Best Practices for Designing Efficient Tableau Workbooks

Author: Alan Eldridge
Sales Consultant, Tableau Software
January 2013
Foreword

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

This document has been written based on the capabilities of Tableau 7. Tableau 8, due for release in Q1 2013, will provide new features and capabilities that may change some of these guidelines.

Introduction

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

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

- **Interactive** – Tableau documents are primarily designed for interactive delivery to users, either on their desktops, over the web or on a mobile device. Unlike in other BI tools that primarily produce print-focused output (either to actual paper or to a document such as a PDF), authors can create rich, interactive experiences that allow users to explore data and be guided through business questions.

- **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 absolutely 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.

- **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.

Consequently, working in Tableau is a new experience for many users and there are techniques and best practices they need to learn in order to create efficient workbooks. However we find many new users try to apply old design approaches with Tableau and get lacklustre results. We want to develop and promote best practices for the use of a new category of business data analysis and a new category of data analysis tools.
What Do We Mean by Efficient Workbooks?

There are several factors that define an “efficient” workbook. Some of these factors are technical and some more user-focused. An efficient workbook is:

- A workbook that takes advantage of the “principles of visual analysis” to effectively communicate the message of the author and the data, possibly by engaging the user in an interactive experience.
- A workbook that responds in a timely fashion. This can be a somewhat subjective measure, but in general we would want the workbook to provide an initial display of information and to respond to user interactions within a couple of (< 5) seconds.

Note that the design of the dashboard can materially affect the technical implication – something that we will cover in more detail in the next section.

When Is Tableau Not the Right Tool?

While Tableau is a rich and powerful tool, 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 here is that these are not the types of problems Tableau was developed to solve and therefore if you pursue them the effort/reward ratio will likely be unfavourable and the resulting solution may perform poorly or inflexibly.

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

- You need a document that has been designed for paper, not the screen. By this, we mean if you have a need to control complex page layouts, need features such as page, section and group headers/footers, or need precise WYSIWYG formatting.
- Tableau can produce multi-page reports but they lack the level of format control that is available in dedicated, banded-style reporting tools.
- You need a complex push-delivery mechanism for documents with personalisation (also called “bursting”) sent via multiple delivery modes.
- Tableau can be used to create push-delivery systems but this is not a core feature of Tableau 7. It requires development of a custom solution using the TABCMD utility. Scheduled delivery of dashboards and reports via email is a planned feature of Tableau 8.
- Your need a document where the primary use case for the reader is to export the data to another format (often a CSV or Excel file). This often means a tabular report with many rows of detailed data.
- To be clear, Tableau does allow users to export data from a view or dashboard to Excel – either at a summary or detail level. However, when the primary use case is to export it means this is an ersatz extract-transform-load (ETL) process. There are much more efficient solutions than a reporting tool to achieve this.
- You need highly complex, crosstab-style documents that perhaps mirror existing spreadsheet reports with complex sub-totalling, cross-referencing, etc. Common examples here are financial reports such as P&L, balance sheet, etc. Additionally there may be the need for scenario modelling, what-if analysis and even write-back of assumption data.
- If the underlying granular data is not available or if the report logic is based on “cell references” rather than rolling up records to totals then it might be appropriate to continue using a spreadsheet for this style of report.

Principles of Visual Analysis

One of the technical breakthroughs in Tableau is VizQL, a technology that allows users to visualise data of any size simply by dragging and dropping. The innovation is a patented formalism that translates actions into a database query and then expresses the response graphically.

This created a fundamentally new way of interacting with databases and spreadsheets. People could now combine visualisation and analysis in the process of visual analysis. Visual analysis means presenting
information in ways that support visual thinking. The right presentation of data makes it easy to organise and understand the information. Computational support for visualisation gives the user the ability to iterate on different presentations of data, asking new questions and exploring it interactively.

This section aims to suggest ways to best utilise this new paradigm. While many of these are just as applicable to traditional BI as to Tableau, the tools for analysis are now in the hands of many new users so it’s helpful to call them out here.

Building Interactive Solutions vs. Reports

“Most people make the mistake of thinking design is what it looks like … That’s not what we think design is. It’s not just what it looks like and feels like. Design is how it works.”

- Steve Jobs, 2003

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 filter/parameter selection page:

We select filters from a series of checkboxes, and click the “go” button to produce the following…

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.

With the quote from Steve Jobs in mind, don’t focus on the presentation of the data (that is important but it’s a topic for a later post). 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 amount of time to execute and stream back to the Tableau engine. The “first contact” experience is critical for setting an end-user’s perception of the solution and if it takes more than a few seconds before anything happens the perception will be a negative one.

- Creating a view with hundreds of thousands to millions of marks (each cell in a crosstab is called a mark) requires a lot of CPU and memory. It also takes time – adding to the negative perception of system responsiveness. On Tableau Server, having many people all generating large crosstabs can result in slow performance, and in a worst case scenario the system could run out of memory. This is technically referred to as “a very bad thing” and can cause server stability issues, errors and all kinds of unpleasant experiences for end users. Sure, you could add more memory to the server to minimise this but this is treating the symptom, not the cause.

- Finally, the users have no contextual guidance on whether their initial set of filters will be too wide or too narrow. How is a report user to know that if they check all available categories their initial query will return tens of thousands of records and exhaust all available RAM on the server? They can’t, other than through painful experience.

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

- The initial query that must be run is highly aggregated and consequently returns only a handful of records. For a well-designed database this is a very efficient activity so the “first contact” response time is very fast, leading to a positive perception of the system. As we drill down, each subsequent query is both aggregated and constrained by the selections from the higher level. They continue to be fast to execute and return to the Tableau engine.

- Although we have more views when the dashboard is fully completed, each view only shows a few dozen marks. The resources necessary to generate each of these views, even when many end users are active on the system, are trivial and it is now much less likely that the system will run out of memory.

- Finally, you can see that for the higher “navigation” levels we’ve taken the opportunity to show the volume of sales in each category. This gives the user some context on whether this selection contains many records or few. We’ve also used
Best Practices for Designing Efficient Tableau Workbooks

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.

The following links provide more detail on the approaches for building interactive documents:

- Guided Analytics
- Filter Actions
- Creating Filter Actions
- Clearing the Dashboard with Actions
- The Power of Tableau Actions

Telling Stories with Data

“...story telling is a thing you’re taught, just as English boys and girls are taught essay-writing. The difference is that people want to hear the stories, whereas I never heard of anyone who wanted to read the essays.”
- C.S. Lewis

It’s tempting to present just the data and facts, but when colleagues and senior management are overwhelmed by data and facts without context, you lose. We have all experienced presentations with large slide decks, only to find that the audience is so overwhelmed with data that they don’t know what to think, or they are so completely tuned out, they take away only a fraction of the key points.

Stories bring life to data and facts. They can help you make sense and order out of a disparate collection of facts. They make it easier to remember key points and can paint a vivid picture of what the future can look like. Stories also create interactivity—people put themselves into stories and can relate to the situation.

The following whitepapers provide further reading on how to develop Tableau workbooks that tell a story:

- Telling Stories with Data
- Tableau Visual Guidebook

Which Chart Should I Use?

Transforming data into an effective visualisation (any kind of chart or graph) is the first step towards making your data work for you. In the following whitepaper you’ll find best practice recommendations for when to use different types of visualisations:

- Which Chart or Graph is Right for you?

The following whitepapers, knowledge base articles, books and blogs are great resources for further reading on the art and science of visual analysis:

- Stephen Few - Show Me the Numbers; Now You See It; Information Dashboard Design; www.perceptualedge.com
- Stephen McDaniel - Rapid Graphs with Tableau Software
- Colin Ware - Visual Thinking for Design
- John Maeda - The Laws of Simplicity
- Tableau Visual Analytics Virtual Classroom

Writing Efficient Workbooks

Basic principles

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

Everything in moderation!

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

If you find yourself with a workbook like this, consider breaking it into several separate files. If your dashboards are overly complex, consider simplifying them and using interactions to guide the end users from view to view.

Remember, we don’t price our software by the document so feel free to spread the data out a little.
If it isn’t fast in the database, it won’t be fast in Tableau.

If your Tableau workbook is based on a query that is slow to run no matter what tool you use to submit it, then your workbook will in turn be slow.

In the following sections we will identify tuning tips for your databases to help improve the time it takes for queries to run. Additionally, we’ll discuss how Tableau’s fast data engine can be used to improve query performance.

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

A workbook that performs slowly in Tableau Desktop won’t get any faster by publishing it to Tableau Server. In general, workbooks will perform slightly slower on Tableau Server because a) there are multiple users all sharing the server resources to generate workbooks simultaneously; and b) the server has to do the work to render the dashboards and charts rather than this being done on the client workstation.

The exception to this rule is if Tableau Desktop is encountering resource limits (e.g. your PC does not have enough RAM to support the data volume you are analysing) that aren’t present on the server. 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.

In the following sections we will discuss how the design of a workbook can be optimised to ensure it runs efficiently on Tableau Server as well as on Tableau Desktop. Note that none of this advice is mandatory – Tableau will generally work fine if none of these practices are followed, however doing so will improve performance, often significantly.

Data sources

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

- File-based data sources
- Relational data sources
- OLAP data sources
- Web-based data sources

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

File-based Data Sources

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

In general we would advise users to import these data source types into the Tableau fast data engine. This will make queries perform much faster and it also results in a much smaller file to store the data values. However, if the file is small or if you need a live connection to the file to reflect changing data you can connect live.

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

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

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

Relational Data Sources

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 also includes Map-Reduce data sources as they are accessed through SQL access layers like Hive.

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

Indexes

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

- Make certain you have indexes on all columns that are part of table JOINs.
- Make certain you have indexes on any column used, within Tableau, in a FILTER.
- Be aware that using discrete date filters in some databases can cause queries to not use indexes on date and datetime columns. We’ll discuss this further in the filter section, but using a range date filter will ensure the date index is used. I.e. instead of using YEAR([DateDim])=2010 express the filter as [DateDim] >= #2010-01-01# and [DateDim] <= #2010-12-31#).
- Ensure you have statistics enabled on your data to allow the query optimiser to create high-quality query plans.
- Many DBMS environments have management tools that will look at a query and recommend indexes that would help.

Referential Integrity

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

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

When you join multiple tables in a data source, Tableau has a nifty (and generally invisible to the user) functionality 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.

Join culling only occurs where referential integrity has been defined between tables. Without it, Tableau generates:

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

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

If the PK/FK relationships are defined in the underlying database, the query is simplified to this:

\[
\text{SELECT} \quad \text{[ProductDim].[Product Category]}, \quad \text{SUM([OrdersFact].[Sales])} \\
\text{FROM} \quad \text{[dbo].[OrdersFact] [OrdersFact]} \\
\text{INNER JOIN} \quad \text{[dbo].[ProductDim] [ProductDim]} \\
\text{ON} \quad ([OrdersFact].[Product ID] = [ProductDim].[Product ID]) \\
\text{GROUP BY} \quad \text{[ProductDim].[Product Category]}
\]

See the following excellent blog post by Russell Christopher for more information, [click here](https://www.russellchristopher.com/).

**Nulls**

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

**Calculations**

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

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

**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.

**OLAP Data Sources**

Tableau supports several OLAP data sources:

- Microsoft Analysis Services
- Microsoft PowerPivot (both PowerPivot for Excel and PowerPivot for SharePoint)
- Oracle Essbase
- SAP BW
- Teradata OLAP
- Oracle OLAP (coming in Tableau 8)

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

See Appendix A for more details.

**Web-based Data Sources**

Tableau currently supports two web-based data sources:
- oData
- Windows Azure Marketplace DataMarket

Both of these 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.

Note – in Tableau 8 we will be introducing several new
web-based data sources:
- Salesforce.com
- Google Analytics
- Google BigQuery

**Queries**

Often a problem with slow-running visualisations is that
you have inadvertently created a query that returns a
large number of records from the underlying table(s),
when a smaller number of aggregated records would
suffice. The time it takes the database to calculate the
results, then stream the records back to Tableau can be
significant. You can check this by looking in the lower-left
corner of the Tableau Desktop workspace and looking at
the number of marks. If this number is very large, you are
potentially pulling a large amount of data from the
database.

Ensure you are not including any unnecessary
dimensions in your visualisation - this will affect the
aggregations in the database and increase the size of the
result set.

Another way to determine if a slow workbook is being
caued by a slow query is review the Tableau Desktop
log file. To find the query being run, look in My
Documents\My Tableau Repository\Logs and find a file
titled log.txt. Open this file and scroll up from the bottom
until you find a section like the following:

```
2012-10-13 21:16:43.515 (1348) Data connection is returning a specification
2012-10-13 21:16:43.515 (1348) CAN NOT DETERMINE: Reading primary query.
2012-10-13 21:16:43.515 (1348) CAN NOT DETERMINE: Reading primary query.
2012-10-13 21:16:43.515 (1348) CAN NOT DETERMINE: Reading primary query.
2012-10-13 21:16:43.515 (1348) CAN NOT DETERMINE: Reading primary query.
```

The section between the query tags is the query that was
passed to the database. The log file shows the time it
took to run and the number of records returned. You
could also copy this text and then use it from a tool like
Access or Excel. If it takes a similar time to return as in
Tableau, then it’s likely the problem is with the query, not
the tools.

**Multiple Tables**

In Tableau, when connecting to a data set that spans
across multiple tables (e.g. a star or snowflake schema
with fact and dimension tables) you can define the
connection using multiple tables or custom SQL.

With multiple tables, Tableau will dynamically create the
SQL based on the fields used in the visualisation – e.g. if
we just select the Sales measure, all the DB has to
process is:

```
SELECT SUM(SALES) FROM ORDERSFACT
```

If we then add the product category, it processes:

```
SELECT PRODUCT_CAT, SUM(SALES)
FROM ORDERSFACT INNER JOIN PRODUCTDIM
GROUP BY PRODUCT_CAT
```
Furthermore, if referential integrity has been defined appropriately between the tables, Tableau can perform “join culling” and drop unused dimension tables from the join expression. See the section on referential integrity for more detail.

**Custom SQL**

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

```sql
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],
) [TableauSQL]
HAVING (COUNT_BIG(1) > 0)
```

A good recommendation is to use custom SQL in conjunction with Tableau’s fast data engine. That way the atomic query is only executed once (to load the data into the data extract) and all subsequent analysis in Tableau is done using dynamic, optimised queries against the data extract. Alternately, see the following section on context filters for another technique to force the database to materialise the results of the custom SQL into a temporary table.

Note that in Tableau 8 we will be introducing the ability to include parameters into custom SQL statements. This will increase the flexibility of custom SQL and make the use of functions and stored procedures more accessible, however the statement will still be executed atomically.

**Blending vs. Joining**

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.

Information on blending as a data integration technique can be found here:

- On-demand training video: Blending
- Information Lab Blog: Tableau for Excel users – Part 3 – Data Blending

If the workbook uses data from more than one data source, you need to either blend data or establish a federated database system. If the workbook uses two data connections from the same data source, you want to join the data tables as this can improve performance and filtering control. If you are adding data to an existing data source or a table in the same data connection, using a join is better.

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

- If there are too many records for a join to be practical, blending data is the better choice.
- If you want to display a summary and details at the same time, blending data is the better choice.

The following article shows several examples of when you may want to use a join and when you want to use data blending, [click here](#).

Note - if you are using a cube data source for data blending, it must be the primary data source to be valid.

Finally, there are several major enhancements to blending being introduced in Tableau 8. You can blend data from multiple sources without including the linked field in the view itself. Additionally, dimensions from secondary data sources can now be filtered regardless of whether they are a linked field.
Excerpts

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.

Our partners at The Information Lab have written an excellent blog post, explaining several use cases where extracts provide benefit (make sure you also read the comments for additional examples from other users).

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

Finally, extracts cannot be created over OLAP data sources such as SQL Server Analysis Services, Oracle Essbase or SAP BW.

Creating Extracts

This bit is simple, assuming you are using Tableau Desktop. After you have connected to your data, go to the DATA menu and choose EXTRACT DATA – then accept the defaults on the dialog box (although more on this later). Tableau will ask where you want to save the extract – choose any location to save the file, although Tableau will probably direct you towards ‘My Tableau Repository | Datasources’ which is just fine too!

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

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

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

The extract creation process requires temp disk space to write working files – it may require up to the square of the size of the resulting extract file (e.g. a 10GB extract may...
require 100GB or temp space). This is done in the directory specified by the TEMP environment variable (usually C:\WINDOWS\TEMP or C:\USERS\USERNAME\APPDATA\LOCAL\TEMP). If this drive has insufficient space, point the environment variable to a larger location.

If it is impossible (or just impractical) to do an initial extract process on a workstation, the following workaround can be done to create an empty extract that is then published to Tableau Server. Create a calculated field that has NOW() in it. Then add it to the extract filters and exclude the single value it shows. This will build and 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.

**Aggregated Extracts**

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

You can define the extract ahead of time by choosing which fields you want and selecting the “Aggregate data for all visible dimensions” check box in Tableau Desktop’s Extract Data dialog box. Alternatively, after doing your analysis and building your dashboard, when you are ready to publish, you can go back into the Extract Data dialog box and click the button for Hide All Unused Fields. Then when you extract the data, it is the absolute minimum required to create the view. People often use that setting for their summary view. Then on another page of the workbook, you can use a lower level of detail, but a larger extract. Since you have filtered that data, it should still perform well. You can keep this process going until you’re connected to live system on the back end. Live systems are good at identifying a small set of rows. In this way, you can mix and match, and aggregate at different levels to resolve nearly any performance issues so that get the results as fast as necessary. Since Tableau is efficient with memory, improving performance this way is usually relatively easy and you can have multiple extracts running at the same time.

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

**Optimising Extracts**

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

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

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

**Refreshing extracts**

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

Note: If the user wants to refresh their data more often than every 15 minutes, they should probably connect to live data, or set up a synchronised report database.

You can choose two refresh schedules for a single extract:

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

**What happens if the refresh takes longer than the increment?**

For example, the schedule is set to refresh the data every hour, but the amount of data is so large that the refresh takes an hour and a half. This situation might actually be desirable.

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

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

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

You also can refresh a workbook incrementally or fully via the Tabcmd command line tool. If you have complex scheduling requirements you can invoke this from an external scheduling tool such as the Windows Task Scheduler. This approach is required if you want a refresh cycle that is shorter that the 15 minute minimum allowed through the Tableau Server interface.
You can set updates to occur on a schedule. Or you can choose to disable the schedule and then manually initiate refreshes when you want them.

Tableau schedules are used only for refreshing extracts. You cannot attach other activities to the schedules. Note that in V8, schedules will also be used for subscriptions – delivering dashboards via email.

**Extracts vs. Live Connections**

The speed of the Tableau fast data engine is relative. You must consider the source data and the processing power you have already given that data, as well as the processing power you have on Tableau before you can determine whether the data engine is going to offer an improvement.

For a non-optimised database or file-based data source, the data engine’s processing is much faster and will result in a better user experience. But a well optimised database with indexing might be just as fast as the Tableau data engine.

At the other extreme, the Tableau data engine will probably be slower than a big cluster of machines like you would find with Data Warehouse appliances such as Teradata or Greenplum. You might create an aggregated extract to offload summary-style analysis, but then drill back to the detailed source data (using actions or blending) which would remain in the DW.

**Filtering**

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

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

### Filtering Categorical Dimensions

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

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

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

**Discrete**

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

The third option is fast in this example because the resulting filter (WHERE STATE="Western Australia") is very simple and can be efficiently processed by the database. However this approach becomes less effective as the number of dimension members needed to express the filter increases – eventually approaching the performance of the lasso and keep-only option.
Ranged
Using the ranged filter approach also allows the database to evaluate a simple filter clause (either WHERE POSTCODE >= 6000 AND POSTCODE <= 7000 or WHERE LONGITUDE < 129) resulting in fast execution. However this approach, unlike a filter on a related dimension, doesn’t become more complex as we increase the cardinality of the dimensions.

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

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

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

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

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

```
SELECT [FactSales].[Order Date], SUM([FactSales].[SalesAmount])
FROM [dbo].[FactSales] [FactSales]
WHERE (([FactSales].[Order Date] >= {ts '2009-01-01 00:00:00'})
AND ([FactSales].[Order Date] <= {ts '2012-12-31 00:00:00'}))
GROUP BY [FactSales].[Order Date]
```
This type of WHERE clause is very efficient for query optimisers, allowing execution plans to leverage indexes and partitions to full effect. If you are observing slow query times when you add discrete date filters, consider replacing them with ranged date filters and see if that makes a difference.

Relative

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

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

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

Context Filters

By default, all filters that you set in Tableau are computed independently. That is, each filter accesses all rows in your data source without regard to other filters. However, you can set one or more filters as context filters for the view. You can think of this as an independent filter. Any other filters that you set are defined as dependent filters because they process only the data that passes through the context filter.

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

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

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

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

```
SELECT SUM([OrdersFact].[Sales])
FROM [dbo].[OrdersFact] [OrdersFact]
INNER JOIN [dbo].[CustomerDim] [CustomerDim] ON ([OrdersFact].[Customer ID] = [CustomerDim].[Customer ID])
INNER JOIN [dbo].[DeliveryDim] [DeliveryDim] ON ([OrdersFact].[Delivery ID] = [DeliveryDim].[Delivery ID])
INNER JOIN [dbo].[LocationDim] [LocationDim] ON ([OrdersFact].[Place ID] = [LocationDim].[Place ID])
INNER JOIN [dbo].[ProductDim] [ProductDim] ON ([OrdersFact].[Product ID] = [ProductDim].[Product ID])
INNER JOIN [dbo].[TimeDim] [TimeDim] ON ([OrdersFact].[Date ID] = [TimeDim].[Date ID])
WHERE (((LocationDim).[Region] >= ‘Africa’) AND ((LocationDim).[Region] <= ‘Oceania’))
HAVING (COUNT_BIG(1) > 0)
```
By setting a context menu on a dimension that returns all dimension members, we force Tableau to materialise the above query and write the results to a temporary table. This results in the same query being regenerated as follows:

```
SELECT SUM([#TABLEAU_3CONTEXT].[SALES])
FROM [#TABLEAU_3CONTEXT]
HAVING (COUNT_BIG(1) > 0)
```

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

**Quick Filters**

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

**Enumerated vs. Non-enumerated**

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

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

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

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

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

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

**Relevant Values**

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

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

- **All Values in Context** – this option is only available when you have active context filters. The quick filter will show all values in the context (i.e. the temporary table generated by the context filter) regardless of other filters on the view. The quick filter does not need to re-query the database when other filters are changed.

- **Only Relevant Values** - when you select this option other filters are considered and only values that pass these filters are shown. For example, a quick filter on State will only show the Eastern states when a filter on Region is set. Consequently, the quick filter must re-query the data source when other filters are changed.

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

**Quick Filter Alternatives**

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

Instead of exposing a quick filter to the users, you could create a parameter and filter based on the users’ selections.
**PROS:**
- Parameters do not require a data source query before rendering
- Parameters + calculated fields can implement logic that is more complex than can be done with a simple field filter
- Parameters can be used to filter across data sources – quick filters operate only within a single data source

**CONS:**
- Parameters are single-value only – you cannot use them if you want the user to select multiple values
- Parameters are not dynamic – the list of values is defined when they are created and does not update based on the values in the DBMS

Another alternative is to use filter actions between views -

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

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

For a detailed discussion on alternate design techniques that don’t rely heavily on quick filters, see this earlier section.

**User Filters**
Any workbooks that utilise user filters – either through the “Create User Filter…“ dialog or through calculated fields that use any of the built-in User functions – cannot use shared result caches when deployed to Tableau Server because they are unique to each user. This can have performance impacts as:
- All workbooks will need to query the underlying data source, even if another user session has just asked the exact same query. This results in more data source I/O.
- More cache space will be required as each user session will create its own query result and model cache. On machines with high load this can result in caches being cleared while still in use, again resulting in more I/O.

See the following section on caching for more detail.

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

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

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

There are three different calculation types in Tableau:
- Basic calculations
- Aggregate calculations
- Table calculations
Basic and Aggregate Calculations

Basic and aggregate calculations are expressed as part of the query sent to the data source and therefore are calculated by the database. For example:

```
SELECT DATEPART(YEAR,[TIMEDIM].[DATE]),
SUM([ORDERSFACT].[SALES])
FROM [DBO].[ORDERSFACT] [ORDERSFACT]
INNER JOIN [DBO].[TIMEDIM] [TIMEDIM]
ON ([ORDERSFACT].[DATE ID] = [TIMEDIM].[DATE ID])
GROUP BY DATEPART(YEAR,[TIMEDIM].[DATE])
```

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

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

Table Calculations

Table calculations, on the other hand, are not executed by the database but rather calculated by Tableau on the query results that get returned. While this means more work for Tableau, it is generally done over a much smaller set of records than are in the original data source.

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

```
WINDOW_AVG(SUM([SALES]))
```

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

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

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

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

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

Calculations vs. Native Features

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

- Grouping dimension members together – consider using Groups;
- Grouping measure values together into “bins” – consider using Bins;
- Changing the displayed values for dimension members – consider using Aliases.

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

Data Types

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

- Integers are faster than Booleans and both are much faster than Strings

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

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

**Performance Techniques**

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

**Use Booleans for Basic Logic Calculations**

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

```
IF [DATE]= TODAY() THEN “TODAY”
ELSE “NOT TODAY”
END
```

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

```
[DATE]=TODAY()
```

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

**String Searches**

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

```
IF FIND([PRODUCT NAME],[PRODUCT LOOKUP])>0
THEN [PRODUCT NAME] ELSE NULL END
```

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

```
CONTAINS([PRODUCT NAME],[PRODUCT LOOKUP])
```

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

**Parameters for Conditional Calculations**

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

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

<table>
<thead>
<tr>
<th>VALUE</th>
<th>DISPLAY AS</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
<td>YEAR</td>
</tr>
<tr>
<td>QUARTER</td>
<td>QUARTER</td>
</tr>
<tr>
<td>MONTH</td>
<td>MONTH</td>
</tr>
<tr>
<td>WEEK</td>
<td>WEEK</td>
</tr>
<tr>
<td>DAY</td>
<td>DAY</td>
</tr>
</tbody>
</table>

Then use it in a calculation like this:

```
IF [PARAMETERS].[DATE PART PICKER]="YEAR"
THEN DATEPART(‘YEAR’,[ORDER DATE])
ELSEIF [PARAMETERS].[DATE PART PICKER]="QUARTER"
THEN DATEPART(‘QUARTER’,[DATE])
…
ELSE NULL END
```

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

<table>
<thead>
<tr>
<th>VALUE</th>
<th>DISPLAY AS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YEAR</td>
</tr>
<tr>
<td>2</td>
<td>QUARTER</td>
</tr>
<tr>
<td>3</td>
<td>MONTH</td>
</tr>
<tr>
<td>4</td>
<td>WEEK</td>
</tr>
<tr>
<td>5</td>
<td>DAY</td>
</tr>
</tbody>
</table>
The calculation would then be as follows – see how the comparisons are numeric rather than string:

```sql
IF [PARAMETERS].[DATE PART PICKER]=1
    THEN DATEPART('YEAR',[ORDER DATE])
ELSEIF [PARAMETERS].[DATE PART PICKER]=2
    THEN DATEPART('QUARTER',[ORDER DATE])
ELSE NULL END
```

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

```sql
DATEPART([PARAMETERS].[DATE PART PICKER],[ORDER DATE])
```

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

### Date Conversion

Users often have date data that is not stored in native date formats – e.g. it could be a string or a numeric timestamp. A common 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):

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

A much more efficient way to do this calculation is:

```sql
DATEADD('DAY', [YYYYMMDD]%100-1,
    DATEADD('MONTH',
        INT((YYYYMMDD)%10000)/100-1,
            DATEADD('YEAR', INT(YYYYMMDD)/10000)-1900,
                #1900-01-01#)))
```

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

### Date Functions

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

### Logic Statements

When working with complex logic statements remember that

```sql
ELSEIF > ELSE IF
```

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

```sql
IF [REGION] = "EAST" AND [CUSTOMER SEGMENT] = "CONSUMER"
    THEN "EAST-CONSUMER"
ELSE IF [REGION] = "EAST" AND [CUSTOMER SEGMENT] <>"CONSUMER"
    THEN "EAST-ALL OTHERS"
END
```

would run much faster as:

```sql
IF [REGION] = "EAST" AND [CUSTOMER SEGMENT] = "CONSUMER"
    THEN "EAST-CONSUMER"
ELSEIF [REGION] = "EAST" AND [CUSTOMER SEGMENT] <>"CONSUMER"
    THEN "EAST-ALL OTHERS"
END
```

but this is faster still:

```sql
IF [REGION] = "EAST" THEN
    IF [CUSTOMER SEGMENT] = "CONSUMER" THEN
        "EAST-CONSUMER"
    ELSE "EAST-ALL OTHERS"
END
```

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.
Similarly, avoid redundant logic checks. The following calculation:

\[
\begin{align*}
&\text{IF } [\text{SALES}] < 10 \text{ THEN } \text{“BAD”} \\
&\text{ELSEIF } [\text{SALES}] \geq 10 \text{ AND } [\text{SALES}] < 30 \text{ THEN } \text{“OK”} \\
&\text{ELSEIF } [\text{SALES}] \geq 30 \text{ THEN } \text{“GREAT”} \\
&\text{END}
\end{align*}
\]

would be more efficiently written as:

\[
\begin{align*}
&\text{IF } [\text{SALES}] < 10 \text{ THEN } \text{“BAD”} \\
&\text{ELSEIF } [\text{SALES}] \geq 30 \text{ THEN } \text{“GREAT”} \\
&\text{ELSE } \text{“OK”} \\
&\text{END}
\end{align*}
\]

**Separating Basic and Aggregate Calculations**

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

**Dashboard and Views**

As already discussed, the design of a view or dashboard has a significant impact on its performance. The following section highlights a number of techniques to ensure your calculations are as efficient as possible:

**Views**

**Only Fetch and Draw What You Need**

Tableau is inherently good at guiding users to display only the fields they need, but sometimes you can have additional fields on the level of details shelf (e.g. measure fields that were being used in the tooltip/title/caption). If these are no longer required, remove them from the view.

**Charts vs. Crosstabs**

As a rule of thumb, charts generally perform better than crosstabs in Tableau. For example, to query and display 1.6M data marks on a map takes ~45 seconds to complete. Displaying that same information on a tabular report took over twice as long and consumed significantly more memory while rendering the table. We have seen some customers attempting to render crosstabs that are so large they cause Tableau to run out of memory.

As mentioned earlier, avoid using Tableau to display crosstabs with large row counts. If you drill into the use cases of these reports you will often find they are nothing more than a back door data extract process, and there are better tools for that task than Tableau.

**Removing Unnecessary Geographic Roles**

If you are using a dimension that has a geographical role, consider setting the role to “none” if you are NOT going to render your data in a map. This will save some time looking up the Tableau generated Latitudes and Longitudes.

**Blending vs. Custom Geographic Roles**

Tableau allows users to add custom data to the inbuilt geocoding engine – either to add more rows to an existing role or to create an entirely new role. This is a useful feature, but customers who use a custom geographic role and then save their file as a packaged workbook might be surprised by how large the resulting file is. One customer saw their average file size jump from ~5MB to almost 80MB, making distribution of the files impractical.

The increase in size is due to the geocoding data being embedded into the workbook. Rather than just copying the custom geographic role data, the entire GEOCODING.FDB file (about 70MB before custom data is added) is added.

One way to avoid this is to include the geographic data by joining or blending, rather than loading it as a custom geographic role. This allows the resulting workbook to just contain the custom geographic data, which will be much smaller.
Dashboards

Less Views, Less Quick Filters

When first learning to create dashboards, many new users often try to put all their views on a single dashboard page. This can result not only in a dashboard that is very difficult to read, but also in a dashboard that is very slow to render.

Each view may require one or more queries of the database to fetch the required data, plus any queries needed to populate any quick filters. Multiply this by several views and there can be a lot of I/O before the dashboard is ready to render.

Furthermore, Tableau processes the views in a dashboard in a serial fashion. Imagine you have a dashboard with 5 views and 5 quick filters, all of which are fast, rendering in < 1 second. Even with these fast components, the dashboard will take at least 10 seconds to render. Consider spreading the information out across multiple pages or consolidating multiple views together into a single combination chart.

Note – in Tableau 8, dashboards will be able to process multiple views in parallel (where possible) which will reduce the impact of this particular bottleneck.

Turn Off Tabs

When a workbook is published to Tableau Server and tabs are turned on, Tableau must process every view in every tab before the initial view can be displayed. This does not mean executing the queries for each sheet but it needs to understand the structure of the tabs in anticipation of any actions or filters.

Imagine you have a workbook with 5 tabs, each containing a dashboard with 5 sheets. Before Tableau Server can display the first view it will have to internally process 25 sheets before sending the queries to the database.

Reducing the number of tabs can sometimes improve performance. To test, add the ‘tabs=no’ parameter to a view URL and compare the performance to the fully tabbed view.

Filter Actions

When using a filter action, select the behaviour on clearing the selection to “exclude all values”. This avoids the potentially expensive query of asking for all of the data that the target view is able to render when the user clears the selection on the source view.

Fixed Size Dashboards

We cover the details of Tableau Server caching in more detail here, but essentially if two users ask Tableau Server for the exact same view but are running browsers with different window sizes, the VizQL server must render the view for each user separately. Different window sizes mean the view is drawn differently for each request.

If you create a dashboard and leave it with the default size setting of “Automatic (Fit To Window)”, you could potentially be making Tableau Server work much harder than it needs to, due to the low hit rate on your caches.

For best performance, choose an exact size for your dashboards.

Check out this blog post from Russell Christopher for more detail.

Tools to Analyse Performance

In the previous sections we have provided many points of guidance on how to improve various aspects of a workbook’s performance – speeding up slow queries, making calculations and filters more efficient, changing dashboard design to make it render faster. But how do you know where to start? This section shows you where to start looking for information on what aspects of your workbook are potentially inefficient.
Tableau Desktop Messages

The first way to understand what is happening with a Tableau workbook is to open it in Tableau Desktop and watch for dialog boxes that indicate different phases of the dashboard display process.

![Executing Query](image)

This dialog indicates that Tableau is executing a query to return records for the view. If this is taking a long time it indicates that the main query is slow. Review the log file to see what queries are taking a long time, and then consider some of the calculation, query and filter techniques outlined earlier to ensure the most efficient query statement possible is being created.

![Computing visualization of Sheet 2...](image)

This dialog indicates that Tableau has received all the data it needs and is now rendering the display of the information. If this is taking a long time, review some of the techniques for calculation optimisation and view design. View layout can take a long time if you are rendering lots of marks, if you are rendering a very large crosstab, or if you have slow-running table calculations.

![Computing quick filters for](image)

This dialog indicates that Tableau is processing the quick filters for this dashboard. You may see this often if you are using “show relevant values” on a lot of quick filters, or if you are displaying enumerated quick filters with lots of values. Consider reviewing the techniques for filter performance and view design to improve performance of this phase.

Log Files

The Tableau log files provide detailed information on the performance of a workbook and should be referred to in order to understand where bottlenecks are occurring. It contains timing data for queries that are being run. An example of the Tableau Desktop log file contents are below:

```
2012-12-17 10:19:14.149 (0004): Executing query
2012-12-17 10:19:14.153 (0004): Done. Time taken: 0.00 sec.
```

As you can see, this information includes information on what Tableau is doing, what communication it is having with the data source and how long each step is taking. We can use this to determine if bottlenecks are occurring within Tableau (and if so, where) or within the database.

For Tableau Desktop, the log file can be found at:
- `C:\Users\username\Documents\My Tableau Repository\Logs`

For Tableau Server, the equivalent information can be found in the VizQL log file, located here:
- `C:\ProgramData\Tableau\Tableau Server\data\tabsvc\vizqlserver\Logs`

The following log files track activities related to the web application, database, and index. They are found in the subdirectories for each component:
- `C:\ProgramData\Tableau\Tableau Server\data\tabsvc\`

Tableau Performance Analyzer from Interworks

While there is a lot of useful information in the Tableau Desktop log file, it can be hard to understand as it is very granular and text-based. Fortunately one of our partners, Interworks, has created a useful tool called the Tableau...
Performance Analyser that reads the log file into Tableau and visualises the data. It produces a graphical analysis of the workbook performance, as shown below:

As you can see, the output identifies slow running dashboards and sheets, showing the time each query takes to run. Use this information to determine where best to focus your tuning efforts.

Click here for more information and instructions on how to download the Performance Analyzer.

Database Performance Monitors

If the data source you are querying is a server-based database, it is highly likely that it provides performance analysis tools of its own. These tools can provide insight in the queries that are hitting it and how the database processes them. The following image shows an example of the SQL Server performance monitor – from here you can drill into a specific query to understand how the query optimiser has broken this down, and receive advice on whether there is additional tuning that could possibly improve query performance.

Tableau 8 Performance Metrics

In Tableau 8, a built-in performance monitor will help you analyse performance so you can tune your workbook for optimal performance. Turn on performance recording to record metrics about the various activities and functions performed by the product. You can then look at the results to better understand what Tableau is doing and find areas to optimise your workbook.

Tableau Server

General Guidelines

Use a 64-bit Operating System

Although Tableau Server runs well on 32-bit Microsoft operating systems, for the best performance, choose a 64-bit edition. It ensures that the 64-bit version of the Tableau data engine is used. It also increases the capacity of the 32-bit processes because they get access to more main memory.

Add More Cores and Memory

Regardless of whether you’re running Tableau Server on one machine or several, the general rule is that more CPU cores and more RAM will give you better performance. Make sure you meet Tableau Server’s recommended hardware and software requirements and see When to Add Workers & Reconfigure to assess whether you should add additional machines.
In January 2012, we performed scalability tests on Tableau Server to help our customers plan for large deployments. We tested three different server configurations—using one, two, or three servers in a dedicated, load test environment. We tested two different types of reports with varying degrees of complexity. We tried to simulate real-world usage by having each user perform a variety of tasks including loading the report, performing a selection, filtering the view and changing tabs. This whitepaper describes those tests and outlines techniques for improving Tableau Server’s performance.

**Configuration**

**Schedule Refreshes for Off-Peak Hours**

If server performance is slow, use the Background Tasks administrative view to see your current refresh task schedules. If you can schedule refreshes for off-peak hours, do so.

**Check the VizQL Session Timeout Limit**

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

**Assess Your Process Configuration**

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

**Monitoring Tableau Server**

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

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Activity</td>
<td>A dashboard view showing activity on the server</td>
</tr>
<tr>
<td>User Activity</td>
<td>A view describing user activity, including login time, duration, and disconnects</td>
</tr>
<tr>
<td>View Performance History</td>
<td>A view describing activity and performance history of views</td>
</tr>
<tr>
<td>Background Tasks</td>
<td>A dashboard view summarizing completed and pending tasks</td>
</tr>
<tr>
<td>Source Libraries</td>
<td>A dashboard view showing utilized libraries and data sources</td>
</tr>
<tr>
<td>Custom Views</td>
<td>A dashboard view showing utilization of custom views</td>
</tr>
</tbody>
</table>

More information on these views can be found here.

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.

**Caching**

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

**Image Tile Cache**

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

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

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

**Model Cache**

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

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

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

**Query Result Cache**

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

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

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

**Maximising Cache Use**

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

**Tuning Caches**

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

<table>
<thead>
<tr>
<th>Minimise queries</th>
<th>Balanced</th>
<th>Most up-to-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each VizQL server will cache models and query results for as long as it can.</td>
<td>Each server will cache models and data for at most the specified number of minutes.</td>
<td>Each server will not cache models or data at all</td>
</tr>
<tr>
<td>Requests get cached data until explicitly refreshed.</td>
<td>Requests get cached data that is at most a known number of minutes old.</td>
<td>Request get the freshest data from the data source</td>
</tr>
<tr>
<td>The cache is held in memory, when memory fills, the cache starts to empty oldest items.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It is possible to force Tableau to bypass all the caches and force a data source query by attaching the “:refresh=yes” measure pair to the view URL. For an example [click here](#).

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

<table>
<thead>
<tr>
<th>Model cache</th>
<th>Query cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>vizqlserver.modelcachesize:30</td>
<td>vizqlserver.querycachesize:64</td>
</tr>
<tr>
<td>The number of models to cache, where there is one model per viz instance in a workbook</td>
<td>The size in megabytes of query results to cache</td>
</tr>
</tbody>
</table>

[Click here](#) for more detail.

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

As this document has shown, there are many factors that affect the efficiency of our workbooks. Sadly, there is no magical “run faster” button we can push to make our dashboards respond instantaneously. Tuning a workbook is a matter of analysing performance to identify the bottleneck, making changes to address it, and then repeating the process until our workbook is as efficient as possible.

Some of the techniques outlined in this document apply to Tableau Desktop and the creation of workbooks, some apply to Tableau Server and their publication, and others apply to you (the author) and their design. By understanding and applying these best practices you can ensure that from the start you are developing your workbooks with efficiency and performance in mind.

Happy tuning!
Appendix A – Relational vs. OLAP Capabilities

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

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

<table>
<thead>
<tr>
<th>Specific Feature</th>
<th>Cube</th>
<th>Relational</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Filters</td>
<td>Cubes have the ability to define user lever security. Tableau has the ability to leverage this logic within your environment. You do not need to re-do your security model inside of Tableau. If you do not have this defined in your cube, you can define it in Tableau.</td>
<td>Relational data sources have the ability to define user level security. You do not need to redefine this logic inside of Tableau. If your database does not have this logic, it can be defined in Tableau.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SECURITY</td>
</tr>
<tr>
<td>Data Blending</td>
<td>In any organization, there is data within a cube, and then there is data not in the cube. Tableau allows users to easily blend relational data with cube data, without moving the data, or modeling it in the cube.</td>
<td>Data blending works with relational databases, across databases, and to cubes. Tableau’s approach to asking questions of disparate data sources is unique. Tableau does not require the data to be moved.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA BLENDING</td>
</tr>
<tr>
<td>Set Analysis</td>
<td>Sets are something that many cube users love. Tableau will leverage these sets. However, if it does not exist in the cube, you can create a dynamic, dimensional set within Tableau.</td>
<td>Sets are a way to save dimensional intersections of data. When using a relational source of data, you can create sets within the data similar to cubes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SETS</td>
</tr>
<tr>
<td>Aggregate Calculation Functions</td>
<td>Aggregates are done ahead of time in a cube, and Tableau responds with the aggregation set in the cube. This is one of the main performance benefits of cubes.</td>
<td>Aggregates are typically not done in a relational database ahead of time. Tableau then allows the user to select the aggregation within Tableau and the database runs the aggregation on demand.</td>
</tr>
<tr>
<td>Tableau “Groups”</td>
<td>Groupings are typically defined in the cube by the developer and are pre-calculated. This results in the performance benefit and standard structure. You can do grouping with simple MDX: <code>[CUSTOMER].[CUSTOMERGEOGRAPHY].[FRANCE] + [CUSTOMER].[CUSTOMERGEOGRAPHY].[GERMANY]</code></td>
<td>Grouping is typically not modeled ahead of time in a relational database. However, Tableau provides you the ability to create groups on the fly at any time during your analysis.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CREATING GROUPS</td>
</tr>
<tr>
<td>Specific Feature</td>
<td>Cube</td>
<td>Relational</td>
</tr>
<tr>
<td>-----------------</td>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td><strong>Tableau “Bins”</strong></td>
<td>Bins, or grouping of measure ranges, are typically modeled in the cube and are attributes within the dimension window. This allows for a common definition for Bins. Users can create or modify bins in Tableau with simple MDX: STR(INT([INTERNET SALES AMOUNT] / [BIN SIZE])) * [BIN SIZE])</td>
<td>Bins are typically not modelled in a relational database. However, Tableau provides the ability for users to create bins on measures. <em>EVEN BINS</em> <em>UNEVEN BINS</em></td>
</tr>
<tr>
<td><strong>String Manipulations</strong></td>
<td>String manipulations are often times not done in a cube ahead of time. String manipulations can be done within Tableau via simple MDX: LEFT([PRODUCT].[PRODUCT CATEGORIES].DATAMEMBER.MEMBERVALUE,LEN([PRODUCT].[PRODUCT CATEGORIES].DATAMEMBER.MEMBERVALUE)-5)</td>
<td>When connecting to relational sources, string manipulations can be accomplished directly with Tableau calculated fields. This can be helpful when changing case, creating zip 5 out of zip 5+4 and many other manipulations. <em>STRING FUNCTIONS</em></td>
</tr>
<tr>
<td><strong>Data Types</strong></td>
<td>Data types (e.g. String, Date, Number) and roles (dimension and measure) are explicitly defined within a cube. This ensures that pre aggregations and attributes show up in the right windows in Tableau.</td>
<td>Tableau will automatically detect the column type of a column in a relational database. This limits the amount of data manipulation you need to do in Tableau. When you want to change data types in Tableau, you can do that with a right click: <em>CHANGE DATA TYPES</em></td>
</tr>
<tr>
<td><strong>KPI / Scorecard</strong></td>
<td>Within a cube, you can define attributes that contain information about what KPI groups a certain member falls within. You can also create threshold KPIs directly within Tableau with a simple Calculation/Parameter: [INTERNET SALES AMOUNT] &gt;= [DESIRED SALES]</td>
<td>Within a relational database, you can create KPI calculations very quickly with simple calculated fields. <em>KPI CALCULATIONS</em></td>
</tr>
<tr>
<td><strong>Actions</strong></td>
<td>Tableau Actions are fully supported with cubes. Actions also work from a cube to relational sources in the same workbook. Cube based actions are not supported in Tableau. <em>ACTIONS</em></td>
<td>Actions work within relational data sources as well as across/between cubes and relational sources. This allows both types of data to communicate between each other.</td>
</tr>
<tr>
<td><strong>Hierarchies</strong></td>
<td>This is one of the key benefits of using Tableau with a cube. Tableau also supports drilling down asymmetrically or raggedly through the hierarchy.</td>
<td>Relational databases do not have hierarchies built within them, but Tableau allows you to easily build these on the fly. <em>BUILDING HIERARCHIES</em></td>
</tr>
<tr>
<td>Specific Feature</td>
<td>Cube</td>
<td>Relational</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Quick Filters</td>
<td>When connected to a cube, you see quick filters as part of the hierarchy and its structure is exposed. You can filter at multiple levels within a hierarchy. If you have a single attribute, you can avoid the structured view and have a single-level hierarchy.</td>
<td>Quick filters in a relational database show up as one level, without structure. You can put these quick filters on the visualizations and dashboard with a single right-click.</td>
</tr>
<tr>
<td>Extracts</td>
<td>Cubes, with their pre-aggregated nature, are inherently fast and do not require extraction.</td>
<td>Relational databases can often be slow. Tableau provides the Fast Data Engine as a method to provide screaming fast performance to relational data. Tableau provides the option to connect live or extract the data.</td>
</tr>
<tr>
<td>Aliases</td>
<td>Business friendly names are a common thing for cubes to house. When connecting to Essbase, you can use any alias file that was defined within the cube.</td>
<td>In relational databases, you can create your alias values to change the name of members of a dimension. This is helpful when you want to group things together and give it a new name.</td>
</tr>
<tr>
<td>Formatting</td>
<td>Formatting of fields (percentages, currency, etc.) are defined within a cube. This ensures that numbers are shown correctly without any intervention by the user.</td>
<td>Relational data sources typically do not have any inherent formatting. Defining percentages and currency can be done directly inside of Tableau. You can even set the default formatting, so that each time that measure is used, it is shown in the correct format.</td>
</tr>
<tr>
<td>Sort order</td>
<td>Cubes provide a developer to define the sort order of members within an attribute/dimension. This ensures that when using attributes, your members are shown in the correct order every time. This is helpful when there is non-standard formatting.</td>
<td>Relational databases do not have default sorting as part of their definition. In Tableau, a user can define a sort driven from a measure (sort state by sales) as well as a default manual sorting.</td>
</tr>
<tr>
<td>Fiscal Calendars</td>
<td>Cubes provide a developer the ability to define different calendars within the cubes. Whether that is a fiscal calendar, 4-4-5 calendar or your own retail calendar, Tableau inherits this structure and behaves the same.</td>
<td>Relational databases typically only store dates and not different calendars. In Tableau, a user can define a new fiscal year start, or write date calculations to mimic a 4-4-5 or retail calendar.</td>
</tr>
</tbody>
</table>
About Tableau

Tableau Software helps people see and understand data. According to IDC in its 2012 report, Tableau is the world’s fastest growing business intelligence company, Tableau helps anyone quickly analyze, visualize and share information. More than 10,000 organizations get rapid results with Tableau in the office and on-the-go. And tens of thousands of people use Tableau Public to share data in their blogs and websites. See how Tableau can help you by downloading the free trial at www.tableausoftware.com/trial.

Tableau and Tableau Software are trademarks of Tableau Software, Inc. All other company and product names may be trademarks of the respective companies with which they are associated.