Best Practices for Designing Efficient Tableau Workbooks

Second Edition

Alan Eldridge

Tableau Software

18 March 2014

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 8.1 with some comments about changes expected in Tableau 8.2. 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;
- the query;
- the calculations;
- the visualisation design;
- some differences between Tableau Desktop and Server.

Why do we care?

In our extensive experience, most performance problems that customers experience are workbook related. If we fix these (or better yet, prevent them in the first place through education) and have optimal, efficient workbooks then we fix the problems.

Basic principles

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

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 very slowly.

If you find yourself with a workbook like this, consider breaking it into several separate files. This is much easier to do in 8.1 as you can now copy dashboards between workbooks. 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.

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

If your Tableau workbook is based on a query that is slow to run no matter what tool you use to submit it then your workbook will in turn 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 a) there are multiple users all sharing the server resources to generate workbooks simultaneously; and b) the server has to do the work to render the dashboards and charts rather than this being done on the client workstation.

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.

Part 1 - Is it the workbook design?

Working in Tableau is a new experience for many users and there are 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 document aims to develop and promote best practices for the use of a new category of business data analysis and a new category of data analysis tools.

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.

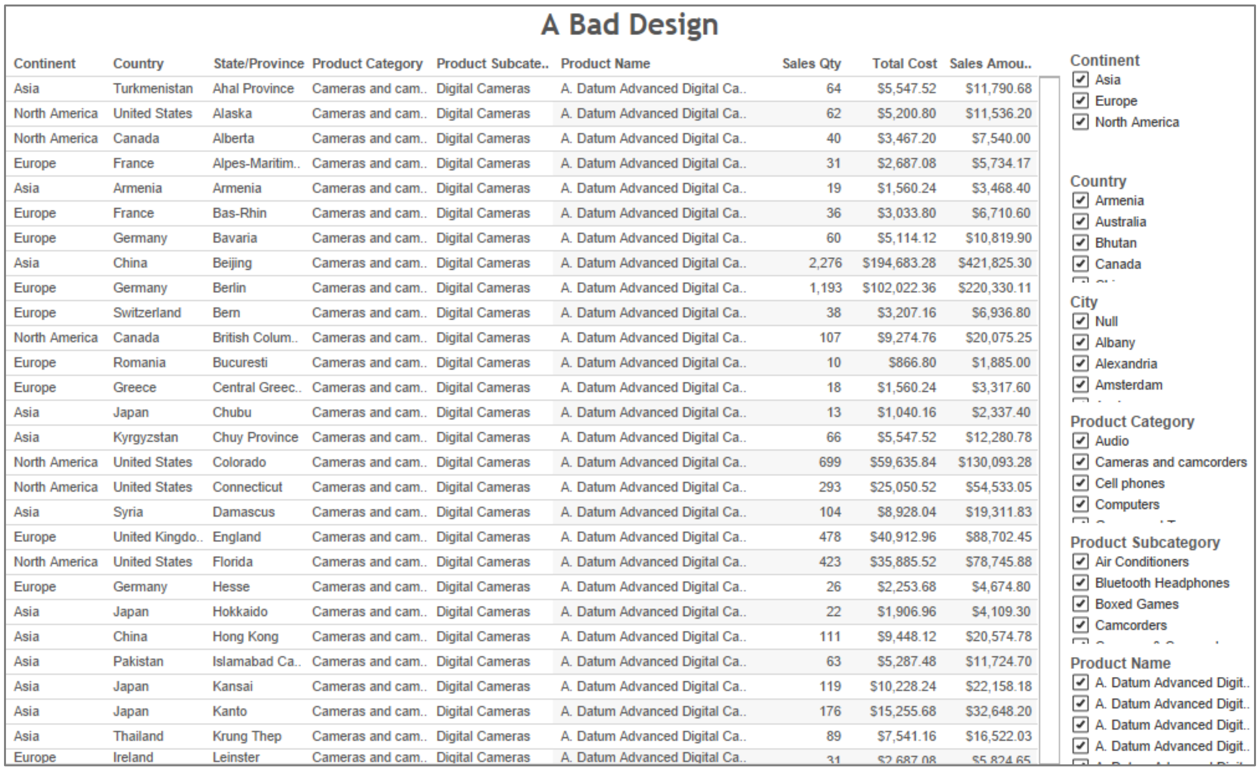
We suggest you consider revisiting your requirements or consider another approach if:

- 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. Tableau 8 introduced the concept of report subscriptions but this is a per-user pull model vs. report bursting.
- The primary 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 primary 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.

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

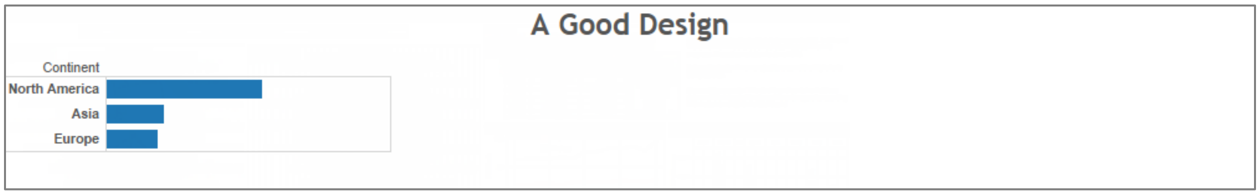
Product Category
- ✓ Audio
- ✓ Cameras and camcorders
- ✓ Cell phones
- ✓ Computers

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

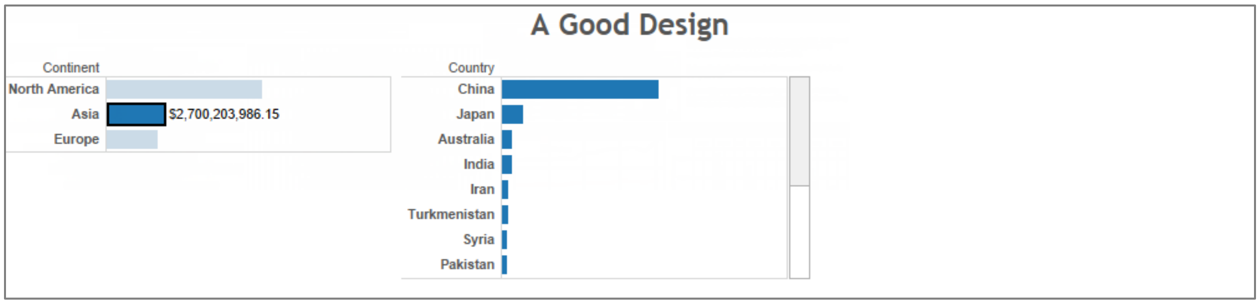
Product Name
- ✓ A. Datum Advanced Digit..
- ✓ A. Datum Advanced Digit..
- ✓ A. Datum Advanced Digit..
- ✓ A. Datum Advanced Digit..

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”.

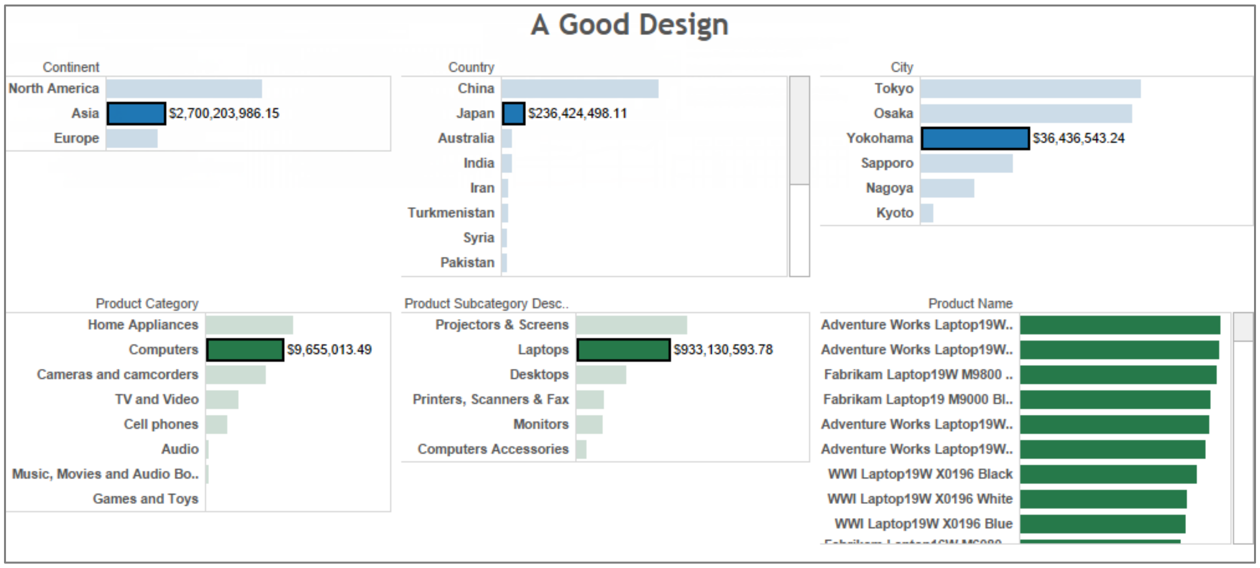
Now consider the following reworking – it’s exactly the same data. We start here at the highest level of aggregation:



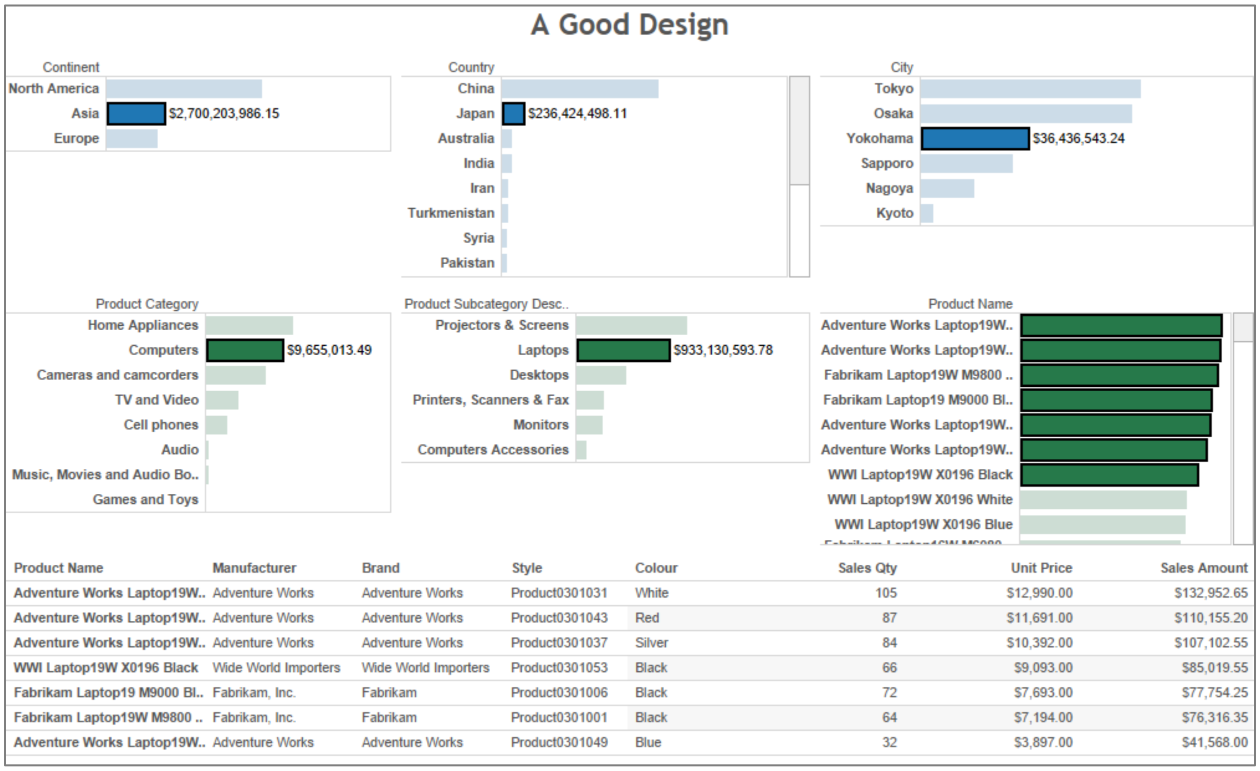
Selecting one or more of the elements shows the next level of detail:



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 is technically referred to as "a very bad thing" and can cause server stability issues, errors and all kinds of unpleasant experiences for end users. Sure, 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.

Part 2 - Is it the data connection?

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,
- Relational data sources,
- OLAP data sources,
- Web-based data sources.

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

Note that in Tableau 8.2 we are introducing support for Tableau Desktop on Mac OS X. Initially the set of supported data sources on Mac will not be the same as on Windows – minimizing the differences between the platforms is something we will work towards but there could be cases where some data sources are only supported on one platform.

Files

This category covers all file-based data formats – text files such as CSV, Excel spreadsheets and MS Access being the most common. Business users are often dealing with data in this format as if they are common for data moving outside the "governed" data sets.

In general we would advise users to import these data source types into the Tableau fast data engine. This will make queries perform much faster and it 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.

In most cases, Tableau uses the Microsoft JET driver to connect to, query and extract these data sources. There are several common issues that end users encounter because of limitations of this interface layer:

- The MS JET driver has limitations on the number and size of columns that can be read. Files read cannot have more than 255 columns, and text fields are truncated at 255 characters. The following forum thread discusses these limitations: http://community.tableausoftware.com/thread/109727
- Because text files and Excel spreadsheets are not "typed" data sources, the MS JET driver scans the top N rows of the file to determine what type of data is contained in each column. Sometimes this is insufficient to correctly identify the type of data stored in a source field. E.g. it might only see values with numbers and determine the data type to be numeric, but further down the table there are alphanumeric entries. The following knowledge base article identifies several workarounds to this problem: http://kb.tableausoftware.com/articles/knowledgebase/tableau-does-not-correctly-recognize-excel-columns
- The MS JET driver does not support COUNT DISTINCT and MEDIAN as aggregations. When using a live connection to a file data source these functions are not available through the Tableau Desktop interface. The simplest workaround is to extract the data into the Tableau fast data engine which does support these functions.

- The MS JET driver cannot read files greater than 4GB in size. While this isn't a problem for Excel or Access files as they have similar file size limits, it can be an issue with extremely large text files. To address this, Tableau introduced a dedicated text parser in V6.1. This allows text files of unrestricted size to be imported into Tableau's fast data engine – you wouldn't want to query a file like this directly as performance would be dreadful. There are limitations on its use though – specifically that it is only used when we are doing a straight-forward read of the text file. If we have any calculations or filters then the import process uses the MS JET driver instead.

In Tableau 8.2 we are releasing a new connector layer for file data sources - specifically text files and Excel workbooks. In part this new connector is to overcome many of the above issues with file data sources but also to support Tableau Desktop for Mac OS X.

This new connector does not use MS JET so the above limitations will not apply. However there will be some differences:

	Text	Excel
Gained	• Much better parsing in general • Much better field type detection – more rows, better logic • Auto-detection of delimiter and code page • Better auto-detection of whether the data has headers • All data engine functions that aren't in JET (COUNTD, MEDIAN, PERCENTILE, DATEPARSE, …) • In/Out Sets • Combined Sets • Fractional seconds as dates • YYYYMMDD as date • X% as numbers • More than 255 columns • Text qualifier (quote) characters • Explicit parsing locale • Works on Mac	• Much better parsing in general • Much better field type detection – more rows, better logic • Auto-detection of whether the data has headers • Better field names • All data engine functions that aren't in JET (COUNTD, MEDIAN, PERCENTILE, DATEPARSE, …) • In/Out Sets • Combined Sets • More than 255 columns • Parsing of columns of integer as integer instead of double • Parsing of columns of date without time as date instead of date & time • Sheets with single quotes in them now work (example of a pretty obscure gain) • Works on Mac
Lost	• JET SQL – RAWSQL and custom SQL • Fixed width file formats in schema.ini • Right and Full joins • Non-equi joins	• JET SQL – RAWSQL and custom SQL • Support for Excel 3 & 4 & 5 files • Support for XLSB • Right and Full joins • Non-equi joins

For users running Tableau on Windows there will be the option to revert to the legacy MS JET based connector if required, however this option will not be available for users running Tableau on Mac.

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 below.

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

Indexes

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

- Make certain you have indexes on all columns that are part of table JOINs.
- Make certain you have indexes on any column used, within Tableau, in a FILTER.
- 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. I.e. 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.

Referential Integrity

Referential integrity information helps Tableau understand how data across multiple tables are related. Ensuring this is configured correctly will help Tableau to construct efficient queries:

- EXPLICITLY define PRIMARY KEYs on all tables, if possible.
- EXPLICITLY define all FOREIGN KEY relationships. This enables Tableau to bypass many of its integrity checks, since it knows the database is doing the verification.

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, ranged 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.

NULLS

Having NULL values in dimension columns can reduce the effectiveness of indexes in many databases. Where possible, define your dimension columns as NOT NULL.

Calculations

In Tableau, calculated fields are expressed as part of the query that is pushed to the database for processing. If you have very complex calculations it is possible that the generated SQL is not the most optimal form and it could be improved.

In these situations, you can either create a custom SQL statement to hand-tune the expression (however this has its own challenges that will be discussed later) or you could implement the expression in a view or function within the database.

Custom SQL

While the ability to create a data connection on a chunk of custom SQL can be very powerful, it has its limitations when it comes to performance. In contrast to single or multiple tables, the custom SQL is never deconstructed and is always executed atomically. We end up with situations where the database is asked to process:

```
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)
```

A good recommendation is to use custom SQL in conjunction with Tableau's fast data engine. 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. Alternately, see the following section on context filters for another technique to force the database to materialise the results of the custom SQL into a temporary table.

Since Tableau 8 it is possible to use parameters in custom SQL - this 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 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.

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.

Stored procedures
In Tableau 8.1 support has been added for stored procedures in Sybase ASE, SQL Server and Teradata. If you have highly complex logic you need to implement (e.g. based on input from the user) then this might be an efficient alternative to custom SQL.

If you do use stored procedures to define a data source for Tableau, keep the following in mind:

- If a stored procedure returns more than one result set, Tableau reads the first one and ignores the rest.
- If a stored procedure has output parameters, Tableau filters out the stored procedure so that it is not listed in the Connection dialog box.
- Stored procedures that have parameters of a non-scalar type are excluded.
- Result set columns that don't have matching types in Tableau (such as varbinary, geometry, and hierarchyid) are logged. If all result set columns map to unknown data types, Tableau displays a message "The result set... has no usable columns."
- Stored procedures that return no result sets are listed in the Connection dialog box but fail if selected.
- If no value is provided for a parameter that the stored procedure requires, an error occurs. Tableau cannot determine in advance whether parameters are required.
- Tableau does not perform any transaction management for stored procedures. That is, stored procedure writers must not depend on Tableau to start transactions before invoking stored procedures, or to commit them afterwards.

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.

Hadoop
In Tableau 8.1, there are three supported Hadoop distributions (along with the supported interfaces):

- Cloudera Hadoop
    - Hive Server
    - Impala

    - Beeswax Server
    - Beeswax Server and Kerberos

- Hortonworks Hadoop Hive

    - HiveServer
    - HiveServer2
    - Hortonworks Hadoop Hive

- MarR Hadoop Hive

    - HiveServer
    - HiveServer2

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). Impala is generally much faster than Hive although this is a rapidly changing area and new technologies such as Apache Shark (designed to be Hive compatible), Hortonworks Stinger and others may reduce or eliminate the differences.

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://kb.tableausoftware.com/articles/knowledgebase/hadoop-hive-performance

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.

See Appendix A for more details.

New in Tableau 8.1 - you can now extract data from SAP BW cubes into Tableau's data engine. 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.

Cloud

Tableau currently supports the following web-based data sources:

- Salesforce.com
- Google Analytics
- oData
- 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
- Google BigQuery
- Microsoft SQL Server Azure (via the SQL Server driver)

While these are also cloud-based data sources they operate like a relational data source and allow both live connections and extracts.

Salesforce

When you connect to Salesforce using Tableau, the data is automatically extracted into a Tableau Data Extract File. In some cases, certain fields cannot be extracted because of character limits. Specifically, text fields that are greater than 4096 characters and calculated fields will not be included in the extract. If you have calculated fields in your data, you will need to recreate them in Tableau after creating the extract.

In addition, 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://support.google.com/analytics/bin/answer.py?hl=en&answer=1042498

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 file…" 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.

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 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://www.theinformationlab.co.uk/2011/01/20/tableau-extracts-what-why-how-etc/

Note – extracts are not a replacement for a data warehouse, rather a complement. While they can be used to collect and aggregate data over time (e.g. 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. However, in Tableau 8.1 we have introduced the ability to 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 MS 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 automatially 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.

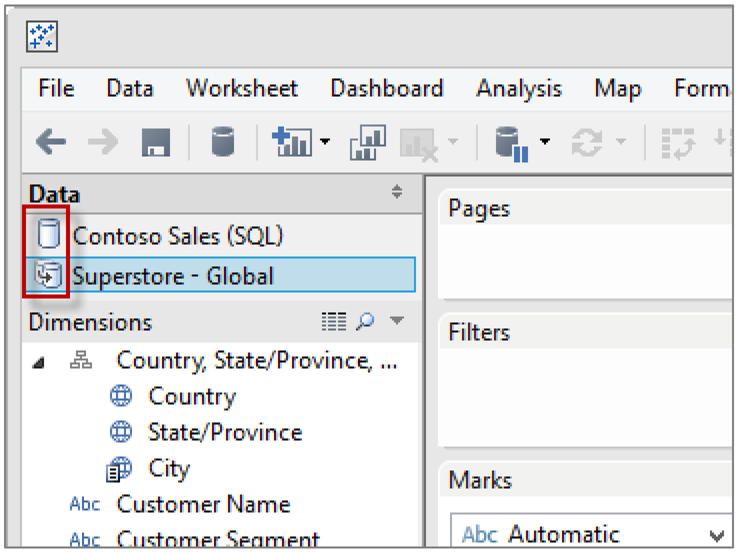
    - The exception to this guideline is if the extract has been published to Tableau Server with data source filters defined and other users are accessing this extract via the data server. Note - you will need to ensure that download permissions are revoked from users to ensure they cannot bypass the enforced filters.

Creating extracts

This bit is simple, assuming you are using Tableau Desktop. 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.

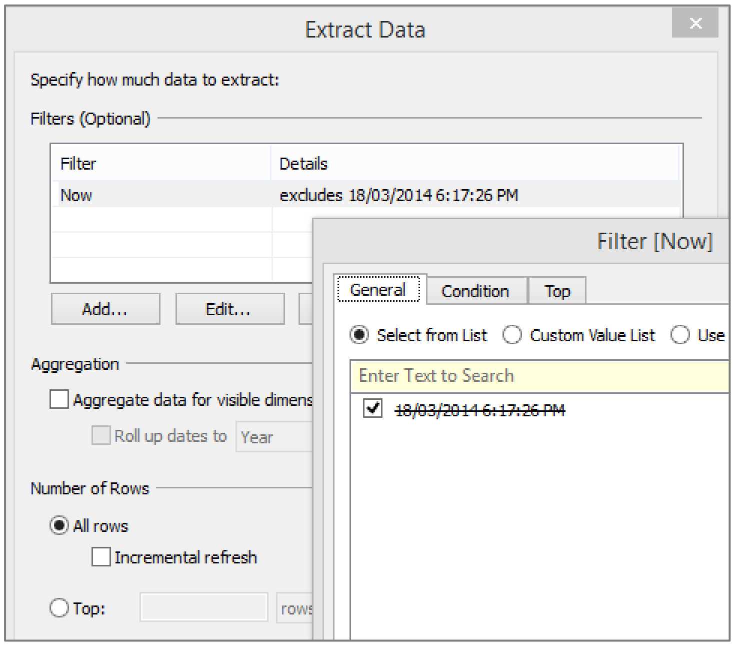
You'll know it's done when the data source icon changes – it will have another database icon behind it, presumably representing a 'copy', which is exactly what an extract is.



Note that the initial creation of an extract is always done in Tableau Desktop or via the Data Extract API and therefore occurs on the workstation. You will need to ensure your workstation 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 – it may require up to the square of the size of the resulting extract file (e.g. a 10GB extract may require 100GB or temp space). This is done 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.


Extract Data

Specify how much data to extract:

Filters (Optional)

Filter	Details
Now	excludes 18/03/2014 6:17:26 PM

Aggregation

☐ Aggregate data for visible dimens

☐ Roll up dates to Year

Number of Rows

● All rows

☐ Incremental refresh

○ Top: rows

Filter [Now]

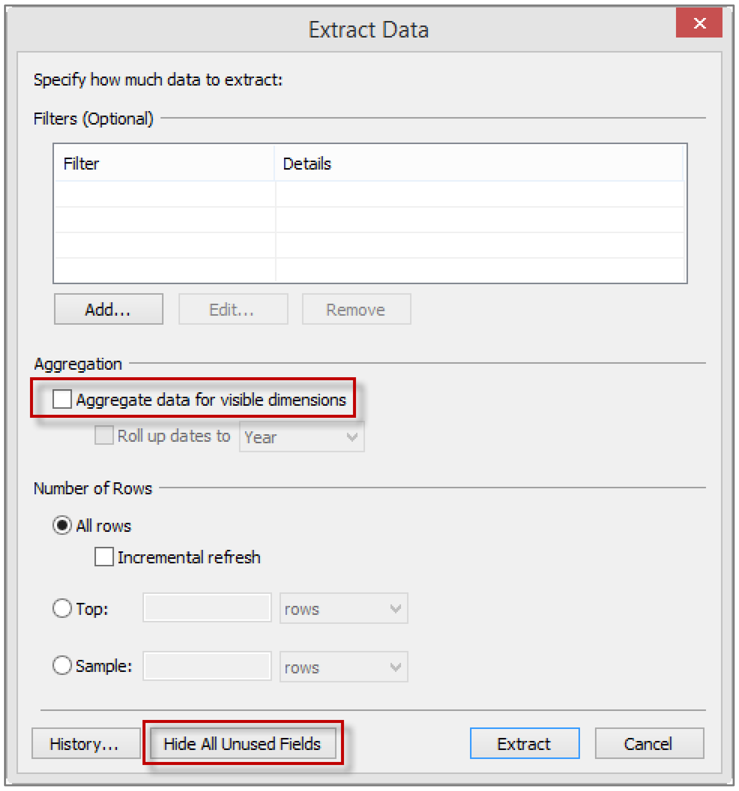
● Select from List ○ Custom Value List ○ Use

☑ ~~18/03/2014 6:17:26 PM~~

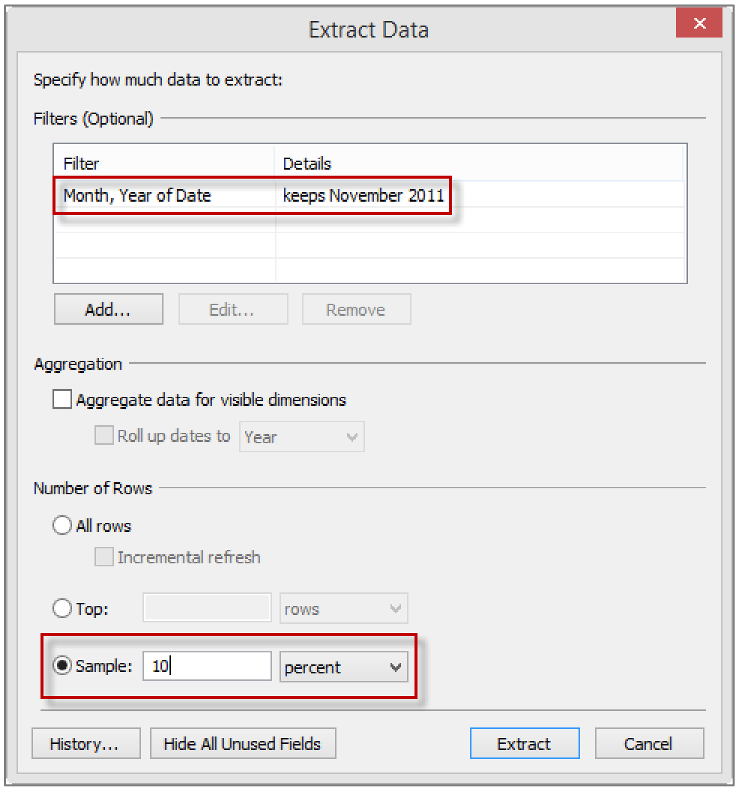
Aggregate extracts

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 is the absolute minimum required to create the view. People often use that setting for their summary view. Then on another page of the workbook, you can use a lower level of detail, but a larger extract. Since you have filtered that data, it should still perform well. You can keep this process going until you're connected to live system on the back end. Live systems are good at identifying a small set of rows. 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.



You can also extract a subset of data – either a filtered subset (say one month of data) or a random sample of data. This will give you the opportunity to create your analytical content and when you are ready to bring the power of Tableau to bear on the complete data set, deselect the Use Extract menu item.


Extract Data

Specify how much data to extract:

Filters (Optional)

Filter	Details
Month, Year of Date	keeps November 2011

Aggregation

- [ ] Aggregate data for visible dimensions
- [ ] Roll up dates to Year

Number of Rows

○ All rows

- [ ] Incremental refresh

○ Top: rows

● Sample: 10 percent

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 that 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 the user wants to refresh their data more often than every 15 minutes, they should probably connect 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 includes changes to existing rows.
- A full refresh discards the current extract and regenerates a new one from scratch from the data source.

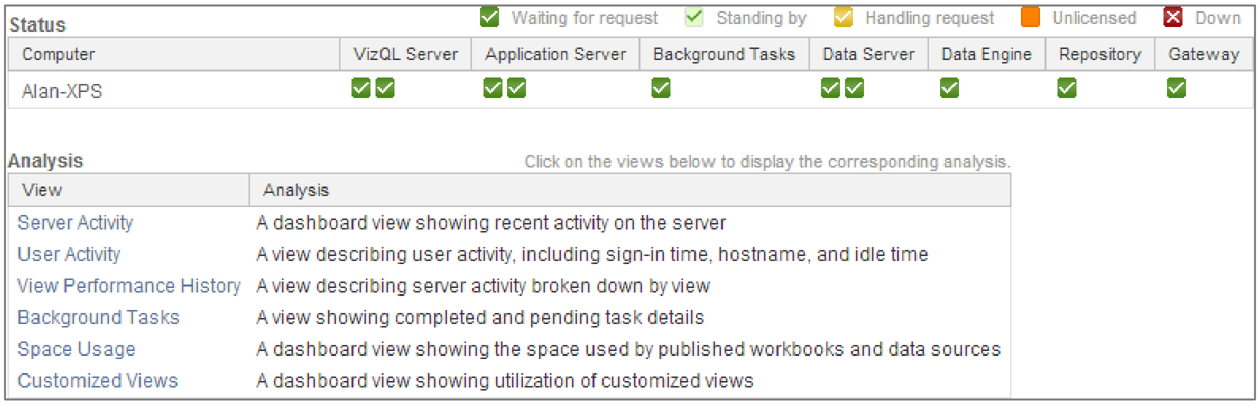
What happens if the refresh takes longer than the increment?

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. This situation might actually be desirable.

- A refresh starts at 1:00 and finishes at 2:30.
- The next refresh starts at 2:00 and finishes at 3:30.
- The next refresh starts at 3:00 and finishes at 4:30.

At 1:00, users are using data that is 1 ½ hours old. If you waited until the 1:00 refresh was finished at 2:30 to start another refresh, that second refresh would not be complete until 4:00. But with the overlapping refreshes, new data is available every hour, at 2:30, 3:30, and 4:30, instead of every 1 ½ hours at 2:30, 4:00, and 5:30. Once a refresh is complete, all new requests are routed to that version of the extract.

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.



Status ✓ Waiting for request ✓ Standing by ✓ Handling request 🟧 Unlicensed ✗ Down

Computer	VizQL Server	Application Server	Background Tasks	Data Server	Data Engine	Repository	Gateway
Alan-XPS	✓✓	✓✓	✓	✓✓	✓	✓	✓

Analysis Click on the views below to display the corresponding analysis.

View	Analysis
Server Activity	A dashboard view showing recent activity on the server
User Activity	A view describing user activity, including sign-in time, hostname, and idle time
View Performance History	A view describing server activity broken down by view
Background Tasks	A view showing completed and pending task details
Space Usage	A dashboard view showing the space used by published workbooks and data sources
Customized Views	A dashboard view showing utilization of customized views

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 that the 15 minute minimum allowed through the Tableau Server interface.

You can set updates to occur on a schedule. Or you can choose to disable the schedule and then manually initiate refreshes when you want them.

Part 3 - 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.

Understanding the queries

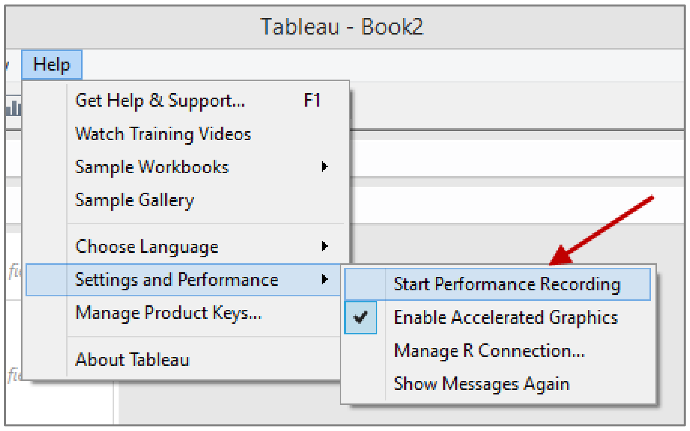
In Tableau you can find the full query text by looking in the log file. The default location is C:\Users\<username>\Documents\My Tableau Repository\Logs\log.txt. This file is quite verbose, but the query will be found between <QUERY> and </QUERY> tags:

```
2014-02-19 12:32:21.574 (-,-,-,-) 16a8: <QUERY protocol='0ec80680'>
2014-02-19 12:32:21.574 (-,-,-,-) 16a8: SELECT [ProductDim].[Product Category]
AS [none:Product Category:nk],
2014-02-19 12:32:21.574 (-,-,-,-) 16a8:   [ProductDim].[Product Subcategory] AS
[none:Product Subcategory:nk],
2014-02-19 12:32:21.574 (-,-,-,-) 16a8:   SUM([OrdersFact].[Sales]) AS
[sum:Sales:ok]
2014-02-19 12:32:21.574 (-,-,-,-) 16a8: FROM ( ( ( ( [OrdersFact]
2014-02-19 12:32:21.574 (-,-,-,-) 16a8:   INNER JOIN [CustomerDim] ON
[OrdersFact].[Customer ID] = [CustomerDim].[Customer ID] )
2014-02-19 12:32:21.574 (-,-,-,-) 16a8:   INNER JOIN [DeliveryDim] ON
[OrdersFact].[Delivery ID] = [DeliveryDim].[Delivery ID] )
2014-02-19 12:32:21.574 (-,-,-,-) 16a8:   INNER JOIN [LocationDim] ON
[OrdersFact].[Place ID] = [LocationDim].[Place ID] )
2014-02-19 12:32:21.574 (-,-,-,-) 16a8:   INNER JOIN [ProductDim] ON
[OrdersFact].[Product ID] = [ProductDim].[Product ID] )
2014-02-19 12:32:21.574 (-,-,-,-) 16a8:   INNER JOIN [TimeDim] ON
[OrdersFact].[Date ID] = [TimeDim].[Date ID]
2014-02-19 12:32:21.574 (-,-,-,-) 16a8: WHERE ((Fix(Fix([TimeDim].[Date])) >=
IIF(ISNULL((-2999)),NULL,DATEADD('d',(-2999),IIF(ISNULL(#02/19/2014
12:32:21#),NULL,DATEADD('d', DATEDIFF('d', #01/01/1970#, #02/19/2014
12:32:21#), #01/01/1970#))))) AND (Fix(Fix([TimeDim].[Date])) <
IIF(ISNULL(1),NULL,DATEADD('d',1,IIF(ISNULL(#02/19/2014
12:32:21#),NULL,DATEADD('d', DATEDIFF('d', #01/01/1970#, #02/19/2014
12:32:21#), #01/01/1970#))))))
2014-02-19 12:32:21.574 (-,-,-,-) 16a8: GROUP BY [ProductDim].[Product
Category],
2014-02-19 12:32:21.574 (-,-,-,-) 16a8:   [ProductDim].[Product Subcategory]
2014-02-19 12:32:21.574 (-,-,-,-) 16a8: </QUERY>
```

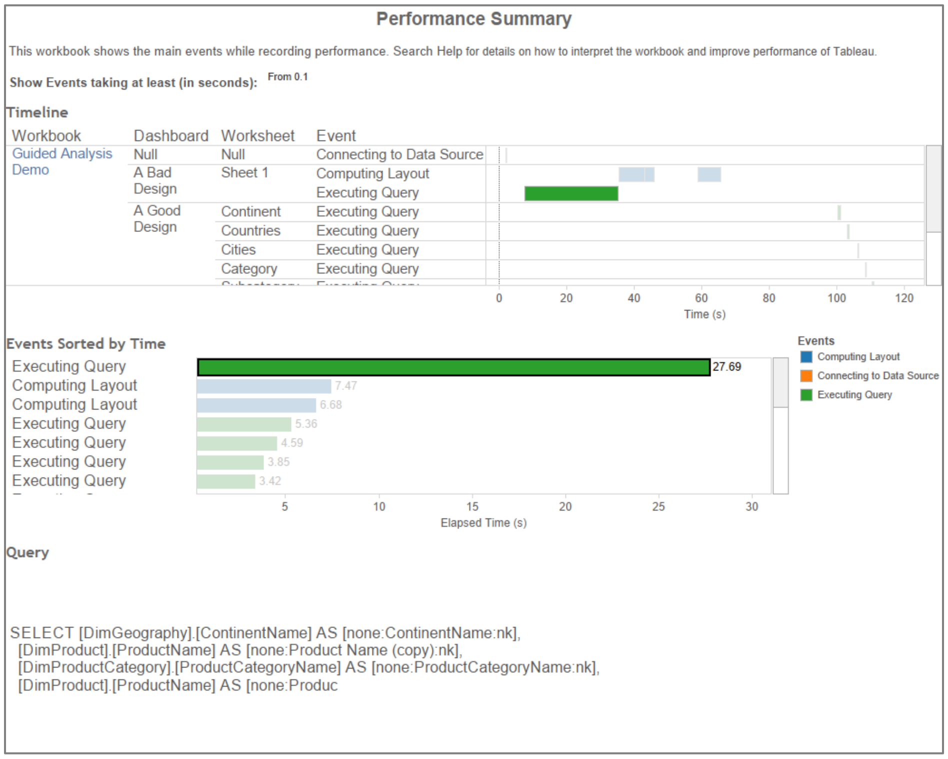
If you are looking on Tableau Server, the logs are in C:\ProgramData\Tableau\Tableau Server\data\tabsvc\vizqlserver\Logs.

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:



```
SELECT [DimGeography].[ContinentName] AS [none:ContinentName:nk],
  [DimProduct].[ProductName] AS [none:Product Name (copy):nk],
  [DimProductCategory].[ProductCategoryName] AS [none:ProductCategoryName:nk],
  [DimProduct].[ProductName] AS [none:Produc
```

You can now identify the actions in the workbook that take the most time - for example in the above image one of the queries from “Sheet 1” on the “Bad Design” dashboard takes 27 seconds to complete. Clicking on the bar shows the text of the query being executed.

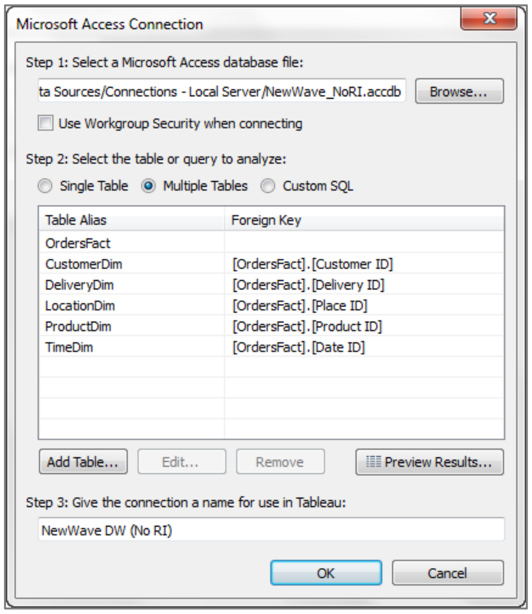
You can use this information to identify those sections of a workbook that are the best candidates for review - i.e. where can you get the best improvement for the time you spend? More information on interpreting these recordings can be found in the following link:

- http://onlinehelp.tableausoftware.com/v8.1/server/en-us/perf_record_interpret_server.htm

Join culling

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:


Microsoft Access Connection

Step 1: Select a Microsoft Access database file:

ta Sources/Connections - Local Server/NewWave_NoRI.accdb

Use Workgroup Security when connecting

Step 2: Select the table or query to analyze:

Single Table [✓ Multiple Tables] Custom SQL

Table Alias	Foreign Key
OrdersFact	
CustomerDim	[OrdersFact].[Customer ID]
DeliveryDim	[OrdersFact].[Delivery ID]
LocationDim	[OrdersFact].[Place ID]
ProductDim	[OrdersFact].[Product ID]
TimeDim	[OrdersFact].[Date ID]

Step 3: Give the connection a name for use in Tableau:

NewWave DW (No RI)

With join culling, Tableau generates the following query:

```
SELECT [ProductDim].[Product Category], SUM([OrdersFact].[Sales])
FROM [dbo].[OrdersFact] [OrdersFact]
  INNER JOIN [dbo].[ProductDim] [ProductDim]
  ON ([OrdersFact].[Product ID] = [ProductDim].[Product ID])
GROUP BY [ProductDim].[Product Category]
```

Without it, Tableau generates a far less efficient query:

```
SELECT [ProductDim].[Product Category], SUM([OrdersFact].[Sales])
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].[TimeDim] [TimeDim]
       ON ([OrdersFact].[Date ID] = [TimeDim].[Date ID])
```

```
    INNER JOIN [dbo].[ProductDim] [ProductDim]
        ON ([OrdersFact].[Product ID] = [ProductDim].[Product ID])
GROUP BY [ProductDim].[Product Category]
```

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 when 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 a new feature has been introduced that allows users to tell Tableau that soft referential integrity is in place and that join culling can be safely used.

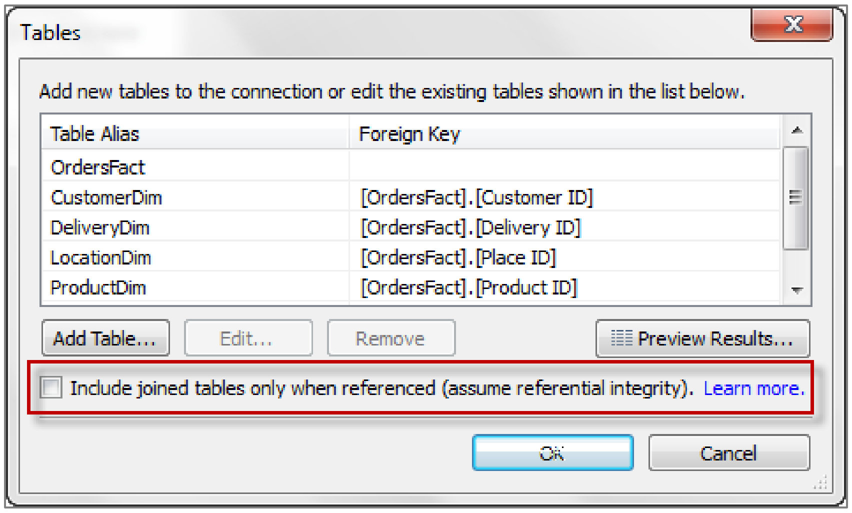

Tables
Add new tables to the connection or edit the existing tables shown in the list below.
Add Table...
Edit...
Remove
Preview Results...
OK
Cancel

Table Alias	Foreign Key
OrdersFact	
CustomerDim	[OrdersFact].[Customer ID]
DeliveryDim	[OrdersFact].[Delivery ID]
LocationDim	[OrdersFact].[Place ID]
ProductDim	[OrdersFact].[Product ID]

☐ Include joined tables only when referenced (assume referential integrity). Learn more.

For more information, see the following series of articles by Russell Christopher on his Tableau Love blog:

- http://tableaulove.tumblr.com/post/11692301750/what-i-learned-about-tableau-join-culling-over-summer
- http://tableaulove.tumblr.com/post/62447366098/what-i-learned-about-tableau-join-culling-over-fall

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.

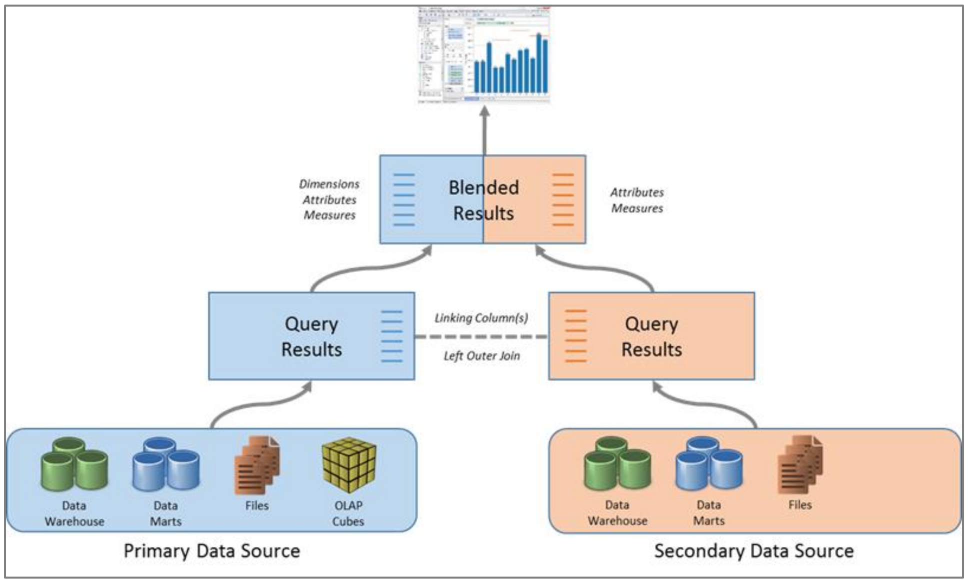
If the workbook uses data from more than one data source, you need to either blend data or establish a federated database system. If the workbook uses two data connections from the same data source, it is generally preferable to join the data tables as this can improve performance and filtering control.

However, there are a few situations where joining data tables may not work as well as blending data. Here are two common situations that may perform better with data blending.

- If you have multiple data connections that are large and take a long time to query, using a join can increase query time dramatically. A better approach may be to aggregate the tables, and then blend the data on the aggregate. For example, you can aggregate data on the year rather than the date, or on the product type instead of the product name. For an example, see the article Data Blending with Summarized Data.
- If you want to see both the summary of a calculation and the breakdown on same view, select Data > Data Connection > Duplicate to blend the data to communicate between two data connections. For more information, see the article Showing Summary and Detail Together.

The following article shows several examples of when you may want to use a join and when you want to use data blending: http://kb.tableausoftware.com/articles/knowledgebase/join-vs-relationship-60

If you are using 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. The release of 64-bit Tableau 8.1 will allow users to blend on more complex data (more unique values) without running out of memory, but these scenarios 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. 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

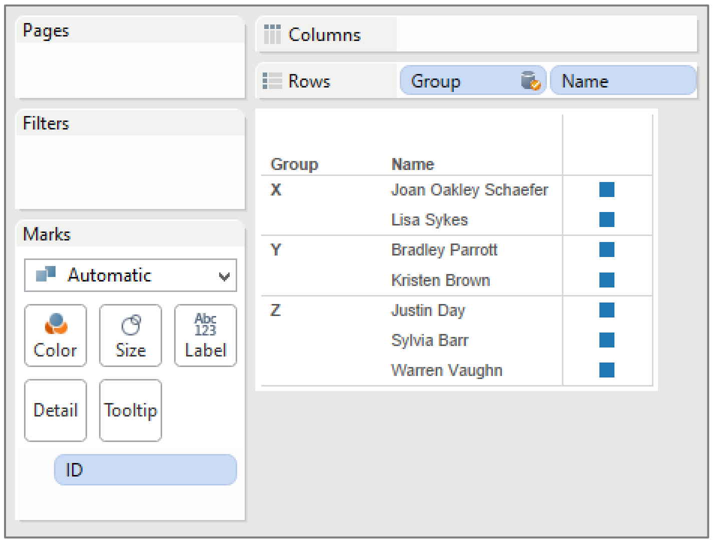
ID	Name
A	Lisa Sykes
B	Joan Oakley Schaefer
C	Bradley Parrott
D	Kristen Brown
E	Sylvia Barr
F	Warren Vaughn
G	Justin Day

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:

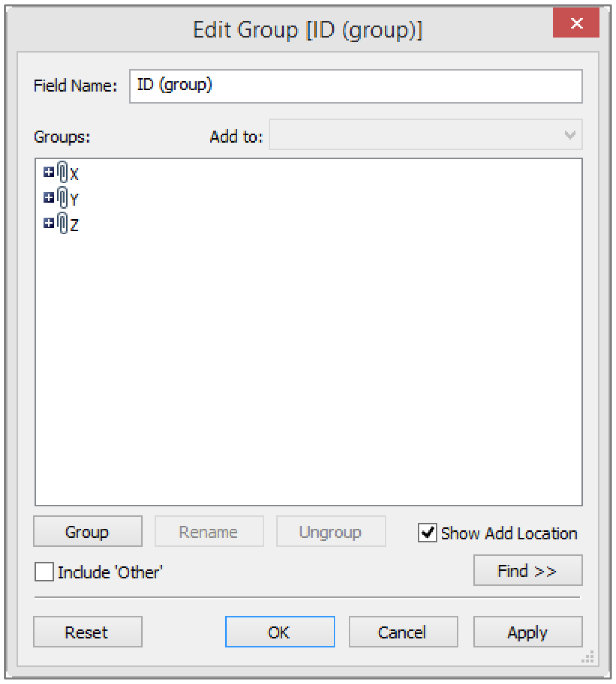
Group	Name
X	Lisa Sykes
	Joan Oakley Schaefer
Y	Bradley Parrott
	Kristen Brown
Z	Sylvia Barr
	Warren Vaughn
	Justin Day

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:



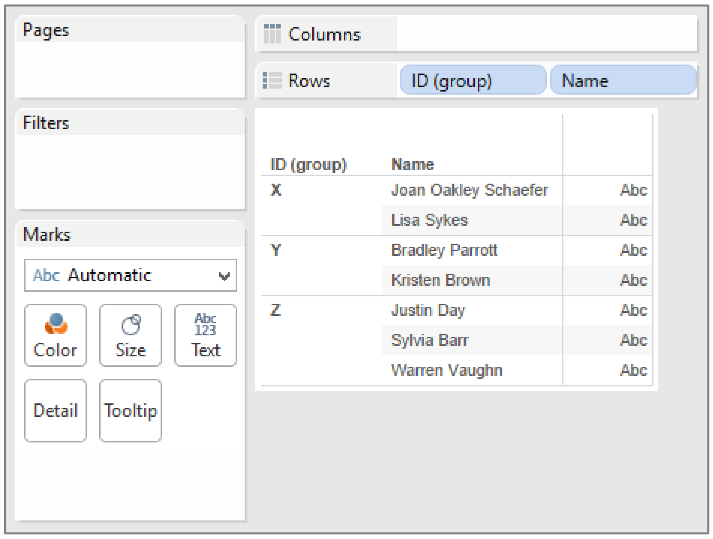
Group	Name	
X	Joan Oakley Schaefer	🟦
	Lisa Sykes	🟦
Y	Bradley Parrott	🟦
	Kristen Brown	🟦
Z	Justin Day	🟦
	Sylvia Barr	🟦
	Warren Vaughn	🟦

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)

We can now recreate this table without requiring a blend:

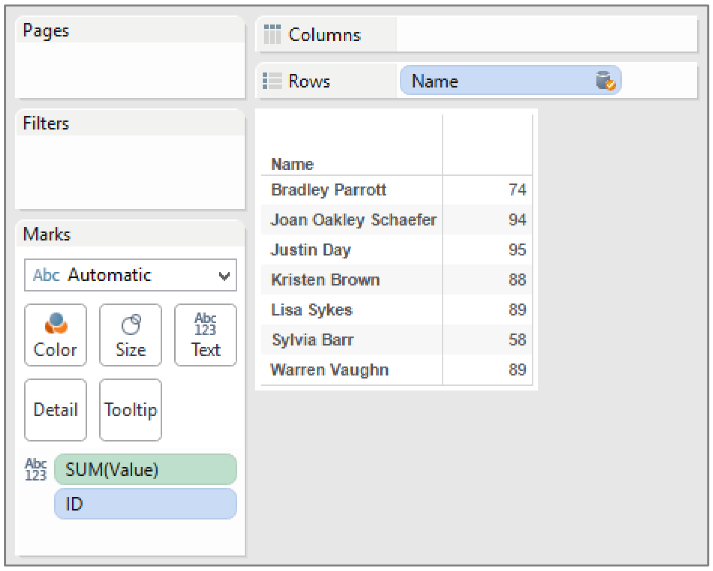


ID (group)	Name	
X	Joan Oakley Schaefer	Abc
	Lisa Sykes	Abc
Y	Bradley Parrott	Abc
	Kristen Brown	Abc
Z	Justin Day	Abc
	Sylvia Barr	Abc
	Warren Vaughn	Abc

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:

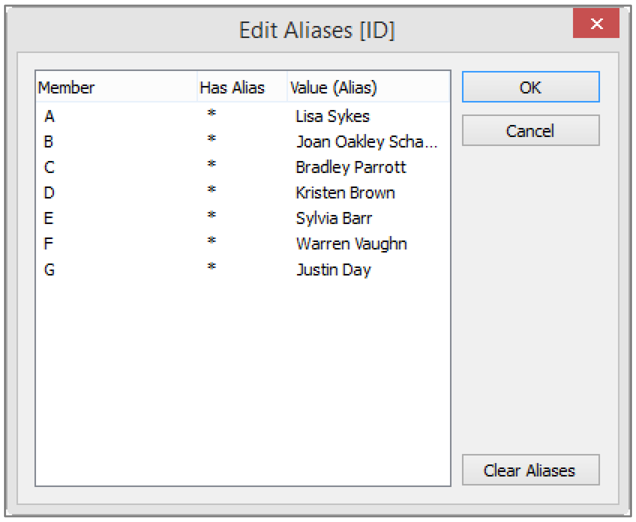
Name	Value
Lisa Sykes	89
Joan Oakley Schaefer	94
Bradley Parrott	74
Kristen Brown	88
Sylvia Barr	58
Warren Vaughn	89
Justin Day	95

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:



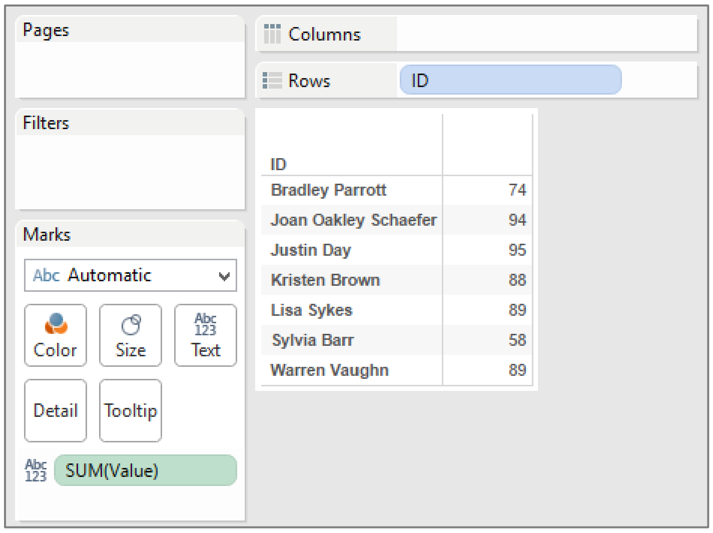
Name	
Bradley Parrott	74
Joan Oakley Schaefer	94
Justin Day	95
Kristen Brown	88
Lisa Sykes	89
Sylvia Barr	58
Warren Vaughn	89

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.

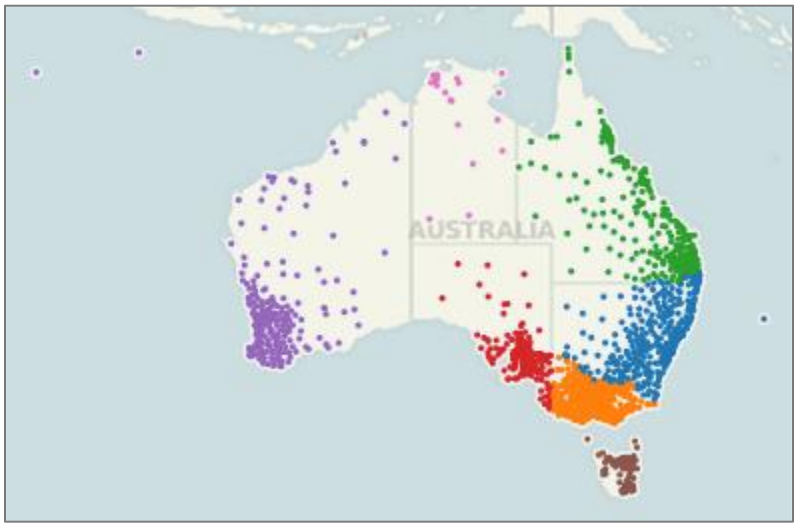
Filtering

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.

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

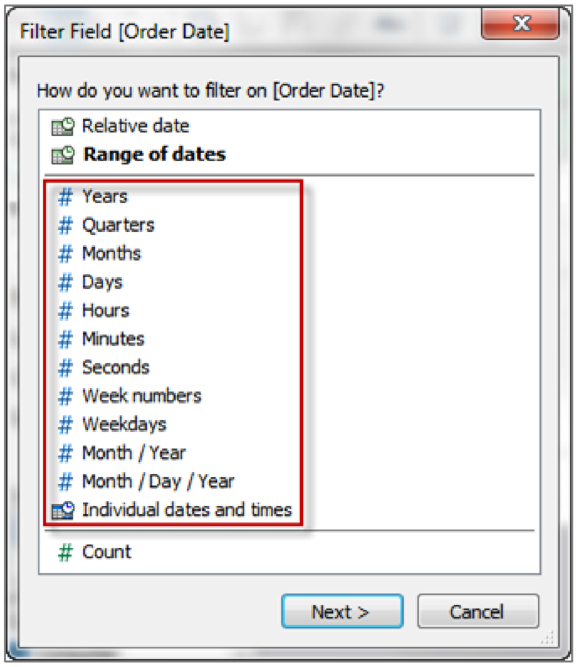
Using the ranged filter approach also allows the database to evaluate a simple filter clause (either `WHERE POSTCODE >= 6000 AND POSTCODE <= 7000` or `WHERE LONGITUDE < 129`) resulting in fast execution. However this approach, unlike a filter on a related dimension, doesn’t become more complex as we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

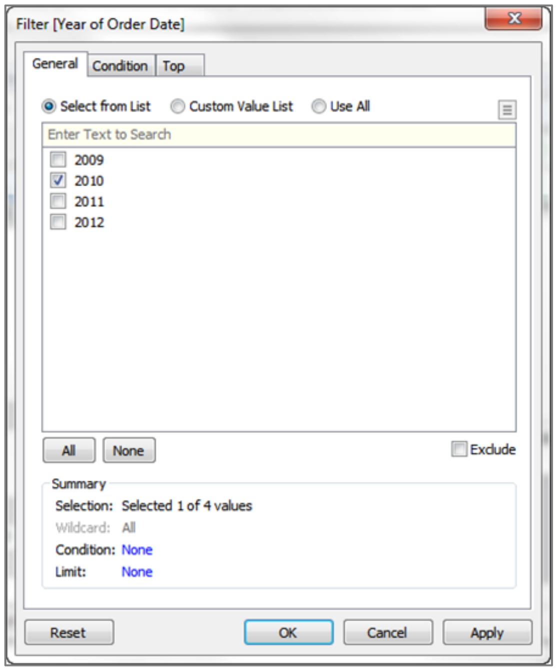
Discrete


Filter Field [Order Date]

How do you want to filter on [Order Date]?

- Relative date
- **Range of dates**
- Years
- Quarters
- Months
- Days
- Hours
- Minutes
- Seconds
- Week numbers
- Weekdays
- Month / Year
- Month / Day / Year
- Individual dates and times
- Count

Next > | Cancel


Filter [Year of Order Date]

General | Condition | Top

[✓ Select from List] Custom Value List | Use All

Enter Text to Search

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

All | None | Exclude

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

Reset | OK | Cancel | Apply

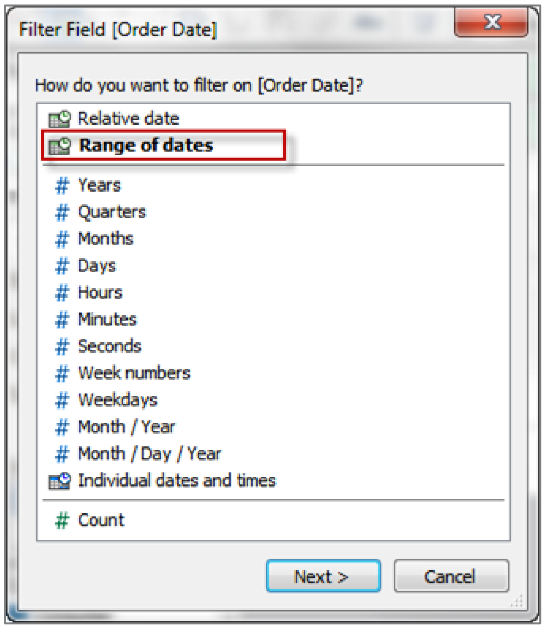
Sometimes you may want to filter to include specific individual dates or entire date levels. This type of filter is called a discrete date filter because you are defining discrete values instead of a range. This filter type results in the date expression being passed to the database as a dynamic calculation:

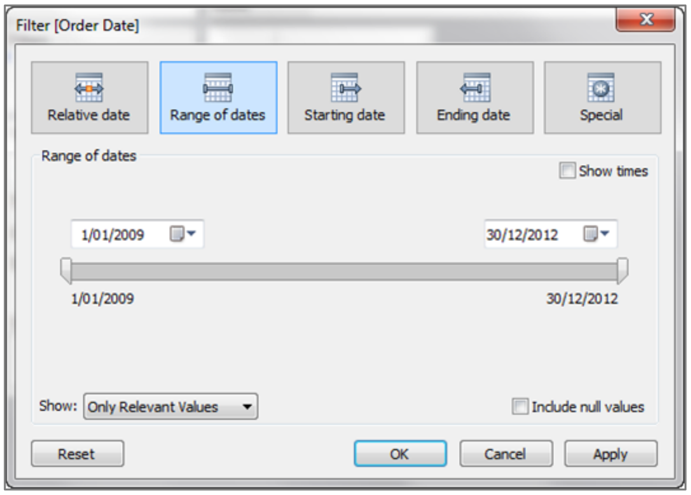
```
SELECT [FactSales].[Order Date], SUM([FactSales].[SalesAmount])
FROM [dbo].[FactSales] [FactSales]
WHERE (DATEPART(year,[FactSales].[Order Date]) = 2010)
GROUP BY [FactSales].[Order Date]
```

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 ranged date filter.

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



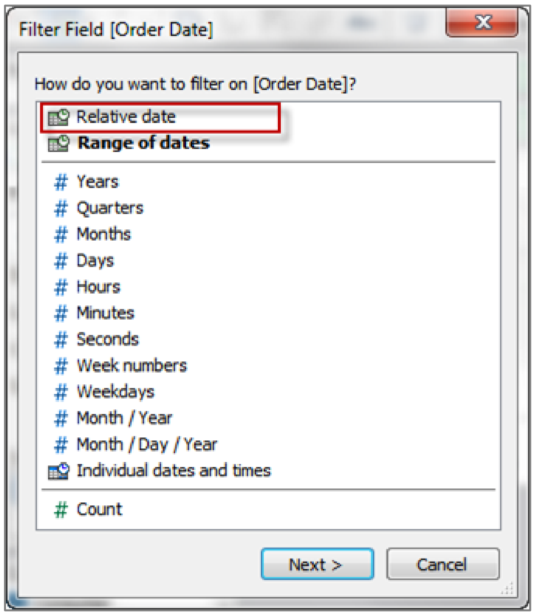


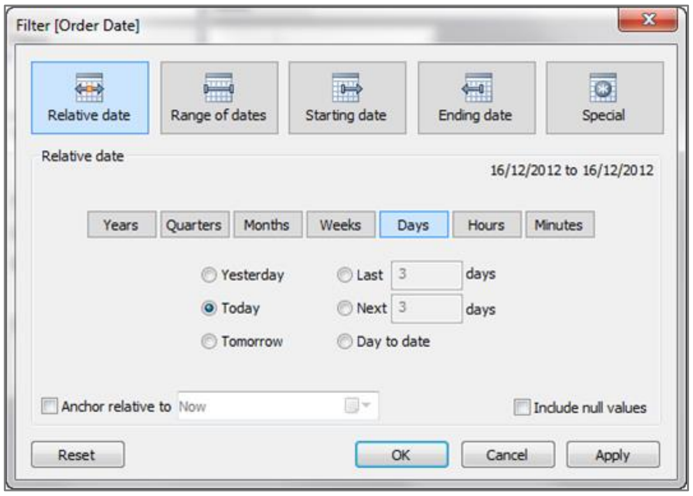
This type of filter is used when you want to specify a range of contiguous dates. It results in the following query structure being passed to the database:

```
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.

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, you can set one or more filters as context filters for the view. You can think of this as an independent filter. Any other filters that you set are defined as dependent filters because they process only the data that passes through the context filter.

Context filters are particularly useful for relational data sources because they are implemented by writing the filter result set to a temporary table. This table then acts as a separate (smaller) data source for subsequent filters and queries, resulting in increased performance when you build data views.

Context filters are often used to improve performance. Note that the improvement occurs because the database writes the results of the context filter to a temporary table. The creation of this temporary table is an expensive activity for the database so this approach is 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 recompute and rewrite the temporary table, slowing performance.

One useful trick that takes advantage of context filter behaviour – they can be used to materialise a data set with complex table joins into a single, denormalised table. For example – consider the following query that Tableau generates:

```
SELECT SUM([OrdersFact].[Sales])
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])
WHERE (([LocationDim].[Region] >= 'Africa')
  AND ([LocationDim].[Region] <= 'Oceania'))
HAVING (COUNT_BIG(1) > 0)
```

By setting a context menu on a dimension that returns all dimension members, we force Tableau to materialise the above query and write the results to a temporary table. This results in the same query being regenerated as follows:

```
SELECT SUM([#Tableau_3_Context].[Sales])
FROM [#Tableau_3_Context]
HAVING (COUNT_BIG(1) > 0)
```

As you can see, this results in a much simpler query for the database to execute, resulting in faster performance. This technique could also be used to optimise workbooks that use a data connection based on a custom SQL statement.

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 shared result 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 following section on Tableau Server caching for more detail.

Part 4 - 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 three different calculation types in Tableau:

- Basic calculations
- Aggregate calculations
- Table calculations

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:

```
SELECT DATEPART(year,[TimeDim].[Date]), SUM([OrdersFact].[Sales])
FROM [dbo].[OrdersFact] [OrdersFact]
  INNER JOIN [dbo].[TimeDim] [TimeDim]
  ON ([OrdersFact].[Date ID] = [TimeDim].[Date ID])
GROUP BY DATEPART(year,[TimeDim].[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.

Table calculations

Table calculations, on the other hand, are not executed by the database but rather calculated by Tableau on the query results that get returned. 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.

Some table calculations are very expensive for the Tableau engine to perform. According to the following blog post from Richard Leeke, for the commonly used WINDOW_XXX and TOTAL table calculations the time to execute increases proportional to the square of the number of rows in the partition being analysed. This makes these functions execute very slowly for large numbers of records.

- http://www.clearlyandsimply.com/clearly_and_simply/2011/01/another-look-at-site-catchment-analysis-with-tableau-6-part-3.html

In this post he offers some workarounds for limiting the number of rows the table calculation engine processes. For example, the WINDOW_AVG calculation shown above can be rewritten as:

```
IF FIRST()==0 THEN WINDOW_AVG(SUM([Sales]),0,IIF(FIRST()==0,LAST(),0)) END
```

This change can yield significant improvement – in one of the examples Richard references, he reduced the time required to render a view from 3 hours to 5 seconds!

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;
- Grouping measure values together into "bins" – consider using Bins;
- 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 are faster than Booleans and both 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:

```
IF [Parameters].[Date Part Picker]="Year"
  THEN DATEPART('year',[Order Date])
ELSEIF [Parameters].[Date Part Picker]="Quarter"
  THEN DATEPART('quarter',[Date])
…
ELSE NULL 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:

```
IF [Parameters].[Date Part Picker]=1
  THEN DATEPART('year',[Order Date])
ELSEIF [Parameters].[Date Part Picker]=2
  THEN DATEPART('quarter',[Order Date])
..
ELSE NULL 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 in Tableau 8.1 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.

Date functions
Use NOW() only if you need the time stamp level of detail. Use TODAY() for date level calculations.

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
```

Separating basic and aggregate calculations

When using extracts and custom aggregations, divide the calculation into multiple parts. Place the row level calculations on one calculated field and the aggregated calculation in a second calculated field. Then extracts can optimise (pre-compute and materialise) the row level calculations.

Part 5 - 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 for many users to consume, performs poorly. The following section identifies areas where the differences between single- and multi-user scenarios can make a difference.

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 all components of Tableau Server can be run as native 64-bit processes. This means all components can address much more RAM and our early benchmarking tests indicate that the 64-bit version of Tableau scales significantly better than the 32-bit version.

Customers are strongly advised to upgrade their Tableau Server installation to 8.1 or later and deploy 64-bit Tableau Server.

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 and see When to Add Workers & Reconfigure to assess whether you should add additional machines.

In July 2013, we performed scalability tests on Tableau Server to help our customers plan for large deployments. We tested three different server configurations-- using one, two, or three servers in a dedicated, load test environment. We tested two different types of reports with varying degrees of complexity. We tried to simulate real-world usage by having each user perform a variety of tasks including loading the report, performing a selection, filtering the view and changing tabs. The following paper describes those tests and outlines techniques for improving Tableau Server's performance.

- http://www.tableausoftware.com/learn/whitepapers/tableau-server-scalability-explained

Configuration

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 you hardware configuration permits you can also move the Backgrounder process(es) to a dedicated worker node.

Check the VizQL session timeout limit

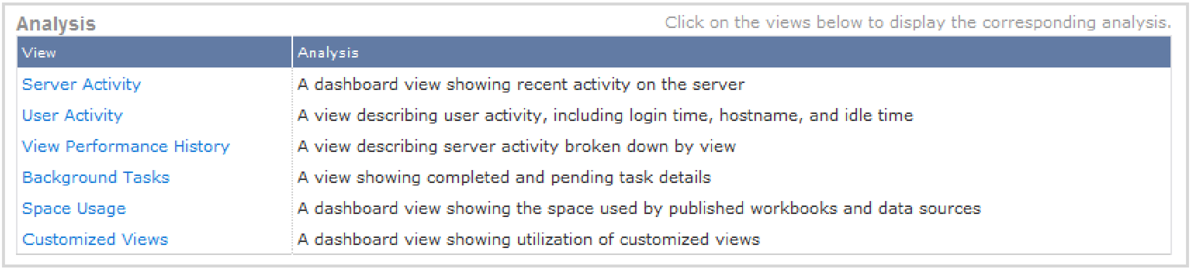
The default VizQL session timeout limit is 30 minutes. Even if a VizQL session is idle, it is still consuming memory and CPU cycles. If you can make do with a lower limit, use tabadmin to change the vizqlserver.session.expiry.timeout setting .

Assess your process configuration

Tableau Server is divided into six different components called server processes. 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.

Monitoring Tableau Server

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



Analysis Click on the views below to display the corresponding analysis.

View	Analysis
Server Activity	A dashboard view showing recent activity on the server
User Activity	A view describing user activity, including login time, hostname, and idle time
View Performance History	A view describing server activity broken down by view
Background Tasks	A view showing completed and pending task details
Space Usage	A dashboard view showing the space used by published workbooks and data sources
Customized Views	A dashboard view showing utilization of customized views

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

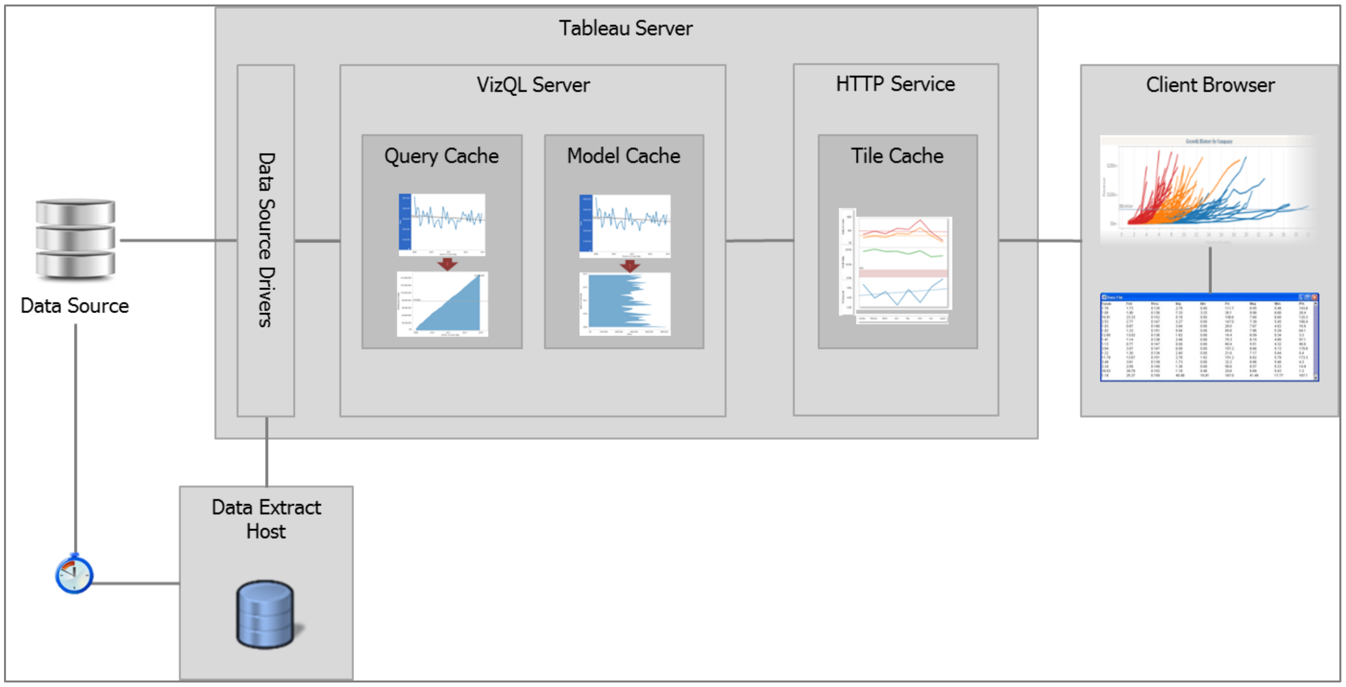
- http://onlinehelp.tableausoftware.com/current/server/en-us/adminview.htm

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://onlinehelp.tableausoftware.com/current/server/en-us/adminview_postgres.htm

Caching

Caching helps Tableau Server respond to client requests quickly, especially for views that connect to live databases. Tableau Server has several layers of caching designed to maximise the reuse of data and calculations across multiple user requests:



Client-side rendering

Before a view's marks and data are displayed in a client web browser, they are retrieved, interpreted, and rendered. Tableau Server can perform this process in the client web browser or on the server. Client-side rendering is the default mode because handling the rendering and all interaction on the server can result in more network data transfer and round-trip delays. With client-side rendering, most view interactions are faster, because they are interpreted and rendered right there in the client.

Some views, however, are more efficiently rendered on the server where there's more computing power. Server-side rendering makes sense for a view that is complex to the extent that image files take up significantly less bandwidth than the data used to create the images. Also, because tablets usually have much slower performance than PCs, they can handle less view complexity. There are cases where a view opened from a PC's web browser might be client-rendered but the same view opened from a tablet's web browser is server-rendered.

Tableau Server is configured to automatically handle all of these situations using The Threshold Calculation as the trigger for rendering a view on the server instead of in the web browser. As the administrator, you can test or fine tune this setting for both PCs and tablets. See the topics below for more information.

You can determine this at any time by adding the following to the view URL. It will show a small “S” or “B” in the view toolbar indicating the rendering mode:

- “?:jsdebug=true”

You can also turn client-side rendering on or off with the following URL parameters:

- Turn off = “?:render=false”
- Turn on = “?:render=true”

Client-side rendering is supported in Internet Explorer version 9.0 or higher, Firefox, Chrome, and Safari. All of these web browsers include the HTML 5 <canvas> element, which is used by client-side rendering. Also, if a view uses polygons, custom shapes, or the page history feature, server-side rendering is performed, even if client-side rendering is otherwise enabled.

Image tile cache

If client-side rendering is not used, then Tableau has three layers of rendering on the server that can help improve view rendering times. The first layer is the image tile cache.

Dashboards are delivered to the client as a series of image “tiles” – these are assembled to show the complete dashboard. Reusing content from this cache is the most efficient form of server response and occurs if:

- The dashboard has already been rendered and the cache time-to-live has not expired;
- The dashboard does not use per-user security;
- The request is asking for the same size dashboard as previously cached – this can occur if the two client browser windows are exactly the same size, or if the dashboard has been designed with an exact size setting.

The image tile cache is disk based and managed by the gateway service. There is one per VizQL worker machine.

Model cache

If we cannot use the image tile cache, the VizQL server must re-render the requested images. To do so, it may be able to re-use all the calculations done previously - calculated fields, table calculations, reference lines, trend lines, etc. These results are held in the VizQL model cache, and we can use them if:

- The requested dashboard has previously been rendered by this VizQL instance and the cache time-to-live has not expired;
- There are no change to the requested data – all filters, parameters and dynamic calculations are the same;
- There have been no changes to the calculations – no changes to reference lines, trend lines, etc.;
- The dashboard does not use per-user security.

The model cache is RAM based and there is one per VizQL server instance. Running multiple VizQL instances can reduce the efficiency of the model cache if the number of users is low.

Query result cache

If we cannot use the model cache, it may be possible to perform all the necessary calculations using data that has already been read from the data source and is held in the query result cache. The query result cache holds the records returned by queries and we can use them if:

- The requested dashboard has previously been rendered by this VizQL instance and the cache time-to-live has not expired;
- There are no changes to the dimensions and measures required from the data source – no fields have been changed, such as through a dynamic calculated field;
- There are no changes to the filters on the dashboard;
- The dashboard does not use per-user security.

Like the model cache, the query result cache is RAM based and there is one per VizQL server instance. Running multiple VizQL instances can reduce the efficiency of the query result cache if the number of users is low.

Maximising cache use

As discussed earlier, the single most effective thing a workbook designer can do to ensure reuse of the image tile and model caches is to set the dashboard size rule to "exact size".

Tuning caches

At the macroscopic level, cache performance on a Tableau Server can be configured to one of three settings via the Tableau Server configuration utility:

Minimise queries	• Each VizQL server will cache models and query results for as long as it can. • Requests get cached data until explicitly refreshed. • The cache is held in memory, when memory fills, the cache starts to empty oldest items.
Balanced	• Each server will cache models and data for at most the specified number of minutes. • Requests get cached data that is at most a known number of minutes old.
Most up-to-date	• Each server will not cache models or data at all • Request get the freshest data from the data source • This option places a much higher load on the servers.

It is possible to force Tableau to bypass all the caches and force a data source query by attaching the "?:refresh=yes" parameter to the view URL. For example:

- http://demo-apac.tableausoftware.com/views/NewWaveDashboard/ExecutiveDashboard?:refresh=yes

Tableau Server administrators can also tune the size of both the model and query result cache. These settings are changed via the tabadmin command line tool.

Model cache	• vizqlserver.modelcachesize:30 • The number of models to cache, where there is one model per viz instance in a workbook
Query cache	• vizqlserver.querycachesize:64 • The size in megabytes of query results to cache

See the following link for more detail:

- http://onlinehelp.tableausoftware.com/current/server/en-us/reconfig_tabadmin.htm

Be aware that changing these settings will increase the amount of RAM used by the VizQL server instances – make sure you have sufficient memory on your Tableau Server machine to support this.

Part 6 - Are there other factors to consider?

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.

Environmental

OS - 32-bit vs. 64-bit

In general, it is preferable to run Tableau on a 64-bit OS. With the release of Tableau 8.1 the entire application is now 64-bit which means the application can use > 3.2GB of RAM, allowing us to deal with much larger queries and render many more data marks. This has a significant impact on scenarios where one is blending on a high-cardinality field or perhaps trying to render (against better advice) a very large crosstab. Practically speaking, this has been a big advantage for customers who are rendering lots of data marks – e.g. custom geographic polygons.

More/faster RAM, CPU

Like most applications, Tableau will benefit from more RAM and a faster CPU. Some of the actions Tableau performs are multi-threaded (e.g. creating or refreshing a data extract) so it will also benefit from more cores.

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.

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. It's a close race in a rapidly evolving field, but at the moment browsers are ranked in the following order:

- Firefox, Chrome
- New IE (10+)
- Old (IE9-), Opera, Safari

This advice is based on figures reported here:

- http://www.zdnet.com/the-big-browser-benchmark-january-2013-edition_p2-7000009776/

Appendix A – Relational vs. OLAP Capabilities

Tableau connects to both cubes (OLAP) and relational data sources with live, native connections. Cubes and relational tables differ greatly in their structure, purpose and generally in their performance as well. Because of these differences, Tableau has optimized connections and functionality to best take advantage of each type of connection.

In general, Cubes are pre-aggregated and structured. Relational sources are generally disaggregated and less-structured. The vast majority of Tableau functionality is identical for both types of data sources.

Specific Feature	**Cube**	**Relational**
User Filters	Cubes have the ability to define user lever security. Tableau has the ability to leverage this logic within your environment. You do not need to re-do your security model inside of Tableau. If you do not have this defined in your cube, you can define it in Tableau. SECURITY	Relational data sources have the ability to define user level security. You do not need to redefine this logic inside of Tableau. If your database does not have this logic, it can be defined in Tableau SECURITY
Data Blending	In any organization, there is data within a cube, and then there is data not in the cube. Tableau allows users to easily blend relational data with cube data, without moving the data, or modeling it in the cube. DATA BLENDING	Data blending works with relational databases, across databases, and to cubes. Tableau's approach to asking questions of disparate data sources is unique. Tableau does not require the data to be moved. DATA BLENDING
Set Analysis	Sets are something that many cube users love. Tableau will leverage these sets. However, if it does not exist in the cube, you can create a dynamic, dimensional set within Tableau. RANKED SET	Sets are a way to save dimensional intersections of data. When using a relational source of data, you can create sets within the data similar to cubes. SETS

Specific Feature	**Cube**	**Relational**
Aggregate Calculation Functions	Aggregates are done ahead of time in a cube, and Tableau responds with the aggregation set in the cube. This is one of the main performance benefits of cubes.	Aggregates are typically not done in a relational database ahead of time. Tableau then allows the user to select the aggregation within Tableau and the database runs the aggregation on demand.
Tableau "Groups"	Groupings are typically defined in the cube by the developer and are pre-calculated. This results in the performance benefit and standard structure. You can do grouping with simple MDX: [Customer].[CustomerGeography].[France] + [Customer].[CustomerGeography].[Germany]	Grouping is typically not modeled ahead of time in a relational database. However, Tableau provides you the ability to create groups on the fly at any time during your analysis. CREATING GROUPS
Tableau "Bins"	Bins, or grouping of measure ranges, are typically modeled in the cube and are attributes within the dimension window. This allows for a common definition for Bins. Users can create or modify bins in Tableau with simple MDX: str(INT(([Internet Sales Amount] / [Bin Size])) * [Bin Size])	Bins are typically not modeled in a relational database. However, Tableau provides the ability for users to create bins on measures. EVEN BINS UNEVEN BINS
String Manipulations	String manipulations are often times not done in a cube ahead of time. String manipulations can be done within Tableau via simple MDX: LEFT([Product].[Product Categories].DataMember.MemberValue,LEN([Product].[Product Categories].DataMember.MemberValue)-5)	When connecting to relational sources, string manipulations can be accomplished directly with Tableau calculated fields. This can be helpful when changing case, creating zip 5 out of zip 5+4 and many other manipulations. STRING FUNCTIONS

Specific Feature	Cube	Relational
Data Types	Data types (e.g. String, Date, Number) and roles (dimension and measure) are explicitly defined within a cube. This ensures that pre aggregations and attributes show up in the right windows in Tableau.	Tableau will automatically detect the column type of a column in a relational database. This limits the amount of data manipulation you need to do in Tableau. When you want to change data types in Tableau, you can do that with a right click: CHANGE DATA TYPES
KPI / Scorecard	Within a cube, you can define attributes that contain information about what KPI groups a certain member falls within. You can also create threshold KPIs directly within Tableau with a simple Calculation/Parameter: [Internet Sales Amount] >= [Desired Sales]	Within a relational database, you can create KPI calculations very quickly with simple calculated fields. KPI CALCULATIONS
Actions	Tableau Actions are fully supported with cubes. Actions also work from a cube to relational sources in the same workbook. Cube based actions are not supported in Tableau. ACTIONS	Actions work within relational data sources as well as across/between cubes and relational sources. This allows both types of data to communicate between each other.
Hierarchies	This is one of the key benefits of using Tableau with a cube. Tableau also supports drilling down asymmetrically or raggedly through the hierarchy.	Relational databases do not have hierarchies built within them, but Tableau allows you to easily build these on the fly. BUILDING HIERARCHIES
Quick Filters	When connected to a cube, you see quick filters as part of the hierarchy and its structure is exposed. You can filter at multiple levels within a hierarchy. If you have a single attribute, you can avoid the structured view and have a single-level hierarchy	Quick filters in a relational database show up as one level, without structure. You can put these quick filters on the visualizations and dashboard with a single right-click.

Specific Feature	**Cube**	**Relational**
Extracts	Cubes, with their pre-aggregated nature, are inherently fast and do not require extraction.	Relational databases can often be slow. Tableau provides the Fast Data Engine as a method to provide screaming fast performance to relational data. Tableau provides the *option* to connect live or extract the data.
Aliases	Business friendly names are a common thing for cubes to house. When connecting to Essbase, you can use any alias file that was defined within the cube	In relational databases, you can create your alias values to change the name of members of a dimension. This is helpful when you want to group things together and give it a new name.
Formatting	Formatting of fields (percentages, currency, etc.) are defined within a cube. This ensures that numbers are shown correctly without any intervention by the user	Relational data sources typically do not have any inherent formatting. Defining percentages and currency can be done directly inside of Tableau. You can even set the default formatting, so that each time that measure is used, it is shown in the correct format.
Sort order	Cubes provide a developer to define the sort order of members within an attribute/dimension. This ensures that when using attributes, your members are shown in the correct order every time. This is helpful when there is non-standard formatting	Relational databases do not have default sorting as part of their definition. In Tableau, a user can define a sort driven from a measure (sort state by sales) as well as a default manual sorting.
Fiscal Calendars	Cubes provide a developer the ability to define different calendars within the cubes. Whether that is a fiscal calendar, 4-4-5 calendar or your own retail calendar, Tableau inherits this structure and behaves the same.	Relational databases typically only store dates and not different calendars. In Tableau, a user can define a new fiscal year start, or write date calculations to mimic a 4-4-5 or retail calendar.