The results of non-deterministic calculations, such as those that involve a parameter or aggregations (such as sum or average) that are calculated at runtime, cannot be stored.

A user might refresh an extract after adding only two rows of data, and notice that the size of the extract has jumped from 100 MB to 120 MB. This jump in size is due to optimisation creating additional columns containing calculated field values, because it is cheaper to store data to disk than to recalculate it every time that data is needed.

One thing to watch out for is if you are making duplicate copies of a connection to a data extract, you need to ensure that all calculated fields exist in the connection you select for the "Optimize" or "Refresh" options, otherwise Tableau will not materialise fields which it thinks are unused. A good habit is to define all calculated fields in the primary data source and copy them as necessary to the other connections and then only ever refresh or optimise the extract from the primary data source.

Refreshing extracts

In Tableau Desktop, to refresh an extract you make a menu selection (Data menu > [your data source] > Extract > Refresh), which updates the data and adds any new rows. But in Tableau Server, during or after the publishing process, you can attach a schedule defined by an administrator to refresh the extract automatically. The smallest schedule increment allowed is every 15 minutes; the schedule can be to refresh at the same time daily, weekly, and so on. You can set up a "moving window" to continually refresh the data to just the most recent.

Note: If you want to refresh your data more often than every 15 minutes, you should consider connecting to live data, or set up a synchronised report database.

You can choose two refresh schedules for a single extract:

- An incremental refresh just adds rows, and does not include changes to existing rows.
- A full refresh discards the current extract and regenerates a new one from scratch from the data source.

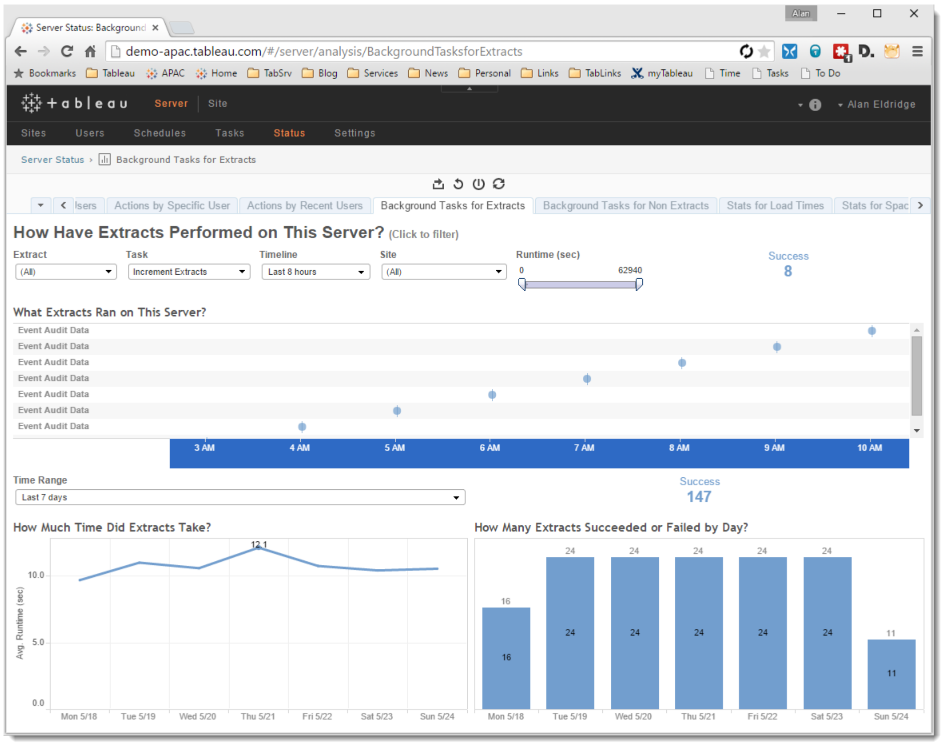
What happens if the refresh takes longer than the increment?

If the refresh time for an extract takes longer than the increment then the intervening refreshes are skipped. For example, the schedule is set to refresh the data every hour, but the amount of data is so large that the refresh takes an hour and a half. Then:

- The first refresh starts at 1:00 and finishes at 2:30.
- The next refresh is schedule to start at 2:00 but because the first is still running it is skipped.
- The next refresh starts at 3:00 and finishes at 4:30.

Extract maintenance

The Maintenance screens show what Background tasks are currently running, as well as those that have run for the past 12 hours. Colour coding is used to show the status of those tasks. The Maintenance screens are available to administrators and, with the appropriate permissions, to some other users, who can have permissions to initiate an ad hoc update to an extract. Also, for example, if a database is going to load, you can set a trigger to initiate an extract after the database finishes loading.



How Have Extracts Performed on This Server? (Click to filter)

Extract: (All)
Task: Increment Extracts
Timeline: Last 8 hours
Site: (All)
Runtime (sec): 0 – 62940
Success 8

Time Range: Last 7 days
Success 147

You also can refresh a workbook incrementally or fully via the `tabcmd` command line tool if you are using Tableau Server or the Tableau.exe command line if you are using Tableau Desktop. If you have complex scheduling requirements you can invoke this from an external scheduling tool such as the Windows Task

Scheduler. This approach is required if you want a refresh cycle that is shorter than the 15 minute minimum allowed through the Tableau Server interface.

Finally, for users of Tableau Online, you can use the Tableau Online sync client to keep on-prem data sources up-to-date via schedules defined on the Online service. You can find more information on this utility here:

> http://tabsoft.co/1fD1NXP

Data preparation

Tableau 9 introduces the ability to perform data preparation functions on data sources – e.g. using the data interpreter against Excel spreadsheets and using the Pivot function to reorient columns of data. When working with large volumes of data these steps can be time consuming and it is recommended that the output be written to a data extract.

Is it the query?

So you have reviewed the data connection and ensured it is following best practice. However your performance is still poor. The next point of review is to understand the specific query (or more likely queries) to ensure they are optimal.

Where can I find the queries?

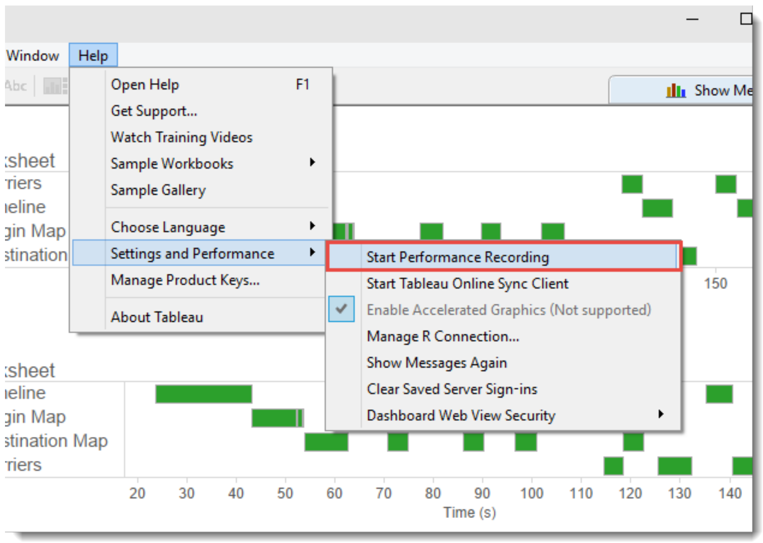
In Tableau you can find the full query text by looking in the log file. The default location is `C:\Users\<username>\Documents\My Tableau Repository\Logs\log.txt`. This file is quite verbose and is written as JSON encoded text, but if you search for "begin-query" or "end-query" you can find the query string being passed to the data source. Looking at the "end-query" log record will also show you the time it took for the query to run and how many records were returned to Tableau:

```
{"ts":"2015-05-24T12:25:41.226","pid":6460,"tid":"1674","sev":"info","req":"-
","sess":"-","site":"-","user":"-","k":"end—query",
"v":{"protocol":"4308fb0","cols":4,"query":"SELECT
[DimProductCategory].[ProductCategoryName] AS [none:ProductCategoryName:nk],\n
[DimProductSubcategory].[ProductSubcategoryName] AS
[none:ProductSubcategoryName:nk],\n  SUM(CAST(([FactSales].[ReturnQuantity]) as
BIGINT)) AS [sum:ReturnQuantity:ok],\n  SUM([FactSales].[SalesAmount]) AS
[sum:SalesAmount:ok]\nFROM [dbo].[FactSales] [FactSales]\n  INNER JOIN
[dbo].[DimProduct] [DimProduct] ON ([FactSales].[ProductKey] =
[DimProduct].[ProductKey])\n  INNER JOIN [dbo].[DimProductSubcategory]
[DimProductSubcategory] ON ([DimProduct].[ProductSubcategoryKey] =
[DimProductSubcategory].[ProductSubcategoryKey])\n  INNER JOIN
[dbo].[DimProductCategory] [DimProductCategory] ON
([DimProductSubcategory].[ProductCategoryKey] =
[DimProductCategory].[ProductCategoryKey])\nGROUP BY
[DimProductCategory].[ProductCategoryName],\n
[DimProductSubcategory].[ProductSubcategoryName]","rows":32,"elapsed":0.951}}
```

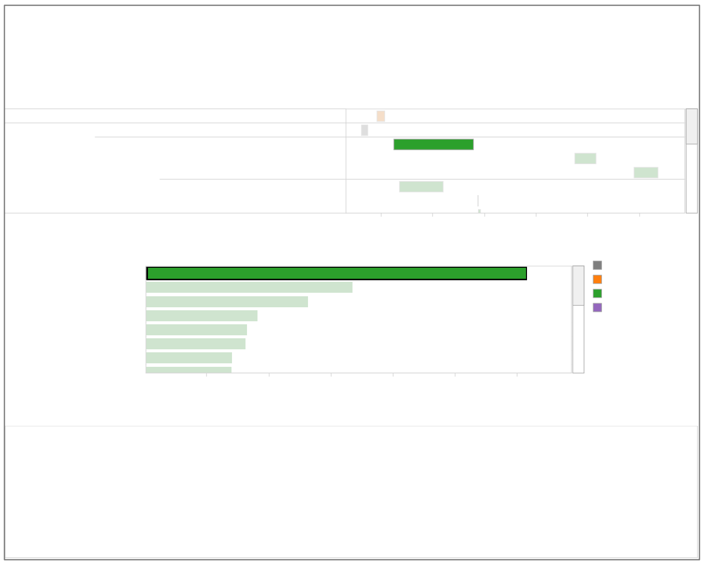
If you are looking on Tableau Server, the logs are in `C:\ProgramData\Tableau\Tableau Server\data\tabsvc\vizqlserver\Logs`.

Performance recordings

Another place to look to understand the performance of a workbook is the Performance Recorder feature of Tableau Desktop and Server. You enable this feature under the Help menu:



Start performance recording, then open your workbook. Interact with it as if you were an end user and when you feel you have gathered enough data go back in the help menu and stop recording. Another Tableau Desktop window will open at this point with the data captured:



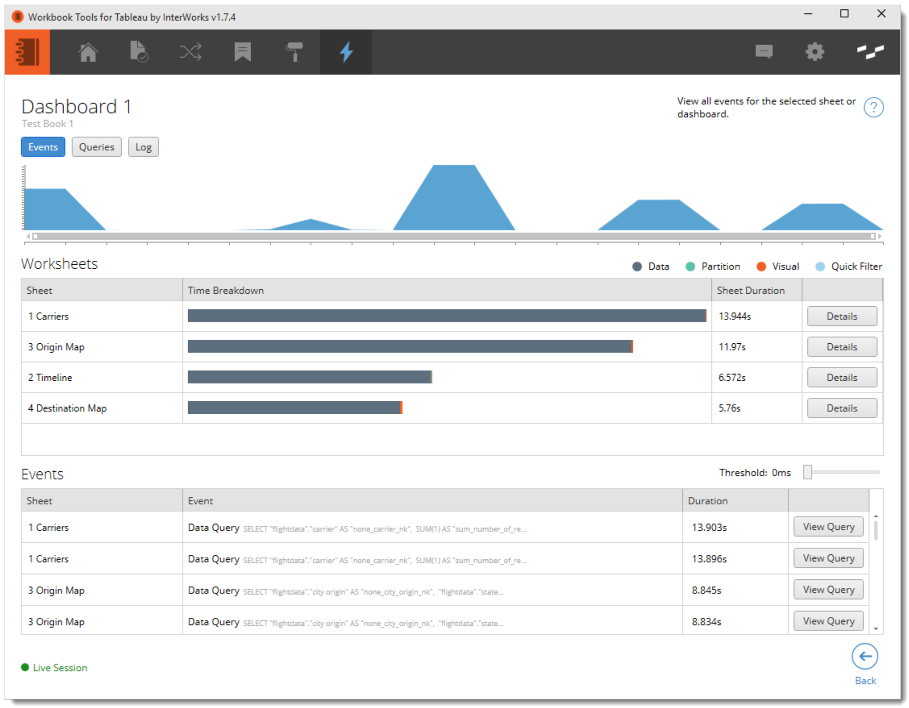
You can now identify the actions in the workbook that take the most time - for example in the above image the selected query from the Timeline worksheet takes 30.66 seconds to complete. Clicking on the bar shows the text of the query being executed.

You can use this information to identify those sections of a workbook that are the best candidates for review - i.e. where can you get the best improvement for the time you spend? More information on interpreting these recordings can be found in the following link:

http://tabsoft.co/1RdF420

Other tools for performance analysis

There are some other 3rd party tools available that will help you identify the performance characteristics of your workbooks. One option is “Power Tools for Tableau” by Interworks which includes a performance analyser that allows you to drill in and understand what sheets and queries are taking the longest time:


Workbook Tools for Tableau by InterWorks v1.7.4
Dashboard 1
Test Book 1
Events
Queries
Log
View all events for the selected sheet or dashboard.
● Data ● Partition ● Visual ● Quick Filter
Threshold: 0ms
Live Session
Back

Worksheets

Sheet	Time Breakdown	Sheet Duration	
1 Carriers		13.944s	Details
3 Origin Map		11.97s	Details
2 Timeline		6.572s	Details
4 Destination Map		5.76s	Details

Events

Sheet	Event	Duration	
1 Carriers	Data Query SELECT "flightdata"."carrier" AS "none_carrier_nk", SUM(1) AS "sum_number_of_re...	13.903s	View Query
1 Carriers	Data Query SELECT "flightdata"."carrier" AS "none_carrier_nk", SUM(1) AS "sum_number_of_re...	13.896s	View Query
3 Origin Map	Data Query SELECT "flightdata"."city origin" AS "none_city_origin_nk", "flightdata"."state...	8.845s	View Query
3 Origin Map	Data Query SELECT "flightdata"."city origin" AS "none_city_origin_nk", "flightdata"."state...	8.834s	View Query

Under the covers improvements

In Tableau 9.0 some of the biggest improvements to workbook efficiency and performance have been made in the query engine – i.e. how Tableau generates the queries required to render the visualisation/dashboard design. Many of these improvements are invisible to the end user but it is useful to understand them so you can ensure your workbooks take full advantage.

Data engine improvements

Tableau 9 includes significant performance improvements for queries against the Tableau data engine. These improvements are achieved by taking advantage of technological advances in CPU technology.

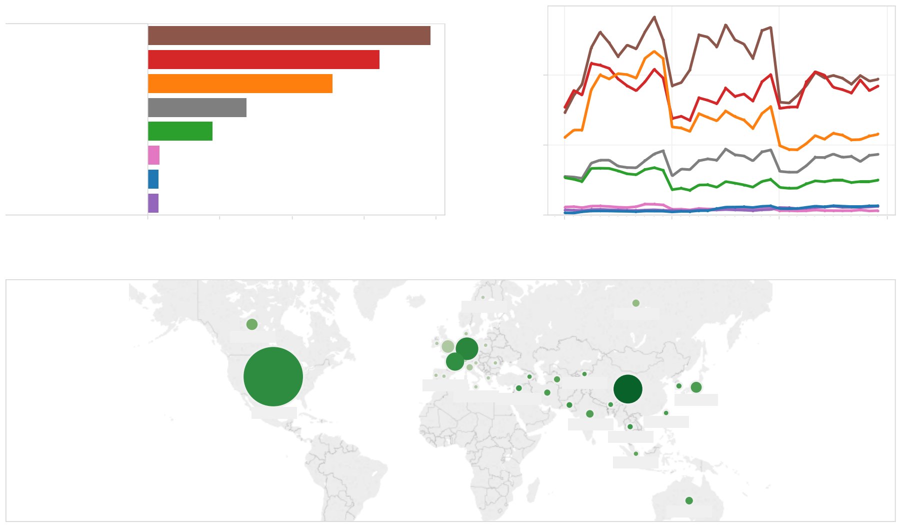
Firstly, the Tableau 9 data engine will run queries faster by using multiple cores where possible. The data engine can now run aggregations in parallel, splitting the work across multiple cores. By default, the maximum degree of parallelism is (number of available logical processors) / 2. This means that query operations on data extracts can run up to N times faster in Tableau 9 (where N is the number of cores in the machine).

Secondly, the data engine will now use SIMD instructions to perform low-level operations such as plus, minus, divide, min, max, sum, etc. on multiple data in parallel. This means that basic computations can be performed more quickly. This improvement is only available on CPUs that support SSE 4.1 however most modern Intel and AMD CPUs do.

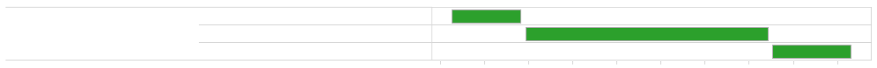
So – how can you take advantage of these improvements in Tableau 9? Simply use data extracts and run your workbooks on a workstation with a modern CPU that has multiple cores.

Query parallelism

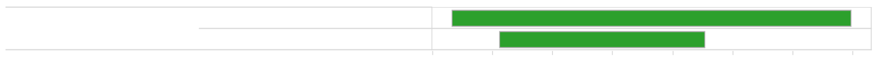
Tableau 9 takes advantage of the ability of source databases to execute multiple queries at the same time. Consider the following dashboard:



When this workbook is opened in Tableau 8.3, you can see that the three queries are run serially taking 4.4 seconds in total:



Opening the same workbook in Tableau 9 shows different behaviour:



The queries in Tableau 9 now run in parallel taking a total of 3.3 seconds – an improvement of 25%. By taking advantage of the data source's ability to handle multiple queries at the same time we can reduce the time it takes to render a dashboard with multiple elements.

The level of query parallelism varies between source systems as some platforms handle simultaneous queries better than others. The default is to run a maximum of 16 parallel queries for all data sources apart from text and Excel files (limit is 1 query at a time), creating SAP BW extracts (limit is 1 query at a time) and Amazon Redshift (limit is 2 queries at a time).

In most cases changing these settings is not necessary and you should leave them at their default settings, however if you have a specific need to control the degree of parallelism this can be set as:

- a global limit on the number of parallel queries for Tableau Server;
- limits for a particular data source type, such as SQL Server;
- limits for a particular data source type on a specific server; or
- limits for a particular data source type, on a specific server, when connecting to a specific database.

These settings are managed by an xml file named connection-configs.xml which you create and save in the app folder on Windows (`C:\Program Files\Tableau\Tableau 9.0`) and Mac (right click the App, click Show Package Contents, and place the file here) for Desktop or in the config directory in the vizqlserver folder (for example: `C:\ProgramData\Tableau\TableauServer\data\tabsvc\config\vizqlserver`) for Server. You must copy this configuration file to all the vizqlserver configuration directories in all worker machines.

You can read more about configuring parallel queries, including the syntax of the connection-configs.xml file, here:

http://tabsoft.co/1HAlztD

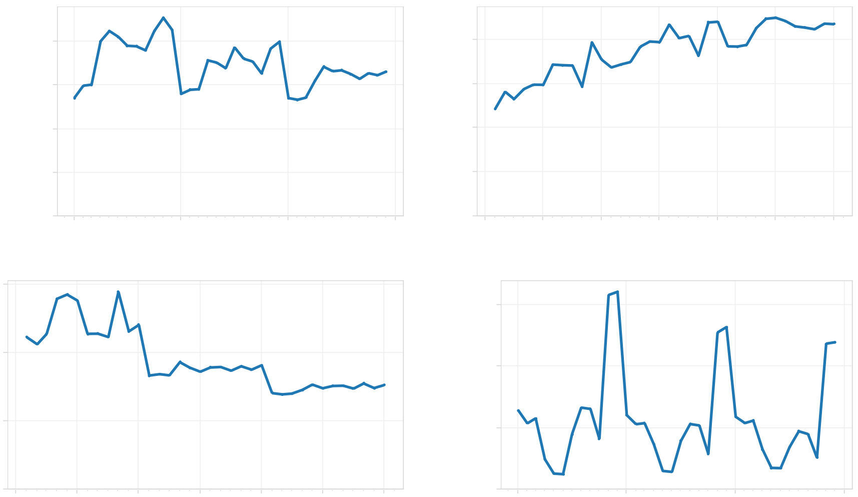
Query batching

You can also see that in Tableau 9 we only executed two queries instead of three. By batching the queries together, Tableau 9 can eliminate redundant queries. Tableau will sort the queries to run the most complex queries first in the hope that subsequent queries can be serviced from the result cache. In this example, because the timeline includes Product Category and because the SUM aggregation of Sales Amount is fully additive, the data for the Category chart can be resolved from the query cache and doesn't require a hit on the data source.

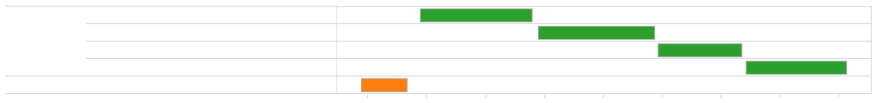
Prior to Tableau 9 it was possible to have less complex queries serviced from the query cache however the order in which Tableau ran the queries was determined a) by the alphabetical name order of the sheets on the dashboard (in Tableau 8.2) or by the order in which the dashboard designer placed the sheet objects on the dashboard (in Tableau 8.0 and 8.1).

Query fusion

What's better than running queries in parallel? Running fewer queries! In Tableau 9, the query engine will look for queries that are at the same level of detail (i.e. they are specified by the same set of dimensions) and will collapse them into a single query that returns all requested measures. Consider the following dashboard:



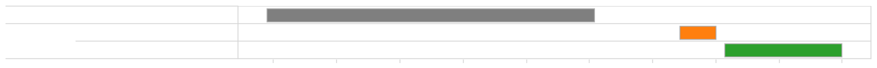
As you can see, this dashboard contains 4 sheets – each showing a different measure over time. They all have the same level of detail as they are showing the data using a continuous month. Running this dashboard in Tableau 8.3 results in 4 queries being run – see the following performance recording:



Each query is fetching a single measure:

```
SELECT DATEADD(month, DATEDIFF(month, 0,
CAST(FLOOR(CAST(CAST([DimDate].[Datekey] as datetime) as float)) as datetime)),
0) AS [tmn:Datekey:ok],
  SUM([FactSales].[DiscountAmount]) AS [sum:DiscountAmount:ok]
FROM [dbo].[FactSales] [FactSales]
  INNER JOIN [dbo].[DimDate] [DimDate] ON ([FactSales].[DateKey] =
[DimDate].[Datekey])
GROUP BY DATEADD(month, DATEDIFF(month, 0,
CAST(FLOOR(CAST(CAST([DimDate].[Datekey] as datetime) as float)) as datetime)),
0)
```

The same workbook opened in Tableau 9 produces a very different result:



Only a single query is run, returning all of the requested measures:

```
SELECT DATEADD(month, DATEDIFF(month, 0,
CAST(FLOOR(CAST(CAST([FactSales].[DateKey] as datetime) as float)) as
datetime)), 0) AS [tmn:Datekey:ok],
  AVG(cast([FactSales].[SalesAmount] as float)) AS [avg:SalesAmount:ok],
  SUM([FactSales].[DiscountAmount]) AS [sum:DiscountAmount:ok],
  SUM(CAST(1 as BIGINT)) AS [sum:Number of Records:ok],
  SUM([FactSales].[SalesAmount]) AS [sum:SalesAmount:ok]
```

```
FROM [dbo].[FactSales] [FactSales]
GROUP BY DATEADD(month, DATEDIFF(month, 0,
CAST(FLOOR(CAST(CAST([FactSales].[DateKey] as datetime) as float)) as
datetime)), 0)
```

As you can see, query fusion can improve the overall performance significantly – in this example the query time was reduced by ~45%, from 3.5 to 1.9 seconds. Where possible, consider setting the same level of detail for multiple sheets on a dashboard.

Caching

What is better than running fewer queries? Running no queries at all! While Tableau has incorporated caching features in the past, Tableau 9 introduces an external query cache for both Tableau Desktop and Tableau Server which can significantly reduce the number of queries that need to be run on the underlying data source.

Desktop

Whenever Tableau 9 uses a file-based data source (data extracts, Excel, Access, text files, but not statistical files) each query generated also generates a key-hash value that is unique for that particular query string and data source combination. We use this key-hash value as a lookup into the external query cache to determine if we have cached results we can use. If we find a hit in the cache, we will simply load the data from there. You can see this activity in the log file:

```
{"ts":"2015-05-25T17:25:18.146","pid":2472,"tid":"2d14","sev":"info","req":"-
","sess":"-","site":"-","user":"-","k":"ec-load",
"v":{"key-hash":"395207117","outcome":"hit","key-size-
b":"1928","cns":"NativeQueryResultsV1","elapsed-ms":"2","value-size-
b":"2092806"}}
```

If we miss the cache, we hit the underlying data source and write the results into the query cache:

```
{"ts":"2015-05-25T17:24:10.280","pid":3784,"tid":"1224","sev":"info","req":"-
","sess":"-","site":"-","user":"-","k":"ec-store","v":{
"key-hash":"1319376522","outcome":"done","key-size-
b":"944","cns":"NativeQueryResultsV1","elapsed-ms":"11","load-time-
ms":"87","value-size-b":"1770608","lower-bound-ms":"20"}}
```

A key difference between this external cache and the caching in previous versions of Tableau is that the external cache data is included when the file is saved as a packaged workbook. This allows Tableau to rapidly render the initial view of the workbook while it continues to unpack the larger data source file in the background. For an end user, this dramatically improves the responsiveness of opening the workbook. Note that if the cache data is > 10% the size of the source data file then it is not included.

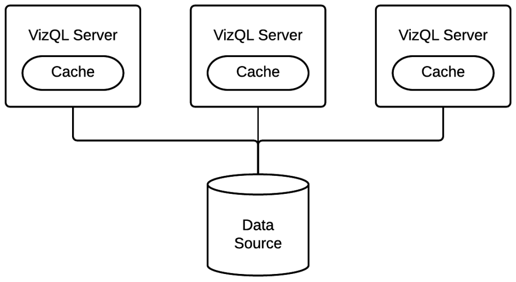
Additionally, the external cache in Tableau 9 is persistent. That means the cache results are retained between sessions in Tableau Desktop so if you use a file data source, restart Tableau Desktop and use it again you will still benefit from the cache.

The external cache is stored in `%LOCALAPPDATA%\Tableau\Caching` on Windows and `~/Library/Caches/com.tableau/` on Mac. By default it is limited to 500MB in total, and it will be invalidated if the user forces a refresh of the data source (e.g. pressing F5, ⌘R).

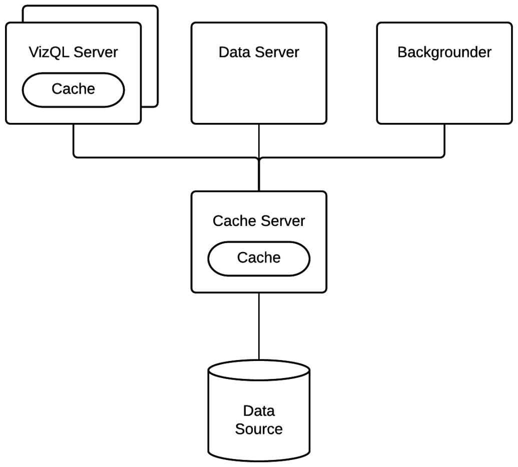
Note that the external query cache is not used if your visualisation uses relative date filters, or if you are using user filters.

Server

The query cache in Tableau Server 9 has been significantly enhanced over previous versions. Previously, the query cache in Tableau Server was local to each VizQL instance:



Now in addition to the local VizQL caches there is an external query cache that is shared – not just across all VizQL instances but across ALL processes that access the underlying data source. A new service called the Cache Server now manages the external cache for all service instances across the cluster:



This change significantly increases the effectiveness of the cache, especially in deployments where there are multiple VizQL instances. The VizQL Server will check its local cache – if it misses, it will check the Cache Server to see if the results are available from a query run elsewhere. Another important change the Cache Server brings is that the external cache is now persistent. Previously the contents of the cache were volatile and were lost whenever the VizQL instance was restarted. Now the cache contents remain between instantiations of the services.

Finally, a useful side-effect of the cache being shared is we can now realistically warm the cache for workbooks that we know are slow on first view by running a subscription for the workbook. Doing this early in the morning before your interactive users arrive at work will ensure that the workbook is run and the query results loaded into the cache. The cache will then be hit when the users view the workbook, making the initial view time fast.

Note – like on Desktop, there are times when the cache is not used. Specifically, when a workbook contains relative data filters, user filters, or parameters.

Joins

Generally, if you are working with multiple tables in Tableau the preferred approach is to define the joins in the data connection window. When you do this you are not defining a specific query – you are simply defining how the tables relate to one another. When you drag and drop fields into a viz Tableau will use this information to generate the specific query necessary to fetch only the required data.

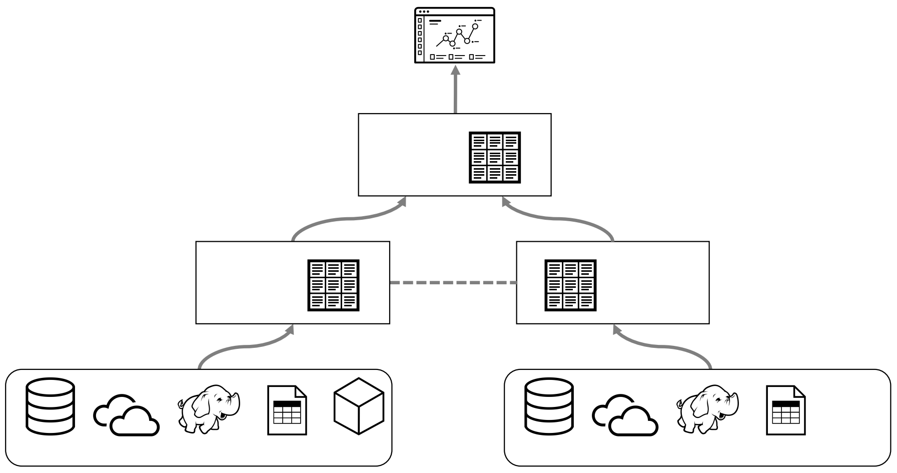
As a general performance rule-of-thumb, the number of joined tables should be minimized to only include the tables needed for a specific worksheet/visualization. Depending on the complexity of the data connection and the worksheets, it may also be worth separating out data connections for each worksheet and creating specific join patterns for those sheets.

Joins perform better when joins are properly constrained between tables. The constraints give Tableau Desktop the freedom to simplify queries down to just the tables necessary to answer certain subsets of a question (Legends, quick filters)

Blending

When deciding between joining data tables and blending data tables in Tableau, consider where the data is coming from, the number of data connections, and the number of records you have in the data.

When you use blending, one of the primary influences on blending performance is not the number of records in each data source but rather the cardinality of the blending field(s) that are linking the two data sets. Blending queries the data from both data sources at the level of the linking fields and then merges the results of both queries together in memory.



If there are many unique values then this can require a large amount memory. It is recommended that you use the 64-bit version of Tableau Desktop when blending on data to avoid running out of memory. However, blends of this nature are still likely to take a long time to compute.

The best practice recommendation for blending is to avoid blending on dimensions with large numbers of unique values (e.g. Order ID, Customer ID, exact date/time, etc).

Primary groups and aliases

If you find it necessary to blend between two data sources because one contains the "fact" records and the other contains dimensional attributes it might be possible to improve performance by creating a primary group or alias.

They work by creating groups and/or aliases in the primary data source to reflect the attributes listed in the secondary data source. Primary groups are used for 1:many relationships and primary aliases are used for 1:1 relationships. You can then use the relevant object from the primary data source, removing the need for blending at view time.

For the following examples, consider the following three tables:

ID	Value
A	89
B	94
C	74
D	88
E	58
F	89
G	95

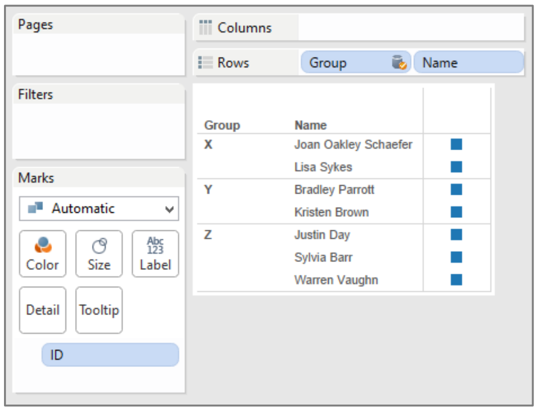
ID	Name
A	Lisa Sykes
B	Joan Oakley Schaefer
C	Bradley Parrott
D	Kristen Brown
E	Sylvia Barr
F	Warren Vaughn
G	Justin Day

ID	Group
A	X
B	X
C	Y
D	Y
E	Z
F	Z
G	Z

Primary groups are useful when the secondary data source contains an attribute that is mapped 1:many back to dimension members in the primary data source. Assume what we want to produce from the above data is this:

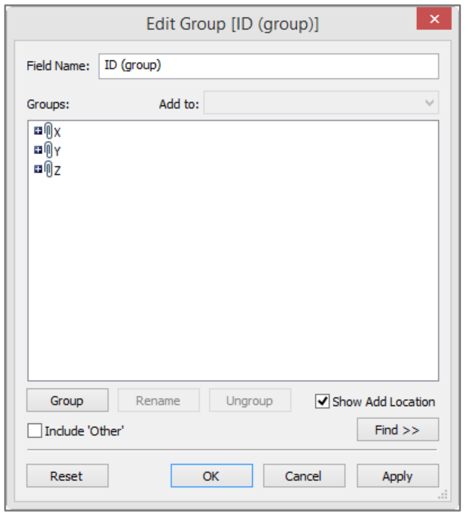
Group	Name
X	Lisa Sykes
	Joan Oakley Schaefer
Y	Bradley Parrott
	Kristen Brown
Z	Sylvia Barr
	Warren Vaughn
	Justin Day

We could create this by blending but as already discussed this would result in very slow performance if there were a large number of IDs:



Group	Name	
X	Joan Oakley Schaefer	🟦
	Lisa Sykes	🟦
Y	Bradley Parrott	🟦
	Kristen Brown	🟦
Z	Justin Day	🟦
	Sylvia Barr	🟦
	Warren Vaughn	🟦

By right-clicking on the Group field and selecting "Create Primary Group" Tableau will create a group object in the primary data source that maps the linking field (in this case ID) to the selected secondary data source dimension (in this case Group).


Edit Group [ID (group)]

Field Name: ID (group)

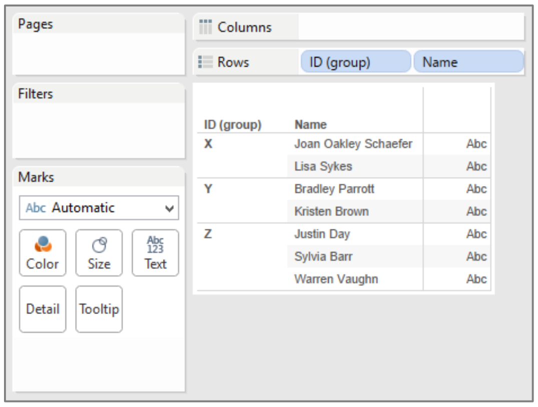
Groups:

- X
- Y
- Z

[✓] Show Add Location

[ ] Include 'Other'

We can now recreate this table without requiring a blend:

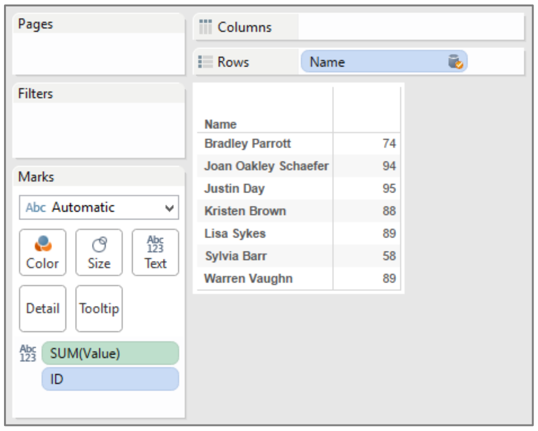


ID (group)	Name	
X	Joan Oakley Schaefer	Abc
	Lisa Sykes	Abc
Y	Bradley Parrott	Abc
	Kristen Brown	Abc
Z	Justin Day	Abc
	Sylvia Barr	Abc
	Warren Vaughn	Abc

Primary aliases are useful when the secondary data source contains an attribute that is mapped 1:1 back to dimension members in the primary data source. Assume what we want to produce from the above data is this:

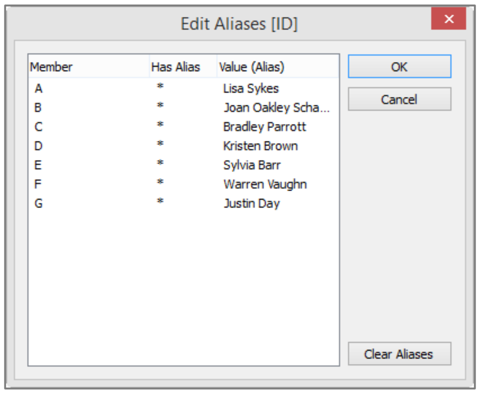
Name	Value
Lisa Sykes	89
Joan Oakley Schaefer	94
Bradley Parrott	74
Kristen Brown	88
Sylvia Barr	58
Warren Vaughn	89
Justin Day	95

We could create this by blending between the two data sources however as already discussed this would result in very slow performance if there were a large number of IDs:



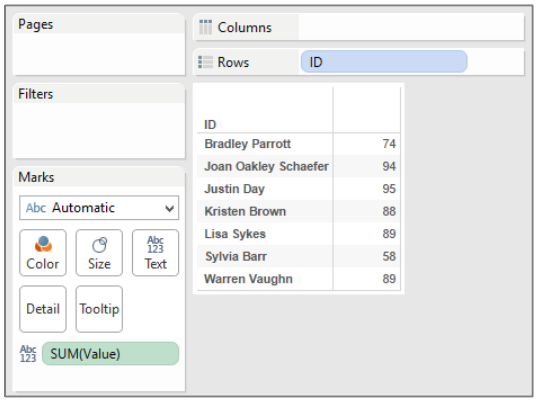
Name	
Bradley Parrott	74
Joan Oakley Schaefer	94
Justin Day	95
Kristen Brown	88
Lisa Sykes	89
Sylvia Barr	58
Warren Vaughn	89

By right-clicking on the Name field and selecting “Edit Primary Aliases” we can do a one-time mapping of the Name to the ID field as alias values:



Member	Has Alias	Value (Alias)
A	*	Lisa Sykes
B	*	Joan Oakley Scha...
C	*	Bradley Parrott
D	*	Kristen Brown
E	*	Sylvia Barr
F	*	Warren Vaughn
G	*	Justin Day

We can now create the required visualisation without a blend which would be much faster:



ID	
Bradley Parrott	74
Joan Oakley Schaefer	94
Justin Day	95
Kristen Brown	88
Lisa Sykes	89
Sylvia Barr	58
Warren Vaughn	89

Note that both primary groups and aliases are not dynamic and would need to be updated if the data changed. Consequently they aren't a great solution for frequently updated data but if you need a quick mapping they can potentially eliminate the need for costly blends.

More information can be found in the following kbase article:

http://tabsoft.co/1PRhkU9

Custom SQL

Sometimes users new to Tableau try to bring old-world techniques to their workbooks such as creating data sources using hand-written SQL statements. In many cases this is counterproductive as Tableau can generate much more efficient queries when we simply define the join relationships between the tables and let the query engine write SQL specific to the view being created. However there are times when specifying joins in the data connection window does not offer all the flexibility you need to define the relationships in your data. For example, you may have data in multiple sheets in an Excel workbook and you need to UNION them together. This can't be done directly in the data connection diagram however you can do this by creating a custom SQL statement:

Creating a data connection using a hand-written SQL statement can be very powerful but as Spider-Man cautions, "with great power comes great responsibility". In some cases custom SQL can actually cause reduce performance. This is because in contrast to defining joins, custom SQL is never deconstructed and is always executed atomically. This means no join culling occurs and we can end up with situations where the database is asked to process the whole query just for a single column, like this:

```
SELECT SUM([TableauSQL].[Sales])
FROM (
  SELECT [OrdersFact].[Order ID] AS [Order ID],
       [OrdersFact].[Date ID] AS [Date ID],
       [OrdersFact].[Customer ID] AS [Customer ID],
       [OrdersFact].[Place ID] AS [Place ID],
       [OrdersFact].[Product ID] AS [Product ID],
       [OrdersFact].[Delivery ID] AS [Delivery ID],
       [OrdersFact].[Discount] AS [Discount],
       [OrdersFact].[Cost] AS [Cost],
       [OrdersFact].[Sales] AS [Sales],
       [OrdersFact].[Qty] AS [Qty],
       [OrdersFact].[Profit] AS [Profit]
  FROM [dbo].[OrdersFact] [OrdersFact]
) [TableauSQL]
HAVING (COUNT_BIG(1) > 0)
```

It's important to ensure that your custom SQL statement does not contain any unnecessary clauses. For example, if your query contains GROUP BY or ORDER BY clauses these are generally going to be overhead as Tableau will create its own clauses based on the structure of the visualisation. Eliminate these from your query if possible.

A good recommendation is to use custom SQL in conjunction with data extracts. That way the atomic query is only executed once (to load the data into the data extract) and all subsequent analysis in Tableau is done using dynamic, optimised queries against the data extract. Of course, all rules have exceptions and this one does too – when using custom SQL create a data extract UNLESS your custom SQL contains parameters.

Using parameters in a custom SQL statement could in some cases make live connections more performant as the base query can be more dynamic (e.g. filter clauses that use parameters will be evaluated appropriately). It could also be used to pass in values for performance limiters such as TOP or SAMPLE to constrain the amount of data returned by the database. However if you are using a data extracts it will be purged and regenerated every time you change the parameter and this can be slow.

Also be aware that parameters can only be used to pass in literal values so they cannot be used to dynamically change the SELECT or FROM clauses.

Alternatives to custom SQL

Instead of using custom SQL directly in Tableau, it is sometimes preferable to move the query into the underlying data source. In many cases this can improve performance as it allows the data source to more efficiently parse the query, or it might mean a complex query only has to be run one time. It also lends itself well to good management practices as it allows the single definition to be shared across multiple workbooks and data sources.

There are several ways to do this:

Views

Almost all DBMS's support the concept views – a virtual table representing the result of a database query. For some database systems, simply taking the query out of a custom SQL statement and instantiating it in the database as a view results in significantly better performance. This is because the query optimiser is able to generate a better execution plan than when the query is wrapped in an outer SELECT statement.

Defining custom query logic into a view instead of in the Tableau workbook also allows it to be reused across multiple workbooks and data sources.

Stored procedures

Stored procedures are similar to views however they can contain much more complex logic and potentially perform multiple steps of data preparation. They can also be parameterised, allowing them to return a targeted set of data based on user input.

Stored procedures are supported in Sybase ASE, SQL Server and Teradata. To avoid running the stored procedure multiple times for a single viz, Tableau executes the stored procedure and writes the result set to a temp table in the database. The actual queries for the viz are then run against the temp table. The execution of the stored procedure and population of the temp table is done on the initial opening of the workbook and whenever the stored process parameters are changed. This process can take some time and these interactions may be slow.

If you are extracting the results of a parameterised stored procedure into a data extract, the extract will be purged and refreshed every time you change the parameter values.

More information on using stored procedures can be found in the Tableau online help:

http://tabsoft.co/1HAD93e

Summary tables

If you have a very large, detailed data set that you typically summarise when querying (e.g. you store individual transactions but typically use the data summarised by day, region, customer, product, etc.) then consider creating a summary table and using Tableau against it to achieve faster query times.

Note – you can use Tableau data extracts to achieve a similar outcome by creating an aggregated data extract. See the section on extracts for more detail.

Initial SQL

Another alternative to custom SQL (if your data source supports it) is to use the custom SQL statement in an initial SQL block. You could use this to create a temporary table which will then be the selected table in your query. Because initial SQL is executed only once when the workbook is opened (as opposed to every time the visualisation is changed for custom SQL) this could significantly improve performance in some cases.

Note that on Tableau Server an administrator can set a restriction on initial SQL so that it will not run. You might need to check to see that this is OK in your environment if you are planning to publish your workbook to share with others.

You can find more information on initial SQL in the online documentation:

http://tabsoft.co/1AsJkXg

Filters

Filtering in Tableau is extremely powerful and expressive. However, inefficient filters are one of the most common causes of poorly performing workbooks and dashboards. The following sections lay out a number of best practices for working with filters.

Note – the efficiency of filters is dramatically impacted by the presence and maintenance of indexes in the data source. See the previous section on indexes for more detail.

Context filters

By default, all filters that you set in Tableau are computed *independently*. That is, each filter accesses all rows in your data source without regard to other filters. However, by specifying a context filter you can make any other filters that you define *dependent* because they process only the data that passes through the context filter.

Depending on the data source type, context filters may be implemented as a subquery, a temp table or even a data extract. The idea is that it creates a smaller data source containing only the data that passes the context filter. Subsequent filters and queries only process this smaller, secondary data source resulting in increased performance.

The creation of a context can be an expensive activity for the data source – particularly when implemented as a temp table or an extract – so this approach is only recommended when:

- the context filter reduces the size of the data set significantly – by an order of magnitude is a good rule of thumb; and
- the context filter is not frequently changed by the user – if the filter is changed the database must recreate the context, slowing performance.

Data source filters

Data source filters are the highest level of filter applied to a data source and like context filters are applied before all others. Unlike context filters though, data source filters are in most cases implemented as subqueries so they can sometimes be an effective way of creating a "context" without generating a temp table or extract.

Slicing filters

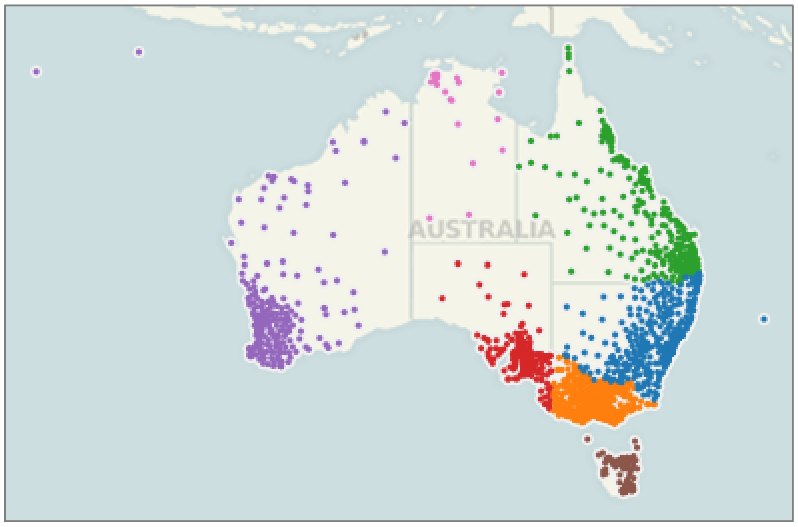
Slicing filters are filters on dimensions that are not used in the viz (i.e. they are not part of the viz level of detail) but have their own aggregation criteria. For example, you might have a viz showing total sales by customer name, but the viz is restricted to only show data for the top two states based on total sales at the state level.

In order for Tableau to display this viz it needs to run two queries – one at the state level to isolate the top two states, and another at the customer level, restricted by the results of the first query. Tableau does this using subqueries.

Beware of using this kind of filter unnecessarily as it can be an expensive form of filtering.

Filtering categorical dimensions

Consider the following visualisation – a map of Australia with the marks for each postcode:



There are several ways we could filter the map to just show the postcodes for Western Australia (the purple dots):

- We could select all the marks in WA and keep-only the selection;
- We could select all the marks outside of WA and exclude the selection;
- We could keep-only on another attribute such the State dimension;
- We could filter by range – either on the postcode values or the latitude/longitude values.

Discrete

Using the first two options we would find the keep-only and exclude options perform poorly – in fact they can often be slower than the unfiltered data set. This is because they are expressed as a discrete list of postcode values that are filtered in or out by the DBMS – either through a complex WHERE clause or by joining with a temp table that has been populated with the selection. For a large set of marks this can result in a very expensive query to evaluate.

The third option is fast in this example because the resulting filter (`WHERE STATE=”Western Australia”`) is very simple and can be efficiently processed by the database. However this approach becomes less effective as the number of dimension members needed to express the filter increases – eventually approaching the performance of the lasso and keep-only option.

Ranged

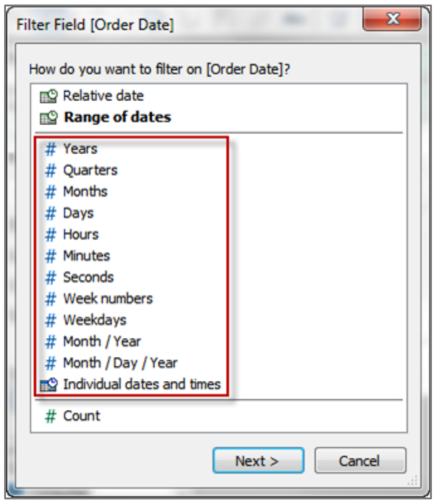
Using the ranged filter approach also allows the database to evaluate a simple filter clause (either `WHERE POSTCODE >= 6000 AND POSTCODE <= 7000` or `WHERE LONGITUDE < 129`) resulting in fast execution. However this approach, unlike a filter on a related dimension, doesn’t become more complex as we increase the cardinality of the dimensions.

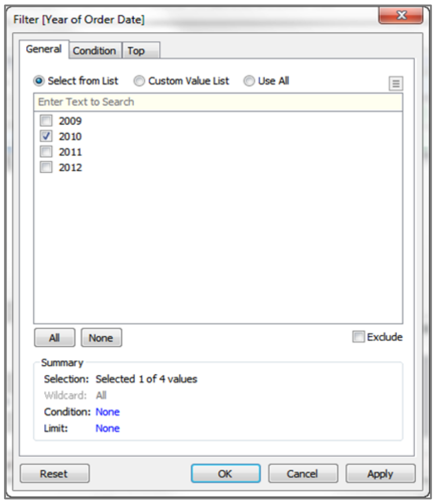
The take-away from this is that ranged filters are often faster to evaluate than large itemised lists of discrete values and they should be used in preference to a keep-only or exclude for large mark sets if possible.

Filtering dates: discrete, range, relative

Date fields are a special kind of dimension that Tableau often handles differently than standard categorical data. This is especially true when you are creating date filters. Date filters are extremely common and fall into three categories: Relative Date Filters, which show a date range that is relative to a specific day; Range of Date Filters, which show a defined range of discrete dates; and Discrete Date Filters, which show individual dates that you've selected from a list. As shown in the section above, the method used can have a material impact on the efficiency of the resulting query.

Discrete





- [ ] 2009
- [x] 2010
- [ ] 2011
- [ ] 2012

Summary
Selection: Selected 1 of 4 values
Wildcard: All
Condition: None
Limit: None

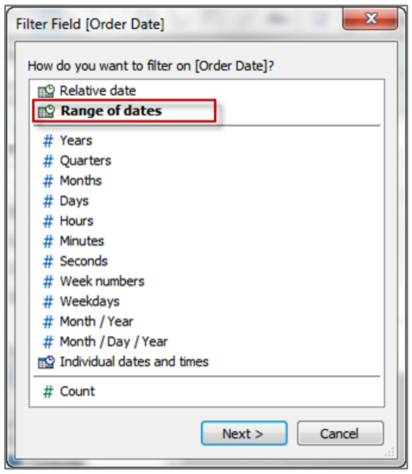
Sometimes you may want to filter to include specific individual dates or entire date levels. This type of filter is called a discrete date filter because you are defining discrete values instead of a range. This filter type results in the date expression being passed to the database as a dynamic calculation:

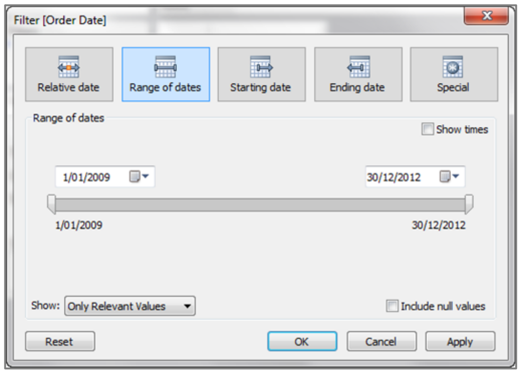
```
SELECT [FactSales].[Order Date], SUM([FactSales].[SalesAmount])
FROM [dbo].[FactSales] [FactSales]
WHERE (DATEPART(year,[FactSales].[Order Date]) = 2010)
GROUP BY [FactSales].[Order Date]
```

In most cases, query optimisers will intelligently evaluate the DATEPART calculation, however there are some scenarios where using discrete date filters can result in poor query execution. For example, querying a partitioned table with a discrete date filter on the date partition key. Because the table isn't partitioned on the DATEPART value, some databases will go off and evaluate the calculation across all partitions to find records that match the criteria, even though this isn't necessary. In this case, you may see much better performance by using a "range of date" filter or a relative date filter with a specified anchor date.

One way to optimise performance for this type of filter is to materialise the calculation using a data extract. First, create a calculated field that implements the DATEPART function explicitly. If you then create a Tableau data extract, this calculated field will be materialised as stored values in the extract (because the output of the expression is deterministic). Filtering on the calculated field instead of the dynamic expression will be faster because the value can simply be looked up, rather than calculated at query time.

Ranged



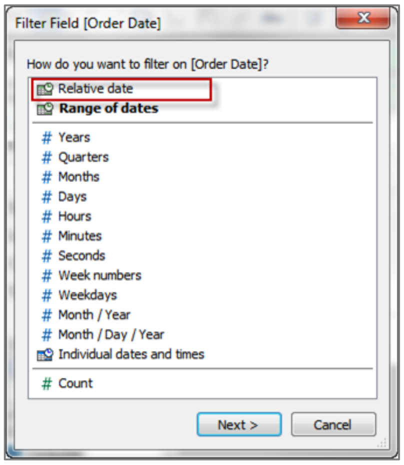


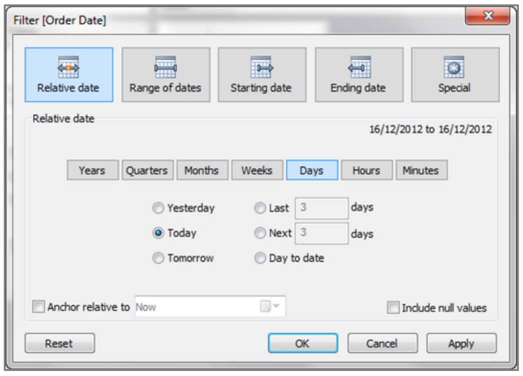
This type of filter is used when you want to specify a range of contiguous dates. It results in the following query structure being passed to the database:

```
SELECT [FactSales].[Order Date], SUM([FactSales].[SalesAmount])
FROM [dbo].[FactSales] [FactSales]
WHERE (([FactSales].[Order Date] >= {ts '2009-01-01 00:00:00'})
  AND ([FactSales].[Order Date] <= {ts '2012-12-31 00:00:00'}))
GROUP BY [FactSales].[Order Date]
```

This type of WHERE clause is very efficient for query optimisers, allowing execution plans to leverage indexes and partitions to full effect. If you are observing slow query times when you add discrete date filters, consider replacing them with ranged date filters and see if that makes a difference.

Relative





A relative date filter lets you define a range of dates that updates based on the date and time you open the view. For example, you may want to see Year to Date sales, all records from the past 30 days, or bugs closed last week. Relative date filters can also be relative to a specific anchor date rather than today.

```
SELECT [FactSales].[Order Date], SUM([FactSales].[SalesAmount])
FROM [dbo].[FactSales] [FactSales]
WHERE (([FactSales].[Order Date] >= DATEADD(year,(-2),DATEADD(year,
  DATEDIFF(year, 0, {ts '2012-12-16 22:37:51.490'}), 0))) AND
  ([FactSales].[Order Date] < DATEADD(year,1,DATEADD(year, DATEDIFF(year, 0,
  {ts '2012-12-16 22:37:51.490'}), 0))))
GROUP BY [FactSales].[Order Date]
```

As you can see, the resulting WHERE clause uses a ranged date filter, so this is also an efficient form of date filter. However be aware that the external query cache is not used for queries that use relative date filters, which can impact the scalability of the workbook when published to Tableau Server.

Quick filters

Despite the name, too many quick filters will actually slow you down, particularly if you set them to use 'Only Relevant Values' and you've got lots of discrete lists. Try a more guided analytics approach and use action filters within a dashboard instead. If you're building a view with umpteen filters in it to make it super customisable, ask yourself whether multiple dashboards with different levels and themes would work better (hint: yes, it probably would).

Enumerated vs. non-enumerated

Enumerated quick filters require Tableau to query the data source for all potential field values before the quick filter object can be rendered. These include:

- Multiple value list
- Single value list
- Compact List
- Slider
- Measure filters
- Ranged date filters.

Non-enumerated quick filters on the other hand, do not require knowledge of the potential field values. These include:

- Custom value list
- Wildcard match
- Relative date filters
- Browse period date filters.

Consequently, non-enumerated quick filters reduce the number of quick filter related queries that need to be executed by the data source. Also non-enumerated quick filters render faster when there are many dimension members to display.

Using non-enumerated quick filters can improve performance however it does so at the expense of visual context for the end user.

Relevant values

Enumerated quick filters can be set to show the potential field values in three different ways:

- All Values in Database - when you select this option all values in the database are shown regardless of the other filters on the view. The quick filter does not need to re-query the database when other filters are changed.

- All Values in Context – this option is only available when you have active context filters. The quick filter will show all values in the context (i.e. the temporary table generated by the context filter) regardless of other filters on the view. The quick filter does not need to re-query the database when other filters are changed.
- Only Relevant Values - when you select this option other filters are considered and only values that pass these filters are shown. For example, a quick filter on State will only show the Eastern states when a filter on Region is set. Consequently, the quick filter must re-query the data source when other filters are changed.

As you can see, the "only relevant values" setting can be very useful for helping the user make relevant selections, but it can significantly increase the number of queries that need to be run while they interact with the dashboard. It should be used in moderation.

Quick filter alternatives

There are alternatives to using quick filters that provide a similar analytic outcome but do so without the additional query overhead.

Instead of exposing a quick filter to the users, you could create a parameter and filter based on the users' selections.

- PROS:
    - Parameters do not require a data source query before rendering
    - Parameters + calculated fields can implement logic that is more complex than can be done with a simple field filter
    - Parameters can be used to filter across data sources – quick filters operate only within a single data source
- CONS:
    - Parameters are single-value only – you cannot use them if you want the user to select multiple values
    - Parameters are not dynamic – the list of values is defined when they are created and does not update based on the values in the DBMS

Another alternative is to use filter actions between views -

- PROS:
    - Actions support multi-value selection – either by visually lassoing or CTRL/SHIFT clicking
    - Actions show a dynamic list of values that is evaluated at run time
    - Actions can be used to filter across data sources – quick filters operate only within a single data source
- CONS:
    - Filter actions are more complex to set up than quick filters

- Actions do not present the same user interface as parameters or quick filters – generally they require more screen real estate to display
- The action source sheet still needs to query the data source, however it benefits from caching within the Tableau processing pipeline

For a further discussion on alternate design techniques that don’t rely heavily on quick filters see the earlier section.

User filters

Any workbooks that utilise user filters – either through the “Create User Filter…” dialog or through calculated fields that use any of the built-in User functions – cannot use the external query caches when deployed to Tableau Server because they are unique to each user. This can have performance impacts as:

- All workbooks will need to query the underlying data source, even if another user session has just asked the exact same query. This results in more data source I/O.
- More cache space will be required as each user session will create its own query result and model cache. On machines with high load this can result in caches being cleared while still in use, again resulting in more I/O.

See the earlier section on the external query cache for more detail.

Is it the calculations?

In many cases your source data will not provide all fields you need to answer all of your questions. Calculated fields will help you to create all the dimensions and measures needed for your analysis.

Within a calculated field you may define a hardcoded constant (like a tax rate, for instance), do very simple mathematical operations like subtraction or multiplication (e.g. revenues minus cost), use more complex mathematical formulas, perform logical tests (e.g. IF/THEN, CASE), do type conversions and much more.

Once defined, a calculated field is available across the entire workbook as long as the worksheets are using the same data source. You can use calculated fields in your workbook in the same way you use dimensions and measures from your source data.

There are four different calculation types in Tableau:

- Basic calculations
- Aggregate calculations
- Table calculations
- Level of detail expressions

A great reference for how to perform complex calculations and a forum where users share solutions to common problems is the Tableau Calculation Reference Library:

http://tabsoft.co/1I0SsWz

Basic and aggregate calculations

Basic and aggregate calculations are expressed as part of the query sent to the data source and therefore are calculated by the database. For example, a viz showing the sum total of sales for each year of order data would send the following query to the data source:

```
SELECT DATEPART(year,[OrdersFact].[Order Date]) AS [yr:Order Date:ok],
  SUM([OrdersFact].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[OrdersFact] [OrdersFact]
GROUP BY DATEPART(year,[OrdersFact].[Order Date])
```

The YEAR calculation is a basic calculation and the SUM(SALES) is an aggregate calculation.

In general, basic and aggregate calculations scale very well and there are many database tuning techniques that can be employed to improve their performance.

Note that in Tableau 9 the actual DB query may not be a direct translation of the basic calculations used in the viz. For example, the following query is run when the viz contains a calculated field Profit Ratio defined as `SUM([Profit])/SUM([Sales])`:

```
SELECT DATEPART(year,[OrdersFact].[Order Date]) AS [yr:Order Date:ok],
  SUM([OrdersFact].[Profit]) AS
[TEMP(Calculation_0260604221950559)(1796823176)(0)],
  SUM([OrdersFact].[Sales]) AS
[TEMP(Calculation_0260604221950559)(3018240649)(0)]
```

```
        FROM [dbo].[OrdersFact] [OrdersFact]
        GROUP BY DATEPART(year,[OrdersFact].[Order Date])
```

Tableau actually fetches the elements of the calculation and performs the division function in the client layer. This ensures that `SUM([Profit])` and `SUM([Sales])` at the Year level are cached and can be used elsewhere within the workbook without having to go to the data source.

Finally, query fusion (which we discussed earlier) can modify the actual query run against the data source to return multiple measures that share a common level of detail.

Table calculations

Table calculations – unlike basic and aggregate calculations – are not executed by the database but rather calculated by Tableau on the query result set. While this means more work for Tableau, it is generally done over a much smaller set of records than are in the original data source.

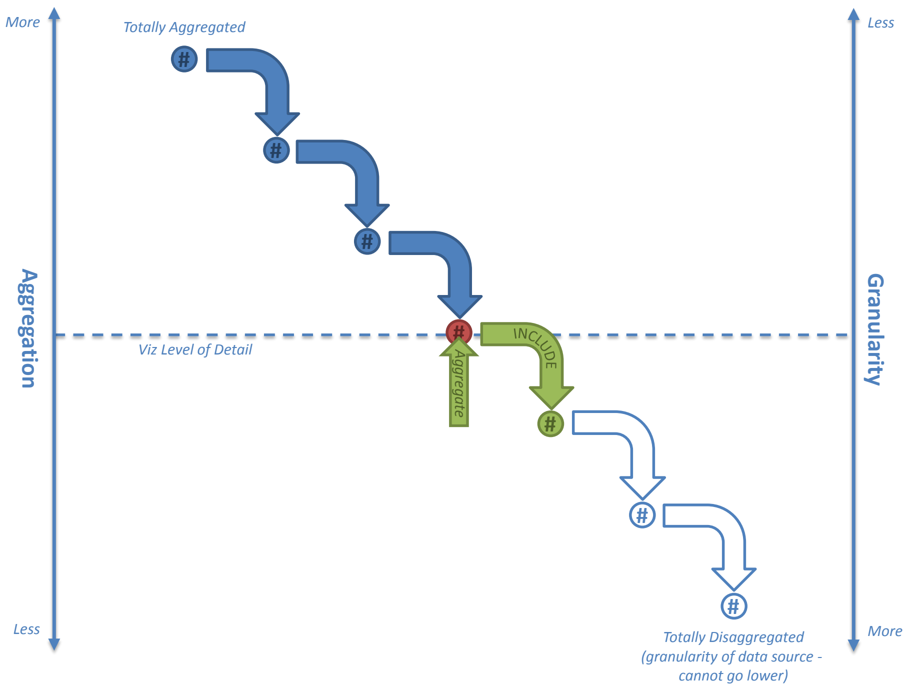
If table calculation performance is a problem (possibly because the result set being returned to Tableau is very large) consider pushing some aspects of the calculation back to the data source layer. One way to do this is to leverage an aggregated data extract. Imagine an example where you want to find the weekly average of daily total sales across multiple stores. You can do this with a table calculation using:

```
        WINDOW_AVG(SUM([Sales])
```

However, if the number of days/stores is very large, this calculation can become slow. To push the SUM([Sales]) back to the data layer, create an aggregate extract that rolls the Date dimension up to the day level. The calculation can then be done by simply AVG([Sales]) as the extract has already materialised the daily totals.

Level of detail expressions

Level of detail (LOD) expressions are a new calculation type introduced in Tableau 9. They allow you to define the level of detail at which a calculation should be performed, independently of the Viz LOD.



LOD expressions allow you to replace calculations you may have authored in more cumbersome ways in earlier versions of Tableau:

- Using table calcs, you may have tried to find the first or last period within a partition. For example, calculating the headcount of an organization on the first day of every month.
- Using table calcs, calculated fields, and reference lines, you may have tried to bin aggregate fields. For example, finding the average of a distinct count of customers.
- Using data blending, you may have tried to achieve relative date filtering relative to the maximum date in the data. For example, if your data is refreshed on a weekly basis, computing the year to date totals according to the maximum date.

Unlike table calcs, LOD Expressions are generated as part of the query to the underlying data source. They are expressed as a nested select so they are dependent on DBMS performance:

```
SELECT T1.[State],SUM(T2.[Sales per County, State])
FROM [DB Table] T1 INNER JOIN
  (SELECT [State], [County], AVG([Sales]) AS [Sales per County, State]
  FROM [DB Table] GROUP BY [State],[County]) T2
ON T1.[State] = T2.[State]
GROUP BY T1.[State]
```

This means that there could be scenarios where it is more efficient to solve a problem using one approach or the other – i.e. a table calc or blending could perform better than a LOD expression, or vice versa. If you suspect performance is slow due to a LOD expression you might try replacing it with a table calc or a data blend (if possible) to see if performance improves. Also, LOD expressions can be heavily impacted by join culling so we suggest you go back and re-read that section if you find your queries are running slowly when LOD expressions are used.

To better understand the mechanics of LOD expressions, read the "Understanding Level of Detail Expressions" whitepaper:

http://tabsoft.co/1MphVai

You can also read the "Top 15 LOD Expressions" blog post from Bethany Lyons that provides a number of examples:

http://tabsoft.co/1Mpi7Gq

Finally, there are a lot of blog posts from the community on the topic that are extremely helpful. A great curated list is available at *data + science*:

http://bit.ly/1MpkFV5

Calculations vs. native features

Sometimes, users create calculated fields to perform functions when these can easily be achieved with native features of Tableau. For example:

- Grouping dimension members together – consider using groups or sets;
- Grouping measure values together into ranged bands – consider using bins;
- Truncating dates to a coarser granularity e.g. month or week – consider using custom date fields;
- Creating a set of values that is the concatenation of two different dimensions – consider using combined fields;
- Changing the displayed values for dimension members – consider using aliases.

This is not always possible (for example, you might require variable width bins which is not possible using basic bins) but consider using the native features wherever possible. Doing so is often more efficient than a manual calculation, and as our developers continue to improve the performance of Tableau you will benefit from their efforts.

Impact of data types

When creating calculated fields, it is important to understand that the data type used has a significant impact on the calculation speed. As a general guideline

- Integers and Booleans are much faster than Strings

Strings calculations are very slow – often there are 10-100 base instructions that need to be executed for each calculation. In comparison, numeric and Boolean calculations are very efficient.

These statements are not just true for Tableau's calculation engine but also for most databases. Because basic and aggregate calculations are pushed down to the database, if these are expressed as numeric vs. string logic, they will execute much faster.

Performance techniques

Consider the following techniques to ensure your calculations are as efficient as possible:

Use Booleans for basic logic calculations

If you have a calculation that produces a binary result (e.g. yes/no, pass/fail, over/under) be sure to return a Boolean result rather than a string. As an example:

```
IF [Date]= TODAY() then "Today"
ELSE "Not Today"
END
```

This will be slow because it is using strings. A faster way to do this would be to return a Boolean:

```
[Date]=Today()
```

Then use aliases to rename the TRUE and FALSE results to "Today" and "Not Today".

String Searches

Imagine you want to be able to show all records where the product name contains some lookup string. You could use a parameter to get the lookup string from the user and then create the following calculated field:

```
IF FIND([Product Name],[Product Lookup])>0 THEN [Product Name] ELSE NULL END
```

This calculation is slow as this is an inefficient way to test for containment. A better way to do this would be to use the specific CONTAINS function as this will be converted into optimal SQL when passed to the database:

```
CONTAINS([Product Name],[Product Lookup])
```

However, in this case, the best solution would be to not use a calculated field at all, but rather use a wild card match quick filter.

Parameters for conditional calculations

A common technique in Tableau is to provide the end user with a parameter so they can select a value that will determine how a calculation is performed. Typically we want to give the user easy to understand options so we create the parameter as a string type. As we have already discussed, numerical calculations are much faster than string calculations so take advantage of the "display as" feature of parameters to show text labels but have underlying integer values for the calculation logic.

As an example, imagine you want to let the end user control the level of date aggregation shown on a view by selecting from a list of possible values. Many people would create a string parameter:

Value	Display As
Year	Year
Quarter	Quarter
Month	Month
Week	Week
Day	Day

Then use it in a calculation like this:

```
CASE [Parameters].[Date Part Picker]
  WHEN "Year" THEN DATEPART('year',[Order Date])
  WHEN "Quarter THEN DATEPART('quarter',[Order Date])
..
END
```

A better performing parameter would be an integer type with text labels, like this:

Value	Display As
1	Year
2	Quarter
3	Month
4	Week
5	Day

The calculation would then be as follows – see how the comparisons are numeric rather than string:

```
CASE [Parameters].[Date Part Picker]
  WHEN 1 THEN DATEPART('year',[Order Date])
  WHEN 2 THEN DATEPART('quarter',[Order Date])
..
END
```

BONUS: For this specific problem there is an even faster way to do this calculation. Using the original string-based parameter, create the calculation as follows:

```
DATEPART([Parameters].[Date Part Picker], [Order Date]))
```

This does away with all the conditional logic elements and allows us to simply substitute the DATEPART string directly into the calculation. This results in the most optimal SQL of all the options.

Date conversion

Users often have date data that is not stored in native date formats – e.g. it could be a string or a numeric timestamp. In Tableau 8.1, a new function DateParse() has been introduced to make it much easier to do these conversions. You now can simply pass in a formatting string:

```
DATEPARSE("yyyyMMdd", [YYYYMMDD])
```

Note that DateParse() is only supported on a subset of data sources:

- MySQL
- Oracle
- PostgreSQL
- Tableau Data Extract

If DateParse() is not supported for your data source an alternate technique to convert this to a proper Tableau date is to parse the field into a date string (e.g. "2012-01-01" – note that ISO strings are preferred as they stand up to internationalisation) and then pass it into the DATE() function.

If the originating data is a numeric field, converting to a string, then to a date is very inefficient. It is much better to keep the data as numeric and use DATEADD() and date literal values to perform the calculation.

For example, a slow calculation might be (converting an ISO format number field):

```
DATE(LEFT(STR([YYYYMMDD]),4)
  + "-" + MID(STR([YYYYMMDD]),4,2)
  + "-" + RIGHT(STR([YYYYMMDD]),2))
```

A much more efficient way to do this calculation is:

```
DATEADD( 'day', INT([yyyymmdd] )% 100 - 1,
  DATEADD( 'month', INT( [yyyymmdd]) % 10000  / 100  - 1,
       DATEADD( 'year', INT( [yyyymmdd] ) / 10000  - 1900,
       #1900-01-01# ) ) )
```

Note that the performance gains can be remarkable with large data sets. Over a 1 billion record sample, the first calculation took over 4 hours to complete, while the second took about a minute.

Logic statements

When working with complex logic statements remember that

- ELSEIF > ELSE IF

This is because a nested IF computes a second IF statement rather than being computed as part of the first. So this calculated field:

```
IF [Region] = "East" and [Customer Segment] = "consumer"
then "East-Consumer"
Else IF [Region] = "East" and Customer Segment] <>"consumer"
then "East-All Others"
END
END
```

would run much faster as:

```
IF [Region] = "East" and [Customer Segment] = "consumer"
       then "East-Consumer"
Elseif [Region] = "East" and [Customer Segment] <>"consumer"
       then "East-All Others"
end
```

but this is faster still:

```
IF [Region] = "East" THEN
  IF [Customer Segment] = "consumer" THEN
   "East-Consumer"
  Else "East-All Others"
  END
END
```

Similarly, avoid redundant logic checks. The following calculation:

```
IF [Sales] < 10 Then "Bad"
Elseif [Sales]>= 10 and [Sales] < 30 Then "OK"
Elseif [Sales] >= 30 then "Great"
END
```

would be more efficiently written as:

```
IF [Sales] < 10 Then "Bad"
Elseif [Sales] >= 30 then "Great"
```

```
else "OK"
END
```

Many little things

There are lots of little things you can do that can have impact of performance. Consider the following tips:

- Distinct counting values is one of the slowest aggregation types in almost all data sources. Use the COUNTD aggregation sparingly.
- Using parameters with a wide scope of impact (e.g. in a custom SQL statement) can affect cache performance.
- Filtering on complex calculations can potentially cause indexes to be missed in the underlying data source.
- Script functions like RAWSQL and SCRIPT_* for integrating with R can be slow, particularly if there are lots of values that need to be passed back and forth from the DBMS/R server.
- Use NOW() only if you need the time stamp level of detail. Use TODAY() for date level calculations.
- Remember that all basic calculations are passed through to the underlying data source – even literal calculations like label strings. If you need to create labels (e.g. for columns headers) and your data source is very large, create a simple text/Excel file data source with just one record to hold these so they don't add overhead on the big data source.

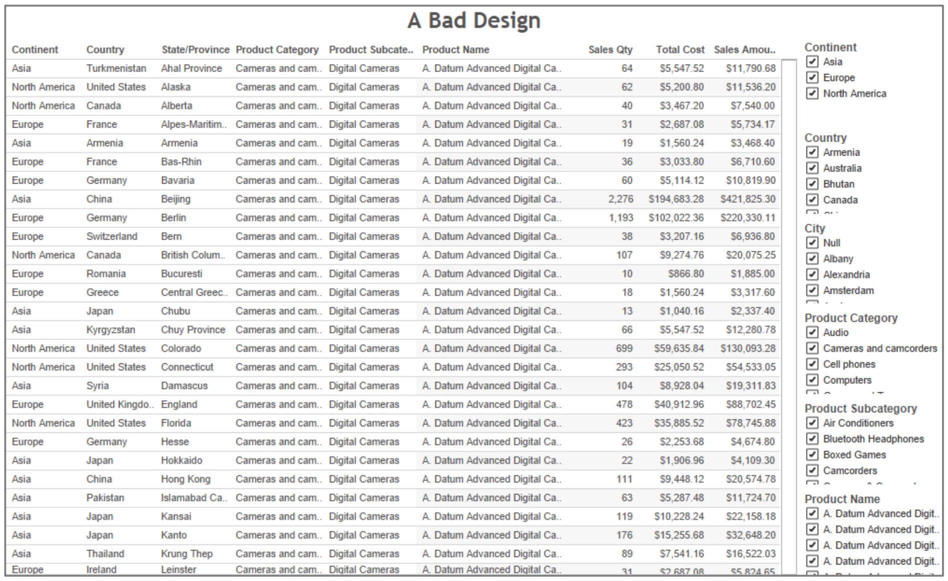
Is it the workbook?

Working in Tableau is a new experience for many users and there are design techniques and best practices they need to learn in order to create efficient workbooks. However, we find many new users try to apply old design approaches with Tableau and get lacklustre results. This section aims to address some the design principles that reflect best practice.

Design approaches - bad vs. good

With Tableau, you are creating an interactive experience for your end users. The final result delivered by Tableau Server is an interactive application that allows users to explore the data rather than just viewing it. So to create an efficient Tableau dashboard, you need to stop thinking as if you were developing a static report.

Here's an example of a dashboard type we see many new authors create – especially if they have previously worked in tools like Excel or Access, or if they come from a background of using "traditional" reporting tools. We start here with a tabular report showing "everything" and a series of filters that allow the user to refine the table until it shows the few records they are interested in:



A Bad Design

Continent	Country	State/Province	Product Category	Product Subcate..	Product Name	Sales Qty	Total Cost	Sales Amou..
Asia	Turkmenistan	Ahal Province	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	64	$5,547.52	$11,790.68
North America	United States	Alaska	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	62	$5,200.80	$11,536.20
North America	Canada	Alberta	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	40	$3,467.20	$7,540.00
Europe	France	Alpes-Maritim..	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	31	$2,687.08	$5,734.17
Asia	Armenia	Armenia	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	19	$1,560.24	$3,468.40
Europe	France	Bas-Rhin	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	36	$3,033.80	$6,710.60
Europe	Germany	Bavaria	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	60	$5,114.12	$10,819.90
Asia	China	Beijing	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	2,276	$194,683.28	$421,825.30
Europe	Germany	Berlin	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	1,193	$102,022.36	$220,330.11
Europe	Switzerland	Bern	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	38	$3,207.16	$6,936.80
North America	Canada	British Colum..	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	107	$9,274.76	$20,075.25
Europe	Romania	Bucuresti	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	10	$866.80	$1,885.00
Europe	Greece	Central Greec..	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	18	$1,560.24	$3,317.60
Asia	Japan	Chubu	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	13	$1,040.16	$2,337.40
Asia	Kyrgyzstan	Chuy Province	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	66	$5,547.52	$12,280.78
North America	United States	Colorado	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	699	$59,635.84	$130,093.28
North America	United States	Connecticut	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	293	$25,050.52	$54,533.05
Asia	Syria	Damascus	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	104	$8,928.04	$19,311.83
Europe	United Kingdo..	England	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	478	$40,912.96	$88,702.45
North America	United States	Florida	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	423	$35,885.52	$78,745.88
Europe	Germany	Hesse	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	26	$2,253.68	$4,674.80
Asia	Japan	Hokkaido	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	22	$1,906.96	$4,109.30
Asia	China	Hong Kong	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	111	$9,448.12	$20,574.78
Asia	Pakistan	Islamabad Ca..	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	63	$5,287.48	$11,724.70
Asia	Japan	Kansai	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	119	$10,228.24	$22,158.18
Asia	Japan	Kanto	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	176	$15,255.68	$32,648.20
Asia	Thailand	Krung Thep	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	89	$7,541.16	$16,522.03
Europe	Ireland	Leinster	Cameras and cam..	Digital Cameras	A. Datum Advanced Digital Ca..	31	$2,687.08	$5,824.65

Continent
- ✓ Asia
- ✓ Europe
- ✓ North America

Country
- ✓ Armenia
- ✓ Australia
- ✓ Bhutan
- ✓ Canada

City
- ✓ Null
- ✓ Albany
- ✓ Alexandria
- ✓ Amsterdam

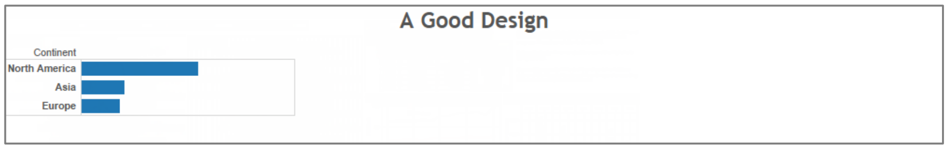
Product Category
- ✓ Audio
- ✓ Cameras and camcorders
- ✓ Cell phones
- ✓ Computers

Product Subcategory
- ✓ Air Conditioners
- ✓ Bluetooth Headphones
- ✓ Boxed Games
- ✓ Camcorders

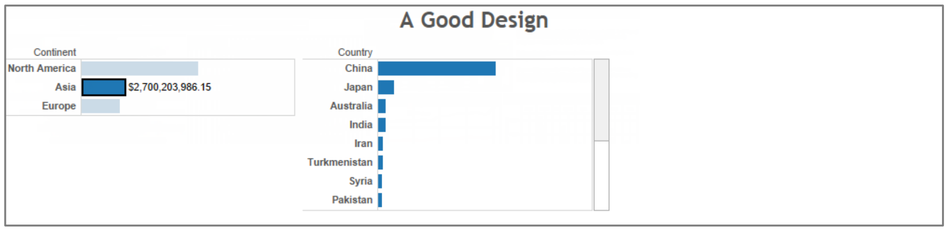
Product Name
- ✓ A. Datum Advanced Digit..
- ✓ A. Datum Advanced Digit..
- ✓ A. Datum Advanced Digit..
- ✓ A. Datum Advanced Digit..

This is not a "good" Tableau dashboard (in fact, it's not a "good" dashboard at all). At worst it's a glorified data extract process because the user wants to take the data to another tool like Excel for further analysis and charting. At best it indicates that we don't really understand how the end user wants to explore the data, so we take the approach of "based on your starting criteria, here's everything… and here are some filter objects so you can further refine the result set to find what you're really after".

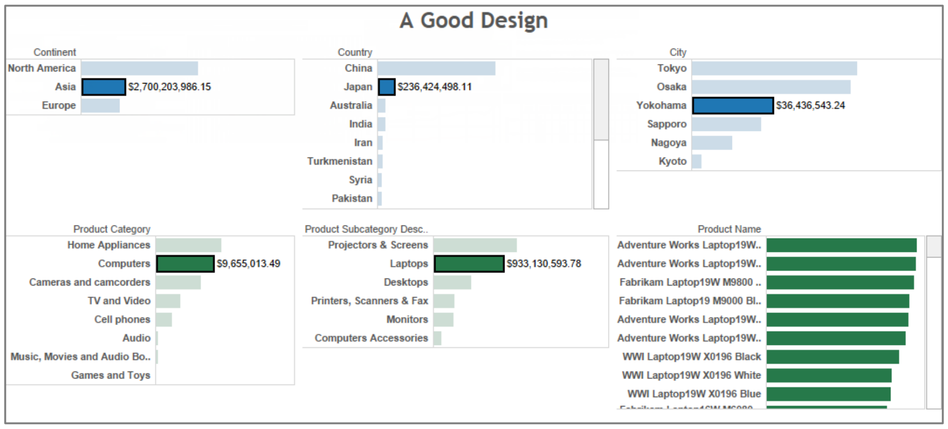
Now consider the following reworking – it's exactly the same data. We start here at the highest level of aggregation:



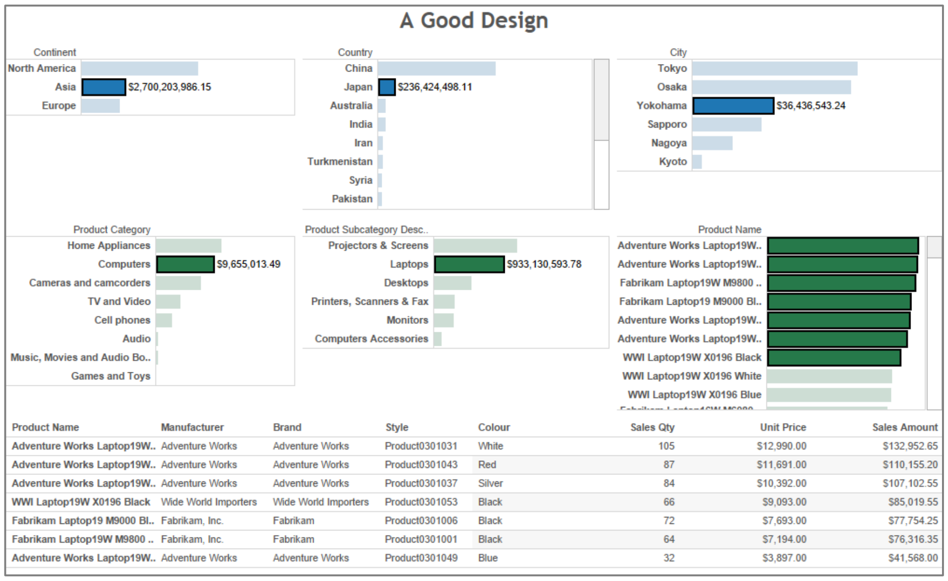
Selecting one or more of the elements shows the next level of detail:



We keep doing this, each time revealing more detail:



Until we finally reveal the ultimate level – the same data that was shown in the crosstab dashboard above.



Product Name	Manufacturer	Brand	Style	Colour	Sales Qty	Unit Price	Sales Amount
Adventure Works Laptop19W..	Adventure Works	Adventure Works	Product0301031	White	105	$12,990.00	$132,952.65
Adventure Works Laptop19W..	Adventure Works	Adventure Works	Product0301043	Red	87	$11,691.00	$110,155.20
Adventure Works Laptop19W..	Adventure Works	Adventure Works	Product0301037	Silver	84	$10,392.00	$107,102.55
WWI Laptop19W X0196 Black	Wide World Importers	Wide World Importers	Product0301053	Black	66	$9,093.00	$85,019.55
Fabrikam Laptop19 M9000 Bl..	Fabrikam, Inc.	Fabrikam	Product0301006	Black	72	$7,693.00	$77,754.25
Fabrikam Laptop19W M9800 ..	Fabrikam, Inc.	Fabrikam	Product0301001	Black	64	$7,194.00	$76,316.35
Adventure Works Laptop19W..	Adventure Works	Adventure Works	Product0301049	Blue	32	$3,897.00	$41,568.00

Don't focus on the presentation of the data (that is important but it's a topic for later). Instead, think about the experience of using this dashboard. Notice how it flows in a natural path, left to right, top to bottom. There can be a lot of data underlying this example but the dashboard guides the end user to drill down gradually to find the focused set of detail records they seek.

The key difference to the two examples provided about is how they guide the end user through the analytic process. The first example starts wide (showing all the possible records you could look at) and then makes the end user reduce the number of records displayed by applying filters. There are inherent problems with this technique:

- The initial query that must be run before anything is shown to the end user is essentially the biggest query you can ask – "give me all records". Over any real-world data set this is going to take a substantial time to execute and stream back to the Tableau engine. The "first contact" experience is critical for setting an end-user's perception of the solution and if it takes more than a few seconds before anything happens the perception will be a negative one.
- Creating a view with hundreds of thousands to millions of marks (each cell in a crosstab is called a mark) requires a lot of CPU and memory. It also takes time – adding to the negative perception of system responsiveness. On Tableau Server, having many people all generating large crosstabs can result in slow performance, and in a worst case scenario the system could run out of memory. This can cause server stability issues, errors and all kinds of unpleasant experiences for end users. Of course you could add more memory to the server to minimise this but this is treating the symptom, not the cause.
- Finally, the users have no contextual guidance on whether their initial set of filters will be too wide or too narrow. How is a report user to know that if they check all available categories their initial query will return tens of thousands of records and exhaust all available RAM on the server? They can't, other than through painful experience.

Contrast this with the second approach where our initial query shows the highest level of aggregation only:

- The initial query that must be run is highly aggregated and consequently returns only a handful of records. For a well-designed database this is a very efficient activity so the "first contact" response time is very fast, leading to a positive perception of the system. As we drill down, each subsequent query is both aggregated and constrained by the selections from the higher level. They continue to be fast to execute and return to the Tableau engine.
- Although we have more views when the dashboard is fully completed, each view only shows a few dozen marks. The resources necessary to generate each of these views, even when many end users are active on the system, are trivial and it is now much less likely that the system will run out of memory.
- Finally, you can see that for the higher "navigation" levels we've taken the opportunity to show the volume of sales in each category. This gives the user some context on whether this selection contains many records or few. We've also used colour to indicate the profitability of each category. Now this becomes extremely relevant as you will be able to see which specific areas require attention, rather than just navigating blindly.

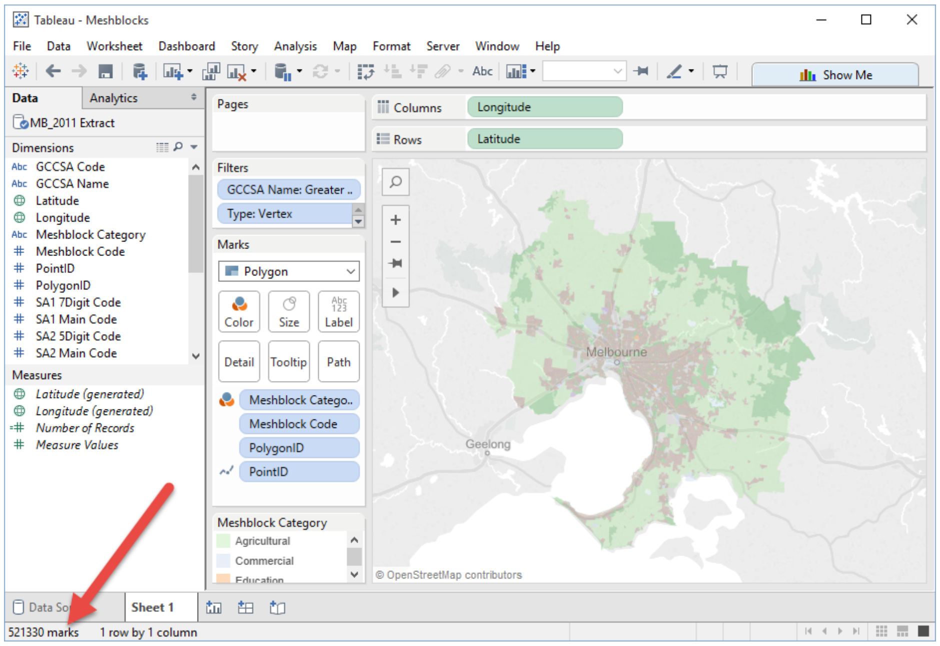
Reduce the complexity

A common mistake for new users is they create dashboards that are overly "complex". Perhaps they are trying to recreate a document they had previously used from another tool, or perhaps they are trying to create something that is specifically designed to be a printed report. The end result is a workbook that performs slowly and inefficiently.

The following points are common contributors to complexity:

Too many marks per viz

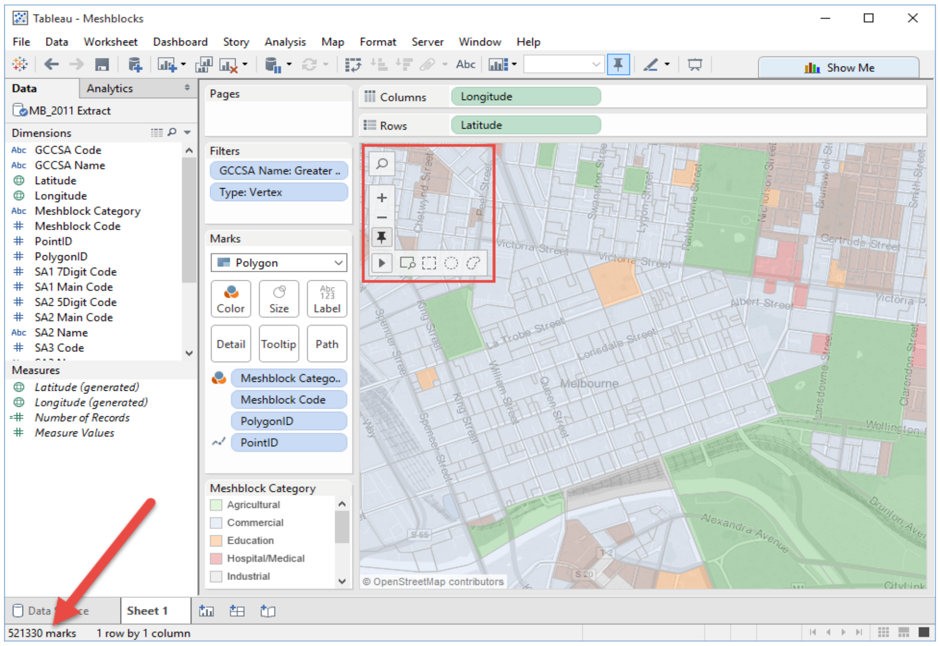
The first point to look at is how many data points are being rendered in each viz. You can easily find this by looking at the status bar of the Tableau Desktop window:



While there is no hard and fast rule on what defines "too many marks" be aware that more marks means more CPU and RAM is required to render them. Watch out for large crosstabs (which is a slow viz type to begin with) and/or maps with complex custom polygons.

Zooming without filtering

When users zoom in on a viz with a large number of marks, we do not filter out the marks you can't see. All we are changing is the viewport over the data. The total number of marks being manipulated is not changing as you can see in the following image:



If you only need a subset of the data, filter out the unwanted data and let the auto-zoom feature of Tableau set the viewport.

Too many viz per dashboard

Another common mistake new users make is to try and put lots and lots of charts/worksheets on a single dashboard.

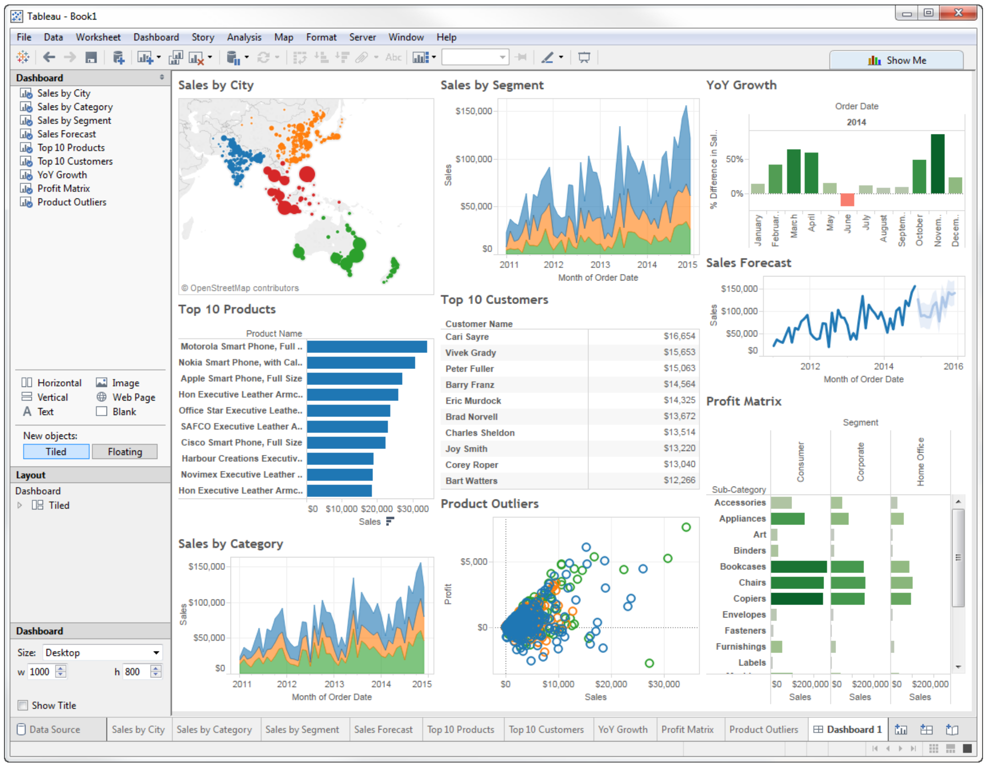


Tableau - Book1
File Data Worksheet Dashboard Story Analysis Map Format Server Window Help
Show Me
Dashboard
Sales by City
Sales by Category
Sales by Segment
Sales Forecast
Top 10 Products
Top 10 Customers
YoY Growth
Profit Matrix
Product Outliers
Horizontal Image
Vertical Web Page
Text Blank
New objects:
Tiled Floating
Layout
Dashboard
Tiled
Size: Desktop
w 1000 h 800
Show Title
Sales by City
© OpenStreetMap contributors
Sales by Segment
YoY Growth
Sales Forecast
Top 10 Products
Product Name
Motorola Smart Phone, Full ..
Nokia Smart Phone, with Cal..
Apple Smart Phone, Full Size
Hon Executive Leather Armc..
Office Star Executive Leathe..
SAFCO Executive Leather A..
Cisco Smart Phone, Full Size
Harbour Creations Executiv..
Novimex Executive Leather ..
Hon Executive Leather Armc..
Sales
Sales by Category
Product Outliers
Profit Matrix
Segment
Consumer Corporate Home Office
Sub-Category
Accessories
Appliances
Art
Binders
Bookcases
Chairs
Copiers
Envelopes
Fasteners
Furnishings
Labels
Month of Order Date
Data Source Sales by City Sales by Category Sales by Segment Sales Forecast Top 10 Products Top 10 Customers YoY Growth Profit Matrix Product Outliers Dashboard 1

Top 10 Customers

Customer Name	
Cari Sayre	$16,654
Vivek Grady	$15,653
Peter Fuller	$15,063
Barry Franz	$14,564
Eric Murdock	$14,325
Brad Norvell	$13,672
Charles Sheldon	$13,514
Joy Smith	$13,220
Corey Roper	$13,040
Bart Watters	$12,266

Remember that each worksheet is going to run one or more queries against the data sources, so the more sheets the longer it is going to take to render the dashboard. Take advantage of the fact that Tableau is designed to deliver interactive dashboards to end users, so spread the data out across multiple dashboards/pages.

Too many quick filters

Quick filters are a very powerful feature of Tableau that allow us to create rich, interactive dashboards for end users. However, each quick filter requires a query in order to populate the options so adding too many to your dashboard can unexpectedly cause the dashboard to take a long time to render.

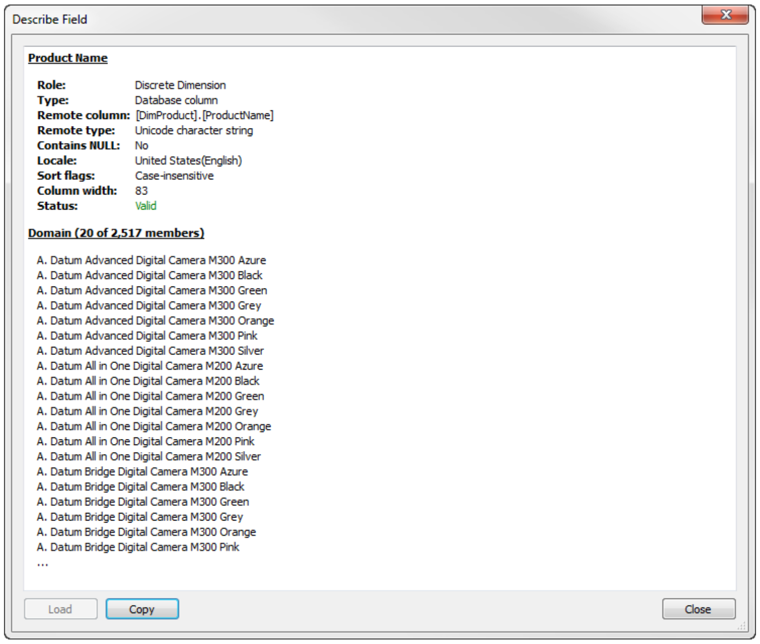
Also, when you use "show relevant values" on a quick filter, it requires a query to update the shown values each time other quick filters are changed. Use this feature sparingly.

Getting to know your data

One of the first things we do when presented with a new set of data is to start to explore it in order to get a "feel" for the structure and the kinds of values in each dimension. Often this is done by dragging a dimension out onto the rows shelf so we can eyeball the data. When we do this, we require Tableau to render the data as a crosstab which is one of the slowest viz types. If the dimension has a particularly high cardinality this can cause the "computing layout" phase to take a long time.

Use describe field

An alternative is to use the "Describe field" option (right click on the field in the data pane) and load the domain. The information is presented as follows:



Product Name

Role:	Discrete Dimension
Type:	Database column
Remote column:	[DimProduct].[ProductName]
Remote type:	Unicode character string
Contains NULL:	No
Locale:	United States(English)
Sort flags:	Case-insensitive
Column width:	83
Status:	Valid

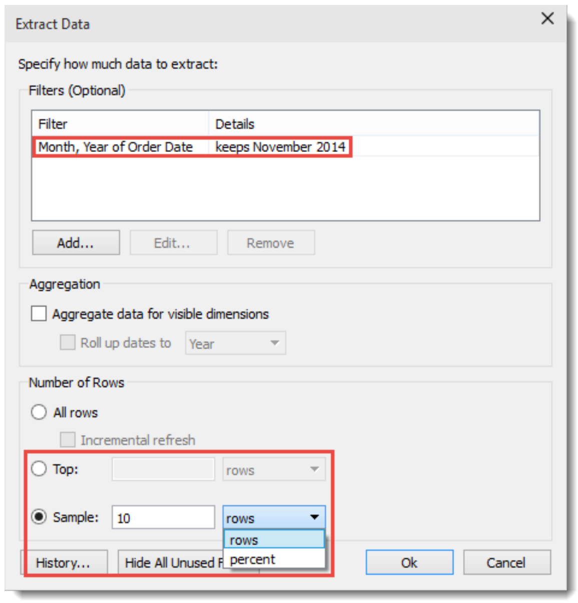
Domain (20 of 2,517 members)

```
A. Datum Advanced Digital Camera M300 Azure
A. Datum Advanced Digital Camera M300 Black
A. Datum Advanced Digital Camera M300 Green
A. Datum Advanced Digital Camera M300 Grey
A. Datum Advanced Digital Camera M300 Orange
A. Datum Advanced Digital Camera M300 Pink
A. Datum Advanced Digital Camera M300 Silver
A. Datum All in One Digital Camera M200 Azure
A. Datum All in One Digital Camera M200 Black
A. Datum All in One Digital Camera M200 Green
A. Datum All in One Digital Camera M200 Grey
A. Datum All in One Digital Camera M200 Orange
A. Datum All in One Digital Camera M200 Pink
A. Datum All in One Digital Camera M200 Silver
A. Datum Bridge Digital Camera M300 Azure
A. Datum Bridge Digital Camera M300 Black
A. Datum Bridge Digital Camera M300 Green
A. Datum Bridge Digital Camera M300 Grey
A. Datum Bridge Digital Camera M300 Orange
A. Datum Bridge Digital Camera M300 Pink
...
```

This provides a description of the data type as reported by the data source as well as a sample of the dimension member values but doesn't attempt to display all the members. Consequently, it can be a more efficient way of familiarising yourself with the data.

Use a subset of data while authoring

When you are working with very large data sets you might only need a sample of the data to create your workbook – either a filtered subset (say one month of data) or a random sample of data.



Filter	Details
Month, Year of Order Date	keeps November 2014

Aggregate data for visible dimensions: unchecked
All rows: not selected
Incremental refresh: unchecked
Top: not selected, rows
Sample: selected, 10 rows (options: rows, percent)

Be aware that in some databases "top" can be interpreted to mean "best" and "sample" can be interpreted to mean "statistically relevant sample". In these cases the workload on the database to identify and select the subset of rows can be significant, but the result is still a subset of data that will be enough to create your analytical content without having to deal with the performance challenges the full data set might cause. When you are ready to bring the power of Tableau to bear on the complete data set, deselect the Use Extract menu item.

Even if you are authoring against a live connection you can reduce the volume of data that will be queried by applying a data source filter across one or more dimensions (e.g. restrict the data to the last 3 months, or the top N customers). Or if you are using a custom SQL statement, consider adding a SAMPLE or TOP clause until you are ready to publish the workbook.

Is it a Desktop vs. Server thing?

There are times when a workbook will perform well in single-user testing but when deployed to Tableau Server (or Tableau Online) for many users to consume, it performs poorly. The following section identifies areas where the differences between single- and multi-user scenarios can make a difference.

General guidelines

Upgrade to the latest release

The development team at Tableau are constantly working to improve the performance and usability of our software. Upgrading to the latest release of Tableau Server can sometimes yield significant improvements in performance without requiring any changes workbook. For example – many customers are reporting 3x (and more) performance improvements in their workbooks just by upgrading from V8 to V9. Of course, this isn't an issue with Tableau Online as it is always being upgraded to the latest versions as they are released.

Use the latest data source drivers

Same as above, database vendors are also working to improve their offerings. Make sure you are using the latest version of the appropriate data source driver as listed on the following web page:

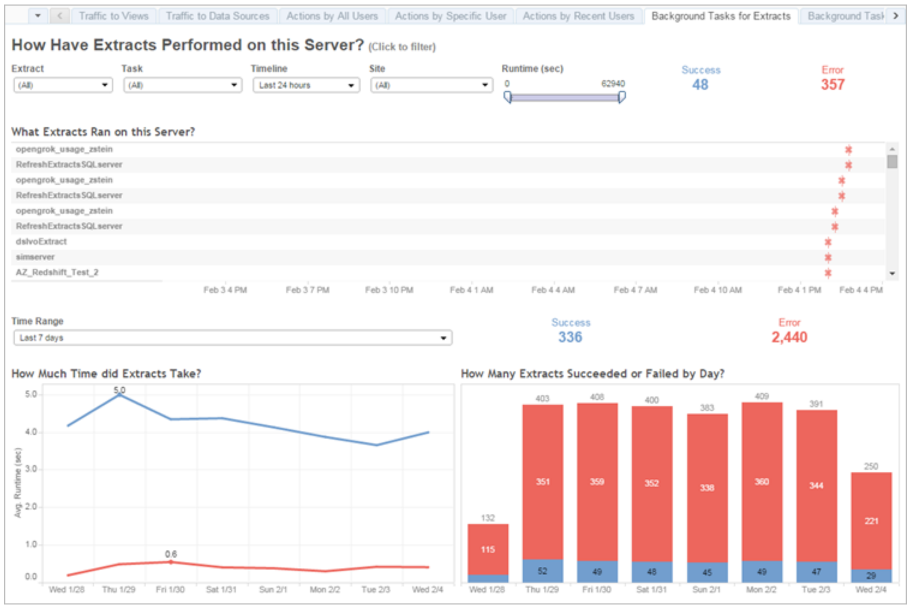
http://tabsoft.co/1RjD4sy

Test on the server using Tableau Desktop

Sometimes you might find yourself in the situation where you have a workbook that performs well when run in Tableau Desktop on your workstation but runs poorly when viewed through Tableau Server. Opening the workbook with a copy of Tableau Desktop installed on the Tableau Server machine can help determine if it is an issue with your workbook or if it is a configuration issue with the server. This approach can identify if it is a driver incompatibility issue, or if there is a problem with the network such as bad routing, DNS or proxy configurations.

Schedule refreshes for off-peak hours

If server performance is slow, use the Background Tasks administrative view to see your current refresh task schedules.



If you can schedule refreshes for off-peak hours, do so. If your hardware configuration permits you can also move the Backgrounder process(es) to a dedicated worker node.

Check the VizQL session timeout limit

The default VizQL session timeout limit is 30 minutes. Even if a VizQL session is idle, it is still consuming memory and CPU cycles. If you can make do with a lower limit, use tabadmin to change the vizqlserver.session.expiry.timeout setting:

http://tabsoft.co/1f17Bd6

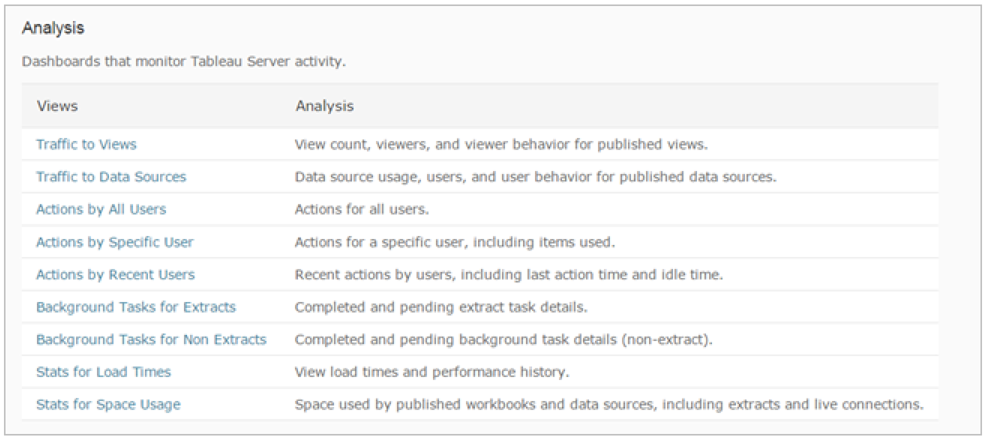
Assess your process configuration

Tableau Server is divided into many different components called services. While their default configuration is designed to work for a broad range of scenarios, you can also reconfigure them to achieve different performance goals. Specifically, you can control on which machines the processes run and how many are run. See Improving Server Performance for guidelines for one-, two-, and three-machine deployments:

http://tabsoft.co/1f17tKK

Monitoring Tableau Server

Tableau Server comes with several views for administrators, to help monitor activity on Tableau Server. The views are located in the Analysis table on the server's Maintenance page:



Analysis

Dashboards that monitor Tableau Server activity.

Views	Analysis
Traffic to Views	View count, viewers, and viewer behavior for published views.
Traffic to Data Sources	Data source usage, users, and user behavior for published data sources.
Actions by All Users	Actions for all users.
Actions by Specific User	Actions for a specific user, including items used.
Actions by Recent Users	Recent actions by users, including last action time and idle time.
Background Tasks for Extracts	Completed and pending extract task details.
Background Tasks for Non Extracts	Completed and pending background task details (non-extract).
Stats for Load Times	View load times and performance history.
Stats for Space Usage	Space used by published workbooks and data sources, including extracts and live connections.

More information on these views can be found at the following link:

http://tabsoft.co/1RjCCL2

Additionally, custom administrative views can be created by connecting to the PostgreSQL database that makes up part of the Tableau repository. Instructions can be found here:

http://tabsoft.co/1RjCACR

Is it environmental factors?

This final section addresses other factors that in many cases are harder to alter. For example, the choice of OS, the hardware on which Tableau is running, etc. – all these things can have an effect on the performance of Tableau.

Russell Christopher has done a couple of fantastic posts on his Tableau Love blog that explores the impact of # cores, CPU speed and IOPS on overall performance. While his experiment was done on AWS it is applicable to all environments:

http://bit.ly/1f17oa3
http://bit.ly/1f17n5O

General guidelines

Use a 64-bit Operating System

Although Tableau Server can run on 32-bit Microsoft operating systems, for the best performance choose a 64-bit edition. With Tableau 8.1 and later all components of Tableau Server can be run as native 64-bit processes. This means all components can address large amounts of RAM and our benchmarking tests indicate that the 64-bit version of Tableau scales significantly better than the 32-bit version.

Customers are strongly advised to install their production environments of Tableau Server using the 64-bit version.

Add more cores and memory

Regardless of whether you're running Tableau Server on one machine or several, the general rule is that more CPU cores and more RAM will give you better performance. Make sure you meet Tableau Server's recommended hardware and software requirements (http://tabsoft.co/1I0RDNs) and see When to Add Workers & Reconfigure (http://tabsoft.co/1I0RBVL) to assess whether you should add additional machines.

Since November 2014, very early in the Tableau 9.0 release cycle, we started performance and scalability testing of new features while they were still being developed. We iteratively incorporated design feedback for new features into the performance and load testing for Tableau Server 9.0 to understand how it would scale across a variety of configurations and workloads.

This white paper explains the scalability tests, methodology and test results.

http://tabsoft.co/1L4Eqnv

Physical vs. virtual infrastructure

Many customers are now deploying Tableau Server on virtualised infrastructure. Virtualisation always has a performance overhead so it will not be as fast as installing on bare metal, however modern hypervisor technology has reduced this overhead significantly.

When installing on virtual machines it is important to ensure that Tableau Server is given dedicated RAM and CPU resources. If it is contesting with other virtual machines for resources on the physical host, performance can be significantly affected.

A good resource for tuning virtual deployments is the “Deploying Extremely Latency-Sensitive Applications in vSphere 5.5” whitepaper from VMWare. Page 15 gives a high level list of the best practices for latency sensitive applications:

http://vmw.re/1L4Fyr1

SSD vs. rotational disk

Some actions of Tableau are heavily IO intensive (e.g. loading/creating/refreshing a data extract) and will benefit from the use of SSDs over rotational disks. The Tableau Love posts linked above show the impact that IOPS has for Tableau environments using data extracts.

Faster network interface

If the data is remote from the Tableau workstation or if you are publishing data or workbooks to Tableau Server, Tableau will benefit from a faster network interface and from a network connection with low latency.

Choice of browser

Tableau heavily utilises JavaScript so the speed of the JavaScript interpreter in the browser has an impact of the speed of rendering. For modern browsers it’s a close race in a rapidly evolving field, but it pays to consider whether the version and performance of your browser can be affecting your experience.