

Best Practices for Designing Efficient Tableau Workbooks

Tableau 10 Edition

Alan Eldridge
Tableau Software

About this document

Once again, I would like to acknowledge that this document is a distillation of materials written by many authors. Much of what I have done is to bring their work together into a single document and apply some structure. Some 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 continued excellent work and lack of copyright infringement claims this document would not exist.

I would also like to thank the many people who have reviewed this document for accuracy and readability. Your attention to detail, and patient explanations have made this into a much more legible document than I could ever have produced on my own.

This document has been updated to reflect the capabilities of Tableau 10. Future releases of Tableau will provide new features and capabilities that may change some of these recommendations.

Thanks,

Alan Eldridge

June 2016

TL;DR

I'm often told by people who have read or recommended this whitepaper that it's too long. My response is that it is as long as it needs to be to cover such a broad scope of material at an appropriate depth.

However, below is a summary of the key points in the document:

- There is no silver bullet for inefficient workbooks. Start by looking at the performance recorder to understand where the time is going. Long-running queries? Lots of queries? Slow calculations? Complex rendering? Use this insight to focus your efforts in the right direction.
- The recommendations in this document are just that – recommendations. While they represent a level of best practice, you need to test if they will improve performance in your specific case. Many of them can be dependent on structure of your data, and the data source you are using (e.g. flat file vs. RDBMS vs. data extract).
- Extracts are a quick and easy way to make most workbooks run faster.
- The cleaner your data is and the better it matches the structure of your questions (i.e. the less preparation and manipulation required), the faster your workbooks will run.
- The majority of slow dashboards are caused by poor design – in particular, too many charts on a single dashboard, or trying to show too much data at once. Keep it simple. Allow your users to incrementally drill down to details, rather than trying to show everything then filter.
- Work with the data you need and no more – both in terms of the fields you reference as well as the granularity of the records you return. It allows Tableau to generate fewer, better, faster queries and reduces the amount of data that needs to be moved from the data source to Tableau's engine. It also reduces the size of your workbooks so they are easier to share and open faster.
- While reducing the data, make sure you use filters efficiently.
- Strings and dates are slow, numbers and Booleans are fast.

Finally, some of the recommendations in this document only have a material impact if you are working with big and/or complex data sets. What is big or complex? That depends... but it doesn't hurt to follow these recommendations in all your workbooks as you never know when your data will grow. Practice makes perfect.

Contents

About this document	2
TL;DR	3
Introduction	6
What is Tableau good for?	6
What is Tableau not good for?	6
Understanding efficiency	8
What is an “efficient” workbook?.....	8
Why should you care about efficiency?.....	8
The laws of physics	9
Tools of the trade.....	11
Performance recorder.....	11
Logs	13
Tableau Server performance views	14
Monitoring and testing	15
Other tools	15
Is it my workbook design?	18
Good dashboard design	18
Tweak your dashboard for performance	21
Good worksheet design	26
Efficient filters.....	32
Is it my calculations?	42
Calculation types.....	43
Analytics	47
Calculations vs. native features	47
Impact of data types	47
Performance techniques.....	48
Is it my queries?	53
Automatic optimisations.....	53
Joins	59
Blending	60
Data integration	63
Custom SQL.....	64
Alternatives to custom SQL.....	66
Is it my data?	68
General advice	68

Data sources.....	68
Data preparation.....	76
Data extracts.....	77
Data governance.....	83
Is it the environment?.....	85
Upgrade.....	85
Test Tableau Desktop on the Server.....	85
Separate refreshes and interactive workloads.....	85
Monitor and tune your Server.....	85
Infrastructure.....	86
Conclusion.....	88

Introduction

What is Tableau good for?

At Tableau, 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 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 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 Server includes the concept of report subscriptions which allows a user to subscribe themselves (and in Tableau 10, others) to receive a report via email, however customers sometimes want a more flexible solution. Tableau can be used to create other forms of push-delivery system but this is not a native feature of Tableau. It requires development of a custom solution built around the TABCMD utility, or the introduction of 3rd party solutions such as *VizAlerts* (<http://tabsoft.co/1stldFh>) or *Push Intelligence for Tableau* from Metric Insights (<http://bit.ly/1HACxul>).
- 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 data 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.

Understanding efficiency

What is an “efficient” workbook?

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:

- **Simple** – Is it easy to create the workbook and will it be easy to maintain in the future? Does it take advantage of the principles of visual analysis to clearly communicate the message of the author and the data?
- **Flexible** – Can the workbook answer multiple questions the users want to ask, or just one? Does it engage the user in an interactive experience or is it simply a static report?
- **Fast** – Does the workbook respond quickly enough for the users? This could mean time to open, time to refresh or time to respond to interaction. This is a subjective measure but in general we want a workbook to provide an initial display of information and to respond to user interactions within seconds.

The performance of a dashboard is impacted by:

- the visual design at both the dashboard and worksheet levels – e.g. how many elements, how many data points, use of filters and actions, etc.;
- the calculations – e.g. what kind of calculation, where is the calculation performed, etc.;
- the queries – e.g. how much data is returned, is it custom SQL;
- the data connections and underlying data sources;
- some differences between Tableau Desktop and Tableau Server;
- other environmental factors such as hardware configuration and capacity.

Why should you care about efficiency?

You should care for several reasons:

- working efficiently as an analyst or workbook author gets you to your answer faster;
- working efficiently keeps you in the “flow” of analysis. This means that you are thinking about the questions and the data, rather than how to manipulate the tool to achieve an outcome;
- creating a workbook with a flexible design reduces the need to create and maintain multiple workbooks that address similar requirements;
- creating a workbook with a simple design makes it easier for others to pick up your workbook and make further iterations on your work;
- perceived responsiveness is an important success factor for end users when viewing reports and dashboards, so making your workbooks run as quickly as possible makes for happier users.

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

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

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

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

The laws of physics

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

If it is slow in the data source, it will be slow in Tableau

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

If it is slow in Tableau Desktop, it will (almost always) be slow in Tableau Server

A workbook that performs slowly in Tableau Desktop won't get any faster by publishing it to Tableau Server. Often users have a perception that their workbook will run faster in Tableau Server because the server has more CPU/RAM/etc. than their local PC. In general, workbooks will perform slightly slower on Tableau Server because:

- there are multiple users all sharing the server resources to generate workbooks simultaneously (although counterintuitively, you might find that your workbook responds more quickly as you start to share it, due to the caching mechanisms within Tableau Server); and
- the server has to do additional work to render the dashboards and charts rather than this being done on the client workstation.

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

The exception to this rule is if Tableau Desktop is encountering resource limits that aren't present on the server – e.g. your PC does not have enough RAM to support the data volume you are analysing, or the server has a faster/lower latency connection to the data source. 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.

Newer is better

The development team at Tableau are constantly working to improve the performance and usability of our software. Upgrading to the latest release of Tableau Desktop and Tableau Server can sometimes yield significant improvements in performance without requiring any changes to the workbook. For example – many customers reported 3x (and more) performance improvements in their workbooks just by upgrading from V8 to V9 and performance improvement continues to be a

focus of Tableau 10 and beyond. Of course, this isn't an issue with Tableau Online as it is always being upgraded to the latest versions as they are released.

This goes for maintenance releases as well as major/minor releases. You can get more information about the Tableau release cycle and specific details for each release on the release notes page:

<http://tabsoft.co/1Oy3LZT>

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

<http://tabsoft.co/1RjD4sy>

Less is more

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

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

Scalability is not the same as performance

Scalability is about ensuring that we can support multiple users viewing shared workbooks.

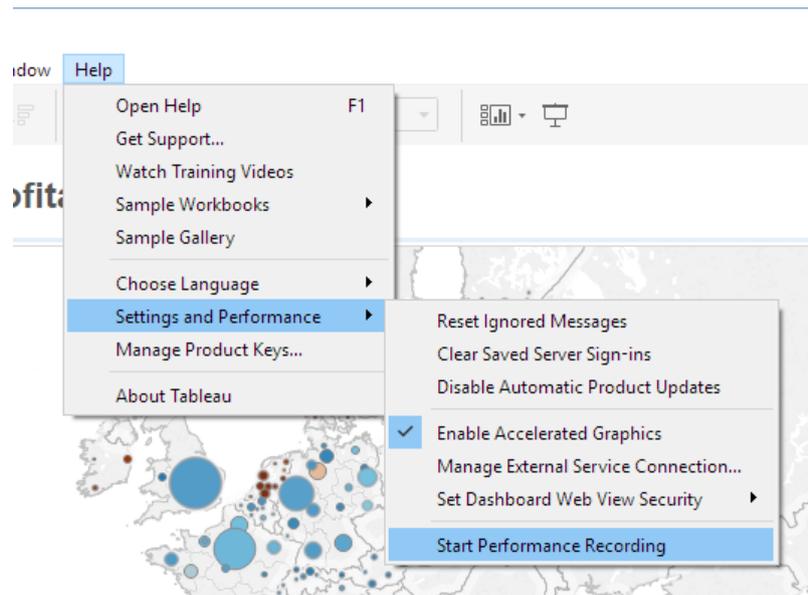
Performance is about ensuring that an individual workbook runs as quickly as possible. While many of the recommendations offered in this document will have a positive influence on scalability for workbooks published to Tableau Server, the principal focus of this document is about improving performance.

Tools of the trade

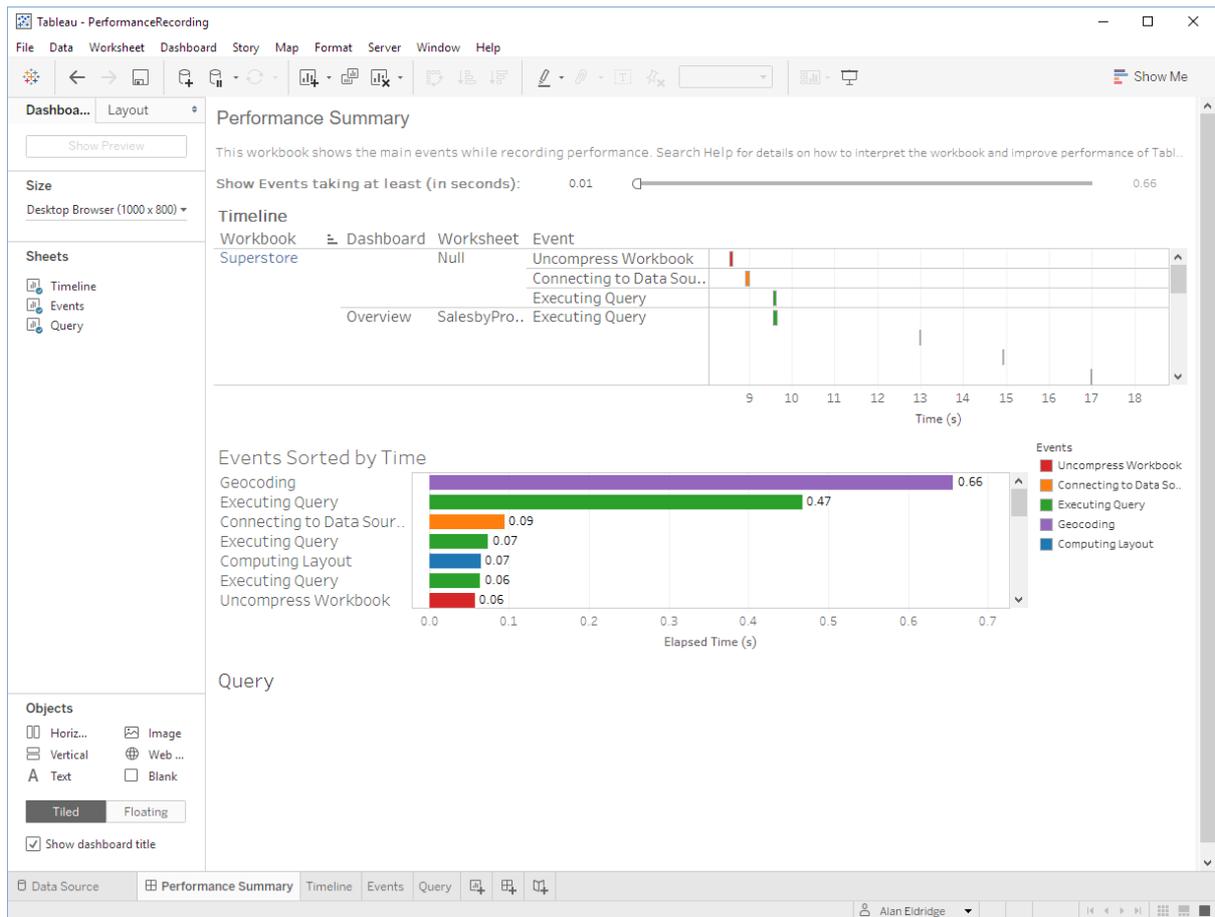
In order to understand the performance of your workbooks, you need to understand a) what is happening and b) how long it is taking. This information is captured in multiple locations depending on where you are running the workbook (i.e. in Tableau Desktop or Tableau Server) and at multiple levels of detail. This section outlines the various options available.

Performance recorder

The first place you should look for performance information is the Performance Recorder feature of Tableau Desktop and Tableau Server. In Tableau Desktop you enable this feature under the Help menu:



Fire up Tableau and start performance recording, then open your workbook (it's best practice to not have other workbooks open while doing this so you are not inadvertently competing for resources). Interact with it as if you were an end user and when you feel you have gathered enough data go back in the help menu and stop recording. Another Tableau Desktop window will open at this point with the data captured:



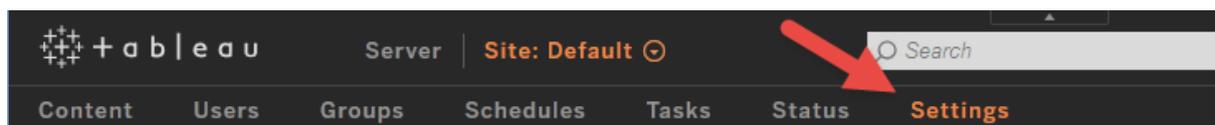
You can now identify the actions in the workbook that take the most time - for example in the above image the selected query from the Timeline worksheet takes 30.66 seconds to complete. Clicking on the bar shows the text of the query being executed. As the output of the performance recorder is a Tableau Workbook, you can also create additional views to explore this information in other ways.

Note: by default, events that take < 0.1 secs are not shown. You can include these by adjusting the filter at the top of the dashboard however your focus should be on the long-running tasks. Setting it to 1 sec is good practice.

It is also possible to create performance recordings on Tableau Server to help identify issues that arise when the workbook is published. By default, performance recording is not enabled on Tableau Server – this is a feature that can be controlled on a per-site basis.

A server administrator can enable performance recording site by site.

1. Navigate to the site for which you want to enable performance recording.
2. Click **Settings**:



3. Under Workbook Performance Metrics, select **Record workbook performance metrics**.
4. Click **Save**.

When you want to create a performance recording:

1. Open the view for which you want to record performance. When you open a view, Tableau Server appends ":iid=<n>" after the URL. This is a session ID. For example:

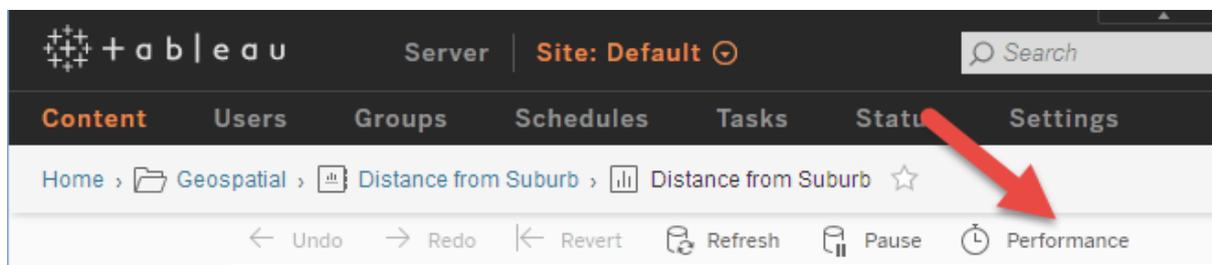
```
http://<tableau_server>/#/views/Coffee_Sales2013/USSalesMarginsByAreaCode?iid=1
```

2. Type `:record_performance=yes&` at the end of the view URL, immediately before the session ID. For example:

```
http://<tableau_server>/#/views/Coffee_Sales2013/USSalesMarginsByAreaCode?record_performance=yes&iid=1
```

3. Load the view.

A visual confirmation that performance recording has started is the **Performance** option in the view toolbar:



When you are done and ready to view the performance recording:

1. Click **Performance** to open a performance workbook. This is an up-to-the-minute snapshot of performance data. You can continue taking additional snapshots as you continue working with the view; the performance data is cumulative.
2. Move to a different page or remove `:record_performance=yes` from the URL to stop recording.

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

<http://tabsoft.co/1RdF420>

Logs

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

```
{ "ts": "2015-05-24T12:25:41.226", "pid": 6460, "tid": "1674", "sev": "info", "req": "-", "sess": "-", "site": "-", "user": "-", "k": "end-query", "v": { "protocol": "4308fb0", "cols": 4, "query": "SELECT [DimProductCategory].[ProductCategoryName] AS [none:ProductCategoryName:nk], \n
```

```

[DimProductSubcategory].[ProductSubcategoryName] AS
[none:ProductSubcategoryName:nk],\n SUM(CAST(([FactSales].[ReturnQuantity])
as BIGINT)) AS [sum:ReturnQuantity:ok],\n SUM([FactSales].[SalesAmount]) AS
[sum:SalesAmount:ok]\nFROM [dbo].[FactSales] [FactSales]\n INNER JOIN
[dbo].[DimProduct] [DimProduct] ON ([FactSales].[ProductKey] =
[DimProduct].[ProductKey])\n INNER JOIN [dbo].[DimProductSubcategory]
[DimProductSubcategory] ON ([DimProduct].[ProductSubcategoryKey] =
[DimProductSubcategory].[ProductSubcategoryKey])\n INNER JOIN
[dbo].[DimProductCategory] [DimProductCategory] ON
([DimProductSubcategory].[ProductCategoryKey] =
[DimProductCategory].[ProductCategoryKey])\nGROUP BY
[DimProductCategory].[ProductCategoryName],\n
[DimProductSubcategory].[ProductSubcategoryName] ", "rows":32, "elapsed":0.951}
}

```

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

Tableau Server performance views

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

Analysis	
Dashboards that monitor Tableau Server activity.	
Dashboard	Analysis
Traffic to Sheets	Usage and users for published sheets.
Traffic to Data Sources	Usage and users for published data sources.
Actions by All Users	Actions for all users.
Actions by Specific User	Actions for a specific user, including items used.
Actions by Recent Users	Recent actions by users, including last action time and idle time.
Background Tasks for Extracts	Completed and pending extract task details.
Background Tasks for Non Extracts	Completed and pending background task details (non-extract).
Stats for Load Times	Sheet load times and performance history.
Stats for Space Usage	Space used by published workbooks and data sources, including extracts and live connections.
Server Disk Space	Current and historical disk space usage, by server node.
Tableau Desktop License Usage	Summary of usage for Tableau Desktop licenses
Tableau Desktop License Expirations	Expiration information for Tableau Desktop licenses

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

<http://tabsoft.co/1RjCCL2>

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

<http://tabsoft.co/1RjCACR>

Monitoring and testing

TabMon

TabMon, is an open source cluster monitor for Tableau Server that allows you to collect performance statistics over time. TabMon is community-supported, and we are releasing the full source code under the MIT open source license.

TabMon records system health and application metrics out of the box. It collects built-in metrics like Windows Perfmon, Java Health, and Java Mbean (JMX) counters on Tableau Server machines across a network. You can use TabMon to monitor physical (CPU, RAM), network, and hard-disk usage. You can track cache-hit ratio, request latency, active sessions, and much more. It displays the data in a clean, unified structure, making it easy to visualize the data in Tableau Desktop.

TabMon gives you full control over which metrics to collect and which machines to monitor, no scripting or coding required. All you need to know is the machine and the metric name. TabMon can run both remotely and independently of your cluster. You can monitor, aggregate, and analyze the health of your cluster(s) from any computer on your network with almost no added load to your production machines.

You can find more information on TabMon here:

<http://bit.ly/1ULFelf>

TabJolt

TabJolt is a “point-and-run” load and performance testing tool specifically designed to work easily with Tableau Server. Unlike traditional load-testing tools, TabJolt can automatically drive load against your Tableau Server without script development or maintenance. Because TabJolt is aware of Tableau’s presentation model, it can automatically load visualizations and interpret possible interactions during test execution.

This allows you to just point TabJolt to one or more workbooks on your server and automatically load and execute interactions on the Tableau views. TabJolt also collects key metrics like average response time, throughput, and 95th percentile response time, and captures Windows performance metrics for correlation.

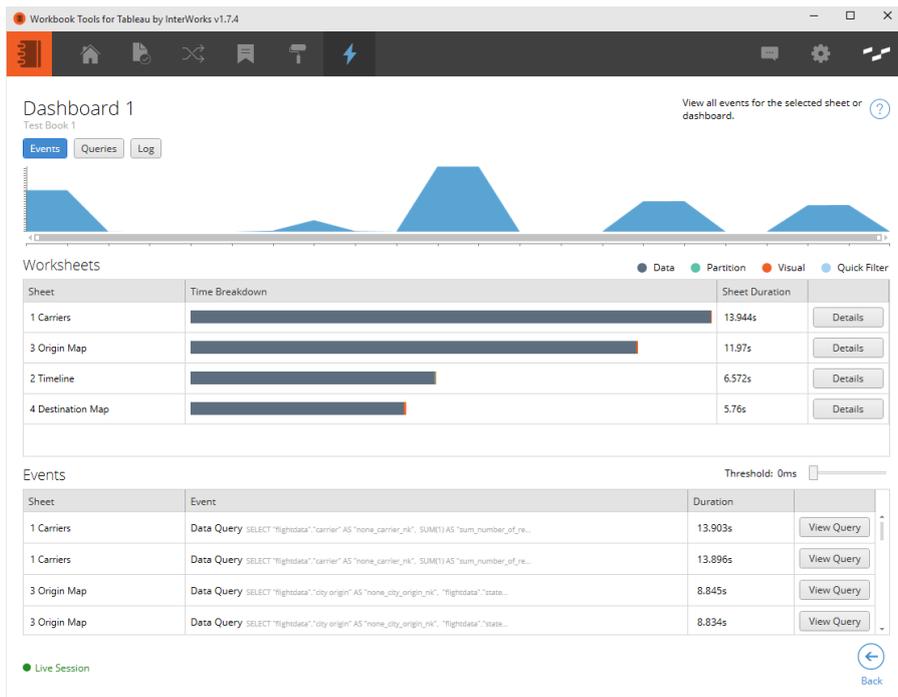
Of course, even with TabJolt, users should have sufficient knowledge of Tableau Server’s architecture. Treating Tableau Server as a black box for load testing is not recommended, and will likely yield results that aren’t in line with your expectations. Tabjolt is an automated tool and cannot easily replicate the variety of human interaction, so consider carefully if your Tabjolt results reflect real-world outcomes.

You can find more information on TabJolt here:

<http://bit.ly/1ULFtgi>

Other tools

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



Palette Software also have a product called Palette Insight that captures performance information from Tableau Server and allows you to do capacity planning, identify resource-intensive users and workbooks, do user access auditing and build chargeback models.



Additionally, most modern DBMS platforms include admin tools that allow the tracing and analysis of running queries. Your DBA can be a source of great assistance if your performance recordings indicate query run time is a key factor.

If you believe the interaction with the client browser and the server is an issue, you can also look at tools like Telerik Fiddler or your browser's developer tools to take a deeper look at the traffic between client and server.

Is it my workbook design?

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

Good dashboard design

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..	2,278	\$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.

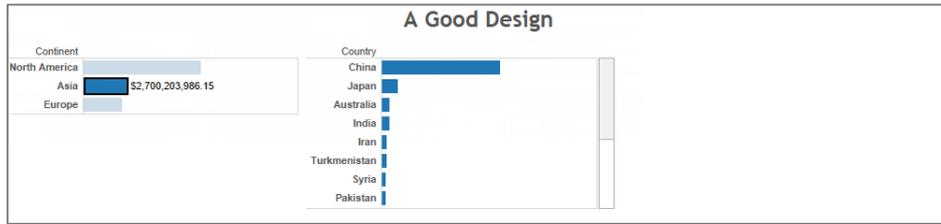
A. Datum Advanced Digit.

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:



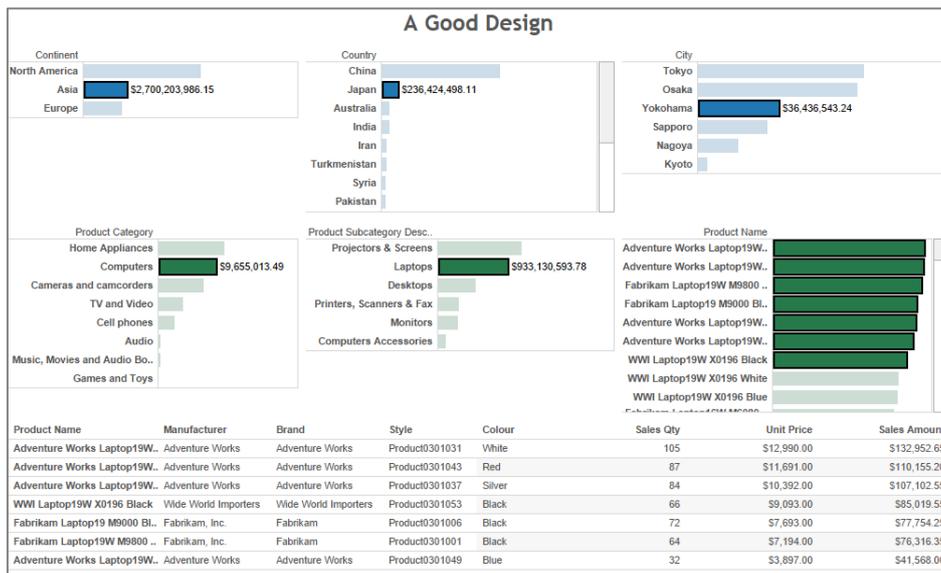
Selecting one or more of the elements shows the next level of detail:



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.



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

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

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

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

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

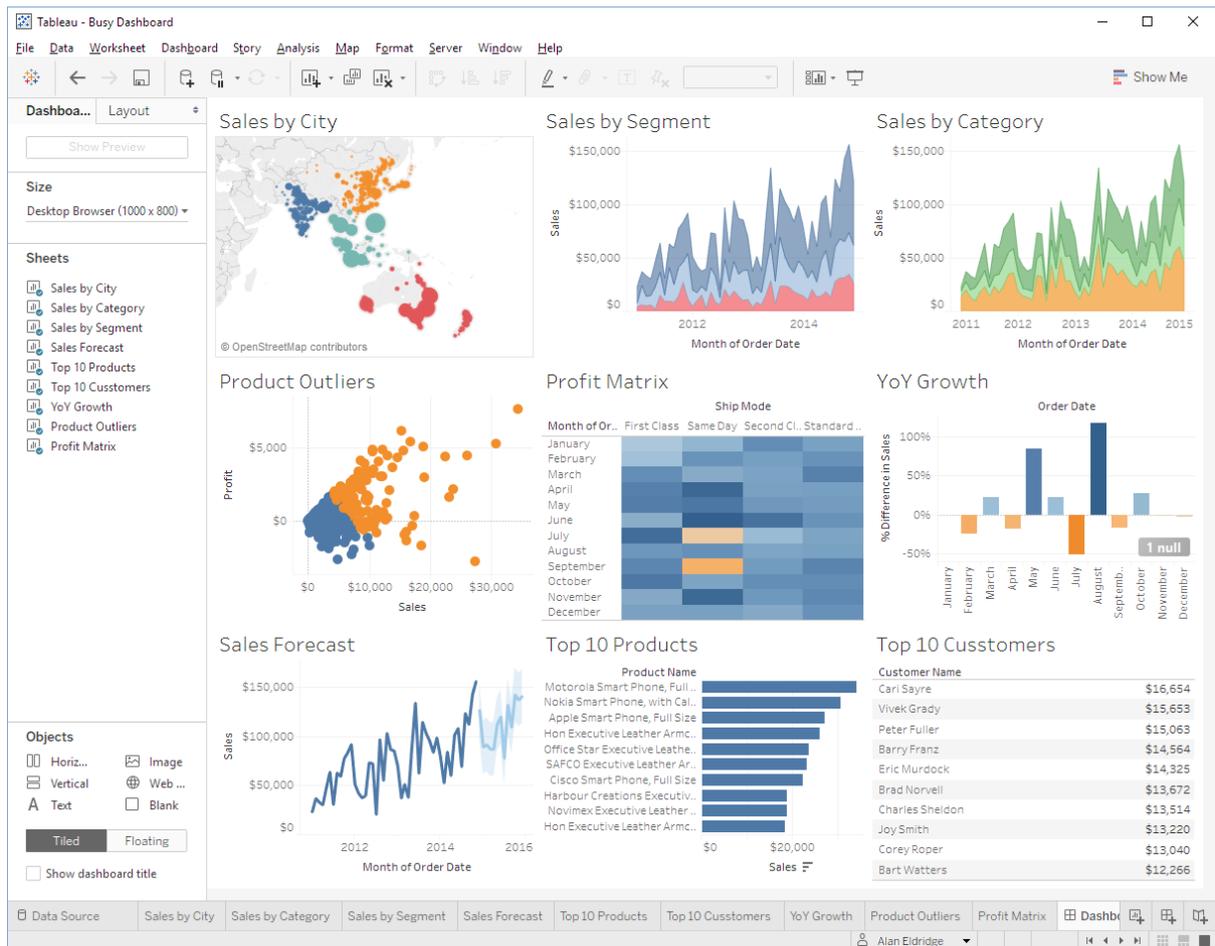
Keep it simple

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

The following points are contributors to complexity:

Too many worksheets per dashboard

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



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

Too many filter cards

Filter cards are a very powerful feature of Tableau that allow us to create rich, interactive dashboards for end users. However, each filter can require a query in order to enumerate the options so adding too many to your dashboard can unexpectedly cause the dashboard to take a long time to render. Also, when you use “show relevant values” on a filter, it requires a query to update the shown values each time other filters are changed. Use this feature sparingly.

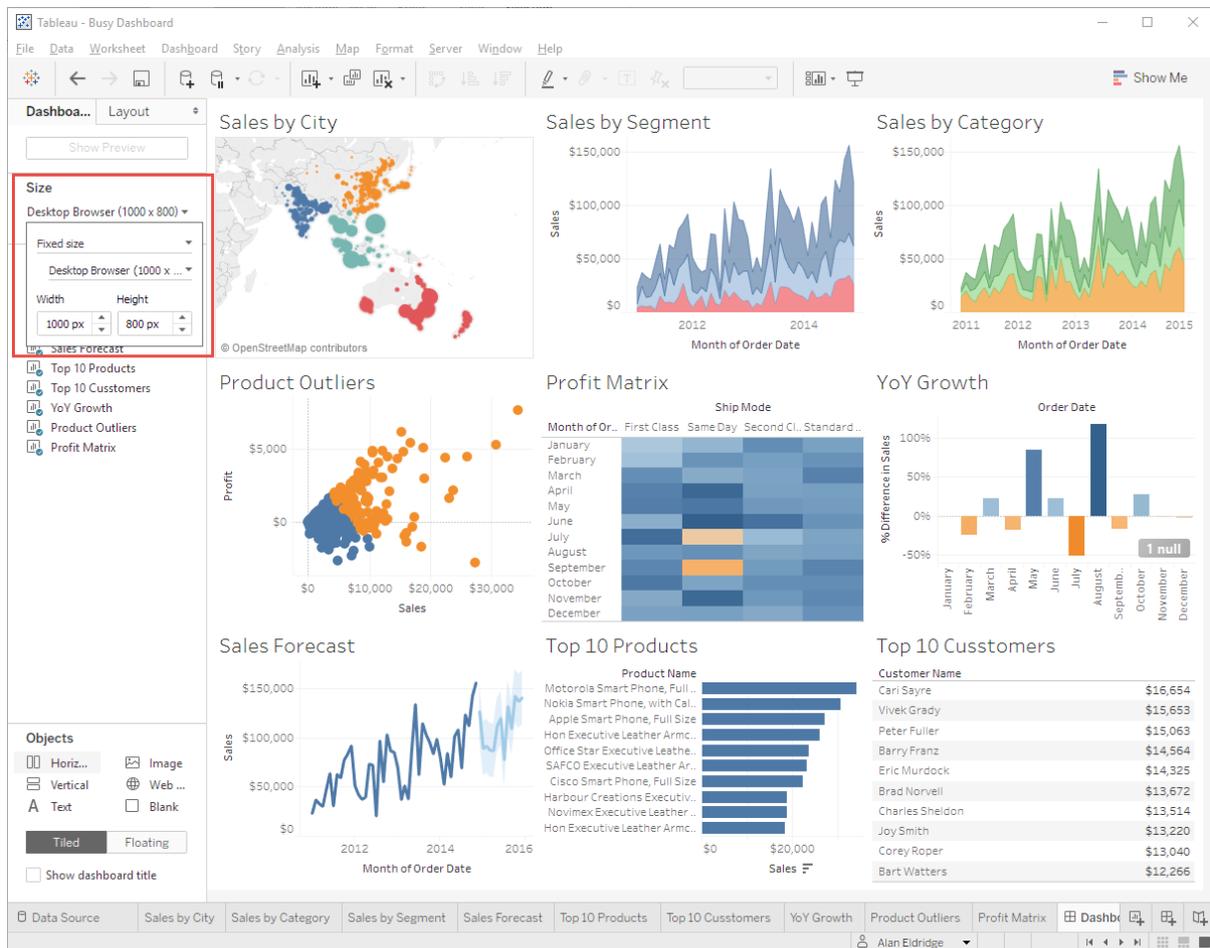
Also, if you have your filters applied to multiple worksheets, be aware that each change will trigger multiple queries as all the affected worksheets that are visible will update (non-visible worksheets are not run). If this takes seconds to complete it can give a poor user experience. If you expect users to make several changes to a multi-select filter type, consider showing the “apply” button so the user can trigger the update when they are done changing their selections.

Tweak your dashboard for performance

Once we have ensured our dashboard is as simple as can be, we can tweak our design to take advantage of caching for further performance improvement.

Fixed-size dashboards

One of the easiest things we can do to improve performance is to check that your dashboard is a fixed size.



Part of the rendering process in Tableau is creating a layout – how many rows/columns to display for small multiples and crosstabs; the number and interval of axis ticks/grid lines to draw; the number and location of mark labels to be shown; etc. This is determined by the size of the window in which the dashboard will be displayed.

If we have multiple requests for the same dashboard but from windows of different sizes, we would need to generate a layout for each request. By setting the dashboard layout to a fixed size we ensure that we only need to create a single layout that can be reused across all requests. For server-side rendering this is even more important as fixed size dashboards mean we can also cache and share the bitmaps that are rendered on the server, improving both performance and scalability.

Device specific dashboards

In Tableau 10 we are introducing a new feature called device specific dashboards. This capability allows you to create custom dashboard layouts that are automatically selected based on the device being used.

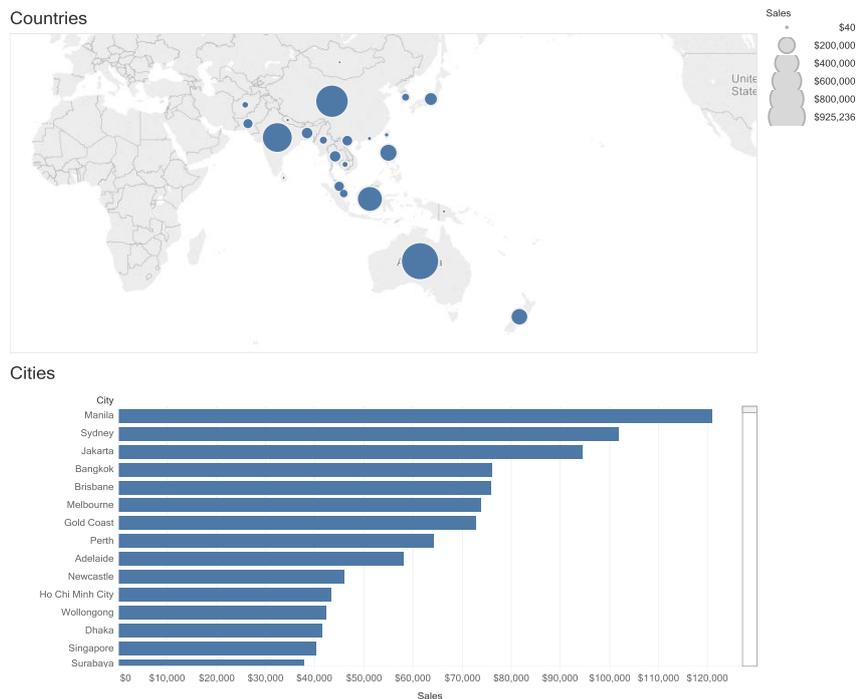
We select the layout to use based on screen size:

- ≤ 500 px on smallest axis – phone
- ≤ 800 px on smallest axis – tablet
- > 800 px – desktop

Given that different devices will have different screen sizes within those ranges and because devices can rotate, you will generally want to set phone/tablet layouts to auto-resize. This will give the best viewing experience across the devices, however it will impact cache reuse (both the presentation model cache and the image tile cache for server-side rendering). In general, the benefit of sizing properly for the device will outweigh the impact on caching but it is something to consider. Note that once users start to use the workbook you will eventually populate the models and bitmaps for most common screen sizes and performance will improve.

Using the viz level of detail to reduce queries

Although best practice is generally to only use the fields you need for each worksheet, sometimes we can improve performance by pulling more information on one worksheet to avoid queries on another. Consider the following dashboard:



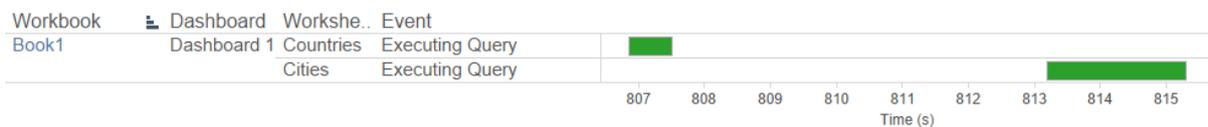
Building this as one would expect, the execution plan will result in two queries – one for each worksheet:

Performance Summary

This workbook shows the main events while recording performance. Search Help for details on how to interpret the workbook and improve performance of Tableau.

Show Events taking at least (in seconds): 0.10 2.11

Timeline



```
SELECT [Superstore APAC].[City] AS [City],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
GROUP BY [Superstore APAC].[City]
```

```
SELECT [Superstore APAC].[Country] AS [Country],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
GROUP BY [Superstore APAC].[Country]
```

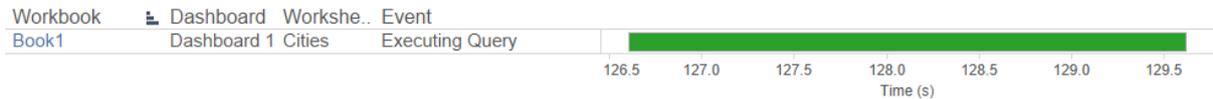
If we make a change to our dashboard design and add Country to the Cities worksheet (on the Detail shelf), Tableau can complete the dashboard with just a single query. Tableau is smart enough to run the query for the Cities worksheet first and then use the query results cache to provide the data for the Countries worksheet. This feature is called “query batching”.

Performance Summary

This workbook shows the main events while recording performance. Search Help for details on how to interpret the workbook and improve performance of Tableau.

Show Events taking at least (in seconds): 0.10 3.02

Timeline

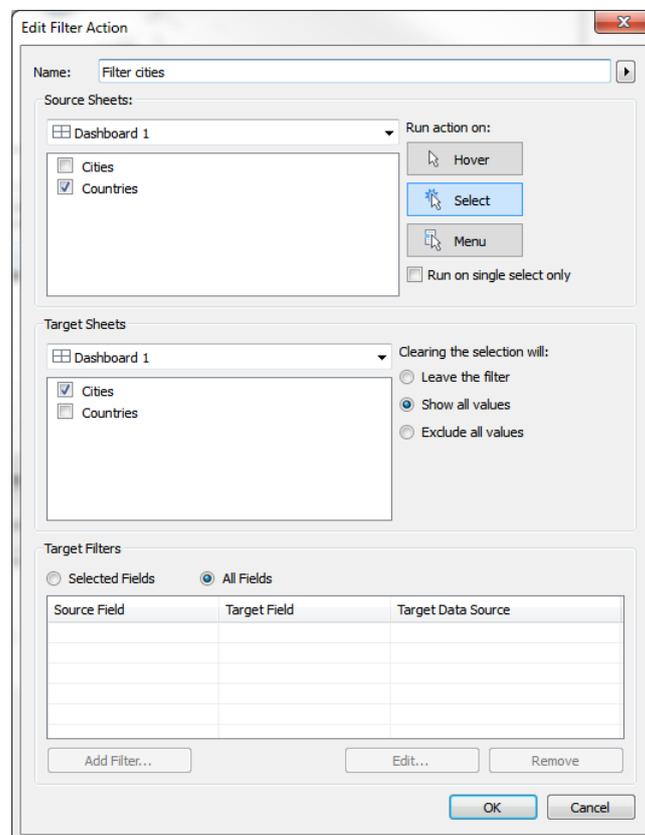


```
SELECT [Superstore APAC].[City] AS [City],
       [Superstore APAC].[Country] AS [Country],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
GROUP BY [Superstore APAC].[City],
         [Superstore APAC].[Country]
```

Clearly this can't be done in all cases as adding a dimension to the viz alters the level of detail so it could result in more marks being displayed. However, when you have a hierarchical relationship in your data like the above example, this is a useful technique as it won't affect the visible level of detail.

Using the viz level of detail to optimise actions

We can use a similar approach with actions to reduce the number of queries we need to run. Consider the same dashboard from above (prior to optimisation), but now we add a filter action from the Countries worksheet to the Cities worksheet.



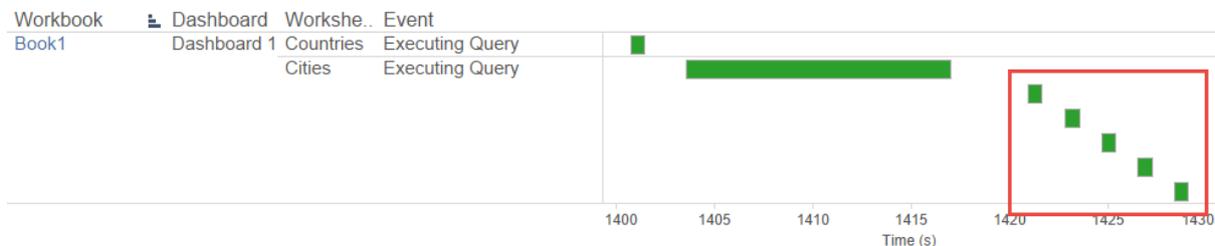
When we trigger this action by clicking on a mark on the map, we see that Tableau needs to run a query to determine the values in the Cities worksheet. This is because there is no data in the query result cache on the city-country relationship.

Performance Summary

This workbook shows the main events while recording performance. Search Help for details on how to interpret the workbook and improve performance of Tableau.

Show Events taking at least (in seconds): 0.10 13.47

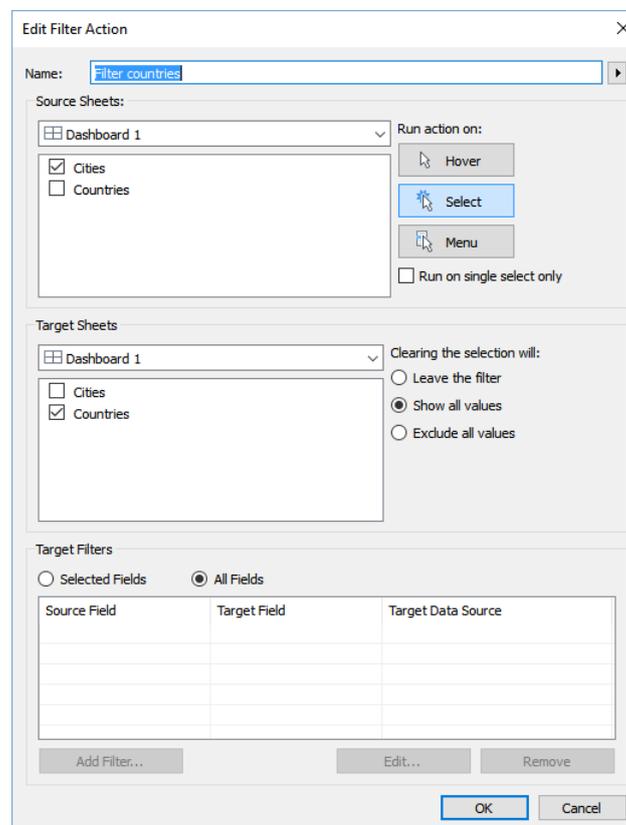
Timeline



```
SELECT [Superstore APAC].[City] AS [City],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
WHERE ([Superstore APAC].[Country] = 'Australia')
GROUP BY [Superstore APAC].[City]
```

If we add Country to the Cities worksheet, we now have sufficient information in the query result cache that we can do these filters without needing to go back to the data source.

We can similarly optimise in situations where the source worksheet is more detailed than the target. If we were to take the default action definition where we filter using “all fields”:



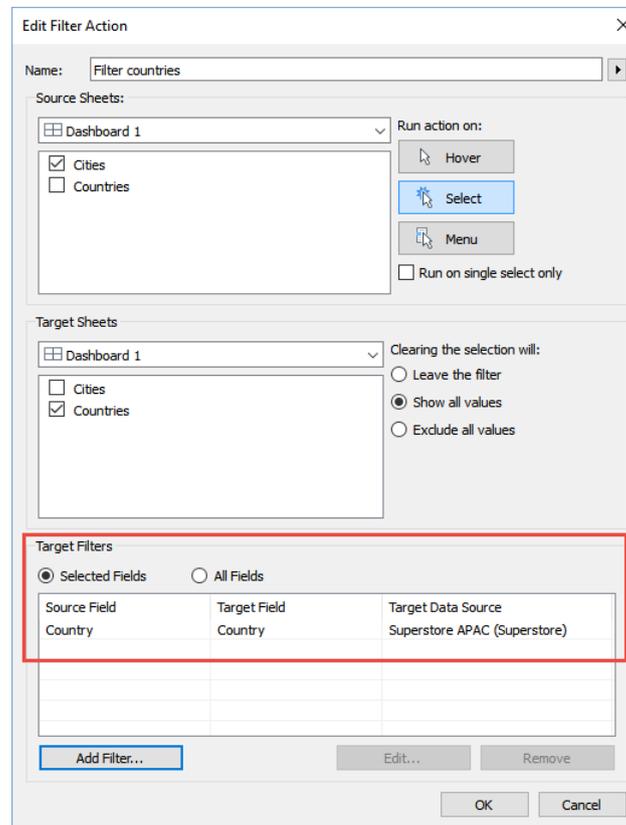
This causes the workbook to run a query for each action because the filter clause references Country and City which can't be served from the query result cache for the Countries worksheet.

```

SELECT [Superstore APAC].[Country] AS [Country],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
WHERE (([Superstore APAC].[City] = 'Sydney') AND ([Superstore APAC].[Country] =
'Australia'))
GROUP BY [Superstore APAC].[Country]

```

If we change the action to just filter based on Country:



We can now satisfy this filter from the query result cache so we don't need to query back to the data source. As above, you need to consider if changing the level of detail will affect the design of your sheet – if not then this can be a useful technique.

Good worksheet design

The next level down from the dashboard is the worksheet. In Tableau the design of the worksheet is intrinsically related to the queries that are run. Each worksheet will generate one or more queries, so at this level we are trying to ensure we generate the most optimal queries possible.

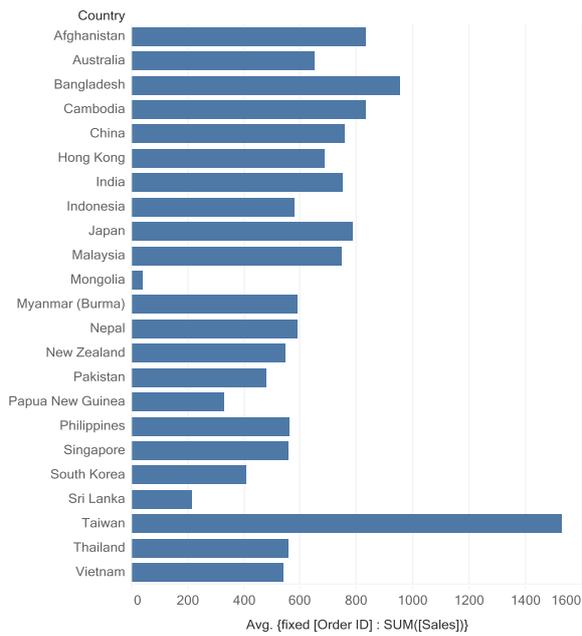
Only include the fields you need.

Look at the detail shelf and remove any fields that aren't directly used on the viz, needed in the tooltip or required to drive the required level of mark detail. This makes the query faster in the data source, and requires less data to be returned in the query results. There are some exceptions to this rule as we've already explored – to help query batching eliminate similar queries in other worksheets – however these are less common and are dependent on not changing the visual level of detail of the sheet.

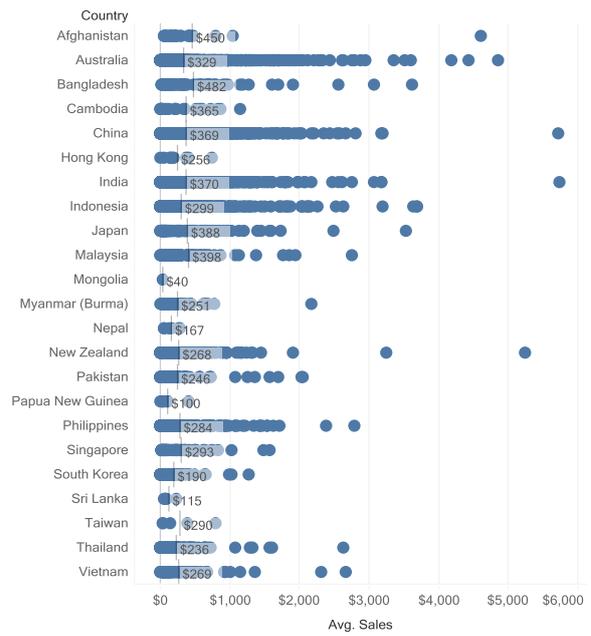
Show the minimum number of marks to answer the question

In Tableau there are often multiple ways to calculate the same number. Consider the following dashboard – both worksheets answer the question “what is the average order size per country?”

Avg Order Size 1



Avg Order Size 2

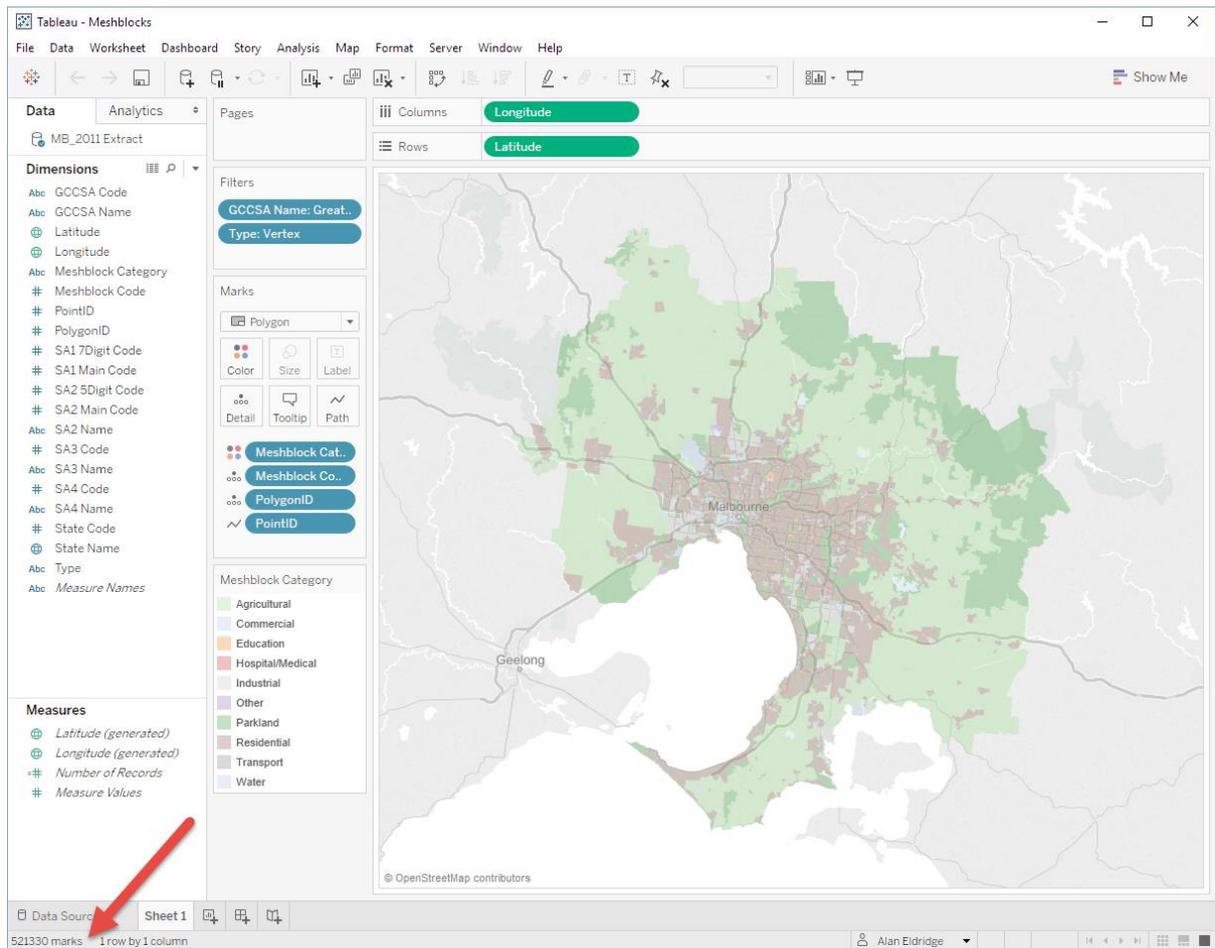


Sheet 1 only shows a single mark, representing the average order size for each country, so we only return 23 records from the data source. However, the Sheet 2 shows a mark for each order within each country and then calculates the average as a reference line. This requires us to fetch 5436 records from the data source.

Sheet 1 is the better solution if we are just interested in the original question – what is the average order size per country – however Sheet 2 answers this but also offers deeper insights into the range of order sizes, allowing us to identify outliers.

Avoid highly complex vizs

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

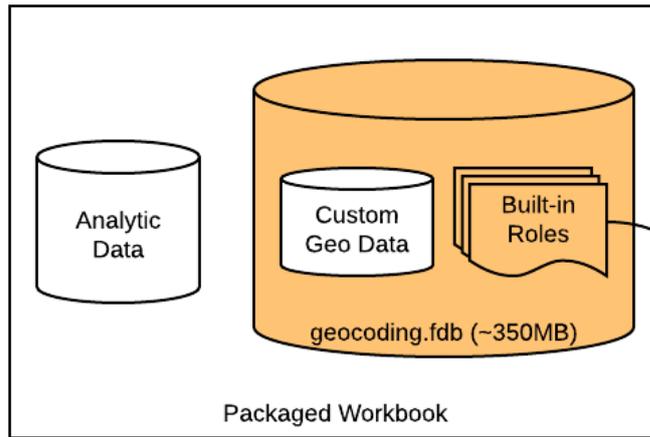


While there is no hard and fast rule on what defines “too many marks” be aware that more marks mean more CPU and RAM is required to render them. Watch out for large crosstabs, scatterplots and/or maps with complex custom polygons.

Maps

Custom geocoding

When you import a custom geocoding role, it is written into the geocoding database – a Firebird DB file that is by default stored in C:\Program Files\Tableau\Tableau 10.0\Local\data\geocoding.fdb. If you use the role in a workbook and save it as a packaged workbook, the entire database file is compressed into the TWBX file – all 350-ish MB of it!

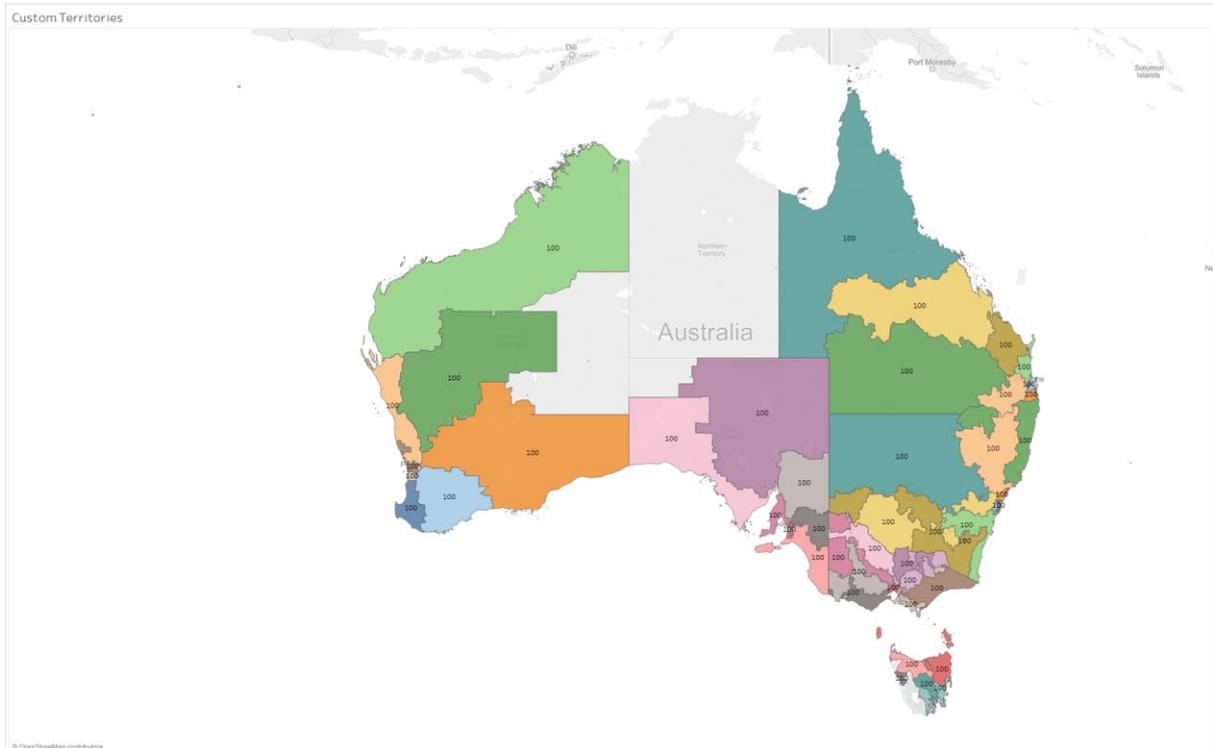


Name	Type	Compressed size	Password ...	Size	Ratio	Date modified
AreaCode.tds	Tableau Datasource	2 KB	No	9 KB	82%	9/06/2016 1:29 PM
City.tds	Tableau Datasource	2 KB	No	13 KB	85%	9/06/2016 1:29 PM
CMSA.tds	Tableau Datasource	2 KB	No	14 KB	86%	9/06/2016 1:29 PM
Congress.tds	Tableau Datasource	2 KB	No	14 KB	86%	9/06/2016 1:29 PM
Country.tds	Tableau Datasource	3 KB	No	17 KB	87%	9/06/2016 1:29 PM
County.tds	Tableau Datasource	2 KB	No	14 KB	86%	9/06/2016 1:29 PM
GEOCODING.FDB	FDB File	118,407 KB	No	359,280 KB	68%	9/06/2016 1:28 PM
State.tds	Tableau Datasource	2 KB	No	14 KB	86%	9/06/2016 1:29 PM
Suburb.tds	Tableau Datasource	3 KB	No	15 KB	86%	9/06/2016 1:29 PM
ZipCode.tds	Tableau Datasource	2 KB	No	8 KB	81%	9/06/2016 1:29 PM

This makes the resulting TWBX file a) very large as the geocoding payload is about 110MB compressed; and b) quite slow to open as the initial decompression has to operate over a much larger data set. A more efficient approach is to not import the data as a custom geocoding role, but rather to use blending within the workbook to combine the analytic data with the geospatial data. Using this approach, the geocoding.fdb file is not embedded and the TWBX only contains the analytic and custom geocoding data.

Custom territories

A new feature of Tableau 10 is custom territories where users can combine areas from the internal geocoding databases to create aggregated regions.



The initial rendering of custom territories based on many lower-level regions can be very slow so use this capability cautiously. Once completed though, the custom territories are cached and good performance can be expected.

Filled maps vs. point maps

Filled map marks (whether used on a map or as a mark on some other type of chart) are expensive when using client-side rendering as we need to send over the polygon data for the shape and this can be quite complex. Consider using a symbol map instead if you are seeing slow rendering performance.

Polygon marks

Any vizzes that use polygon marks will force Tableau Server to perform [server-side rendering](#) which can impact the end user experience. Use them sparingly.

Other factors

Large crosstabs

In earlier versions of this document we recommended you to avoid large crosstabs as they rendered very slowly. The underlying mechanics of this viz type have been improved in recent releases and crosstabs now render as quickly as other small multiple chart types. However, we still advise you to think carefully about using large crosstabs as they require lots of data to be read from the underlying data source and are not analytically useful.

Tooltips

By default, placing a dimension on the tooltip shelf causes it to be aggregated using the ATTR() attribute function. This requires two aggregations to be performed in the underlying data source – MIN() and MAX() – and both results to be passed back in the result set. See the [using ATTR\(\)](#) section later in the document for more information.

If you are not concerned with the possibility of having multiple dimension values, a more efficient solution is to just use one aggregation instead of the default ATTR(). Pick either MIN() or MAX() – it doesn't matter which you use, but pick one and stick to it to maximise your chances of a cache hit.

Another option – if you know it won't affect the visual level of detail of your viz – is to place the dimension on the level of detail shelf instead of the tooltip shelf. This results in the dimension field being used directly in the SELECT and GROUP BY clauses of the query. We recommend you test if this will perform better than the single aggregation as it can depend on the performance of your data platform.

Legends

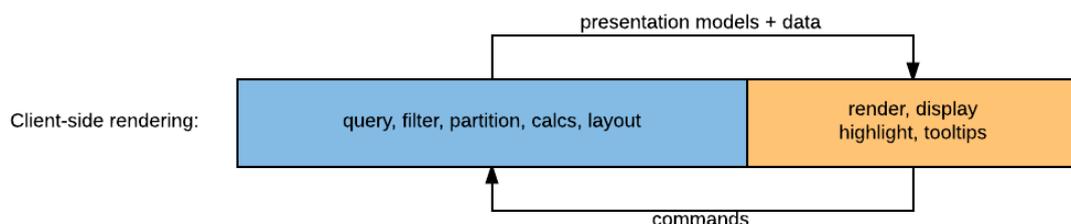
Generally, legends are not a cause of performance problems as their domains are fulfilled from the query result cache. However, they can add to the rendering overhead if the enumerated domain is large because the data needs to be transferred to the client browser. If this is the case the legend is generally not useful, so just remove it.

Page shelf

Some users think that the page shelf works in the same way as the filter shelf – that it reduces the number of records being returned from the data source. This is not correct – the query for the worksheet will return records for all marks across all pages. If you have a page dimension with a high degree of cardinality (i.e. lots of unique values) this can increase the size of the worksheet query significantly, affecting performance. Use this capability sparingly.

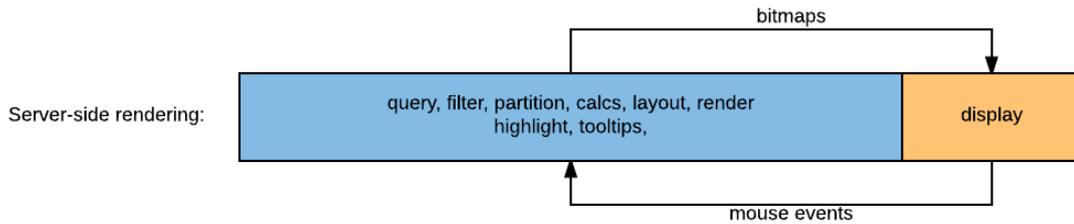
Client-side vs. server-side rendering

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, many view interactions are faster because they are interpreted and rendered right there in the browser.



(Blue - done on server; orange - done in client browser.)

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. The trade-off though is that simple interactions such as tooltips and highlighting can be slow in server-side rendering as they require a server round-trip.



(Blue - done on server; orange - done in client browser.)

Tableau Server is configured to automatically handle all of these situations using a complexity threshold as the trigger for rendering a view on the server instead of in the web browser. The threshold is different for PCs and mobile devices so there are cases where a view opened from a PC web browser might be client-rendered but the same view opened from a tablet web browser is server-rendered. Filtering can also change the rendering behaviour – a workbook might initially open using server-side rendering but then change to client-side rendering when a filter is applied. Also, if a viz uses polygon marks or the pages shelf it will only use server-side rendering, even if it would otherwise meet the criteria for client-side rendering, so use these features with caution.

As an administrator, you can test or fine tune this setting for both PCs and tablets. See the following link for more details:

<http://tabsoft.co/1WgBLLh>

Efficient filters

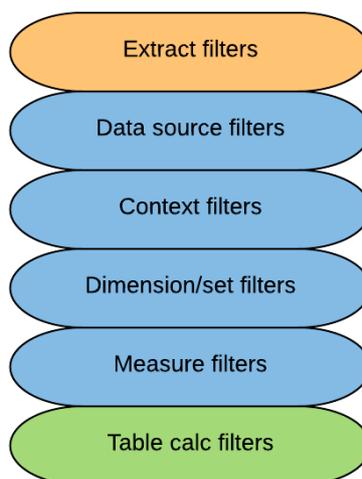
Filtering in Tableau is extremely powerful and expressive. However, inefficient filters are one of the most common causes of poorly performing workbooks and dashboards. The following sections lay out a number of best practices for working with filters.

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

Filter types

Filter priority

In Tableau, filters are applied in the following order:



Extract filters

These filters are only applicable when using data extracts, but in that case are logically applied before all other filters. They limit the data that is fetched from the underlying data source, and can be either dimension or measure filters. Additionally, they can perform either a TOP or SAMPLE to reduce the number of records returned, depending on the source data platform.

Data source filters

Data source filters are the highest level of filter available on live connections. A key difference between data source filters and context filters is that data source filters are scoped to the whole data source whereas context filters are set for each worksheet. This means that when used in a published data source, data source filters can be enforced whereas context filters are applied at the worksheet level.

Data source filters can be an effective way of putting a constraint on a data source to prevent end users from accidentally running a massive query – e.g. you could put a data source filter limiting queries on a transaction table to only be for the last 6 months.

Context filters

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

You should use context filters when they are required to get the correct answer (e.g. a filtered TopN). For example, consider a view that shows the top 10 products by SUM(Sales), filtered by region. Without a context filter, the following query is run:

```
SELECT [Superstore APAC].[Product Name] AS [Product Name],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
     INNER JOIN (
       SELECT TOP 10 [Superstore APAC].[Product Name] AS [Product Name],
                  SUM([Superstore APAC].[Sales]) AS [$_alias__0]
       FROM [dbo].[Superstore APAC] [Superstore APAC]
       GROUP BY [Superstore APAC].[Product Name]
       ORDER BY 2 DESC
     ) [t0] ON ([Superstore APAC].[Product Name] = [t0].[Product Name])
WHERE ([Superstore APAC].[Region] = 'Oceania')
GROUP BY [Superstore APAC].[Product Name]
```

This returns the contribution in Oceania for the global top 10 products. If what you really wanted was the top 10 products within the Oceania region, you would add the Region filter to the context and the following query would be run:

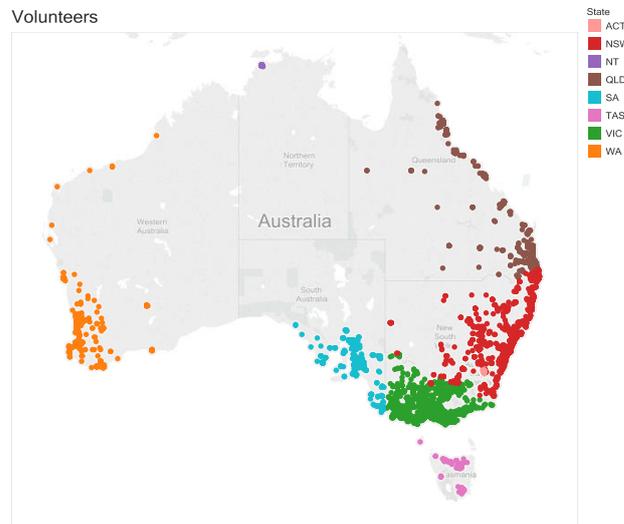
```
SELECT TOP 10 [Superstore APAC].[Product Name] AS [Product Name],
             SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok],
             SUM([Superstore APAC].[Sales]) AS [$_alias__0]
FROM [dbo].[Superstore APAC] [Superstore APAC]
WHERE ([Superstore APAC].[Region] = 'Oceania')
GROUP BY [Superstore APAC].[Product Name]
ORDER BY 3 DESC
```

Historically, context filters were implemented as a temp table in the data source. This is no longer the case – in almost all situations context filters are implemented as part of the data source query (as shown above) or processed locally in the data engine.

You should no longer use context filters as a mechanism for improving query performance.

Filtering categorical dimensions

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



There are several ways we could filter the map to just show the postcodes for Western Australia (the orange 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 as 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 `WHERE IN` clause or, if there are many values, by creating a temp table populated with the selected values and using an `INNER JOIN` with this and the main table(s). 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.

Range of value

Using the range of values 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 range of value 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.

Slicing filters

Slicing filters are filters on dimensions that are not used in the viz (i.e. they are not part of the viz level of detail). For example, you might have a viz showing total sales by country, but the viz is filtered by region. In this example the query run is as follows:

```
SELECT [Superstore APAC].[Country] AS [Country],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
WHERE ([Superstore APAC].[Region] = 'Oceania')
GROUP BY [Superstore APAC].[Country]
```

These filters become more complex if we are slicing on the result of an aggregation. For example, if we filtered the above viz not by region, but to show the sales for the top 10 profitable products, Tableau needs to run two queries – one at the product level to isolate the top 10 profitable products, and another at the country level restricted by the results of the first query:

```
SELECT [Superstore APAC].[Country] AS [Country],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
INNER JOIN (
  SELECT TOP 10 [Superstore APAC].[Product Name] AS [Product Name],
              SUM([Superstore APAC].[Profit]) AS [$__alias__0]
  FROM [dbo].[Superstore APAC] [Superstore APAC]
  GROUP BY [Superstore APAC].[Product Name]
  ORDER BY 2 DESC
) [t0] ON ([Superstore APAC].[Product Name] = [t0].[Product Name])
GROUP BY [Superstore APAC].[Country]
```

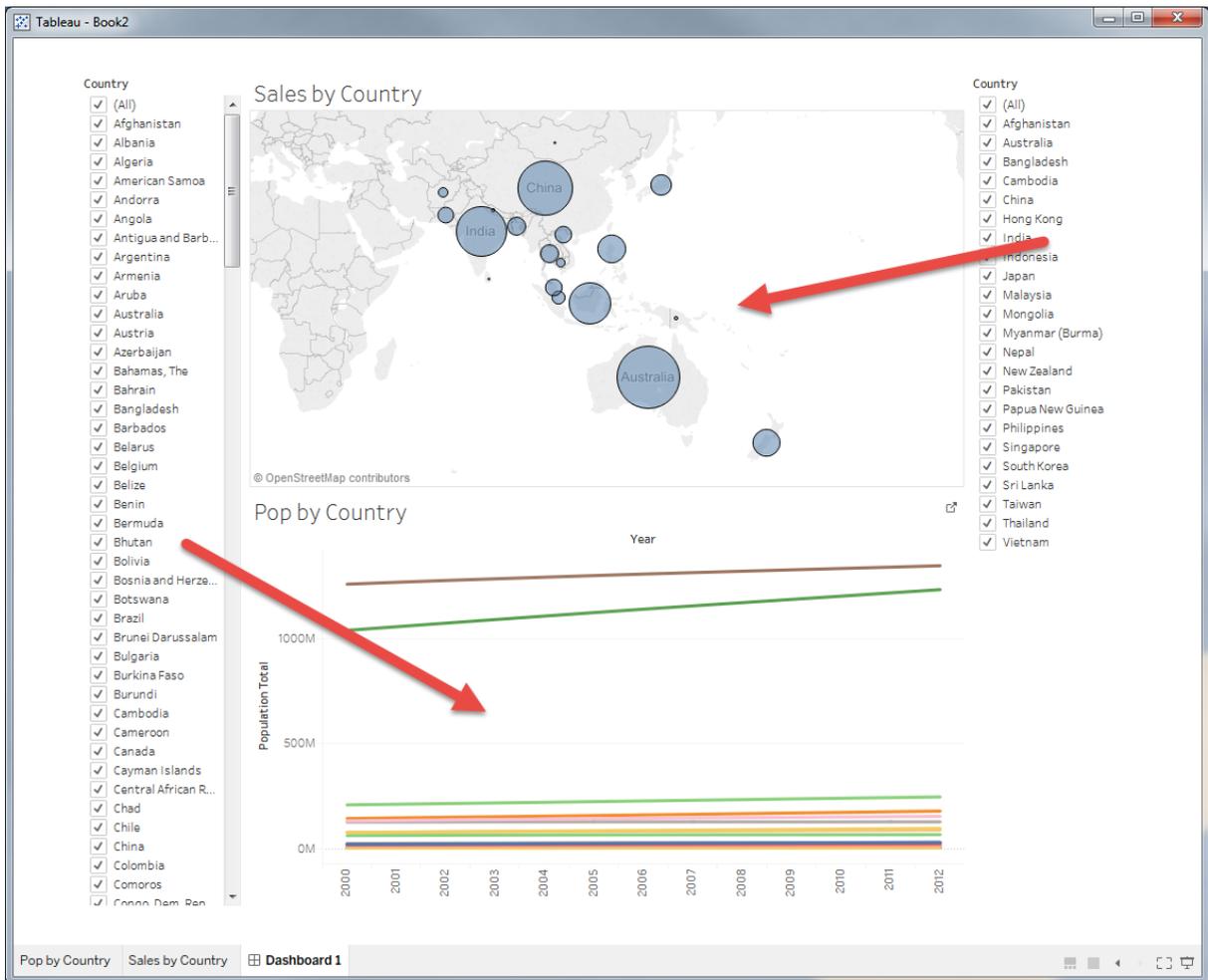
Use caution when using slicing filters as they can be expensive to evaluate. Also note that because the dimension isn't part of the query result cache, we cannot do fast in-browser filtering on slicing filters (see the [earlier section](#) about client-side vs. server-side rendering).

Cross data source filters

Cross data source filtering is a new feature in Tableau 10. It allows a filter to be applied to multiple data sources that have one or more fields in common. Relationships are defined the same way as for blending – automatically based on name/type matches or manually via custom relationships in the Data menu.

Cross-database filters have the same performance implications as quick filters on a dashboard. When you change them they can cause multiple zones to update, potentially requiring multiple queries. Use them judiciously, and if you expect users to make multiple changes, consider showing the “apply” button to only fire queries when selections are complete.

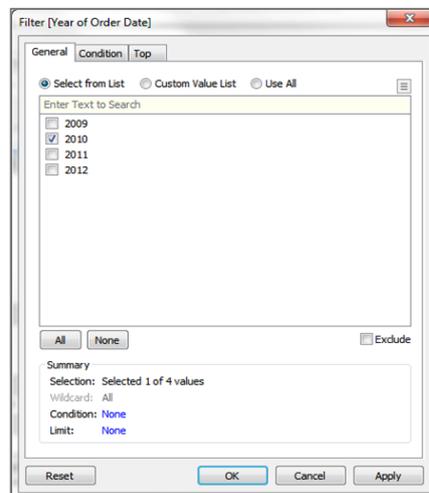
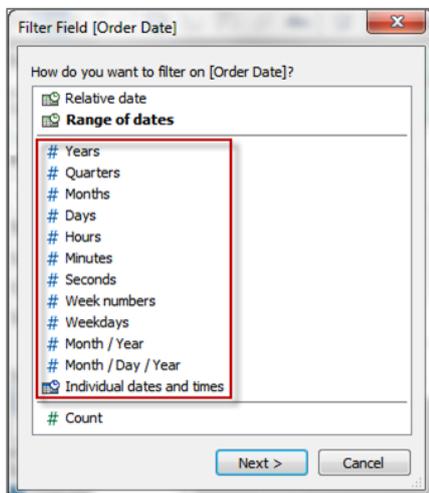
Note also that the domain of the filter is taken from the “primary” data source – i.e. the first-used data source of the sheet where the filter was created. If a related field has different domains in different data sources, you need to be careful which one you use as the same filter can end up showing different values, as shown below:



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



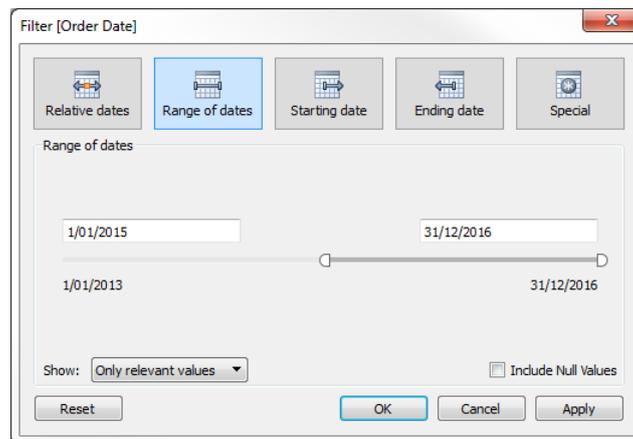
Sometimes you may want to filter to include specific individual dates or entire date levels. This type of filter is called a discrete date filter because you are defining discrete values instead of a range. This filter type results in the date expression being passed to the database as a dynamic calculation:

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

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

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

Range of date

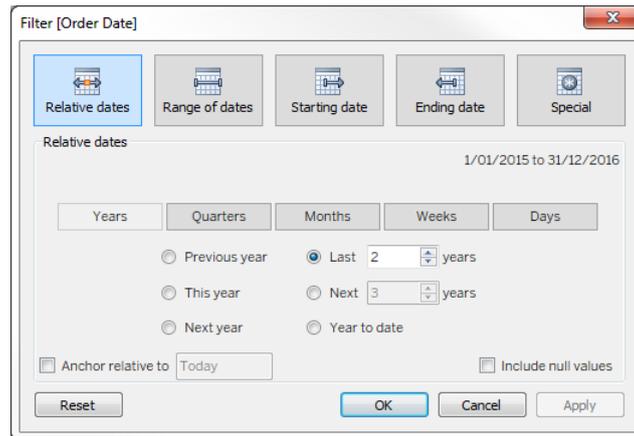


This type of filter is used when you want to specify a range of contiguous dates. It results in the following query structure being passed to the database:

```
SELECT SUM([factOrders].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[factOrders] [factOrders]
WHERE (([factOrders].[Order Date] >= {d '2015-01-01'}) AND
([factOrders].[Order Date] <= {d '2016-12-31'}))
GROUP BY ()
```

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 SUM([factOrders].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[factOrders] [factOrders]
WHERE (([factOrders].[Order Date] >= {ts '2015-01-01 00:00:00'}) AND
([factOrders].[Order Date] < {ts '2017-01-01 00:00:00'}))
GROUP BY ()
```

As you can see, the resulting WHERE clause uses a range of dates syntax so this is also an efficient form of date filter.

Note that due to the changing nature of date filters that are relative to the current date/time - either via the relative date filter option above or by an explicit use of NOW() or TODAY() in a filter formula – the query cache is not as efficient for queries that use them. Tableau flags these as "transient queries" in the cache server and their results are not kept as long as other query results.

Filter cards

Showing too many filter cards will 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 lots of filters in it to make it highly interactive, ask yourself whether multiple dashboards with different levels and themes would work better (hint: yes, it probably would).

Enumerated vs. non-enumerated

Enumerated filter cards require Tableau to query the data source for field values before the filter card can be rendered. These include:

- Multiple value list – *all dimension members*
- Single value list – *all dimension members*
- Compact List – *all dimension members*
- Slider – *all dimension members*
- Measure filters – *MIN and MAX values*
- Ranged date filters – *MIN and MAX values*

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

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

Relevant values

Enumerated filter cards 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 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 filter card will show all values in the context regardless of other filters on the view. The filter does not need to re-query the database when dimension or measure filters are changed however it does need to re-query if the context changes.
- Only Relevant Values - when you select this option other filters are considered and only values that pass these filters are shown. For example, a filter on State will only show the Eastern states when a filter on Region is set. Consequently, the 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.

Filter card alternatives

There are alternatives to using filter cards that provide a similar analytic outcome but do so without the additional query overhead. For example, 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 – for versions prior to Tableau 10, filters operate only within a single data source. In Tableau 10 and later, filters can be set to work across related data sources

- 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 – for versions prior to Tableau 10, filters operate only within a single data source. In Tableau 10 and later, filters can be set to work across related data sources
- CONS:
 - Filter actions are more complex to set up than filter cards
 - Actions do not present the same user interface as parameters or filter cards – 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

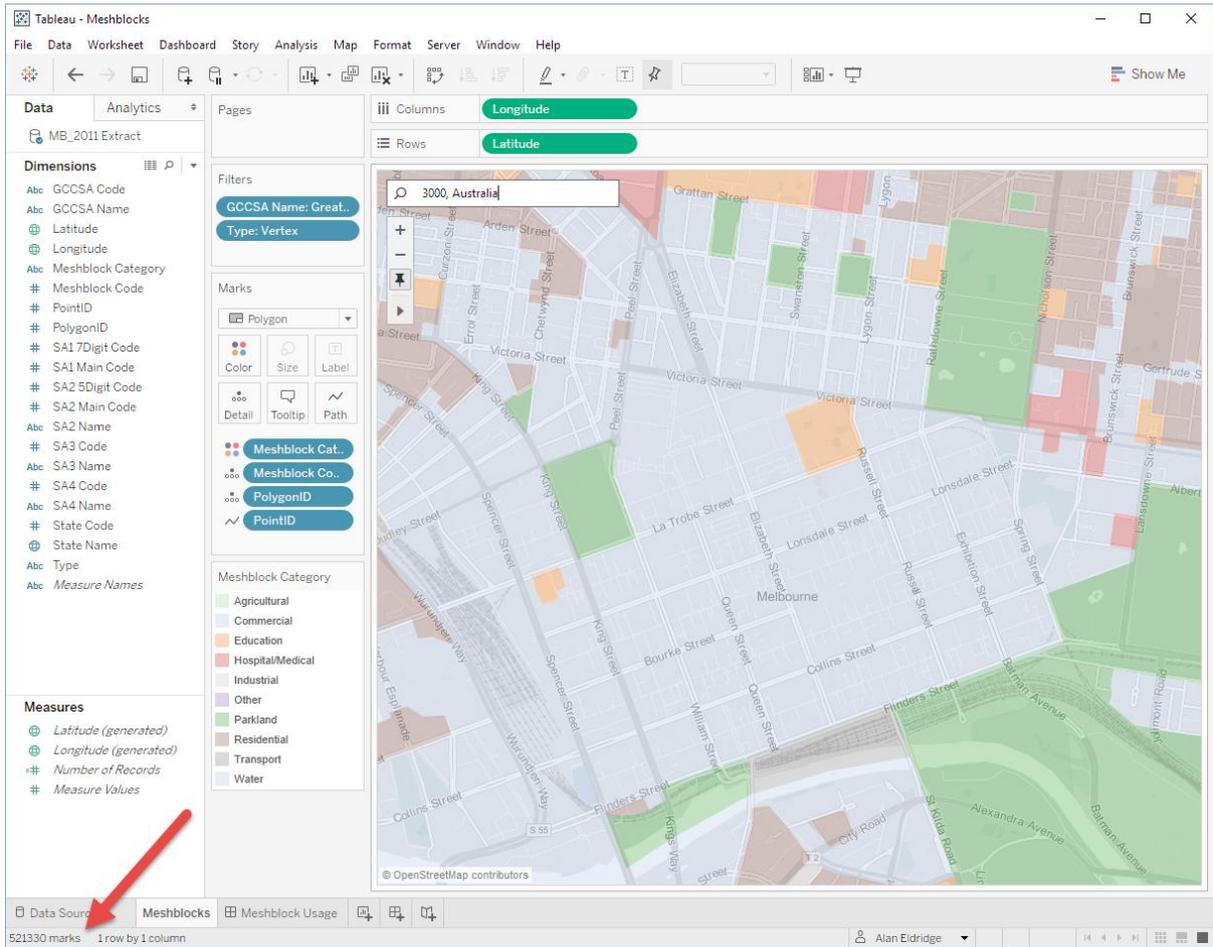
For a further discussion on alternate design techniques that don't rely heavily on filter cards, see the [earlier section](#) on good dashboard design.

User filters

When a workbook contains user filters – either through the “Create User Filter...” dialog or through calculated fields that use any of the built-in User functions like ISMEMBEROF() – the model cache is not shared across user sessions. This can result in a much lower rate of cache reuse, which in turn can mean more work for the Tableau Server. Use these filter types with consideration.

Zooming vs. filtering

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



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

Is it my calculations?

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

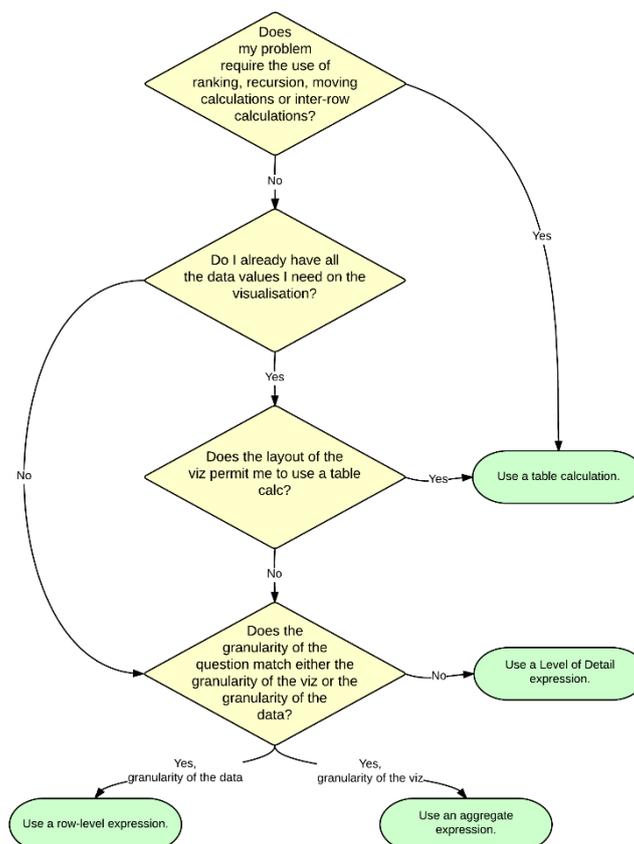
Within a calculated field you may define a hardcoded constant (like a tax rate, for instance), do very simple mathematical operations like subtraction or multiplication (e.g. revenues minus cost), use more complex mathematical formulas, perform logical tests (e.g. IF/THEN, CASE), do type conversions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

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

There are four different calculation types in Tableau:

- row-level calculations,
- aggregate calculations,
- table calculations, and
- Level of Detail expressions.

Use the following flowchart as a guide for how to select the best approach:



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

<http://tabsoft.co/1I0SsWz>

Calculation types

Row-level and aggregate calculations

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

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

The **YEAR** calculation is a row-level calculation and the **SUM(SALES)** is an aggregate calculation.

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

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

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

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

Finally, query fusion (combining multiple logical queries into a single actual query) can modify the query run against the data source. It can result in multiple measures from multiple worksheets being combined into a single query if those other worksheets share the same granularity.

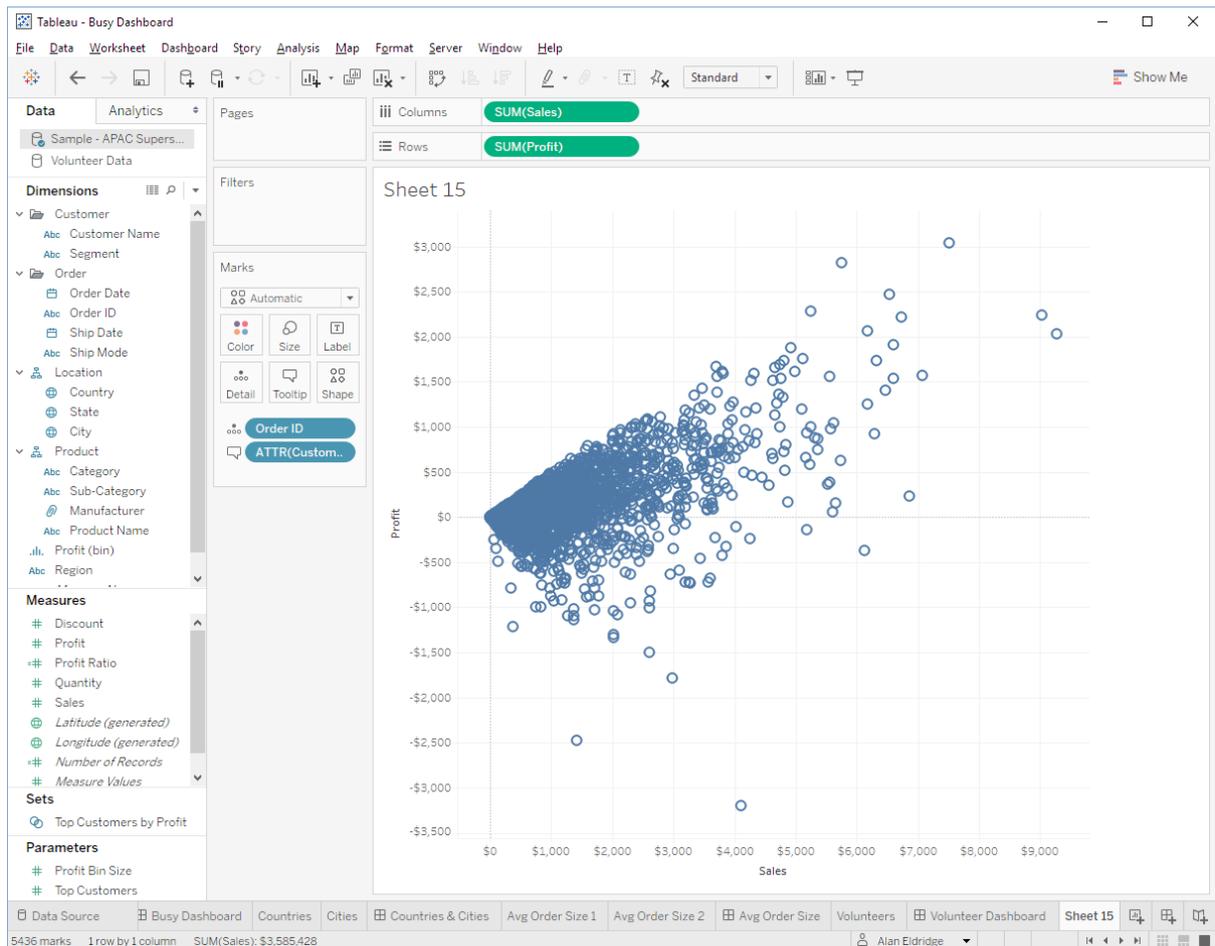
Using ATTR()

ATTR() is a powerful function, often used as an aggregation for categorical dimensions. In simple terms, it returns a value if it is unique, else it returns *. Technically, the function performs the following logical test:

```
IF MIN([dimension]) = MAX([dimension])
  THEN [dimension]
  ELSE "*"
END
```

This requires two aggregations to be performed in the underlying data source – MIN() and MAX() – and both results to be passed back in the result set. If you are not concerned with the possibility of having multiple dimension values, a more efficient solution is to just use one aggregation – either MIN() or MAX(). It doesn't matter which you use, but pick one and stick to it to maximise your chances of a cache hit.

To show how ATTR() impacts the data source, we created the following viz:



By default, placing a dimension on the tooltip shelf uses ATTR() as the default aggregation. This causes the following query:

```
SELECT [Superstore APAC].[Order ID] AS [Order ID],
       MAX([Superstore APAC].[Customer Name]) AS [TEMP(attr:Customer
Name:nk) (2542222306) (0)],
       MIN([Superstore APAC].[Customer Name]) AS [TEMP(attr:Customer
Name:nk) (3251312272) (0)],
       SUM([Superstore APAC].[Profit]) AS [sum:Profit:ok],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
GROUP BY [Superstore APAC].[Order ID]
```

If we know there aren't multiple customers for each order ID, we can change the aggregation to MIN() and make the query simpler:

```
SELECT [Superstore APAC].[Order ID] AS [Order ID],
       MIN([Superstore APAC].[Customer Name]) AS [min:Customer Name:nk],
       SUM([Superstore APAC].[Profit]) AS [sum:Profit:ok],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
GROUP BY [Superstore APAC].[Order ID]
```

The following table shows the different in performance for the two options (results are seconds for the query to complete, sourced from the log file):

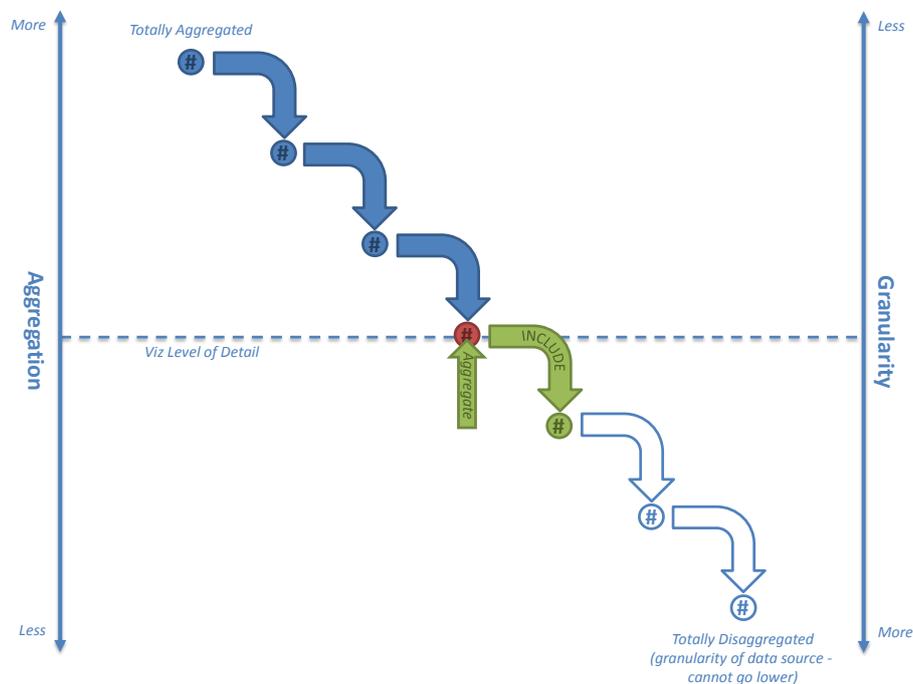
Test #	ATTR	MIN
1	2.824	1.857
2	2.62	2.09
3	2.737	2.878
4	2.804	1.977
5	2.994	1.882
Average	2.7958	2.1368
% improvement		24%

You can read more information on ATTR() in the following blog post from InterWorks:

<http://bit.ly/1YEuZhX>

Level of detail calculations

Level of detail (LOD) expressions allow you to define the level of detail at which a calculation should be performed, independently of the viz LOD.



LOD expressions can (in some cases) allow you to replace calculations you may have authored in more cumbersome ways in earlier versions of Tableau:

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

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

```

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

```

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

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

<http://tabsoft.co/1MphVai>

You can also read the “Top 15 LOD Expressions” blog post from Bethany Lyons that provides a examples for a number of common problem types:

<http://tabsoft.co/1Mpi7Gq>

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

<http://bit.ly/1MpKfV5>

Table calculations

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

If table calculation performance is a problem (possibly because the result set being returned to Tableau is very large) consider pushing some aspects of the calculation back to the data source layer. One way to do this is via a level of detail expression. Another would be 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.

In some use cases, we might know that the value of the aggregate factor is not changing over the partition. When this happens, use MIN() or MAX() as the aggregation function instead of AVG() or ATTR() as they are faster to evaluate. And once you have picked one – stick to it to it to improve your chances of a cache hit!

External calculations

Tableau has mechanisms where complex logic can be passed to an external engine for calculation. These include making calls to R (via a Rserve connection) or Python (via a Dato server connection).

Within Tableau, these operations are similar to table calculations so they are calculated across partitions. This means they may be called multiple times for a single viz which can have performance overhead. They are also not optimised or coalesced in any way so consider whether you can obtain multiple return values in a single function call (e.g. in a concatenated string) instead of making multiple function calls. Finally, the time taken to transfer data to/from the external computation engine can also be an overhead.

Analytics

The analytics pane provides quick access to a number of advanced analyses such as:

- Totals
- Reference lines
- Trend lines
- Forecasts
- Clustering (new in Tableau 10)

These analytics do not generally require additional queries to run – they perform more like table calculations over the data in the query result cache. Like table calculations, if you have a very large set of data in the query results then the computation process can take some time, but in general these are not going to be a big contributor to overall poor workbook performance.

Another contributor to performance for totals and reference lines is whether the measure aggregations are additive or non-additive. For additive aggregations (e.g. SUM, MIN, MAX) then calculating totals or reference lines can be done locally in Tableau. For non-additive aggregations (e.g. COUNTD, TOTAL, MEDIAN, PERCENTILE) we need to go back to the data source to calculate the total/reference line values.

Calculations vs. native features

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

- Grouping dimension members together – consider using [groups](#) or [sets](#);
- Grouping measure values together into ranged bands – consider using [bins](#);
- Truncating dates to a coarser granularity e.g. month or week – consider using [custom date fields](#);
- Creating a set of values that is the concatenation of two different dimensions – consider using [combined fields](#);
- Displaying dates with a specific format, or converting numeric values to KPI indicators – consider using built-in formatting features;
- Changing the displayed values for dimension members – consider using [aliases](#).

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

Impact of data types

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

- Integers and Booleans are much faster than strings and dates

String and date 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 row-level 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 filter card.

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. 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
Year
Quarter
Month
Week
Day
```

Then use it in a calculation like this:

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

A better way would be to use the built-in DATEPART() function and create the calculation as follows:

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

In older versions of Tableau, the latter approach was much faster. However there is no longer a performance difference between these two solutions as we collapse the conditional logic of the CASE statement based on the parameter value and just pass the appropriate DATEPART() to the data source.

Don't forget that you also need to think about the future maintainability of the solution, in which case the latter calculation would probably be a better option.

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. The function DateParse() makes it easy to do these conversions. You can simply pass in a formatting string:

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

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

- non-legacy Excel and text file connections
- 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 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#)))
```

An even better way would be to use the MAKEDATE() function if it is supported on your data source:

```
MAKEDATE(2004, 4, 15)
```

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

Logic statements

Test for the most frequent outcome first

When Tableau evaluates a logical test, it stops working through the possible outcomes once it finds a match. This means you should test for the most likely outcome first so that for the majority of the cases being evaluated, they stop after the first test.

Consider the following example. This logic:

```
IF <unlikely_test>
  THEN <unlikely_outcome>
  ELSEIF <likely_test>
    THEN <likely_outcome>
  END
```

Will be slower to evaluate than this:

```
IF <likely_test>
  THEN <likely_outcome>
  ELSEIF <unlikely_test>
    THEN <unlikely_outcome>
  END
```

ELSEIF > ELSE IF

When working with complex logic statements remember that ELSEIF > ELSE IF. This is because a nested IF computes a nested CASE statement in the resulting query rather than being computed as part of the first. So this calculated field:

```
IF [Region] = "East" AND [Segment] = "Consumer"
  THEN "East-Consumer"
  ELSE IF [Region] = "East" and [Segment] <> "Consumer"
    THEN "East-All Others"
  END
END
```

Produces the following SQL code with two nested **CASE** statements and 4 conditional tests:

```
(CASE
  WHEN (([Global Superstore].[Region] = 'East')
  AND ([Global Superstore].[Segment] = 'Consumer'))
    THEN 'East-Consumer'
  ELSE (CASE
    WHEN (([Global Superstore].[Region] = 'East')
    AND ([Global Superstore].[Segment] <> 'Consumer'))
      THEN 'East-All Others'
    ELSE CAST(NULL AS NVARCHAR)
  END)
END)
```

This would run faster as rewritten below with the ELSEIF instead of the nested IF and more efficient use of conditional tests:

```
IF [Region] = "East" AND [Segment] = "Consumer"
```

```

    THEN "East-Consumer"
    ELSEIF [Region] = "East"
        THEN "East-All Others"
    END

```

This results in only a single CASE statement in the SQL code:

```

(CASE
    WHEN (([Global Superstore].[Region] = 'East')
    AND ([Global Superstore].[Segment] = 'Consumer'))
        THEN 'East-Consumer'
    WHEN ([Global Superstore].[Region] = 'East')
        THEN 'East-All Others'
    ELSE CAST(NULL AS NVARCHAR)
    END)

```

However, this is faster still. While it does use a nested IF statement, it is the most efficient use of the conditional tests:

```

IF [Region] = "East" THEN
    IF [Segment] = "Consumer"
        THEN "East-Consumer"
        ELSE "East-All Others"
    END
END

```

The resulting SQL code:

```

(CASE
    WHEN ([Global Superstore].[Region] = 'East')
    THEN (CASE
        WHEN ([Global Superstore].[Segment] = 'Consumer')
            THEN 'East-Consumer'
            ELSE 'East-All Others'
        END)
    ELSE CAST(NULL AS NVARCHAR)
    END)

```

Avoid redundant logic checks

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

```

Generates the following SQL:

```

(CASE
    WHEN ([Global Superstore].[Sales] < 10)
        THEN 'Bad'
    WHEN (([Global Superstore].[Sales] >= 10)
    AND ([Global Superstore].[Sales] < 30))
        THEN 'OK'
    WHEN ([Global Superstore].[Sales] >= 30)
        THEN 'Great'
    ELSE CAST(NULL AS NVARCHAR)
    END)

```

This would be more efficiently written as:

```

IF [Sales] < 10 THEN "Bad"

```

```
ELSEIF[Sales] >= 30 THEN "Great"  
ELSE "OK"  
END
```

Producing the following SQL:

```
(CASE  
  WHEN ([Global Superstore].[Sales] < 10)  
    THEN 'Bad'  
  WHEN ([Global Superstore].[Sales] >= 30)  
    THEN 'Great'  
  ELSE 'OK'  
END)
```

Many little things

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

- Dealing with date logic can be complicated. Rather than writing elaborate tests using multiple date functions – e.g. MONTH() and YEAR() – consider using some of the other built-in functions such as DATETRUNC(), DATEADD() and DATEDIFF(). These can significantly reduce the complexity of the query generated to the data source.
- Distinct counting values is one of the slowest aggregation types in almost all data sources. Use the COUNTD aggregation sparingly.
- Using parameters with a wide scope of impact (e.g. in a custom SQL statement) can affect cache performance.
- Filtering on complex calculations can potentially cause indexes to be missed in the underlying data source.
- Use NOW() only if you need the time stamp level of detail. Use TODAY() for date level calculations.
- Remember that all basic calculations are passed through to the underlying data source – even literal calculations like label strings. If you need to create labels (e.g. for columns headers) and your data source is very large, create a simple text/Excel file data source with just one record to hold these so they don't add overhead on the big data source. This is especially important if your data source uses stored procedures – see [this section](#) for more details.

Is it my queries?

One of the very powerful features of Tableau is that it can use both in-memory data (i.e. extracted data connections) as well as in-situ data (i.e. live data connections). Live connections are a very powerful feature of Tableau as they allow us to take advantage of the processing power already in place in the data source. However, because we are now dependent on the source data platform for performance it becomes critical that we generate our queries to the data source as efficiently and optimally as possible.

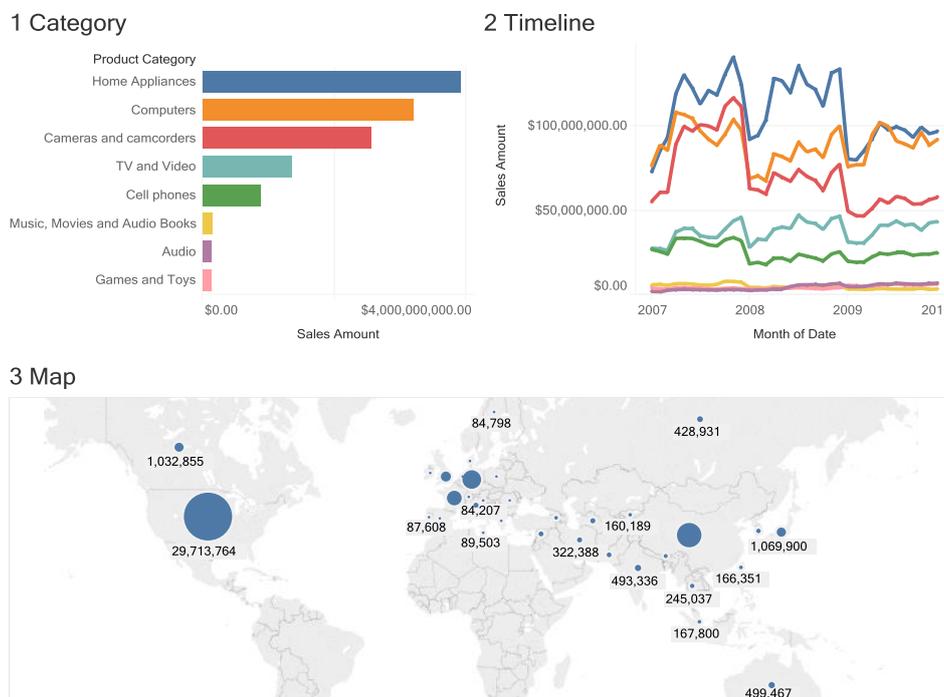
As we have already discussed, the more complex the question, the more data you display, the more elements you include on the dashboard – these all mean more work for the data source. To make our workbooks as efficient as possible, we want to minimise the number of queries, ensure the queries are evaluated as efficiently as possible, minimise the amount of data returned by the queries, and reuse data as much as possible between requests.

Automatic optimisations

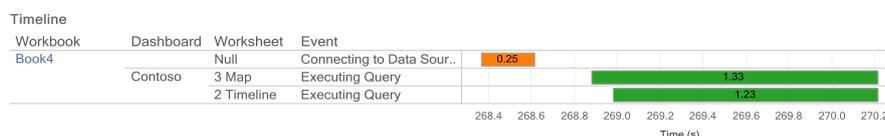
While there are many things we can deliberately do to improve the performance of our queries, there are many optimisations that Tableau will perform automatically to ensure your workbooks execute efficiently. Most of these are things you can't directly control and in most cases shouldn't need to think about, but if you understand them well you can take advantage of them to improve performance.

Parallel execution – running multiple queries at once

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



Opening this workbook in Tableau shows the following queries:



As you can see, run serially the queries would take 2.66 secs. But by running them in parallel we only take the length of the longest query (1.33 secs).

The level of query parallelism varies between source systems as some platforms handle simultaneous queries better than others. The default is to run a maximum of 16 parallel queries for all data sources apart from text, Excel and statistics files (limit is 1 query at a time). Other data sources might have limits set below the default – see the following link for details:

<http://tabsoft.co/1TjZvxo>

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

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

These settings are managed by an xml file named connection-configs.xml which you create and save in the app folder on Windows (C:\Program Files\Tableau\Tableau 10.0) and Mac (right click the App, click Show Package Contents, and place the file here) for Tableau Desktop or in the config directory in the vizqlserver folder (for example:

C:\ProgramData\Tableau\TableauServer\data\tabsvc\config\vizqlserver) for Tableau Server. You must copy this configuration file to all the vizqlserver configuration directories in all worker machines.

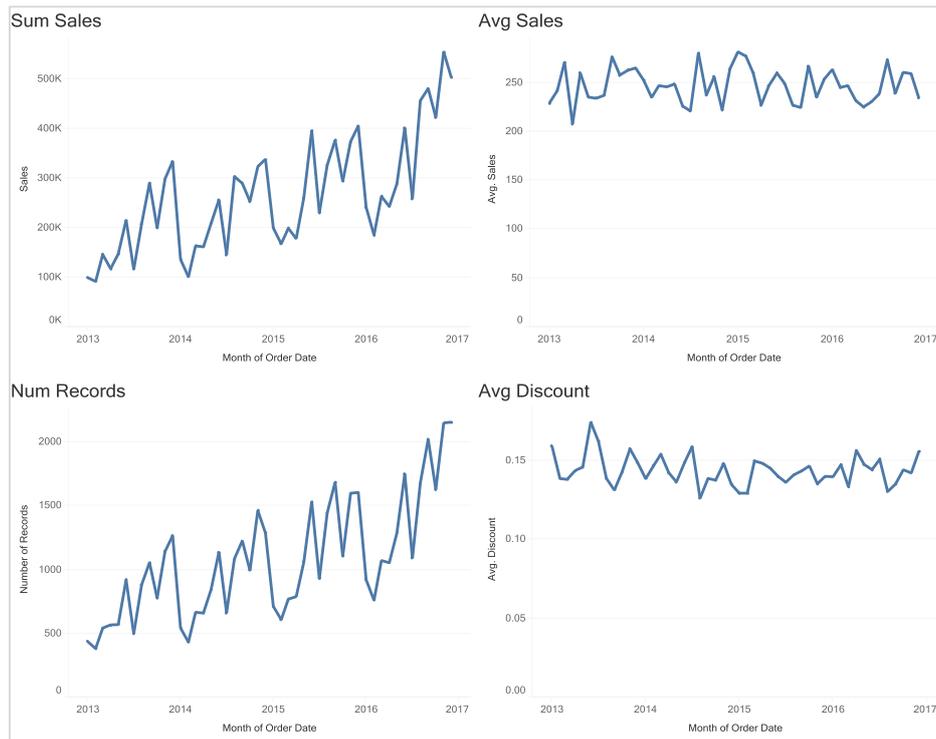
You can read more about configuring parallel queries, including the syntax of the connection-configs.xml file, here:

<http://tabsoft.co/1HA1ztD>

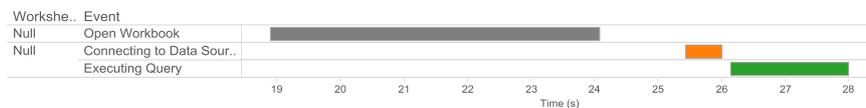
Query elimination – running less queries

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

The query optimiser will also look for queries that are at the same level of detail (i.e. they are specified by the same set of dimensions) and will collapse them into a single query that returns all requested measures. Consider the following dashboard:



As you can see, this dashboard contains four sheets – each showing a different measure over time. They all have the same level of detail as they are showing the data using a continuous month. Instead of running four separate requests of the database, Tableau combines these together into a single query:



The query is as follows:

```
SELECT AVG(cast([Global Superstore].[Discount] as float)) AS
[avg:Discount:ok],
AVG(cast([Global Superstore].[Sales] as float)) AS [avg:Sales:ok],
SUM(CAST(1 as BIGINT)) AS [sum:Number of Records:ok],
SUM([Global Superstore].[Sales]) AS [sum:Sales:ok],
DATEADD(month, DATEDIFF(month, CAST('0001-01-01 00:00:00' AS datetime2),
[Global Superstore].[Order Date]), CAST('0001-01-01 00:00:00' AS datetime2))
AS [tmn:Order Date:ok]
FROM [dbo].[Global Superstore] [Global Superstore]
GROUP BY DATEADD(month, DATEDIFF(month, CAST('0001-01-01 00:00:00' AS
datetime2), [Global Superstore].[Order Date]), CAST('0001-01-01 00:00:00' AS
datetime2))
```

As you can see, this optimisation (called “query fusion”) can improve the overall performance significantly. Where possible, consider setting the same level of detail for multiple sheets on a dashboard.

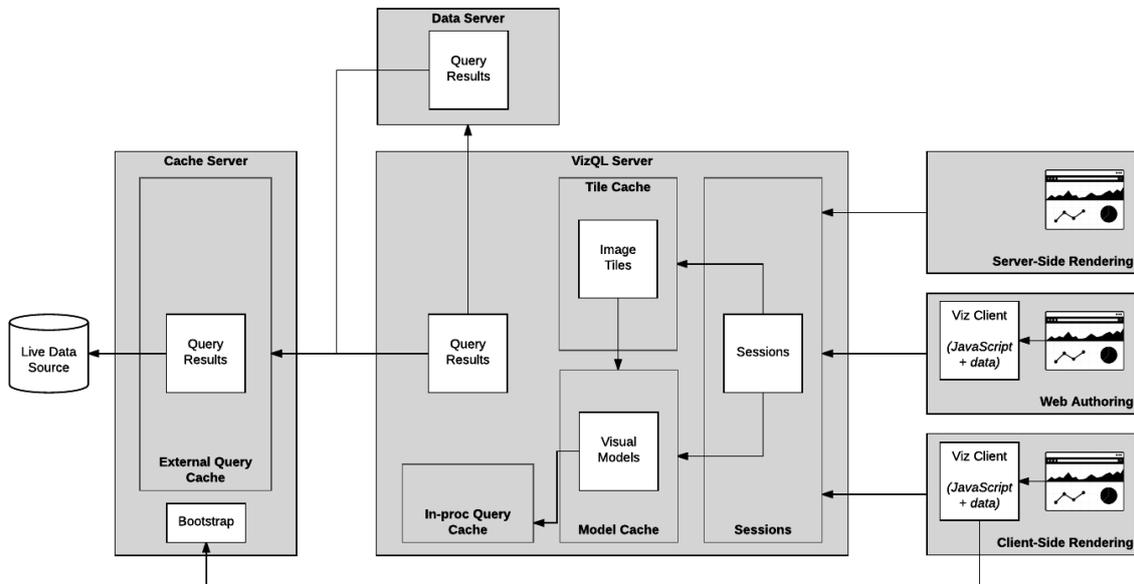
Note that query fusion does not happen for queries against TDE data sources.

Caching – not running queries at all

What is better than running fewer queries? Running no queries at all! Tableau has extensive caching in both Tableau Desktop and Tableau Server which can significantly reduce the number of queries that need to be run on the underlying data source.

Tableau Server

Caching in Tableau Server has multiple layers:



The first layer of caching depends on whether the session is using client-side or server-side rendering. See this [earlier section](#) for more information on the two rendering models and how Tableau determines which it should use.

If the session is using client-side rendering, the first thing the browser needs to do is load the viewer client. This involves loading a set of JavaScript libraries and the data so it can render the initial view – a process called “bootstrapping” which can take several seconds. Knowing that dashboards are likely to be viewed by multiple users and in order to reduce the wait time for subsequent view requests, the bootstrap responses are cached. Each session first checks the bootstrap cache to see if there is a response already created that it can use. If it finds one, the browser simply loads the response from the cache, making it very quick to render the initial view. Once the viewer client is loaded in the client browser, some interactions can be fulfilled entirely with the local data (e.g. highlighting, tooltips) resulting in a fast, smooth user experience.

If the dashboard is being displayed via server-side rendering, the server renders the dashboard elements as a series of image files (called “tiles”). These tiles are passed to the browser where they are assembled to display the viz. As above, we expect that dashboards will be viewed by multiple users, so the server caches these images on disk. Each request checks the tile cache to see if the image has already been rendered, and if it finds one it simply loads the file from cache. A hit on the tile cache makes the response time faster and reduces the workload on the server. One simple way to increase the effectiveness of the tile cache is to set your dashboards to have a [fixed size](#).

Overall, client-side rendering produces smoother, more responsive user interactions and places less workload on the Tableau Server. Where possible, you should design your workbooks so they can be rendered client-side.

The next layer of caching is called the visual model. A visual model describes how to render an individual worksheet so viewing a dashboard can reference multiple visual models – one for each worksheet used. It includes the results from local calculations (e.g. table calcs, reference lines, forecasts, clustering, etc.) and the visual layout (how many rows/columns to display for small multiples and crosstabs; the number and interval of axis ticks/grid lines to draw; the number and location of mark labels to be shown; etc.).

Visual models are created and cached in-memory by the VizQL Server and it does its best to share results across user sessions where possible. The key factors in whether a visual model can be shared are:

- The size of the viz display area – models can only be shared across sessions where the size of the viz is the same. Setting your dashboards to have a fixed size will benefit the model cache in much the same way as it benefits the tile cache – allowing greater reuse and lowering the workload on the server.
- Whether the selections/filters match – if the user is changing filters, parameters, doing drill-down/up, etc. the model is only shared with sessions that have a matching view state. Try not to publish workbooks with “show selections” checked as this can reduce the likelihood that different sessions will match.
- The credentials used to connect to the data source – if users are prompted for credentials to connect to the data source then the model can only be shared across user sessions with the same credentials. Use this capability with caution as it can reduce the effectiveness of the model cache.
- Whether user filtering is used – if the workbook contains user filters or has calculations containing functions such as USERNAME() or ISMEMBEROF(), then the model is not shared with any other user sessions. Use these functions with caution as they can significantly reduce the effectiveness of the model cache.

The final layer of caching is the query cache. It stores results from queries that have been run in anticipation that we can use them to service future queries. Hitting this cache is very effective as it allows us to avoid repeatedly running queries in the data source – we just load the data from the cache. It also allows us, in some circumstances, to service queries using the result of another query. We discussed the benefits of this [earlier](#) when talking about query elimination and query fusion.

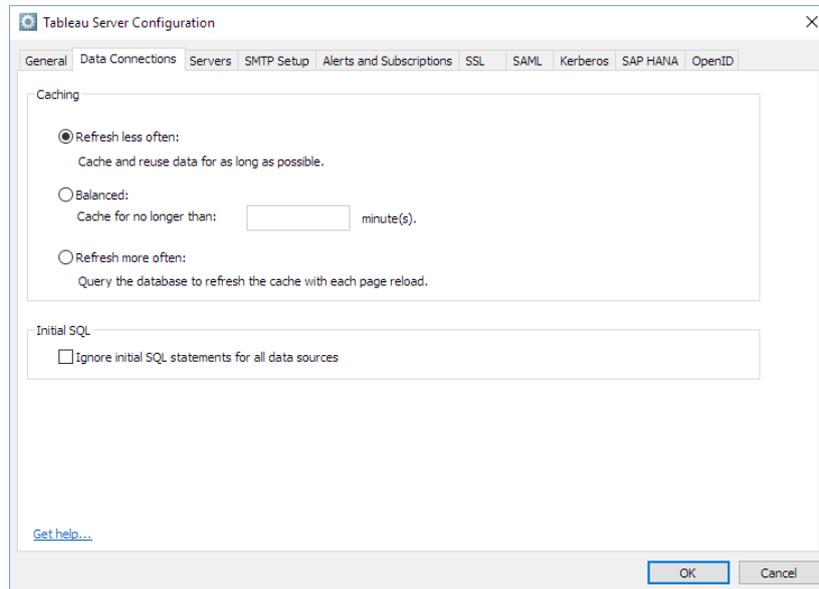
There are two parts to this cache – one is held within the VizQL process (called the “in-proc cache”) and the other is shared across multiple processes via the Cache Server (called the “external cache”). If a query is sent to a VizQL instance that has run the same query previously, the request is serviced from the in-proc cache. Note that this cache is local to each VizQL process and is maintained in-memory.

In addition to these in-proc caches there is an external cache that is shared – not just across VizQL instances but across ALL processes that access the underlying data source (e.g. backgrounders, data servers, etc.). A service called the Cache Server manages the external cache across the whole cluster. Note that not all queries are written to the external cache. If a query runs very quickly, it can be faster to run it again rather than check the cache, so there is a minimum query time threshold. Also, if the result of a query is very large, it might not be efficient to write it to the Cache Server, so there is a maximum size threshold as well.

The external cache significantly increases the effectiveness of caching in deployments where there are multiple VizQL instances. In contrast to the in-proc caches which are volatile, the external cache

is persistent (the Cache Server writes the data to disk) and remains between instantiations of the services.

To maximise the effectiveness of the caches, you can adjust the settings of the Tableau Server installation to retain them for as long as possible:



If you have the server capacity available, you can increase the size of the caches. There is no penalty to increasing these settings, so if you have sufficient RAM you can increase the numbers significantly to avoid content being pushed from the cache. You can monitor the effectiveness of the caches via the [log files](#), or via [TabMon](#).

- **Model cache** – the default setting is to cache 60 models per VizQL instance. You can adjust this setting through the *tabadmin* command – if you have the RAM available you could increase this to match the number of views published on your server:
 - `tabadmin set vizqlserver.modelcachesize 200`
- **In-proc cache** – the default setting is to cache 512MB of query results per VizQL instance. This might not sound like much, but remember we are caching the query results which are from aggregate queries. You can adjust this setting through the *tabadmin* command:
 - `tabadmin set vizqlserver.querycachesize 1024`

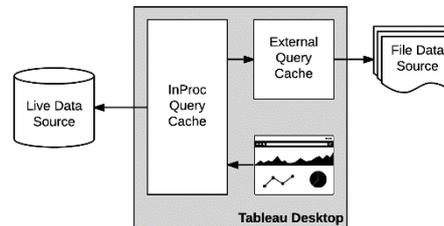
You can also increase the capacity and throughput of the external query cache by adding more instances of the Cache Server. Best practice is to run one instance of the Cache Server for every VizQL Server instance.

- **External cache** – the default setting is to cache 512MB of query results for each instance of the Cache Server.

Finally, a useful side-effect of the cache being shared is we can warm the cache for workbooks that we know are slow on first view by running a subscription for the workbook. Doing this early in the morning before your interactive users arrive at work will ensure that the workbook is run and the query results loaded into the cache. Assuming the results are not pushed from cache or expired in the intervening time, the cache will then be hit when the users view the workbook, making the initial view time fast.

Tableau Desktop

Caching in Tableau Desktop is much simpler than Tableau Server because it is a single user application and we don't need to manage multiple sessions. Only the query cache is present in Tableau Desktop:



Like Tableau Server, there is an in-proc and an external cache. The in-proc, volatile, memory-based cache is used for connections to all data sources, however the external cache is only used for file-based data source (e.g. data extracts, Excel, Access, text files, statistical files, etc.). Like in Tableau Server, the external cache is persistent. That means the cache results are retained between sessions in Tableau Desktop so if you use a file data source, restart Tableau Desktop and use it again you will still benefit from the cache.

The external cache is stored in `%LOCALAPPDATA%\Tableau\Caching` on Windows and `~/Library/Caches/com.tableau/` on Mac. By default, it is limited to ~750MB in total, and it will be invalidated if the user forces a refresh of the data source (e.g. pressing F5, ⌘R).

Finally, in Tableau Desktop the external query cache data is included when the file is saved as a packaged workbook. This allows Tableau Desktop and Tableau Reader to rapidly render the initial view of the workbook while it continues to unpack the larger data source file in the background. For an end user, this can dramatically improve the responsiveness of opening the workbook. Note that the cache data is only included if the data is $\leq 10\%$ the size of the source data file, and queries that use relative date filters are not included.

Lazy connections

Prior to Tableau 10, when a user opened a workbook with multiple data sources we would initially connect to all non-credentialed data sources (i.e. data sources that don't require the user to enter a username/password). This meant we could potentially be spending time connecting to data sources that weren't being used on the sheet/dashboard/story the user was viewing.

In Tableau 10 we now only connect to a data source when a user needs it to view a selected sheet/dashboard/story. This change will reduce the time to load a workbook with a tabbed view (on Tableau Desktop) so users can start exploring their data sooner.

Joins

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

As a general performance rule-of-thumb, the number of joined tables should be minimized to only include the tables needed for a specific worksheet/visualization. Depending on the complexity of the

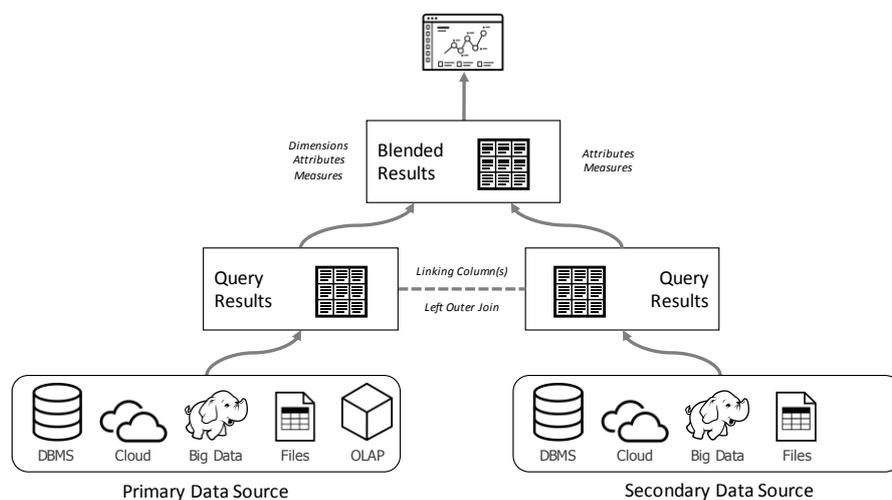
data connection and the worksheets, it may also be worth separating out data connections for each worksheet and creating specific join patterns for those sheets.

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

Blending

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

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



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

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

Primary groups and aliases

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

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

For the following examples, consider the following three tables:

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

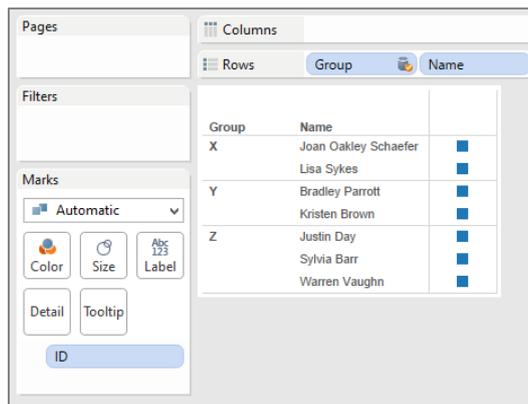
ID	Name
A	Lisa Sykes
B	Joan Oakley Schaefer
C	Bradley Parrott
D	Kristen Brown
E	Sylvia Barr
F	Warren Vaughn
G	Justin Day

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

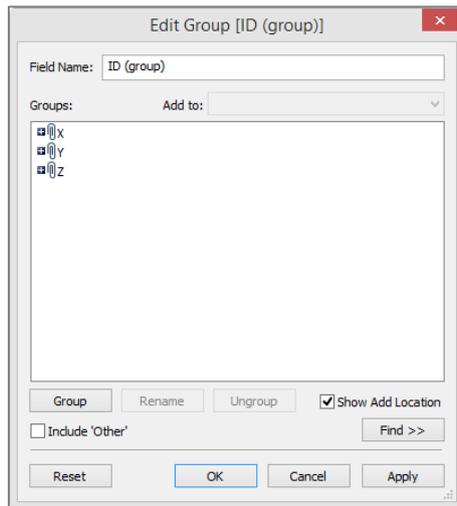
Primary groups are useful when the secondary data source contains an attribute that is mapped 1:many back to dimension members in the primary data source. Assume what we want to produce from the above data is this:

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

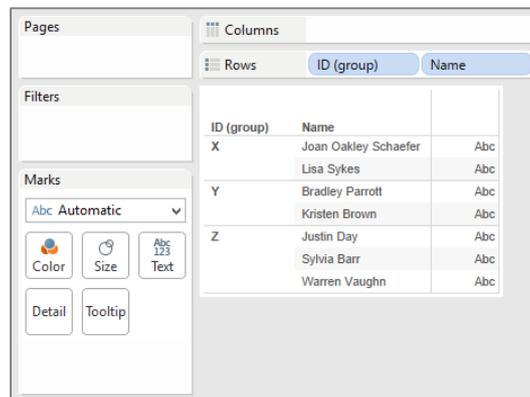
We could create this by blending but as already discussed this would result in very slow performance if there were a large number of IDs:



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).



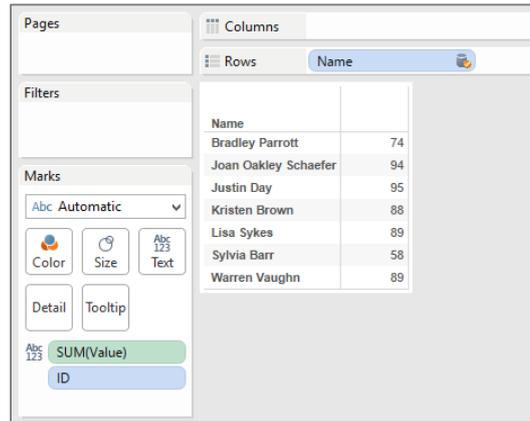
We can now recreate this table without requiring a blend:



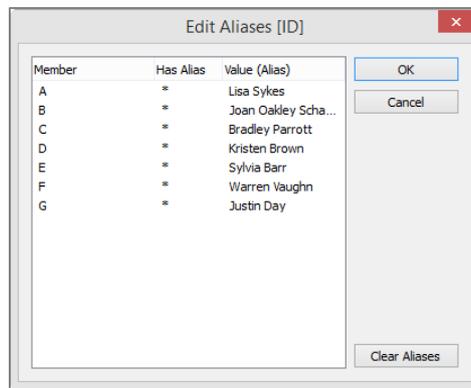
Primary aliases are useful when the secondary data source contains an attribute that is mapped 1:1 back to dimension members in the primary data source. Assume what we want to produce from the above data is this:

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

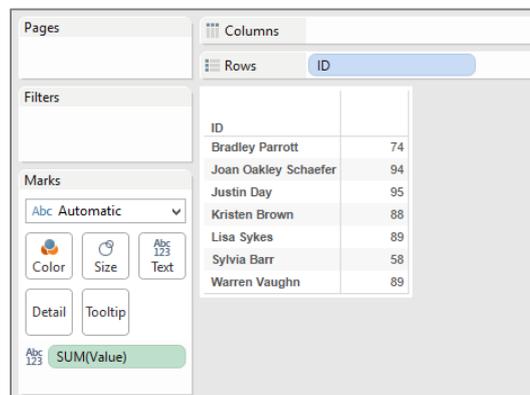
We could create this by blending between the two data sources however as already discussed this would result in very slow performance if there were a large number of IDs:



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:



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



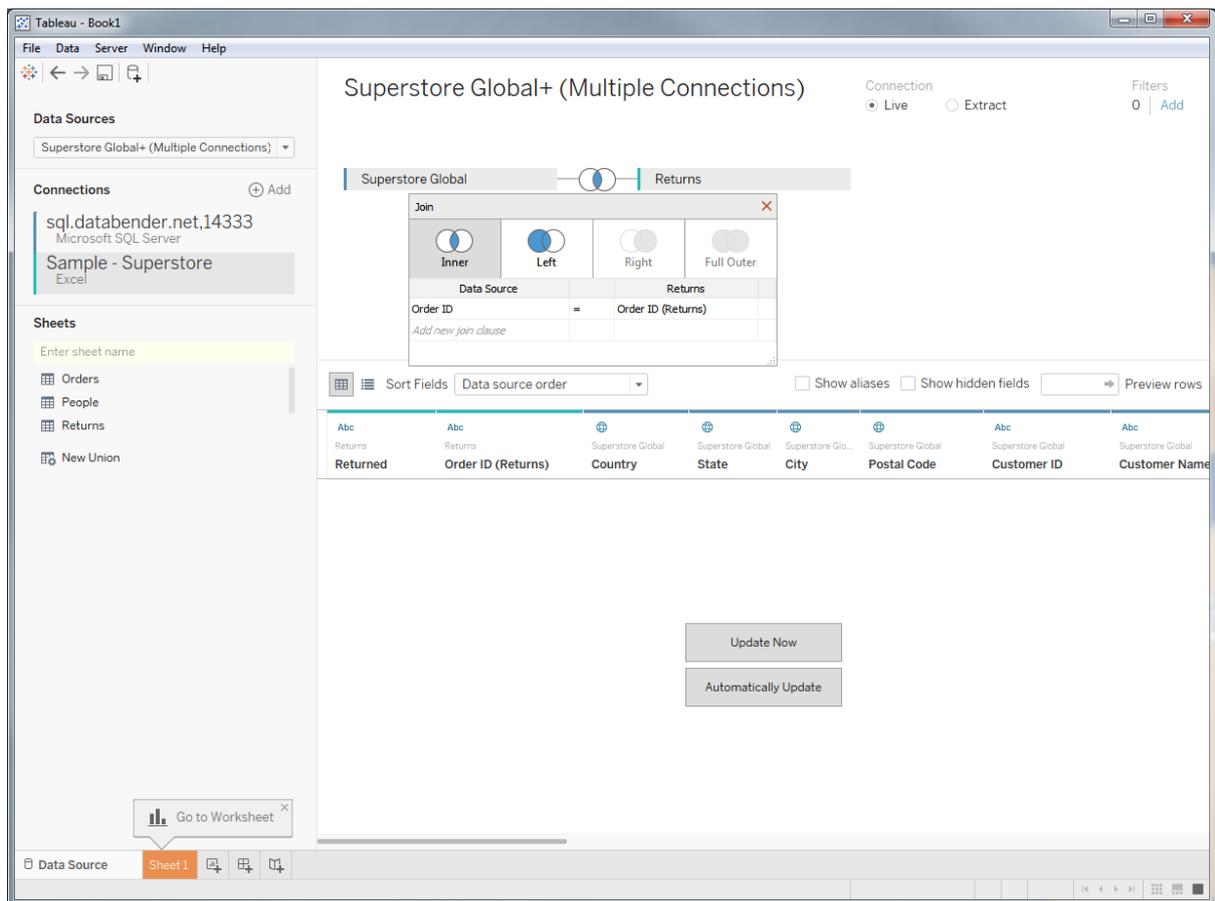
Note that both primary groups and aliases are not dynamic and would need to be updated if the data changed. Consequently they aren’t a great solution for frequently updated data but if you need a quick mapping they can potentially eliminate the need for costly blends.

More information can be found in the following kbase article:

<http://tabsoft.co/1PRhkU9>

Data integration

Data integration is a new feature, introduced in Tableau 10, that allows data sources to combine data from multiple, potentially heterogeneous, data connections.



A key difference between data integration and data blending is that data integration is a row-level join whereas data blending is performed on the aggregated result set from each data source. This means data integration is sensitive to the size of the underlying data sources – there are 4 key considerations:

- The more data we have to move, the longer it takes – data is extracted at the row level from each data connection so the volume of data is a key factor;
- The further data must travel, the longer it takes – performance will be impacted if you are joining to a data source that is on a high-latency connection;
- The slower the rate of data flow, the longer it takes – performance will be impacted if you are joining to a data source that is on a bandwidth-constrained connection;
- The more things need to be matched, the longer it takes – similar to point 1, performance is affected by the number of records we must join together.

In Tableau 10, data integration can only be used with extracted data sources – i.e. the results from each data connection must be extracted into a TDE file. This extracted data source can then be published to Tableau Server for others to use.

In Tableau 10 it is not possible to do data integration on live data sources, nor is it possible to do data integration against published data sources. These are capabilities we are looking to add as future enhancements.

Custom SQL

Sometimes users new to Tableau try to bring old-world techniques to their workbooks such as creating data sources using hand-written SQL statements. In many cases this is counterproductive as

Tableau can generate much more efficient queries when we simply define the join relationships between the tables and let the query engine write SQL specific to the view being created. However, there are times when specifying joins in the data connection window does not offer all the flexibility you need to define the relationships in your data.

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

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

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

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

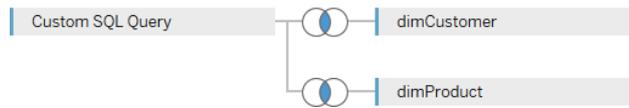
Using parameters in a custom SQL statement could in some cases make live connections more performant as the base query can be more dynamic (e.g. filter clauses that use parameters will be evaluated appropriately). It could also be used to pass in values for performance limiters such as TOP or SAMPLE to constrain the amount of data returned by the database. However, if you are using a data extracts it will be purged and regenerated every time you change the parameter and this can be slow. Also be aware that parameters can only be used to pass in literal values so they cannot be used to dynamically change the SELECT or FROM clauses.

Finally, it is also possible to join tables to custom SQL:

Custom SQL Query+ (Superstore)

Connection
 Live Extract

Filters
0 | Add



This allows you to write a more specific custom SQL that references a subset of the total schema. The joins from this to the other tables can then (potentially) be culled like normal table joins, creating more efficient queries.

Alternatives to custom SQL

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

There are several ways to do this:

Views

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

<http://tabsoft.co/1su5YMe>

Defining custom query logic into a view instead of in the Tableau workbook also allows it to be reused across multiple workbooks and data sources. Additionally, many DBMS's support the concept of materialised views or snapshots where the results of the view query are pre-computed and cached, allowing for much faster responses at query time. They can be similar to summary tables (see below) but are maintained by the DBMS automatically.

Stored procedures

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

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

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

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

<http://tabsoft.co/1HAD93e>

Summary tables

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

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

Initial SQL

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

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

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

<http://tabsoft.co/1AsJkXg>

Is it my data?

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

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

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

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

General advice

Use native drivers

Tableau 10 supports native connectivity to over 40 different data sources. This means Tableau has implemented techniques, capabilities and optimizations specific to these data sources. Engineering and testing activities for these connections ensure they are the most robust Tableau has to offer.

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

If there is a native driver for the data source you are querying, you should use this over the ODBC connections as it will generally provide better performance. Also note that ODBC connections are only available on Windows.

Test as close to the data as possible

As stated earlier, a general principal is that if a data source performs queries slowly then the experience in Tableau will be slow. A good way to test the raw performance of the data source is to (if possible) install Tableau Desktop on the machine where the data source resides and to run some queries. This will eliminate factors such as network bandwidth and latency from the performance and allow you to better understand the raw performance of the query in the data source.

Additionally, using the *localhost* name for the data source instead of the DNS name can help determine if environmental factors such as slow name resolution or proxy servers are adding to the poor performance.

Data sources

Files

This category covers all file-based data formats – text files such as CSV, Excel spreadsheets and MS Access being the most common, however this also includes data files from statistical platforms SPSS,

SAS and R. Business users often use data in this format because it is a common way to get data out of “governed” data sets – either by running reports or performing a query extract.

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

Shadow extracts

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

You may notice that the first time you use a large file it can take several seconds to load the data preview pane. This is because Tableau is extracting the data from the file and writing it to a shadow extract file. By default, these files are created in

C:\Users\

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

Legacy connectors for Excel and text files

In earlier versions of Tableau, connections to Excel and text files made use of Microsoft’s JET data engine driver. Since Tableau 8.2 though, we default to a native driver that provides better performance and works with larger, more complex files. However, there are some situations in which you might prefer to use the legacy drivers – e.g. if you want to use custom SQL. In these scenarios users have the option to revert back to the legacy JET driver. Note that reading MS Access files still uses the JET driver.

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

<http://tabsoft.co/1HACvmj>

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

Relational Databases

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

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

Row-based vs. column-based

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

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

SQL as an interface

There are many systems that are not based on traditional RDBMS technologies but still appear as a relational source as they provide a SQL-based interface. For Tableau this includes several NoSQL platforms (e.g. MarkLogic, DataStax, MongoDB, etc.) as well as query acceleration platforms such as Kognitio, AtScale and JethroData.

Indexes

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

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

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

NULLS

Having NULL values in dimension columns can reduce the effectiveness of queries. Consider a viz where we want to show the total sales by country for the top 10 products. Tableau will first generate a subquery to find the top 10 products and then join this back to the query by country to produce the final result.

If the product column is set to ALLOW NULL, we need to run a query to check if NULL is one of the top 10 products returned by the subquery. If it is, we need to perform a NULL-preserving join which is much more computationally expensive than a regular join. If the product column is set to NOT NULL, we know that the result of the subquery cannot contain NULL so we can perform a normal join, skipping the “check for NULL” query.

This is why, where possible, you should define your dimension columns as NOT NULL.

Referential Integrity

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

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

#	CustomerDim Customer ID (Cust...)	CustomerDim Name	#	DeliveryDim Delivery ID (Delive...)	DeliveryDim Shipping Mode	#	LocationDim Place ID (Location...)	LocationDim Region
	75.00	Altd		8900.00	First Class		1051.00	Europe
	1195.00	Systed Systems, Inc		8300.00	First Class		661.00	Asia
	1004.00	Seas		2400.00	Standard Class		885.00	Asia
	1439.00	Vellutems Corporation		0.00	Standard Class		595.00	Asia
	883.00	Mics Consulting		0.00	Standard Class		786.00	Asia
	984.00	Quan		2400.00	Standard Class		173.00	Africa
	1029.00	Sems Group		4200.00	Standard Class		31.00	Africa
	953.00	Nettpaq USA		2400.00	Standard Class		130.00	Africa
	683.00	Inetems Research		6000.00	Second Class		282.00	America
	1302.00	TER		5400.00	Standard Class		541.00	Asia
	1054.00	Silian Corporation		0.00	Standard Class		151.00	Africa

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

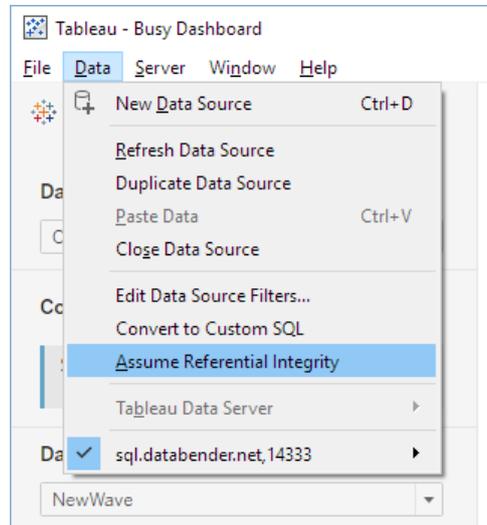
```
SELECT SUM([OrdersFact].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[OrdersFact] [OrdersFact]
GROUP BY ()
```

Without it, a far less efficient query is generated:

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

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

In the past, “hard” referential integrity was required where the rule was enforced by the DBMS. 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. Users can tell Tableau that soft referential integrity is in place and that join culling can be safely used by enabling “assume referential integrity”:



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

<http://bit.ly/1HACmPV>

<http://bit.ly/1HACqPn>

Partitioning

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

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

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

Temp Tables

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

OLAP

Tableau supports several OLAP data sources:

- Microsoft Analysis Services

- Microsoft PowerPivot (both PowerPivot for Excel and PowerPivot for SharePoint)
- Oracle Essbase
- SAP Business Warehouse
- Teradata OLAP

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

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

<http://tabsoft.co/1HAF47P>

SAP BW data extracts

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

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

<http://tabsoft.co/1SuYf9d>

Big Data

Big data is a heavily overloaded term in the world of data analysis, however in this document we are particularly using it to refer to platforms that are based on Hadoop. In Tableau 10 there are four supported Hadoop distributions that support Hive and/or Impala connections:

- Amazon EMR
 - HiveServer
 - HiveServer2
 - Impala
- Cloudera Hadoop
 - HiveServer
 - HiveServer2
 - Impala
- Hortonworks Hadoop Hive
 - HiveServer
 - HiveServer2
 - Hortonworks Hadoop Hive
- MapR Hadoop Hive
 - HiveServer
 - HiveServer2

- Spark SQL
 - SharkServer *
 - SharkServer2 *
 - SparkThriftServer

** Note that SharkServer and SharkServer2 connections are provided for your use, but are not supported by Tableau.*

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

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

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

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

<http://tabsoft.co/1BkFo62>

Cloud

Tableau currently supports the following cloud data sources:

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

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

Tableau also supports the following cloud-based data platforms:

- Amazon Aurora
- Amazon Redshift
- Amazon RDS
- Google BigQuery
- Google Cloud SQL
- Google Sheets
- Microsoft Azure SQL Data Warehouse
- Snowflake

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

Finally, Tableau also provides a generic data connector for importing data from web-based services that publish data in JSON, XML, or HTML formats – the Web Data Connector.

Salesforce

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

There is no “connect live” option

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

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

The initial extract can be very slow

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

Join options are limited

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

You cannot pre-filter data

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

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

Formula columns cannot be migrated over

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

There is a 10k character limit on queries

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

Google Analytics

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

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

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

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

<http://bit.ly/1BkFoTG>

To avoid sampling, there are two approaches to take:

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

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

Web data connector

A Tableau web data connector gives you a way to connect to data that doesn't already have a connector. Using a web data connector, you can create and use a connection to almost any data that is accessible over HTTP. This can include internal web services, JSON data, XML data, REST APIs, and many other sources. Because you control how the data is fetched, you can even combine data from multiple sources.

You create a web data connector by writing a web page that contains JavaScript and HTML. After you've written a web data connector, you can share it with other Tableau users by publishing it to Tableau Server.

To help you create web data connectors, we have created a software development kit (SDK) that includes templates, example code, and a simulator that lets you test web data connectors. This documentation also includes a tutorial that walks you through how to create a web data connector from scratch. The web data connector SDK is an open source project. For more information, see the Tableau [webdataconnector](#) page on GitHub.

More information on web data connectors is available here:

<http://tabsoft.co/1NnGsBU>

Data preparation

The data we need to use in our workbooks is sometimes not perfect. It may not be clean; it may not be the right shape; and it might be spread across multiple files or databases. Historically, Tableau

worked best when it was fed with clean, normalised data which meant that you sometimes needed to prepare your data using other tools before you loaded it into Tableau.

This is no longer always the case – Tableau now has multiple features to help load messy data. These features include:

- Pivot
- Union
- Data interpreter
- Merge columns

Performing data preparation on the fly a) helps keep you in the flow of your analysis and b) allows users without access to other tools to continue analysing their data in Tableau. However, these operations are generally computationally expensive (especially over large volumes of data) and they can have performance implications for report users. It is recommended that where possible you use a data extract to materialise the data after using these operations.

You should also consider these capabilities to be complementary to existing ETL processes and consider if pre-processing the data upstream of Tableau is a better and more efficient approach if the data is used across multiple reports or shared with other BI tools.

Data extracts

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

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

An extract is:

- A persistent cache of data that is written to disk and reproducible;
- A columnar data store – a format where the data has been optimised for analytic querying;
- Completely disconnected from the database during querying. In effect, the extract is a replacement for the live data connection;
- Refreshable, either by completely regenerating the extract or by incrementally adding rows of data to an existing extract;
- Architecture-aware – unlike most in-memory technologies it is not constrained by the amount of physical RAM available;
- Portable – extracts are stored as files so can be copied to a local hard drive and used when the user is not connected to the corporate network. They can also be used to embed data into packaged workbooks that are distributed for use with Tableau Reader;
- Often much faster than the underlying live data connection.

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

<http://bit.ly/1F2iDnT>

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

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

When to use extracts? When to use live connections?

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

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

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

- Real-time data - because extracts are a point of time snapshot of data they would not be appropriate if you need real-time data in your analysis. It is possible to automatically refresh extracts using Tableau Server and many customers do this at intra-day frequencies but true real-time data access would require a live connection.

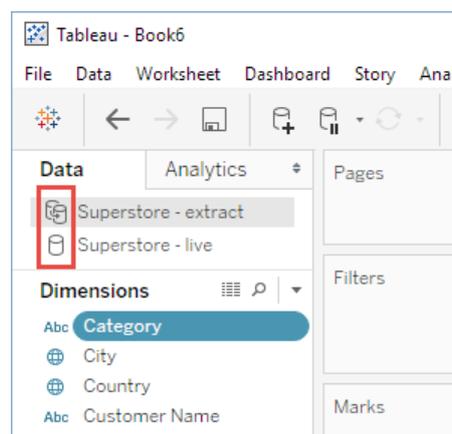
- Massive data - if the volume of data you need to work with is massive (the definition of “massive” will vary from user to user but generally it will be millions to billions of records) then extracting this may not be practical. The resulting extract file may be excessively large or the extract process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

Creating extracts in Tableau Desktop

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

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

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



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

The extract creation process requires temp disk space to write working files – up to 2x the final extract file size. This working space is allocated in the directory specified by the TEMP environment variable (usually C:\WINDOWS\TEMP or C:\Users\USERNAME\AppData\Local\Temp). If this drive has insufficient space, point the environment variable to a larger location.

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

Creating extracts using the data extract API

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

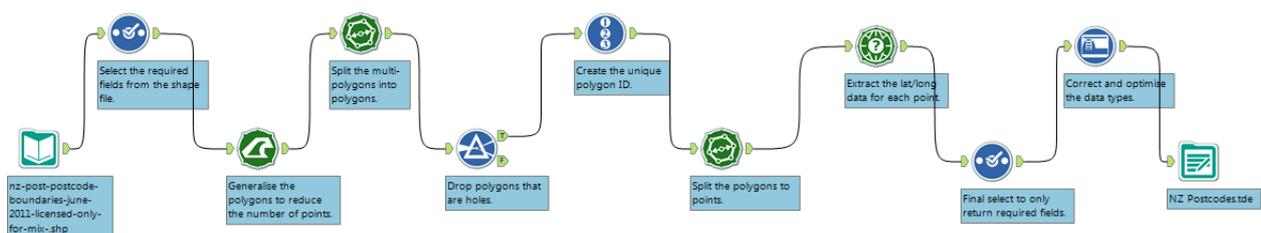
This API is available for developers in Python and C/C++/Java on both Windows and Linux. You can find more information on the API here:

<http://tabsoft.co/1Sv1n55>

Creating extracts using 3rd party tools

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

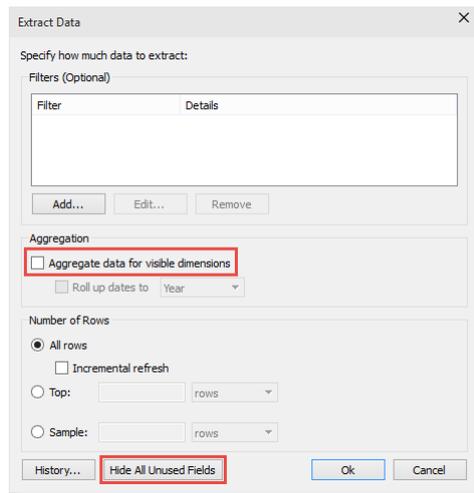
If you have complex data preparation requirements, tools like Alteryx and Informatica can be used to efficiently perform the ETL stages and then directly output the prepared data into a TDE file for use in Tableau Desktop.



Aggregate extracts

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

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



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

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

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

Optimising extracts

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

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

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

Caveat: a trusted resource in our development team passed on the comment that although it is possible to create multiple connections to a single TDE, it was never really developed to support this. It can cause lots of problems if the connections get out of sync with the TDE structure. His advice – just don't do it.

Refreshing extracts

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

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

You can choose two refresh schedules for a single extract:

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

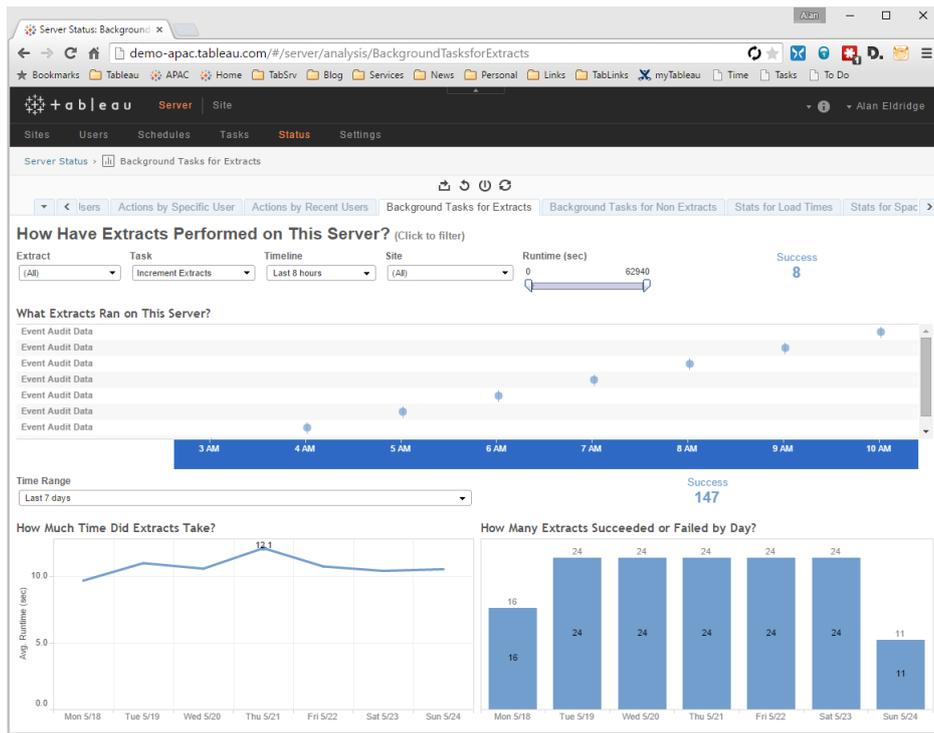
What happens if the refresh takes longer than the increment?

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

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

Extract maintenance

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



You also can refresh a workbook incrementally or fully via the `tabcmd` command line tool if you are using Tableau Server or the `Tableau.exe` command line if you are using Tableau Desktop. If you have complex scheduling requirements you can invoke this from an external scheduling tool such as the Windows Task Scheduler. This approach is required if you want a refresh cycle that is shorter than the 15 minute minimum allowed through the Tableau Server interface.

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

<http://tabsoft.co/1fD1NXP>

Data governance

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.

Is it the environment?

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

Upgrade

The development team at Tableau are constantly working to improve the performance and usability of our software. Upgrading to the latest release of Tableau Server can sometimes yield significant improvements in performance and stability without requiring any changes workbook.

Check the Tableau Release Notes page and upgrade to the most recent build you can:

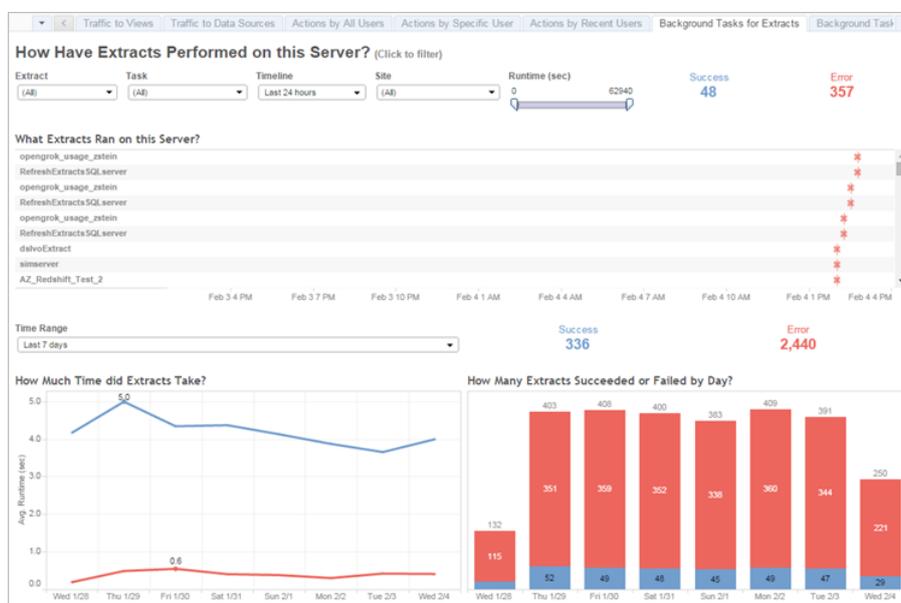
<http://tabsoft.co/1Xwjtkk>

Test Tableau Desktop on the Server

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

Separate refreshes and interactive workloads

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



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

Monitor and tune your Server

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

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

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

<http://tabsoft.co/1RjCCL2>

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

<http://tabsoft.co/1RjCACR>

Check the VizQL session timeout limit

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

<http://tabsoft.co/1f17Bd6>

Assess your process configuration

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

<http://tabsoft.co/1f17tKK>

Infrastructure

64-bit

Versions of Tableau Server prior to Tableau 10 can run on 32-bit Microsoft operating systems, however customers are strongly advised to install their production environments using the 64-bit version. 32-bit installations are only recommended for DEV/TEST installations.

From Tableau 10 and onwards, Tableau Server is only available as a 64-bit application.

Add more CPU/RAM

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

TabMon (mentioned earlier in this document) is an excellent tool for collecting utilisation data that can help your capacity planning process.

Don't disregard IO

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

<http://bit.ly/1f17oa3>

<http://bit.ly/1f17n5O>

Physical vs. virtual

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

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

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

<http://vmw.re/1L4Fyr1>

Browser

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

Conclusion

As someone wise once said (and advised me for this document) – “tell ‘em what you’re going to tell ‘em, tell ‘em, then tell ‘em what you told ‘em”. Sage advice, indeed.

So again, here are the key points I hope you take away from reading this document:

- There is no silver bullet. Performance can be slow for a number of reasons. Collect data to help identify where the time is going. Then focus your improvements in the most expensive area, then move to the next area, and so on. Stop when it’s fast enough or the effort/reward ratio is no longer viable.
- Many slow dashboards are caused by poor design. Keep it simple. Don’t try to show too much at once and use guided analysis designs where possible.
- Don’t work with data you don’t need. Use filters, hide unused fields and aggregate.
- Try not to fight against the data engine. It’s smart and it’s trying to help you, so trust that it will generate efficient queries.
- The cleaner your data is and the better it matches the structure of your questions, the faster your workbooks will run and the happier your life will be.
- Extracts are a quick and easy way to make most workbooks run faster. If you don’t need real-time data and aren’t working over billions of rows of data, you should try them.
- Strings and dates are slow, numbers and Booleans are fast.
- Although this document is based on the best advice from a lot of smart people, its recommendations are just that – recommendations. You will need to test which ones will improve performance in your specific case.
- When you are working with small data volumes much of this won’t matter – you can just brute force through bad technique. But it doesn’t hurt to follow these recommendations in all your workbooks as you never know when your data will grow.
- Practice makes perfect.

Now go forth, and make lots of efficient workbooks!