Designing Efficient Production Dashboards

Written by Ben Bausili
Additional Research by Mat Hughes

Contents

About This Document

Sections:

Introduction

Why do we care about efficiency?

Tableau use cases and essential takeaways

Talking to users about performance

About This Document

This document summarizes work from multiple people across the Tableau community, InterWorks, Inc, and the landmark work of Alan Eldridge in the Designing Efficient Workbooks whitepaper from 2016. Many of the insights found here are from years of free community education I've benefited from during Twitter conversations, blog posts, videos, Tableau community groups, and of course, the Tableau Conference. The Tableau community is and remains the most remarkable thing about this fantastic tool. Thanks to all of you who have made Tableau welcoming and for contributing knowledge and guidance so willingly. A special thanks to Mat Hughes for helping lead testing utilizing **Scout** to validate recommendations.

This document addresses Tableau features through 2021.1. Future releases of Tableau may change some of the recommendations or open up new optimization methods.

Introduction

This paper focuses on helping Dashboard authors build and maintain their workbooks efficiently in production environments. More specifically, on the methods to deliver efficient workbook performance to your end-users, from the first user request to the final page load. We'll also consider the author's time and energy in creating and maintaining dashboards. If you're looking for troubleshooting help, feel free to skip to the troubleshooting flowchart at the end of this document. Dashboard authoring covers a broad scope of material, but this paper will limit itself to the areas the average dashboard author will have the ability to change. As such, where other parts of the data or Tableau ecosystem come into play, the goal of this paper will be to point in the direction of additional resources and help facilitate conversations with others, such as with your Tableau Server admins. This paper will not discuss the environmental factors such as network throughput, disk I/O, architecture, or infrastructure constraints such as VM oversubscribing. If you are experiencing widespread performance issues in your Tableau environment, consider seeking **consulting services** or an **in-depth server assessment**.

Why do we care about efficiency?

You never get a second chance at a first impression. Load times are one of the very first aspects of our dashboards visible to the user. We can begin to lose our audience before they even begin to experience our dashboard. That negative impression can color the rest of the experience, no matter how well thought out. If our goal is to deliver answers to that audience, capturing and keeping their attention matters a lot. In my experience, most performance problems that the dashboard user encounters are due to mistakes in design.

By Building dashboards with a view towards efficiency, we can fix these mistakes and deliver fast, flexible, and compelling dashboards. We care about efficiency because we care about the user. We want to respect their time, and we want the experience to be impactful. That experience has a few key characteristics:

- Delivery of answers fast
- Keeps the user in the "flow" of analytics: focused on their questions, not the tool
- Flexible enough to answer multiple questions, reducing the need to create and maintain numerous dashboards
- Easy to understand, interpret, and iterate upon
- Responsive (or perceived responsiveness)

Of course, Performance and efficiency are complicated topics, and definitions vary across people and organizations. There isn't a single acceptable load time that fits everybody. Some groups are willing to trade performance for features, while others pursue speed at all costs. This paper will focus on the technical perspective, looking at the choices you can make as an author that will often be more efficient than the alternatives. Still, even the technical aspects involve many complex systems interacting in complex ways. There will rarely be a single fix for performance, and not all recommendations will work best 100% of the time.

Most of all, it's important to remember the user in all of this discussion. As the dashboard author, you need to understand the bigger picture of your project and where the user fits. This paper may give you the best way to implement dashboard requirements, but you determine the requirements. There will be times when you have to choose between performance and feature requests; keeping the user's needs in mind will help you make the right call. The user, after all, is why we care about efficiency in the first place.

Tableau use cases and essential takeaways

As mentioned before, the focus of the paper is on building and maintaining efficient production dashboards. However, production dashboards are just one of the use cases for Tableau, and often the dashboard journey has much humbler origins as ad-hoc analysis or rough prototyping. Let's take a look at the three main analytic use-cases for Tableau and some key efficiency takeaways for each. If you're in a place where you're looking for some additional quick insight, a good place to start is the **flowchart** at the end of this document.

Accelerating the cycle between question and answer with data is one of Tableau's missions. It achieves this by making data a visual experience (through **VizQL**) and embracing the design process as interactive and iterative. As such, it's a fantastic ad-hoc analytical tool, letting the dashboard author ask a question, see the answer quickly through visuals, and then iterate with another question. Ad-hoc reports often turn into quick prototypes that evolve into production dashboards. That journey from ad-hoc, to prototype, to production is a natural one. It also has a way of pulling inefficiencies into the final dashboard product. Features of Tableau that help speed ad-hoc insight, like being able to easily connect to data sources, make new calculations, and add as many filters as you desire, may result in additional workload or complexity if not streamlined for production.

Across all these use cases, there are a few things that will always remain true:

- **If the data source is slow, it will be slow in Tableau.** Tableau extracts are a great solution to this problem. If you must use live data connections, it's best to consult your Database Administration or other IT resources to optimize your data source.

- **If it is slow in Tableau Desktop, it will most likely be slow in Tableau Server.** Many users believe that because Tableau Server has more CPU/RAM/etc., it will be faster than their local PC. This is rarely the case. Tableau server is running multiple workloads for multiple users. Invest your time making sure it's fast on a desktop before contacting your Tableau Server admin about tuning the server.
- **Newer is generally better.** The Tableau development teams are constantly improving performance and usability. Upgrading to the latest release can often help with performance. For production use cases, look for an X.1 release or later (For example, for Tableau version 2020.4, you'd want to be on 2020.4.1 or later).
- **Less is more**. Minimize and focus is the central recommendation. Just because you can have 50 sheets on a dashboard doesn't mean you should. Thoughtfully reducing complexity is the key to outstanding performance.

With those essential takeaways in mind, let's take a look at each stage, the goals of that stage, what efficiency looks like for that stage, and what modifications you can make to smooth out the path to production.

Ad-hoc analytics

The goal of any ad-hoc analysis is exploration. You are approaching your data to reach a better understanding. As such, there are very few rules and guidelines on how to approach ad-hoc analytics. Nevertheless, it's important to remember that the goal is not application efficiency but speed to insight. Premature optimization leads to a lot of wasted time. To speed up this stage, remember:

- **Avoid premature optimization.** It's tempting to spend time immediately looking at making things fast, but often it can have unintended side effects. For instance, it might be tempting to summarize your data to create a smaller, faster dataset. In production, aggregating data is a powerful tool for performance, but early-on aggregating can hamper your exploration, causing you to spend time modifying your dataset to better answer questions.
- **Don't worry about formatting.** Everybody loves a beautiful chart but leave the fine-tuning until you know what charts you want to keep and share. Generating charts without worrying about design helps you explore the data rapidly.
- **Make use of Tableau's duplicate sheet capability.** Tableau's iterative process makes it easy to continue to modify and change charts from one visual to another. With infinite undo and redo, it's also easy to explore without fear of permanently messing things up. One trick, however, that I've found indispensable is to duplicate the current sheet anytime I've created a helpful chart. It's an easy way to save a trail of insights as you continue to iterate and explore.
- **Follow good naming conventions and other best practices.** As you create lots of calculations and visualizations, it's easy to allow the default naming to remain and may even seem less efficient to take time to rename your assets. However, quickly giving meaningful names will help you in the long run and make your workbooks legible to you and others. Not only that, but you'll have an easier time leveraging your ad-hoc work in production builds if it follows any naming standards your organization may have.
- **Use production data sources (such as those certified on Tableau Server or Online)**. Leveraging already completed work, especially when it comes to data quality, will help accelerate your time to insight and give you more trust in the answers.

Prototype dashboards

The goal of prototypes is to get user feedback as fast as possible. In this phase, you're defining the core question or questions a dashboard aims to answer and getting it in front of your stakeholders to get feedback. Prototypes are an excellent place to use the charts and insights you generated during ad-hoc analysis.

To speed up this stage, remember:

- **Frame the discussion.** The most crucial part of building prototypes is the people component. Determine what questions you're trying to answer with your prototype and frame the discussion to your test users around those specific questions.
- **Don't build everything all at once.** When given a list of requests, it can be tempting to build out every dashboard or feature for your stakeholders—working in large chunks of work results in long development cycles. Prioritize many small iterations to get user feedback faster.
- **Don't spend too much time on fine-grain details.** It's a good idea to spend time formatting your tooltips to make them relevant and readable, but it can also be time-consuming, especially as changes build-up from user feedback. In the prototype stage, it's good to limit those sorts of modifications to a single example before implementing them more fully across all your dashboards.
- **Use ad-hoc features for adding data structure.** Additional dimensions or hierarchies are often needed. For a prototype, use tools in the reporting layer rather than forcing changes to the source data.
- **Consider using fake data.** Building with actual data is one of the joys of using Tableau. Using actual data allows you to find insights and see how the data will affect layout choices. Sometimes, however, data is unavailable or can result in conversations about the problems with the data (and other known issues) instead of design aspects like layout, color, and user stories. Fake or mocked-up data can help focus the conversation and allow you to explore various states of your dashboard that may not currently exist. A good example is KPI or Scorecard type dashboards, where we may want to explore designs for over or underperforming KPIs.

Production

The goal of production is to deliver insight to your users as quickly as possible. When working on the prototype, you spend time defining the core question(s) to address within your dashboard and finding the best ways to answer those questions. The production phase takes those answers and makes them repeatable, maintainable, and accurate. This process can be deceptively time-consuming in any tool, including Tableau, and is one reason to focus on using the prototype stage to find the right thing to build. Once we know the right thing to make, we can make it right.

One important thing to note is that data volume may be the single most significant factor in performance, and it tends to grow over time as dashboards age. If you're working with small data sizes, many recommendations will not be critical. If performance is already adequate at the prototype stage, focus on things that will help you maintain the dashboard over the long term. However, if you find yourself working with slow data or with data that has grown over time, remember that streamlining performance will always require a series of trade-offs. The following guidelines are things to weigh in your decision process:

- **Focus on the user.** Often we measure performance from the first interaction to dashboard loaded. It's important to remember that true performance is about getting insight to our users. Limiting the number of clicks or reducing the time spent interpreting results may not get us faster load times, but it does create happier users.
- **Extracts solve many problems.** Changing from a live connection to your data source to a Tableau Hyper extract will make most workbooks run faster. If you're not running extracts, this is the first place to start, even just as a test to rule out other data issues.
- **Simple is fast.** When looking at slow workbooks, there's almost always complexity that doesn't serve the user's needs. Too many charts, filters, marks, calculations, or rows of data can have your dashboard working harder than it needs to.

- **Limit the data.** When you are exploring or authoring, you often want to look at all the data, but as you move into production, it's best to only publish with the data you need. Each unneeded row adds additional processing overhead that you just don't need. Consider adding a data source filter, dropping unnecessary columns, or aggregating details to the appropriate level. Do you need hourly or daily data? Do you need years of history? You may, but often you don't; in those cases, summarize the data before using it in Tableau.
- **Show the essential.** Production dashboards should be focused and only contain the necessary complexity. Consider ways to reduce the number of filters, marks, sheets, columns, and rows of data. Ask yourself questions like "how will users make decisions with this dashboard?" and "What time frame is important?". Do user testing and measure the actual use of features.
- **Move complexity to the data layer.** While exploring data or building prototypes, you did a lot of things to prove the concept. You used features such as Level of Detail calculations (LOD), string calculations, window calculations, custom SQL, or maybe even an external service. All of these may add additional processing complexity to dashboard load, but we can often negate this. Pre-compute the data and calculations either in an extract or your backend data source. By moving this additional processing to the data layer, we do the processing before the user even requests the dashboard.
- **Use extracts wherever possible to accelerate performance.** Tableau gives you lots of built-in tools to achieve the above recommendations. Hide unused and confidential fields. Roll up data granularity by pre-aggregating or filtering. Break hierarchies to only visible dimensions.
- **Fixed dashboard size.** Determine the screen size or sizes that you want to deliver to your end-users and build towards that. Automatic sizing is less efficient than exact dashboard size.
- **Don't forget about Tableau Prep.** Tableau Prep provides you another way to work with your data and make it fast and focused for your dashboards. Do calculations, aggregate data, and reshape it to best suit your needs.

Take a step back and ask if what you're trying to do is unrealistic. I once had a client trying to work with live data from across the globe. Processing billions of rows of data and transmitting it across the ocean will take more than a few seconds. You may wish to bypass the laws of Physics, but I can assure you they won't cooperate.

Areas outside of Tableau's primary use-case

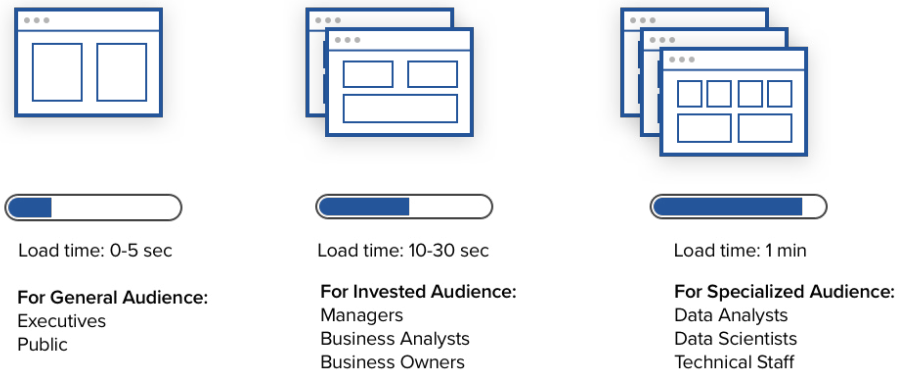
Tableau is an easy-to-use tool. It's so easy that sometimes it's tempting to use it for things outside its core focus. Users can and often do these things, but it's important to note that Tableau may not be the best solution or at least recognize the trade-offs you're making when you go down one of these routes. Consider revisiting your requirements or taking a different approach if:

- **Exporting data for production use cases.** I've seen a dashboard underperform too often because of a large crosstab designed to allow a quick CSV or Excel download used for other business processes. It makes sense, Tableau makes it easy to query data and provides built-in export capabilities. However, there are more efficient solutions for this ad-hoc ETL (extract-transform-load) process, including Tableau Prep or 3rd party query tools.
- **Building spreadsheet style views with highly complex formatting,** such as P&Ls, balance sheets, etc. The Tableau community has many tips and tricks for getting desired results, but the performance can be lacking, especially on large datasets. It will better serve many end-users to leverage Tableau's **View Data** capability or **Web Authoring**. Also, consider leveraging some of the available **Tableau extensions** that provide capabilities such as spreadsheets and write-back.
- **Authoring pixel-perfect designs,** such as for complex print layouts. The Tableau community and Tableau Public are filled with eye-catching, beautiful designs and are fully capable of doing multi-page reports. However, other tools may provide you with better formatting control if you need to do complex layouts.

Talking to users about performance

I know many of you are already asking yourself how to convey this information to your end-users. Knowing how to build efficient dashboards can sometimes feel very different from meeting stakeholder requests. There will always be tension between feature requests and focusing a dashboard on the essentials. Two key concepts that I've found helpful to this conversation with stakeholders are performance budget and technical debt.

When thinking about budget, it is vital to keep your audience's expectations and **use cases** in mind. Certain audiences will expect quicker dashboards that get to the point. Others will expect more exploration capability and functionality at the cost of speed. Here's one way to look at a typical production environment:



Performance budget

When talking about efficiency and performance, putting it in terms of a budget can be helpful. We deal with budgets in business all the time. Performance is no different. If we're aiming for a 5 second load time, we only have so much time (and processing power) that we can leverage. Each feature we pile on to the dashboard is eating away at that budget. In fact, in a cloud-first world, the time and compute power needed for a dashboard to load can be directly tied to real-world infrastructure budgets. To be effective and make the most of your budget, you should:

- **Establish a "good" load time.** That target load time is your budget and will be different depending on the situation. For some companies, it's 5 seconds for every dashboard; for others, you may have groups willing to wait because of complicated business requirements. Setting this number determines your budget.
- **Frame requests in terms of impact to the budget.** Help stakeholders and users understand the impact of new features and the potential benefit of removing others. You can use this paper as a guide for understanding relative impact, but **running performance tests** can give you more concrete numbers to discuss.
- **Take a performance recording of your initial design.** Once your initial dashboard design is locked in, a performance recording can establish a baseline for future change requests. Many production dashboards start efficient and slowly change over time. Having a baseline to refer back to can help understand the impact of changes and better discuss their relative benefit.

We'll talk about performance recordings later, but helping your stakeholders and users understanding the impact of requests can help you have a more productive discussion and deliver a better result for everybody.

Technical debt

The second concept I find helpful is Technical Debt. In short, it's a concept from software development that there is often a long-term cost to short-term decisions. For example, choosing an easy but limited solution now can cost more than using a better approach that takes longer when you add up the time and resources spent over the entire lifecycle. Take, for instance, a production dashboard that will be in use for several years. It can be a significant time-savings to quickly push a prototype in production, neglecting documentation, using insufficient naming, and complicated calculations. Over the years of production use, however, the long-term costs (in maintenance, performance, and loss of knowledge) will outweigh the short-term time savings. As with all debt, if not repaid, it accumulates interest that builds over time. As it accrues, you need to be aware of and actively manage your total debt. Think of the situation of handing off responsibility for a project to a new employee. When documentation isn't up to date, the amount of time that the new employee spends to understand and reverse engineer the project can be substantial.

Technical debt shows up in big and small ways. It can be as simple as a good naming convention but can be as big as making sure you have good data pipelines. Business Intelligence groups will spend time shaping and cleaning data to make it useful for reports. This work can be a source of significant technical debt if not prepared for long-term maintenance. Moving this data work from bespoke department tools to your IT group's standard tools and processes can significantly improve the long-term stability and reliability.

I'm not saying all debt is bad. In many cases, when building prototypes, it's fundamental that we make tradeoffs for speed of development. When we're talking about production use-cases, we must manage our debts and consider the entire lifecycle of our project. In particular, when talking with stakeholders, technical debt can be a valuable metaphor for explaining why we must invest time developing the right way.

Examples of Tableau-specific technical debt include using calculated fields to derive dimensions, blending across multiple data sources, manually implementing row-level security, or even just poor naming conventions. Often these things make sense for a prototype or test, but once in production, they consume time and resources from us and those tasked with supporting the production solution. Establishing a Tableau-centric and organization-wide **governance model** can help manage the technical debt related to Tableau by determining when and how to promote content to production.

Summary of Performance Factors

Sections:

The Anatomy of Performance

VizQL and how Tableau Works

Summary of Performance Factors

With the high-level recommendations and concepts in mind, let's dive into the details of production dashboards. We'll start by detailing each of the various components from the source data system to visuals to the dashboard layout before diving further into the methods to make each efficient.

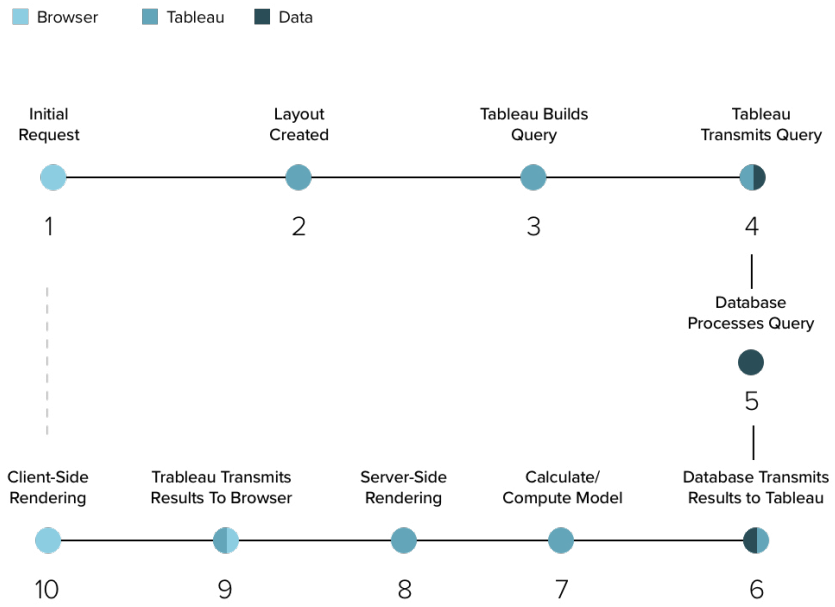
The anatomy of performance

A Tableau Dashboard has four core elements: Data, Calculations, Worksheets, and Dashboard Layout. We start by connecting to our data, enhancing it by creating calculations, using Tableau's interface to query and visualize the data, and finally building out our dashboards. The path is rarely linear and has many iterations, but these are the elements we control as dashboard authors that impact efficiency. These elements provide the structure that the rest of the paper will follow. Before specific recommendations, let's examine the processes behind each of these elements.

When your end-user decides to load a dashboard, Tableau writes queries processed by your data source. Tableau is, at its core, a visual query generator. Much of the work Tableau does is translating your designs into queries your underlying data source can understand. Those query results are then used by Tableau to perform additional calculations on the data (such as window calculations) as well as the internal calculations Tableau may need for your dashboard. Your worksheets/visualizations are rendered based on the query results and calculations before being placed in their final layout. This process, when broken down, can be summarized into just four components that impact performance:

- Query Time
- Calculation Time
- Rendering Time
- Layout Computation

We impact each of those components with the elements of our dashboard—data and calculation complexity impact query speed. Calculations and worksheet design affect the calculation and rendering time. Dashboard and worksheet design impact layout computation time. If you want to dig a bit deeper into the actual VizQL process, follow me to the next section to look at VizQL or skip ahead if you want to dive right into our recommendations.



VizQL and how Tableau works

Tableau does the work of translating design into queries through the VizQL Server. When a dashboard is requested, VizQL must first determine its layout. In the browser, we call this the DOM or Document Object Model. To figure out the layout, VizQL must parse the Tableau Workbook file (TWB) for your dashboard. Tableau stores this file in its repository as a binary large object (BLOB). One important implication is that Tableau has to pull the entire dashboard file to render any single dashboard in it. Once the TWB file is pulled into memory, VizQL will parse it. The TWB file defines your dashboard with multiple XML elements such as window, dashboard, and worksheet. The bottom line is that workbook size matters. Items that will often contribute to a large TWB file that takes longer to process:

- Dashboards with many elements (containers, worksheets, filters, features)
- Many dashboards or worksheets
- Large image files
- Multiple layouts
- Datasources with a high number of fields
- High-cardinality fields used for colors or filters

Once VizQL has parsed the TWB file and located the correct XML elements, it's ready to compute the dashboard layout and then query the data. Many customers I've worked with imagine that Tableau is retrieving the data via one large query, but in reality, Tableau is often generating multiple small queries. A complex dashboard can result in a large volume of queries. The impact this has on performance depends a lot on your underlying data source.

Tableau will do its best to combine queries using Query Fusion and run queries in parallel. Tableau will also create temporary tables and leverage materialized views for additional performance gains if it is able. Once the limit of parallelism has been reached (by either Tableau or the data source), queries will queue up and wait for other queries to finish. Once a query has been completed, and the results have been streamed back to Tableau, the next query will begin. In practice, that means that even small, sub-second queries in enough numbers can block the dashboard from rendering and noticeable load times. Once the queries finish, Tableau can move on to rendering the visualizations. Keeping in mind how vital VizQL and the queries it generates are to performance, let's dive into how you can achieve good performance.

Roadmaps to Good Performance

Sections:

Data Source Best Practices

Calculation Best Practices

Worksheet Best Practices

Dashboard Best Practices

Optimizing Existing Workbooks

Gathering Data On Performance

Roadmaps to Good Performance

The road from raw data to a final dashboard is marked with many decisions. This section gives you the best practices to keep in mind on your journey as you create your data source, enrich it with calculations, transform it into visualizations, and bring it all together in the final dashboard. The later chapters of this paper will tackle the performance of each of these phases in more depth, but if you just want high-level advice, you're in the right place.

Data source best practices

A few recommendations apply broadly to all data source and model types, including doing proper data prep, summarizing data, excluding unused data, and using Hyper when possible. For more detailed advice and insight on building dashboards, see the **Making Data Sources**. For now, let's break down our general advice:

Data speed is key

Your visualizations can only perform as fast as your underlying data sources. That is the giant roadblock that can turn great dashboards into headaches. Many sources have unique optimization strategies, such as database indexes. You can consult your administrators to discuss possible options, but one technique that always works is limiting the size of the data. Smaller data means less info to process and transmit. Use only the data that is needed at the grain necessary for the worksheet to perform its analysis.

Use native drivers when possible

As of writing, Tableau has over 80 native connections to data sources, besides general-purpose ODBC and JDBC. A native connection benefits from Tableau's engineering, implementing specific techniques, capabilities, and optimizations. For the best performance, stick with native drivers. When a native driver isn't available, you can turn to JDBC or ODBC if your data source supports it, or check the **extensive list of web data connectors** for additional options. Using a Hyper Extract will help performance in these cases, but also ensure you have access to the full suite of Tableau features which can sometimes be limited by generic drivers.

Use Hyper extracts when possible

Hyper is a powerful tool for following best practices. Leverage data source filters, aggregating to visible dimensions, and hide unused columns to shrink the size of your extracts and accelerate your dashboards. When your Tableau extract is embedded in your Tableau workbook (vs. being published to the Tableau data server), Tableau also does further optimizations making traditionally slow elements of your dashboard faster, such as Filters set to show only relevant values. With a focused extract, you'll be well on your way to a fast-loading dashboard.

Optimize and materialize your calculations

In production, perform calculations in the database when possible. This provides multiple benefits, including centralizing the logic to be used by multiple Tableau data sources and authors, but it also has significant performance benefits. When a calculation is created and stored in the database, you're reducing overhead in Tableau. This is especially the case for row-level calculations that do not need to be done by Tableau when a user requests the dashboard. Aggregate calculations are an example of something that often needs to be done specific to a user's requests and should be calculated fields in Tableau.

Calculation best practices

Calculations are where we can control much of the complexity and speed of queries to the underlying data source. Reducing that complexity by materializing calculations or doing it in advance in your data source are two of the most effective ways to achieve good performance. For more detailed advice and insight on calculations and functions, see **creating calculations**. On a high level, when looking at your calculations you should:

Materialize your calculations

Calculations in Tableau typically run when a user requests or interacts with a dashboard. If we can do the calculation before the user arrives, we'll save them time and deliver the dashboard faster. There are a few different ways you can achieve this speed boost. The easiest is if you are using a Tableau Hyper Extract. As mentioned in the data section of this paper, Tableau will materialize your calculations (i.e. calculate them and store the results in the Extract) for you when you create your extract or if you ask it to by going to the Data menu in Tableau Desktop, selecting your data source, and then Extract > Compute Calculations Now.

On a live connection, you can perform calculations in the data source. Row-level calculations can easily be added to a database view or table, but in some cases you may be able to perform aggregate calculations as well. Look for LOD calculations or Window Calculations that aren't going to change depending on user interaction as potential candidates to add to the data. Due to the numerous ways to perform and store the calculations in your data source, you may need to talk to the appropriate administrator about the best methods for your situation.

Reduce the granularity of LOD or Tableau calculations

In general, when you aggregate more it'll be faster. The closer the calculation is to each individual row of data, the slower it will perform. For LODs, look at the number of unique dimensions members in the calculation and reduce if possible. For Tableau Calculations, the more marks you have in the view the longer it will take to calculate. A lot of "Tableau tricks" online cleverly use LODs and Table Calculations in ways that work on smaller data sets, but can result in long loads on larger, production data sets.

MIN and MAX are faster than ATTR and AVG

In a lot of visualizations, there is a need to display a column that contains duplicate values and would display the same result for AVG, MIN, MAX, and ATTR. In those cases, MIN and MAX will be computationally faster and is a better choice if you're sure there are not multiple values present. For ATTR, Tableau calculates both a MIN and a MAX calculation, so choosing one of those reduces the amount of calculations performed.

Make groups with CASE statements

One sometimes surprising best practice is to create calculations that use CASE statements to group items instead of Tableau's built-in group functionality. The built-in grouping loads the entire domain of the dimension, whereas your calculated field only loads the named members of the domain. In our testing, grouping methods performed in the following order (from best to worst):

1. CASE Statements
2. Tableau Sets
3. Native Group function
4. If / Else If Statements

Use IN instead of OR

In Tableau 2020.3, the IN operator was introduced. This not only gave us a way to clean up syntax of conditional statements using OR, but also a way to make it even more performant. Take for example this statement:

This statement will return TRUE if Project contains the values "Project 1" or "Project 2" and FALSE otherwise. From a performance perspective, when a data source is passed a series of OR statements, it typically evaluates each condition individually before resulting the entire statement. This makes sense, given these logical comparisons often can have multiple types of comparisons within them. If we use IN instead, it can be assumed that we're checking a single dimension against a list, which is a much simpler task from a performance perspective. This allows us to have very clean calculations as well as a performance. Translating the previous example to using IN results in the following:

Reduce the number of nested calculations

Creating calculations built upon other calculations can often happen as you iterate and validate your work, but each layer can add complications and additional processing. This is especially true when you layer IF statements or other performance intensive functions. This doesn't mean to consolidate all your calculations into a single block as readability and your ability to maintain calculations overtime is important. Focus on situations with many layers and intensive calculations (such as conditional statements). Also, keep in mind that pushing the calculations to the data source or materializing within a Tableau extract will resolve the performance issue with nested calculations without further rewrites.

COUNTD is slow

There's a lot going on in this simple calculation to ensure it only counts the distinct items in your list. Whenever you can, use COUNT, [Number of Records], or in the new object model you can use the CNT function that allows you to count the rows of a single table.

Materialize string manipulation in your data

Sometimes you need to use strings in calculations. Find, search, replace, etc are all comparatively expensive calculations. You'll see better performance by making sure your data has the string values you need to use already in it.

Materialize date conversions in your data

When building for production, it's important that you minimize the amount of date calculations and conversion you have to do in Tableau at run-time. If you do have to do them, leverage DATEPARSE and MAKEDATE before other methods.

Don’t worry about string values

It’s been widespread advice in the past to use integers or booleans instead of strings for results in calculations (or for parameter values). This may have once been solid advice based on common computer programming principles, but Tableau developers have done a lot of optimization over the years to make this old advice irrelevant. In testing the results of using strings vs. integers or booleans has been a toss-up at best and sometimes strings have been faster. Use strings wherever it makes sense and is clearer for others making sense of your work.

Worksheet best practices

Worksheets are where we can control much of the rendering that Tableau will need to accomplish. Tableau has amazing rendering capabilities, but the number and complexity can quickly add up across multiple visualizations. Also keep in mind that filtering visualizations have the added benefit of reducing both the amount of marks to render as well as the size of query results. For more detailed advice on worksheets, see **dashboard best practices**. When building your worksheets:

Visuals are better than text

Tableau has been optimized to do visualizations. Yet, it is not uncommon to see dashboards that are just a table of raw data with a long list of filters on the side. These views struggle with performance as it loads lots of unseen data (only available after scrolling) and includes more details than your users probably need. In many cases these exist to allow users to extract data to another tool like Excel for further analysis. It’s not a good dashboard, however, and the experience is lackluster.

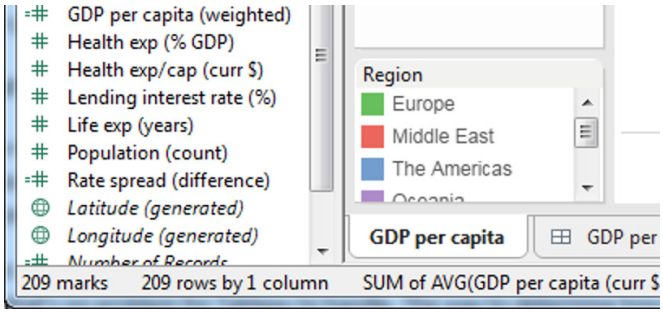
When working with clients, this often indicates a lack of understanding of end-user needs. It’s “one dashboard to rule them all” that provides everything as a catch all. It expects the user to do the hard work to find their insights. You’ll get better performance for your users if you give them a way to step through and drill down to the data that answers their questions. A good dashboard will:

- A clearly defined purpose and audience
- Share data in meaningful and insightful ways
- Use the knowledge of human cognition to show the “best” view for users
- Leverage informative yet concise visuals
- Leverage **pre-attentive attributes**.

- You can learn more about these sorts of best practices in many locations, including in Tableau's Blueprint article, **Why Visual Analytics** as well as **Visual Best Practices**. The good news is if you follow many of these guidelines, you'll end up with fast and beautiful dashboards.

Avoid high mark counts

A mark is any point, plot, or symbol on your visualizations. They can be the bars in your bar graph or the points in your trendline. In large text tables each text element is considered a mark. You can see the number of marks that Tableau is rendering by looking at the bottom-left corner of Tableau Desktop. The more marks the more work Tableau is performing rendering your visualization.

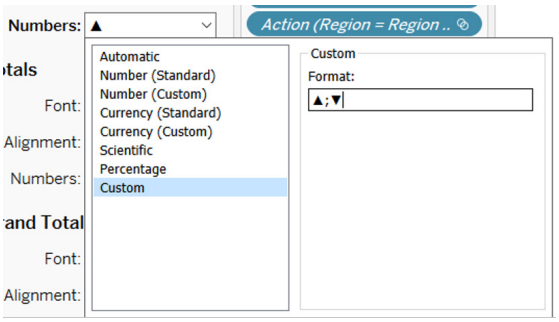


It can be tempting to show every data point in your dataset and Tableau is certainly capable of handling millions of marks at a time, but when considering performance, it's important to ask if it's necessary to convey what you want from the visualization. In some cases, the answer will be yes, but in most you can likely aggregate to a higher level or use some additional tricks. Look to filter the marks or break out the results into different worksheets or views for better visibility into the data.

Two very effective methods that can make for both strong visuals as well as faster ones are Density Marks and Hexbins. **Density Marks** in particular require just a change of the mark type and create a more streamlined visual that indicates the number of marks in an area with color and limits the number of individual items Tableau will have to draw on the screen.

Consider using text formatting instead of calculations and shapes

Sometimes you need to convert a numeric value to a KPI indicator of some sort, such as ▲(up) or ▼(down). A neat trick is to use Tableau built in custom formatting to display these marks and you get a performance boost by skipping another calculated field and assigning it a shape.



Minimize the size of images

Visuals, such as custom shapes, can help the readability of your visualizations, as well as giving it visual flair. Like any type of mark, using lots of images can mean more work for Tableau. For images, however, there's an added issue of the file size. Since you're providing the file to Tableau, it's possible for these files to be larger than they need to be. Our suggestion is to keep images below 50kb when possible with a 32x32 pixel count. Keeping file sizes down will also help in other ways, such as publishing and export. If you commonly export Tableau reports to another medium, such as PDF, these custom shapes and images can substantially increase the size of your document.

Use transparent background PNGs instead of JPGs

Another way to ensure smaller file sizes for your custom shapes and images is simply to use PNGs. PNGs support transparent backgrounds, which means less to store. It also can make the images more useful since they will work on multiple background colors without introducing a box around your shape.

Avoid polygon marks

Polygon marks are powerful in that they let you define custom shapes and paths in Tableau, really opening up the potential visualizations you can create. However, any visualization that uses polygon marks will for Tableau Server or Tableau Online to use server-side rendering, which can impact the end-user experience. More on client-side vs. server-side rendering in the server section.

Use the Page Shelf sparingly

Some dashboard authors believe the page shelf has the same performance as the filter shelf due to their similar appearance. However, when you use pages, Tableau will query for all the pages at once. The page shelf does not reduce the data. Stop and ask: would the view perform well if I added the dimension value on the page shelf to the display.

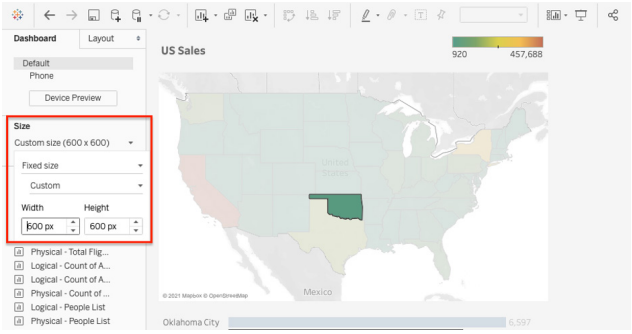
Dashboard best practices

Dashboards are where it all comes together. If you've followed best practices with your data, calculations, and worksheet design, you'll be in a strong position to create well performing dashboards. In a Tableau dashboard, we're typically bringing multiple visualizations together and tying them together through actions, filters, buttons, and other elements to create the final product our end-users experience. The primary way we control performance on the dashboard level is by limiting the number of elements (worksheets, filters, containers, images, etc).

Dashboards perform best when they are designed for a clear purpose, with focused content. Keeping the number of visualizations and filters to the essential is the number one control you have over your dashboard's performance. Remember that everything we've discussed is also still at play. The number of marks being rendered matters to the individual visualizations, but the number of marks across all your visualizations on a dashboard matters even more. A visualization may perform fast on its own but combined with many others can result in a slower dashboard. If you have a complicated chart with many marks, it may be best to keep that the primary focus of a dashboard. When building dashboards:

Use fixed sized dashboards

When you design your dashboard, you can choose from a few different options, including setting fixed dimensions or letting one (or both) dimensions fit the end-user's screen. When you choose fixed dimensions, Tableau is able to better cache the results since it is a predictable size.

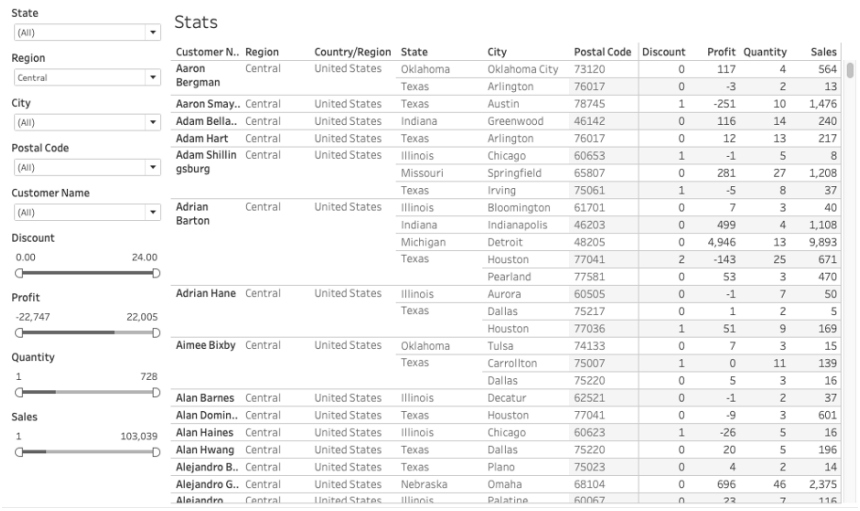


Reduce the number of views per dashboard

Every visualization you add to your dashboard increases the number of queries to execute and marks to render. It can be tempting to create a single dashboard with all your visualizations but breaking them up across multiple dashboards will result in better performance. If you have more than 5 visualizations on your dashboard, begin to consider how to break apart your analysis.

Reduce the number of filters

All of our **previous guidance on efficient filters** is important to follow, but when considering the design of your dashboard the greatest factor in performance is the number of quick filters that you are providing your users. Adding filters increases rendering and query complexity. This is especially true if they have a high cardinality (a large number of unique values) or are set to Show Only relevant Values. A good target is 3-5 filters. If you have more, consider the actual needs of your users and alternative ways to deliver them the functionality they desire.



Drill into detail

Sometimes you need to give your users a lot of details or even the raw data. In use cases like this, it's best to think about how you can keep all the details from being pulled on initial load. Many dashboards suffer from long load times because of the amount of data that needs to be loaded by these granular views, but never end up being used by the end-user. Consider using actions, global filters, or putting these detailed views in their own purpose built dashboard.

Hide Worksheets until Needed

When you float a container in Tableau, you have the additional capability to **add a show/hide button**. Tableau utilizes "lazy loading" which means that whatever is in the container when hidden will not be rendered. This gives you additional ways to provide drill-downs, alternative chart views, or other creative uses that won't impact the render time.

Remove unneeded device specific dashboards

Tableau gives you the ability to create device specific dashboards so that you can better serve your users on phones or tablets. Tableau will even automatically generate a mobile version of your dashboard by default. If these device specific dashboards are not required, however, removing them will create a smaller TWB file and help your dashboard load faster.

Clean-up containers

Containers in Tableau help us align and group content in our dashboard. Often during the process of design and iteration, numerous nested containers end up in our dashboard. Cleaning up the design will uncomplicate the rendering and calculation that Tableau has to do when loading your dashboard.

Optimizing Existing Workbooks

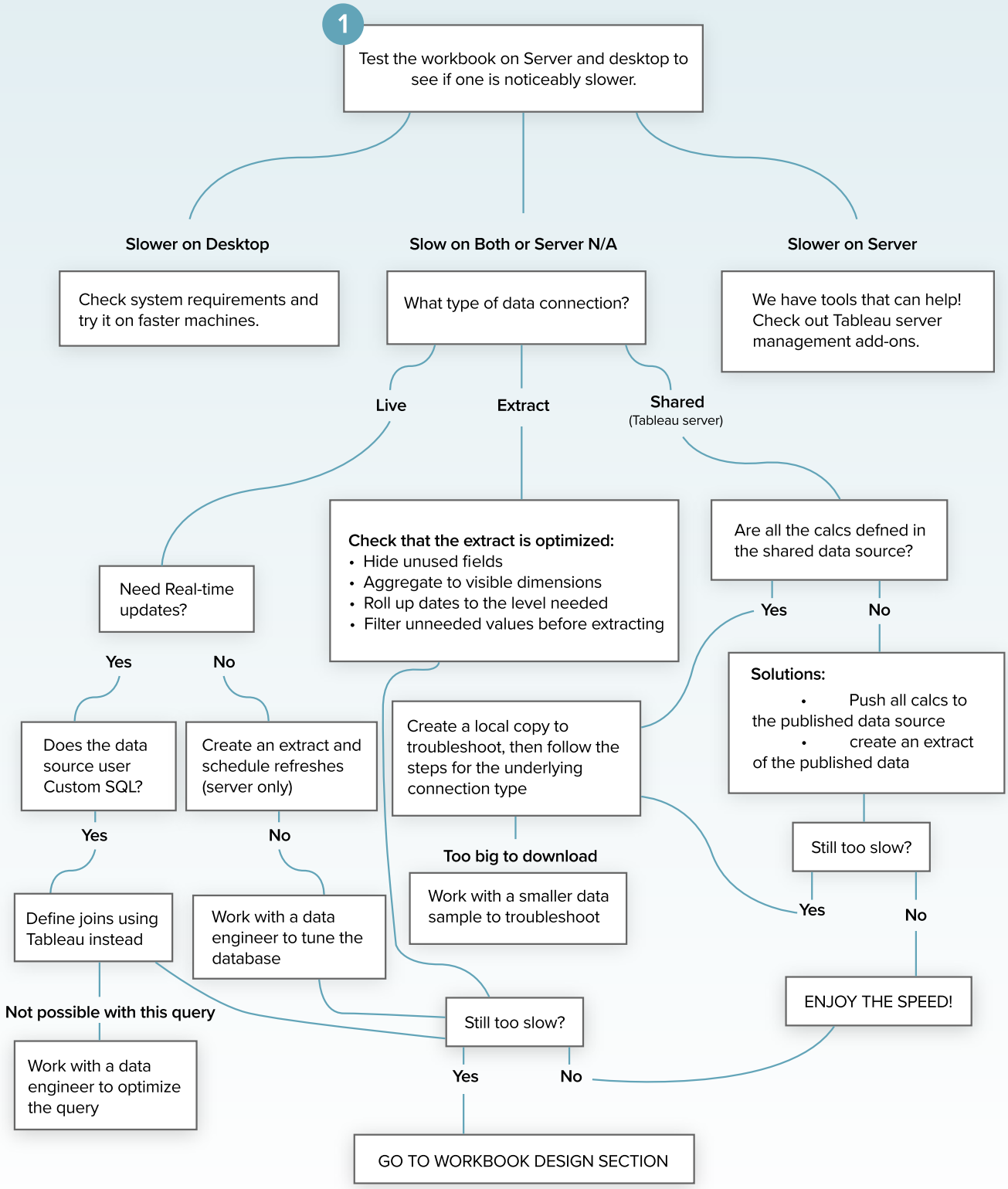
Everyone wants a faster dashboard. What defines "fast" varies from one person to another, so it's important to set a realistic benchmark and consider the time that's worth investing to achieve it. This section provides a flow chart to help simplify the process of determining where to focus your optimization efforts. The key areas that will be focused on determining if there's a strain on Tableau Server resources, issues with Database responsiveness (including Tableau extracts), too much data to render, or design aspects to address. The flowchart is built upon the expertise and battle tested by consultants across the globe and is organized by the most likely areas you'll be able to achieve large improvements. At a high level it shows the thought process of experts for tackling performance problems when you're coming to a workbook for the first time.

(see performance optimization recommendations on next 3 pages)

Performance Optimization

SERVER AND DATA SOURCES

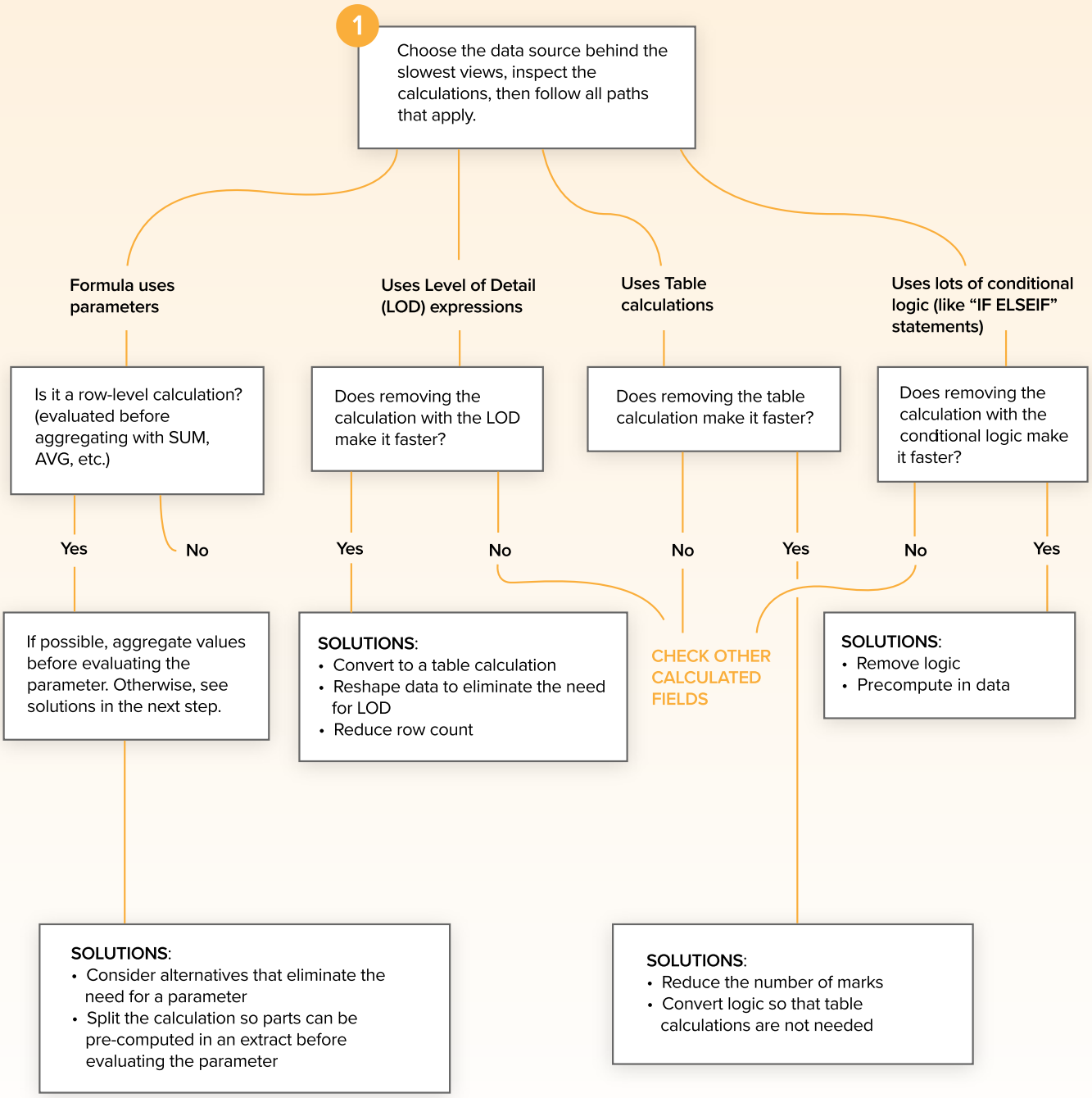
Find the workbook(s) with the slowest load times. Here's how:



Performance Optimization

CALCULATIONS

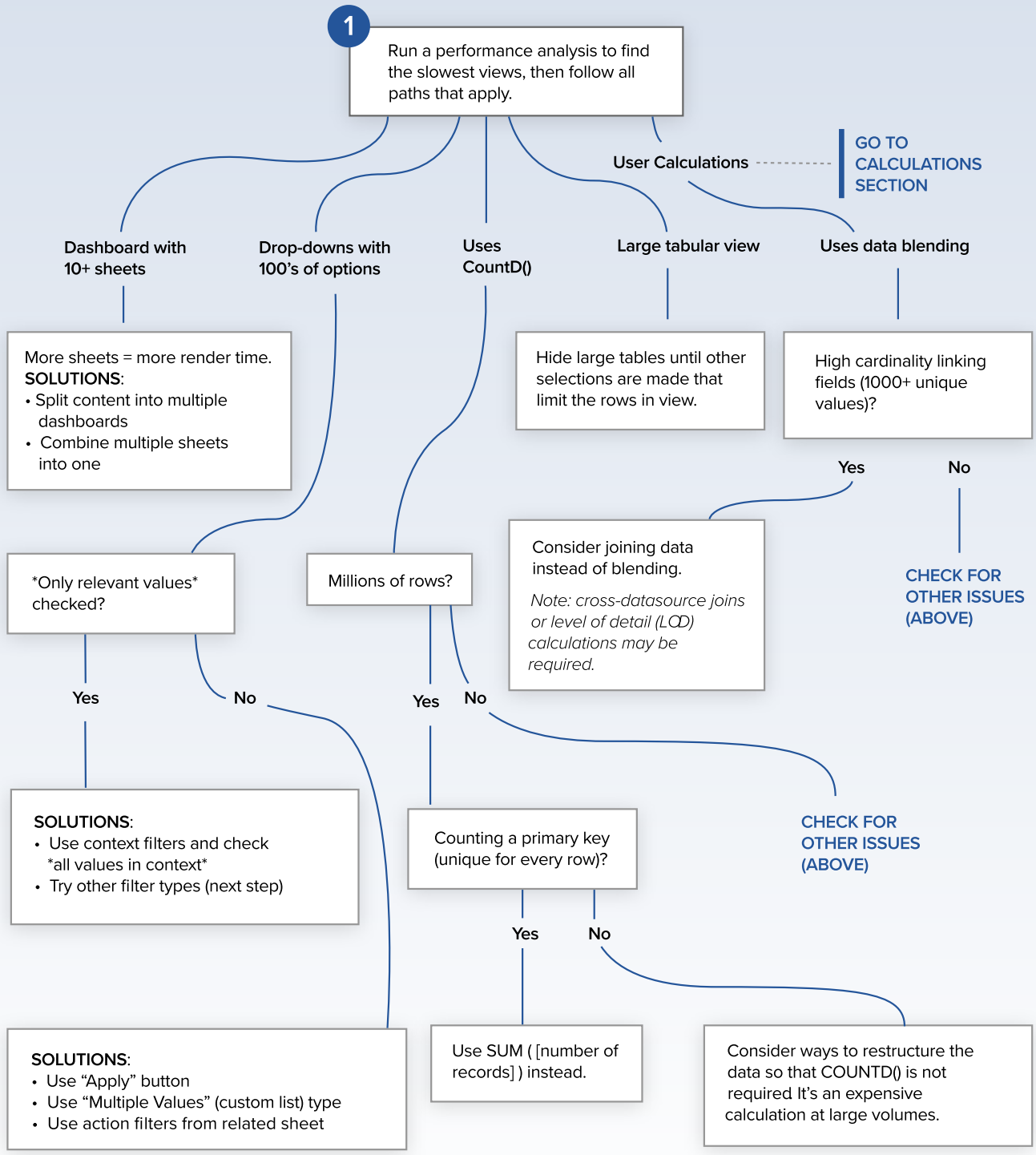
This section assumes you are using a Tableau data extract to optimize performance. If using a live connection, pre-compute as much as possible at the database layer.



Performance Optimization

WORKBOOK DESIGN

This section assumes you have ruled out issues caused by the database or Tableau Server.



Gathering data on performance

The flowchart is an excellent way to tackle big problems when users ask you for help, but when you're really fine tuning a production dashboard, there's no replacement for gathering data to really provide insight on the problem. The best and most accessible way to do this is with a **Tableau performance recording**.

When you start a performance recording, it will begin tracking key events as you interact with your workbook. When you stop the recording, you'll be able to view performance metrics in a workbook that Tableau creates to analyze and troubleshoot different events that are known to affect performance:

- Query execution
- Compiling query
- Geocoding
- Connections to data sources
- Layout computations
- Extract generation
- Blending data
- Server rendering (Tableau Server only)

On Desktop

To start recording performance, follow this step:

Help > Settings and Performance > Start Performance Recording

To stop recording and view a temporary workbook containing results from the recording session, follow this step:

Help > Settings and Performance > Stop Performance Recording

On Server

A server administrator will have to enable performance recording on a site by site basis. To start a recording:

1. Open the view for which you want to record performance.

   When you open a view, Tableau Server appends "*:iid=<n>*" after the URL. This is a session ID. For example:

```
http://10.32.139.22/#/views/Coffee_Sales2013/USSalesMarginsByAreaCode?:iid=1
```

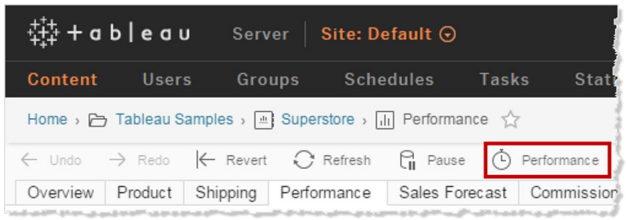
2. Type *:record_performance=yes&* at the end of the view URL, immediately before the session ID. For example:

```
http://10.32.139.22/#/views/Coffee_Sales2013/USSalesMarginsByAreaCode?:record_
perfor mance=yes&:iid=1
```

3. Click the Refresh button in the toolbar.

4. Load the view.

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



Making Data Sources

Sections:

Data Sources in Tableau

Data Model Options

Ways Tableau Can Optimize Queries

Hyper Extract Specific Recommendations

Options for Creating Extracts

Maintaining Extracts

Live Data connections Specific Recommendations

Custom SQL

Shared vs. Embedded Data Sources

Row-level Security

Blending

Making Data Sources

As we've said, Tableau is, at its core, a visual query generator. It takes your designs and translates them into multiple queries to your data source. Well-designed data sources can give your production dashboard a fast foundation.

A lot happens in these queries and is part of the magic of Tableau. It's essential to understand that Tableau offloads as much of the calculation and aggregation work to the underlying data source as possible. This gives you a lot of flexibility in handling the processing through data source optimization and is another reason why **Hyper Extracts** are such an important part of your toolkit for performance.

Data sources in Tableau

Tableau has an extensive ability to connect to data across many different platforms. For the purposes of talking about performance, we're going to broadly group these connections to the following types:

- File-based data sources – such as Excel and CSV;
- Relational database data sources – such as Snowflake, Oracle, Teradata and SQL Server, as well as specialized analytic appliances such as HP Vertica, IBM Netezza, etc;
- OLAP data sources – such as Microsoft Analysis Services and Oracle Essbase;
- No SQL data sources – such as Hadoop;
- Cloud-based data sources – such as Salesforce, Google, etc.

For most connections, Tableau can either query the data live or create a Hyper Data Extract (a .hyper file). When querying live, that means that Tableau will generate queries to the underlying data source as you build visualizations and dashboards. Since data sources come in all shapes and sizes, slow and fast, the performance of your data source will have a huge impact on your dashboard. If your data source is slow, the Tableau Hyper Extract is your secret weapon for performance. In my experience, the majority of environments would benefit from using extracts. A Tableau Hyper Extract is:

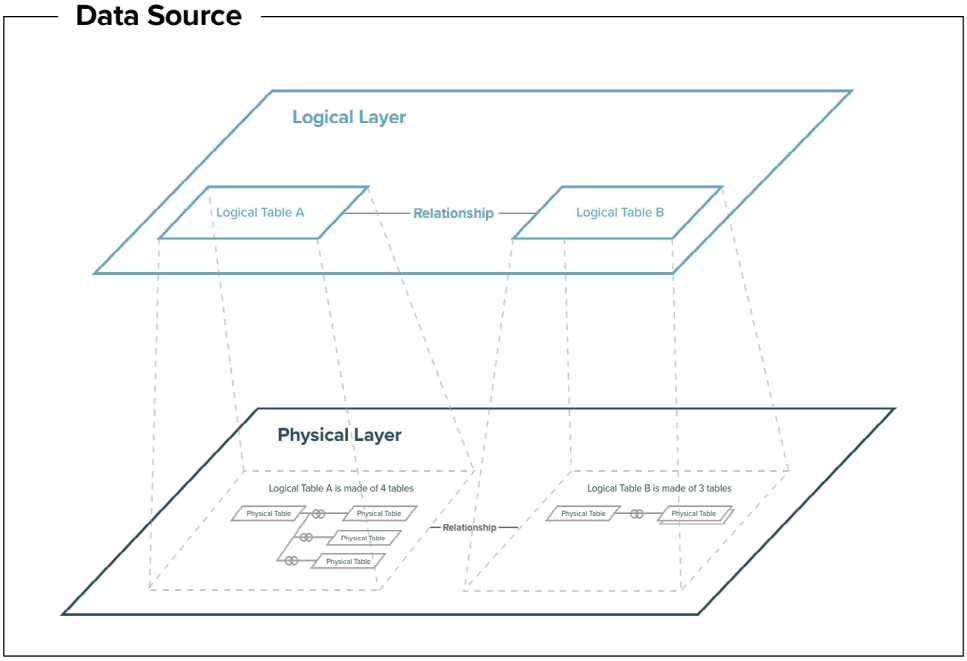
- A persistent cache of data. It is written to disk and reproducible.
- A columnar data store – this is a format of storage that is particularly good for analytics
- Separate from your underlying data source. All Tableau queries will go to Hyper instead of live data connection. This has the added benefit of helping reduce resource contention in your underlying data source.
- Refreshable on a schedule. You can completely regenerate an extract or incrementally add rows of data to an existing extract via Tableau Server and Tableau Online, or do more complex updates via the Hyper API. Most production use cases require full extract refreshes for a number of reasons.
- In-memory and Architecture-aware – Hyper is an in-memory technology but is not constrained by physical RAM available, unlike many similar technologies.
- 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.

If you are using a live connection in Tableau and you suspect your underlying data source could be causing performance problems, creating a Hyper Data Extract is often one of the easiest ways to improve performance. Beyond being a very capable analytic database, leveraging Hyper gives you additional ways to improve the speed of your data that you might not have access to with other systems either because of the nature of those data sources or because of the administrative rights needed. It's also worth mentioning that Hyper is workbook-aware in certain circumstances. When it is embedded with a workbook, Hyper Extracts will automatically create summary tables and other artifacts to help speed up your dashboard. This is a level of customization that would be hard (if impossible) to achieve in other data sources.

Data model options

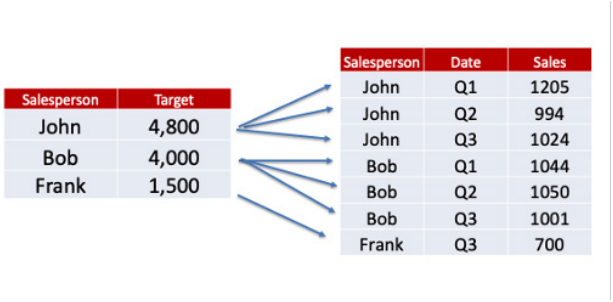
In version 2020.2, Tableau introduced major changes to the ways you can do data modeling. Previously, you defined the connections between tables with joins. These were physical connections that were specifically defined. One way to think about this is that it behaved similarly to if it was all brought into a single table. You can still use this method in 2020.2 and later through what Tableau calls the Physical Layer.

Now, however, Tableau has introduced relationships, which is a more flexible alternative to the existing joins and allows Tableau to query the tables independently. The goal of relationships is to provide an easy, fast and more flexible alternative to using joins. It focuses on how two tables relate to each other, based on common fields, but does not merge the tables together. This relationship is defined in the Logical Layer.



One limitation of joins in the Physical Layer is that they are defined as a row-by-row connection without taking granularity of the data into account. This results in a lot of workarounds when working with data of different grains. Take for instance a sales team. Each salesperson has a target sales number for the month. The granularity of that target is at the level of the person and month. Sales, on the other hand, is at a different granularity and is stored at the level of the moment of sale. If we define a join between these two tables, Targets, and Sales, we end up duplicating our Target data substantially with it showing up for each and every sale. Beyond an increase in storage, this has the result of making analysis harder.

Imagine you want to aggregate the targets to the level of the sales team. With a join between these two tables, if we simply summarize the target, we'll get an unrealistically high number.



Salesperson	Target
John	4,800
Bob	4,000
Frank	1,500

Salesperson	Date	Sales
John	Q1	1205
John	Q2	994
John	Q3	1024
Bob	Q1	1044
Bob	Q2	1050
Bob	Q3	1001
Frank	Q3	700

Figure 1. Joins happen on a row by row basis

Sales Person	Date	Sales	Target
John	Q1	1205	4,800
John	Q2	994	4,800
John	Q3	1024	4,800
Bob	Q1	1044	4,000
Bob	Q2	1050	4,000
Bob	Q3	1001	4,000
Frank	Q3	700	1,500

Figure 2. The result is that we can't safely aggregate the target column

With relationships in the Logical Layer, Tableau can respect the granularity of each table, allowing you as the dashboard developer to ask for a sum of Target and get the result you expect. It also has some significant performance benefits to your calculations and queries. So, when possible, consider using the Logical Layer. You can learn more in **Tableau's Relationship blog series**.

There may be a few cases where the benefits of the logical layer are limited, including:

- Dirty data in tables or the tables are not well-structured. For instance, if your tables contain a mix of measures and dimensions in multiple tables it can make multi-table analysis more complex. In database terms, common **star schemas** or **snowflake schemas** will provide best results.
- Tables with NULLs or lots of unmatched values across the relationship.
- When using data source filters. These provide substantial benefits to speed in the form of data reduction but can also limit Tableau's ability to do **join culling** in the data which allows Tableau to simplify queries by removing unnecessary joins.

With the new data modeling in mind, let's look at how you can optimize your performance depending on if you're using a Hyper Extract or a Live Connection.

Ways Tableau can optimize queries

One way Tableau speeds up your queries is by running multiple queries at the same time. The level of this parallelism varies between data sources as some platforms handle simultaneous queries better than others. Local files like text, Excel, and statistic files are limited to 1 query at a time. For others, Tableau will default to a max of 16 parallel queries. If the results of a performance recording show queries running serially, try taking an extract of your data and rerunning the test. If queries are running in parallel on a Hyper extract, you may want to talk to an admin about the settings or capabilities of your underlying data source.

Tableau will also attempt to eliminate and combine queries. Through the use of query fusion, Tableau will look at ways to satisfy the needs of multiple visualizations with as few queries as possible. To get Query Fusion to apply, visualizations need to be on the same level of detail. Tableau's query optimizer will also sort the queries to run the most complex queries first in the hope that subsequent queries can be serviced from the result cache. One way to get this to happen more frequently is by tying together visualizations with higher level dimensions they may share in common. For example, you may have a worksheet that looks at State level information and another that looks at City. On the City visualization, add State to the level of detail to show Tableau how the two worksheets are related.

Finally, Tableau will try to avoid running queries at all by caching. Caching in Tableau is multi-layered and you as the Tableau author do not have much control over how it is leveraged, other than to set if it's used and for how long. In the Tableau Server section of this paper, there is a section on **caching** if you want to learn more.

Hyper extract specific recommendations

Tableau's Hyper Extracts are a purpose-built analytic data store that allows you to offload processing from your underlying data source to Tableau's Hyper engine. With it, you gain additional tools for fine tuning your data for Tableau consumption that you may not have the permission for in other data sources. For most production use cases, using an extract is recommended due to its numerous benefits, but even if it doesn't work for your particular situation creating an extract is a great step for troubleshooting performance by comparing its performance to your live data sources performance. Let's take a look at the ways you can get the most out of your Tableau Hyper Extracts.

Reduce the size of your extract

The biggest impact thing you can do is to focus your Extract to the exact needs of your workbook. Thankfully, Tableau gives you a lot of built-in options for managing your extract and making it ready for production. We recommended considering these in particular, from greatest potential impact to least.

- Pre-aggregate data and "aggregate data for visible dimensions" When you create a data extract, Tableau also gives you the option to aggregate your data for all visible dimensions. When these pre-aggregations are used, the data extract is called an "aggregated extract." This type of extract does not contain the row-level data. Rather, it contains only the aggregated data. It's ideal for some visualizations, depending on the type of analysis performed. An aggregated extract is smaller than a standard data extract, and it creates another level of efficiency in generating fast performance with your dashboards and worksheets.
- Use the appropriate data storage. With the new data model, the old "single-table" extract and "multi-table" extract has been replaced by the concept of data storage. You can store data in the extract as either logical tables or physical tables, which is tied to the layers of the new data model as described in the **data model options** section. This change means putting a little more thought into your selection for best results. The physical table storage option is the most granular, storing the data separately for each table you've added to the physical layer. The logical storage will group everything as you see it on the logical layer, so it will complete any joins you've defined on the physical layer and store them as a single table in the extract. You can think of logical storage as an aggregate of physical storage based on how you've defined the relationships on the logical lever. This split between logical and physical gives us a ton of control as authors to define exactly how tables are stored. One

potential thing to look out for is if you create the extract on Tableau Server it will default to logical tables for data storage. In cases where you've defined physical joins, this could result in larger than expected data volumes. A real-life example we saw was with a user security table join that ended up being stored as a single, very large table which slowed down performance..

- Use data source filters. There are many ways to filter your data in Tableau, including context filters or even regular filters on your views. Data source filters easily provide the most performance benefit out of them all, especially when working with extracts. You can add a data source filter in the create extract dialog. This filter will leave out any data from the extract, making it smaller and faster.
- Remove unneeded columns by pressing the "Hide All Unused Fields" button when you create a Tableau Extract. When you initially create the extract, it's best to skip this step, since you might need columns you didn't expect at first and bringing a column back will require the extract to be regenerated. However, in production, this is a quick and easy way to decrease the size of your extract and speed up your queries. When you press this button, Tableau will remove any field in your data that isn't referenced by your Tableau workbook. For best results, make sure you've cleaned up your production workbook of any unused calculations, sheets, or dashboards so that Tableau can truly optimize it for the production workload.

Embedded extracts provide additional optimization

When it's time to publish your Tableau workbook, you have two options for connecting to your Hyper Extract. First, you can publish your workbook with your extract, this is called an Embedded Extract. You don't have to do anything special to make this happen. The second is to publish your data source to the Tableau Data server. This centralizes your extract so that it can be used by multiple Tableau workbooks and authors.

Embedded extracts have two important performance advantages: (1) they are optimized for your workbook and (2) they reduce round-trips over the network. When you embed the extract, Tableau analyzes your workbook and optimizes the extract by creating temporary tables to speed up elements of your dashboards. For instance, you might have multiple filters set to show only relevant values. Tableau can create smaller datasets to make these look-ups easier to perform when users are interacting with your dashboards.

One important note here is that in a production environment, with many authors publishing work, you may have limitations on how many embedded extracts are feasible due to space and maintenance considerations. Each embedded extract means additional data hosted on your Tableau Server or Tableau Online instance and the need to keep it up to date. Weigh those considerations against the potential performance improvements (and test when in doubt). In some environments, it may be best to save this technique for the highest impact and most visible dashboards.

Materialize calculations in your extracts

If you're using a Tableau extract, Tableau will automatically materialize your calculations (if possible) when it builds an extract. This essentially pre-calculates the results before a user even requests the dashboard. This technique can be applied to any data source, however, with a bit of manual work. By taking the time to build out row level calculations or even more complicated calculations (like Level of Detail or Window Calculations) in your data source, you're offloading the processing from Tableau at load to a time when a user isn't waiting for results. These are particularly effective when dealing with string calculations, which are far more resource-heavy than numeric or date calculations.

Calculations created after the initial extract have not been materialized. You should use the **Compute Calculations Now** option to further optimize your extract for the new calculations. To do so, go to the Data menu, find your data source, and select **Extract > Compute Calculations Now**. On Server, Tableau Server site administrators and data source owners have another option and can use tabcmd to materialize calculations by adding the –**add calculations** option.

Similarly, if you're working from a published data extract, your calculations will not be materialized. Since it's published, you'll need to go through the extra step of downloading the published extract, opening it in Tableau Desktop, and then performing **Compute Calculations Now**, before republishing to the Data Server. If you don't have the proper permissions, talk to the owner of the published data source or talk to a site administrator.

There are a few calculations that will not be materialized in a Tableau Extract, particularly:

- Calculations that use unstable functions such as NOW() and TODAY()
- Calculations that use external functions such as RAWSQL and R
- Table calculations
- Level of detail (LOD) calculations
- Aggregations

It's also important to note that there are some rare cases where materialization isn't faster. In particular, Tableau doesn't consider if your calculations are nested. If you have many calculations that derive from other calculations, there might be cases where materializing the calculations is slower, especially if materializing substantially increases the size of the Extract. If you suspect you might be running into one of these rare circumstances, consider setting up a performance test.

Options for creating extracts

There are many ways to create Tableau extracts to benefit from the increased speed and optimization they often provide. These include:

- Locally with **Tableau Desktop**
- In a browser via **Tableau Server** or **Tableau Online**
- With **Tableau Prep**
- With the **Hyper API** in code
- With 3rd Party tools

Leveraging Tableau Prep or a 3rd party tool can give you an expanded range of options for cleaning, aggregating, and transforming your data to exactly the needs of your dashboard. Consider the needs of your dashboard and advice above to determine if this could help deliver a better, targeted data source for your use case.

Maintaining extracts

For production use cases, you can set a schedule to refresh your extracts on Tableau Server or Tableau Online. The smallest schedule increment allowed is every 15 minutes on Tableau Server and every hour with Tableau Online; the schedule can be to refresh at the same time daily, weekly, and so on. You can choose two refresh schedules for a single extract.

- Incremental refresh, which adds rows, but does not change existing data.
- Full refresh which discards the current data and regenerates a new one from scratch.

Incremental will be the fastest and require fewer resources on your Tableau Server. It's best practice to use incremental when you can and to set the appropriate refresh increment that your users actually need. Often daily and weekly refreshes are the most common need for the business. Users may ask for more, but consider if they can make legitimate business decisions on the additional data.

- It's possible to trigger refreshes using the **REST API** or **tabcmd** which is a command-line tool provided by Tableau. This is a good way to integrate refresh schedules into 3rd party tools, especially for cases where there might be complex scheduling requirements across multiple tools in your data ecosystem.

Live data connections specific recommendations

When using a live data connection, you'll be very dependent on the underlying data source's ability to process your requests in a timely manner. In cases where you're connecting to a relational database, your database administrator has a role in helping performance by doing things such as index tuning and proper data modeling. That said, as workbook authors, we do have some best practices we can follow to insure we're getting the most out of our Tableau.

Use referential integrity

If your underlying database supports the concept of foreign keys, Tableau will automatically be able to assume referential integrity because it is pre-set in the database. This allows Tableau to do join culling, which optimizes query performance. In short, if a Table is not referenced in a query, referential integrity tells Tableau that it can drop it from the query without affecting the results. To do this manually, you can select the **Assume Referential Integrity option** from the Data menu. Doing this manually will be slower than the former as it only affects performance on Tableau's end, and the results may be unreliable if it isn't a many-to-one relationship. If you're not confident about referential integrity in your data, do not enable it in Tableau.

Leverage the relational data model

The new data model (discussed above) allows for more flexible relationships between tables. This not only results in a better user experience as an author, but also has some significant benefits to query optimization. In particular, testing showed performance was better on more intensive calculations, like Count Distinct (COUNTD) when it could leverage a small dimension table in place of one large table or traditional join. Additionally, if you are counting a dimension, such as User ID, that is tied directly to a dimension table (such as a user table) you can use an even faster method to count the rows on the dimension table directly. Tableau provides this COUNT function for each table in a relationship (CNT(Table)) and it will often be the fastest method.

Consider using multiple data sources

It's tempting to make a single data source to build from, especially since it's often faster to set up and can help ease development. For production, however, this can result in unoptimized queries. Consider using multiple data sources so each type of analysis is built off a well-designed join.

The opposite extreme, multiple data sources when not needed, is also a pitfall. If all the sheets in your dashboard are the same level of granularity, a single table is likely preferable. However, when dealing with complex data sources at different levels of detail, it's often better to use them separately then to try to force them to coexist with the complex queries that can result. Remember that Tableau has many features, such as data blending and **cross data source filters** that can make a seamless end-user experience despite multiple data sources being used behind the scenes.

Advice for talking to data engineers and database administrators

There is only so much tuning we can do on the Tableau side. If you've optimized what you can in Tableau and your performance is hitting a bottleneck specifically on queries, then it might be time to talk to your data team about tuning the data source. Here are some things they can specifically do to help Tableau (though not all will apply to every data source type):

- **Move complex calculations to the database.** When optimizing performance, having Tableau do the least amount of work while users are waiting for results is key. Production database servers are designed to handle significant query loads and are a great option for moving processing out of Tableau. Anything you can do before the data even comes into Tableau will help optimize performance. Row-level calculations are good candidates to move into the data source. Other calculations such as window calculations and level of detail calculations should be considered, though they are harder to generate and sometimes cannot be moved over if they are based on user inputs. One benefit of moving calculations over is that they are better centralized in the data source for use by other Tableau workbook authors or even other toolsets.
- **Set indexes on join dimensions.** Tableau is generating queries to your database based on the joins in your data source connection. Just like any query, performance will be better if these are properly indexed.
- **Use a star schema when possible.** There are many ways to model data, but a single fact table with related dimension tables will often result in more efficient queries from Tableau.
- **Set appropriate primary and foreign keys.** When Tableau knows how tables are related, it can implement things like Join Culling for better queries. Specifically, Join Culling allows Tableau to only include the tables it needs in a query, instead of every one defined in the data connection in Tableau.
- **Set Index on filtering dimensions.** Tableau is going to be doing a lot of filtering. By looking at the filters used in Tableau, you have a good idea of where you may need indexes. You can also take a performance recording in Tableau Desktop and see the queries being generated if you want more specific examples to use in your performance tuning.
- **Cast data types in data source.** A lot of times values need to be cast from one type to another (string to integer, decimal to integer, etc) so that it's properly displayed or can be used properly in calculations. Doing this in Tableau in every relevant query is a lot of wasted processing. By making sure you have the right data types stored in your data source you can significantly speed up query results.
- **Make sure database permissions support creating temp tables.** Sometimes Tableau needs to make temp tables, such as in the case of initial SQL. Making sure Tableau has the ability to create temp tables gives it one more option in getting the best performance out of its queries.

Custom SQL

Custom SQL is the capability Tableau gives you to write your own queries in SQL to your underlying data source. This is a powerful tool that allows you to transform your data, perform more complicated logic, or leverage existing queries to build your dashboards. In my experience, having this available has helped the ad-hoc and prototyping phases of a project get off the ground faster than alternatives. However, when it comes to production use cases, custom SQL has some performance trade-offs. In particular, Tableau leverages the custom SQL as a subquery. Each query Tableau creates will contain your custom SQL surrounded by the Tableau specific elements. In short, Custom SQL creates long and confusing looking queries. For many databases the resulting complex queries will often achieve worse results, despite their best efforts to optimize. If possible, avoid custom SQL in production. Thankfully, when we do, we have a few options to deal with our custom sql.

Use initial SQL when possible

One way to avoid a custom sql query being run multiple times in Tableau is to leverage Initial SQL if it's available. As the name would indicate, Initial SQL is only run when the workbook is first opened and can be used to create a temporary table. That temporary table can then be used like any other table by Tableau and could significantly improve performance. One downside of this method, however, is that the temporary table remains the same during the duration of the user session, so updates to the underlying data will not be reflected in the temporary table until a new session has been opened.

Take a Hyper extract

Another way to avoid the complex query issue is to simply take a Tableau Hyper Extract. This pulls the data into Tableau's data store, meaning the query only runs on each refresh. You'll have the benefits of your custom SQL, but in the speedy data engine of Hyper. One point of consideration, however, is long-term maintenance. It may be better for your organization to keep your SQL in the same location. In which case, another solution to consider would be a database view.

Use a database view or custom table

If you're connecting to a relational database, then it's likely that your database administrator, or developers could create a database view. A database view is a way to save a query in a database for reference later. Tableau treats a view in the same way as a table, making the queries cleaner and easier for many database engines to optimize. Further, many databases do further optimization on views or allow for additional optimization options (such as materializing the data). In certain cases, it may be better to create a specific table or tables for your analysis instead of views, if you think that's the case talk with your database administrator or database developers on the options they think are best for your workload.

Simplify the query as much as possible

Finally, if you have no other choice then custom SQL, make it as simple as you possibly can. Avoid including every column (I know it's tempting to save time with Select *) by specifying exactly the columns your dashboard will require. Similarly, adding features like Tableau parameters to your customer SQL can make it harder to process conditional logic and adds overhead. Finally, look out for unnecessary clauses such as ORDER BY that Tableau will be handling itself based on the requirements of the visualizations.

Shared vs. embedded data sources

When you create a new data source, Tableau gives you the flexibility to use it in a couple of ways. By default, the new data source is embedded in your workbook. When you publish your dashboard, the data source goes with it. Alternatively, you can publish your data source to Tableau's Data Server. This allows for multiple workbooks to utilize the same shared data source. This reusability can help immensely with maintainability and developer productivity, as well as ensuring dashboards are built off of reliable and proven data sources.

Embedded data sources benefit from the ability to be targeted to your use case. When a data source doesn't have to serve everybody's needs, you're free to drop columns, aggregate rows, and materialize your calculations. Many of the recommendations throughout this paper talk about the benefits of focusing your data source as it's one of the biggest and quickest ways to affect performance. However, there's an additional benefit you get with Embedded Hyper Extracts in particular.

When you use an Embedded Hyper Extracts, Tableau optimizes it specifically for your dashboard. For instance, Hyper will actually create smaller tables to be used for your filters. In cases where you're using a more processor intensive feature, such as showing only relevant values in your filters, this optimization can be a significant gain. This obviously means a trade-off between the maintainability of a shared data source and performance of an embedded extract. Not to mention the additional space requirements of maintaining additional Hyper Extracts.

In thinking about the trade-off, it may be helpful to think about the relative importance of the dashboard. Certainly, there exist dashboards that are higher profile, impact a wider audience, or are considered mission critical. In those cases, opting for the performance gained with an embedded extract is likely worth the potential extra maintenance cost. Other factors that should play into your considerations should include the size of the data set, the time it takes to refresh, and the number of dashboards that can leverage the shared data source.

Row-level security

Row-Level Security (RLS) is a requirement for many companies and has several performance implications. In particular, this sort of user-specific content severely limits Tableau's ability to do any caching and means extra filtering overhead that has to be done when the user requests the workbook. Making sure that the recommendations above have been implemented can be even more important given these factors.

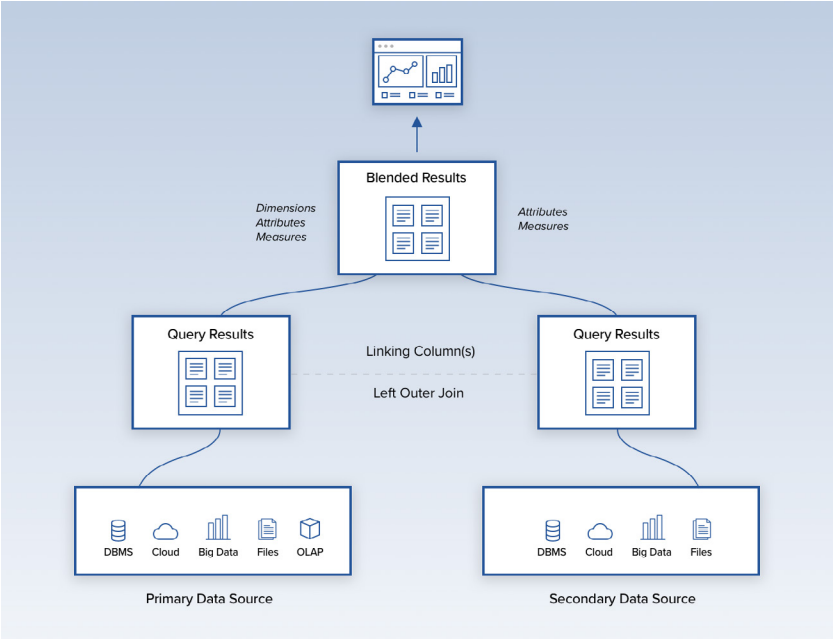
For best practices, see the whitepaper **Best Practices for Row Level Security with Entitlement Tables** or **How to Set Up Your Database Row Level Security with Tableau.**

A few important things to keep in mind:

- Set row-level security with a data source filter. Create a calculated field with a user function and filter on the appropriate results. For example [Username] = USERNAME()

If using a live data connection, consider if the security can be applied by the database, either through initial SQL or features like **Kerberos Delegation** or **SQL Server impersonation**.

- Properly model the user security tables in your database with proper indexing and foreign keys.
- If you are using Extracts, make sure you're using the right data storage method. For instance, if you define the table joins on the Physical Layer, but select to store as Logical Tables, you'll result in a very large table where each row is multiplied by the number of users that have access to it.
- Consider if there are alternatives to row-level security, such as aggregating sensitive data to an appropriate level for a general audience or targeting your workbooks more closely to groups with similar access.



Blending

Data Blending allows you to query two different data sources and "blend" the results in Tableau. As time has gone on, Tableau has added additional features like cross database joins, level of detail calculations, and the relationship model that have significantly reduced the need for blending. The primary driver of blending performance is the cardinality of the fields you're using to link the data sources. Blending queries the data from both data sources at the level of those linking fields before merging the results in memory in Tableau. The more unique values, the larger the query results, the more amount of memory used, and the larger amount of processing it requires.

For production use cases, blending should be used minimally and only on low-cardinality dimensions. Blends are not meant to replace row-level joins and if you can use one of the other methods for your use case, you should. As a general rule, relate or join when you can, blend when you must. Blending is an important, but infrequent tool in your toolkit.

Creating Calculations

Sections:

Types of Calculations

Native Features vs. Custom Calculations

String Searches and Manipulations

Conditional Calculations and Parameters

Level of Detail Calculations

Table Calculations

Advanced Calculations

Creating Calculations

Calculations use various functions, formulas, and aggregations to transform data into analysis. This section should help guide you with best practices in creating the needed calculations, but it is not intended to be an introduction to calculations in Tableau. If you need a primer, check out this help article **Get Started with Calculations in Tableau**.

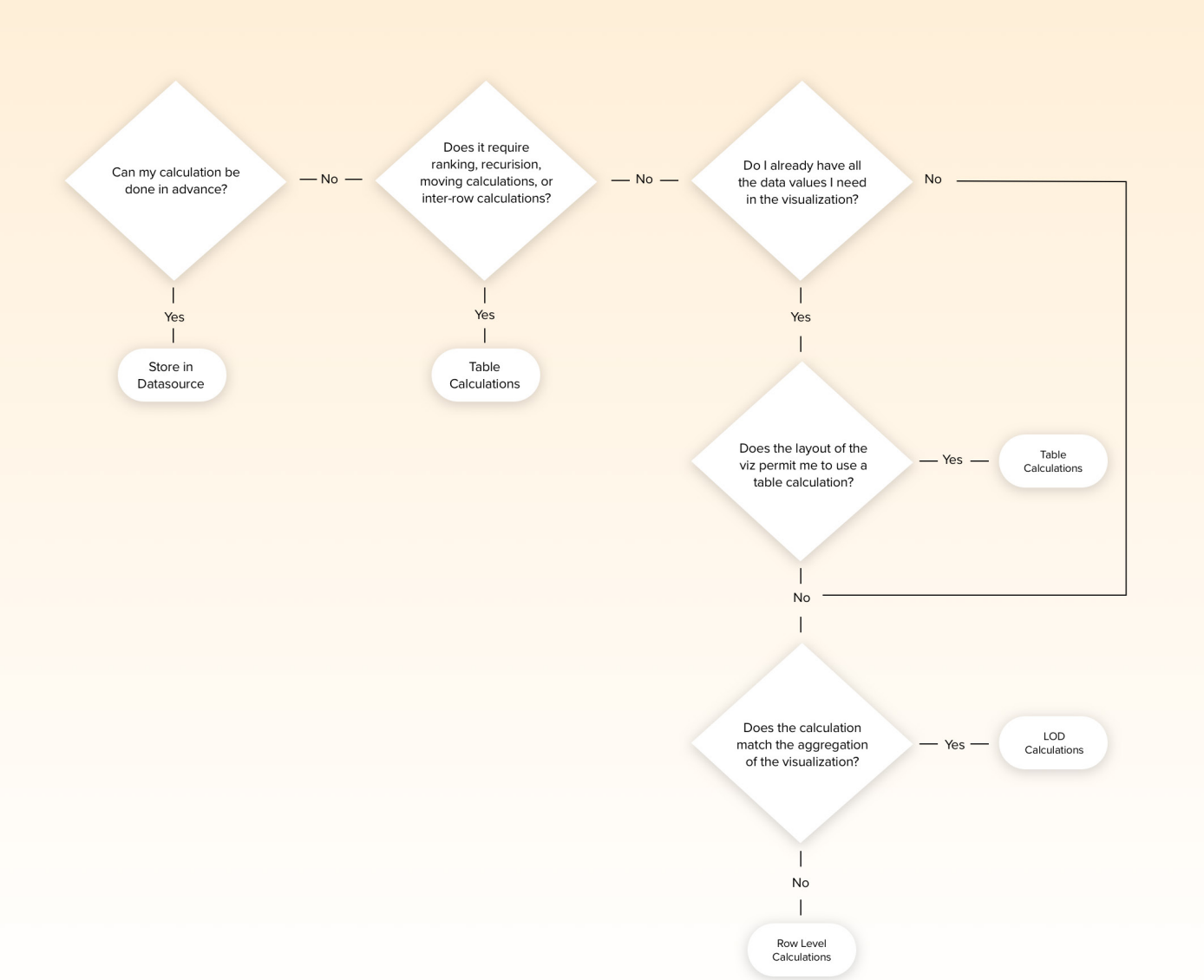
Once you have built a calculated field, it is available to any workbook using the same data source. These calculated fields can be used just like other measures and dimensions already in your data. The way you define and use these calculations affects both how Tableau queries the data source and the processing time in Tableau. Using **best practices** when assembling your calculations will help ensure you get the best performance possible.

Types of calculations

For the purposes of this paper, we'll consider 5 types of calculations:

1. Row-Level Calculations - Row-level calculations are calculations that happen within a row of data. These can include date functions, string functions, and mathematical operations to manipulate and combine the fields. You can add the results of a row-level calculation to the raw data without changing the results.
2. Aggregate Calculations - Aggregate calculations use aggregate functions (like SUM, MAX, MIN, ATTR, etc) to combine multiple rows of data and create the result of the view. These calculations summarize a column of data to produce our visualization within a worksheet. This summary and the results of the aggregate calculations is determined by the level of detail of the worksheet, so the results can change from worksheet to worksheet.
3. Table Calculations - Table Calculations work on the level of detail of your worksheet. In essence, they allow a second pass on your data after an aggregation. This enables you to perform calculations that couldn't be done in a single aggregation such as the average of your aggregation results, running total, or % of total. The key here is that the calculation is working on the results of your view.
4. Level of Detail (LOD) Calculations - LOD calculations give you more control over the level of granularity you want to do your calculation. They can be performed completely independently with the FIXED expression, at a more granular level with the INCLUDE expression. or a less granular level with the EXCLUDE.
5. External Functions (External Services and Pass-through Functions) - these are calculations that leverage external systems (Einstein, R, Python, Matlab, etc) to enrich the data with results from models or specialized functions not available natively within Tableau.

See Tableau's **Types of Calculations** help doc for more details. Use the following flowchart as a guide for how to select the best approach:



Native features vs. custom calculations

Native features in Tableau makes developing analytics fast and easy. In certain cases, like grouping data, it is often faster to do a custom calculation with a CASE statement or to add the groups into your data set. In many others, the native features have been optimized to the point that they will be substantially faster than anything custom. In many cases, any performance gains here are likely minimal, so weigh that against building a workbook that is easy to maintain. When in doubt, use Native functionality. It's also important to remember that if you are using an optimized Tableau extract you're likely getting many of the benefits automatically without need to rewrite calculations or change to native functionality. With that said, let's take a look at some key features and their performance implications:

Groups
Groups allow us to aggregate a dimension into a higher-level dimension we've defined. For instance, if we had a dataset that included Countries, we could group those countries into global regions. They are quick and easy to create and have the ability to be materialized in a Tableau Extract. When they have not been materialized or you are connecting live to another data source, the native Group functionality will often perform slower than a custom calculation using a CASE statement, as well as slower then Sets. If you are in this particular situation, consider using one of the faster methods or better yet, adding the Group to your data.

Sets
Sets are a custom field that are created in Tableau based on dimensions in your data and can be either created manually or computed dynamically. Like Groups, dimensions can be used to determine what is included or excluded but Sets also can use measures for this determination. In my professional experience, Sets are one of the underutilized features of Tableau in the wild, with authors often defaulting to Groups or Calculations first. In testing, Sets performed almost as fast as CASE statements for grouping dimensions while also providing a ton of flexibility in analysis, such as being able to use the IN/OUT functionality. Sets behave differently than Groups, of course, allowing you to easily filter to a subset of your data or to visually group members. In comparison to custom calculations, there are cases where CASE statements are faster, though less flexible. Sets will also almost always beat IF statements while providing similar functionality if you need to compute the grouping based on a measure.

Aliases
Often the values in the data are not what we want to display. It may be a case of "dirty" data or simply a case of business names changing over time. Aliases give us a mechanism for changing the display value of dimension members without creating a new field. Often, dashboard authors will instead choose to use Calculations or Groups to change the display value, but using the native Alias functionality will take less storage in the workbook and will generally perform faster than alternative methods. In cases where aliasing isn't possible or there needs to be more significant data cleaning, consider doing the clean-up prior to the workbook, either in the data set or by leveraging Tableau Prep.

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

- Totals
- Reference Lines
- Box Plots
- Trend Lines
- Models (Forecast, Cluster)

These analytic capabilities run and perform like table calculations over the data in the query result cache. Like table calculations, the more marks you visualization has the longer the computation process will take. Another factor in performance is if the measure aggregation is additive or not. Measures such as SUM, MIN, or MAX can be done locally by Tableau and are faster, while non-additive aggregations such as COUNTD, TOTAL, MEDIAN, and PERCENTILE will need to go back to the data source to get the proper result. Generally, the features available in the analytics pane are not going to be a big contributor to poor performance, but in certain cases may be worth examining.

Small things can add up

There are many things in calculations that can add up to impact performance. Pay attention to the exact needs of your workbook and pay attention to things like:

- Date logic can be complicated. Use built in functions such as DATETRUNC(), DATEADD(), and DATEDIFF() instead of writing elaborate logical tests. Look at storing values in your data or creating a Calendar table that can hold important business dates and be referenced by multiple data sources.
- Parameters can affect Tableau's ability to cache, especially when they are used in custom SQL Statements on live connections
- Filtering on calculations, especially complex ones, can result in slow running queries. On relational databases in particular, you can miss the index which results in the entire table needing to be scanned.
- Use TODAY() instead of NOW() when you don't need a timestamp
- Avoid calculations with blends when possible, since they involve data from at least two data sources, requiring multiple queries. Remember that Tableau supports cross data source joins which works well in situations where you are using Tableau Hyper Extracts.

String searches and manipulation

Tableau has done a lot of optimizing on strings over the years, to the point where string values as the result of our calculations or in our parameters do not have a significant performance impact. Calculations that manipulate strings or search them, however, are still performance intensive. When you use string functions such as LEFT, RIGHT, MID, FIND, SPLIT, or any of the REGEX functions, it's best to look at how these calculations can be added to your underlying data or materialized in your Extract.

It's also important to think about how you use the functions. Incorporating IF statements is going to be slower than allowing a function such as CONTAINS to return just a Boolean result. Take for example a case where you want to look up a product name. One way to write this would be:

```
IF FIND([Project],'Project 1') > 0
THEN [Project]
ELSE NULL
END
```

A faster way to write this so that there's only one comparison in the IF statement would be:

And even faster still would be to remove the IF statement:

```
IF CONTAINS([Project], 'Project 1')
THEN [Project]
ELSE NULL
END
```

Of course, it's worth saying that if you can change this sort of calculation into a Filter or Set, you may find even better results in performance.

```
CONTAINS([Project], 'Project 1')
```

Conditional calculations and parameters

Conditional calculations give us a lot of power to change with our data or with user inputs. For instance, parameters provide the end-user a control for providing inputs to the dashboard. These can be used to create all kinds of great user interactions, such as enabling what-if analysis with user provided assumptions or providing a way to dynamically choose what level of detail to view or metric to display. Many recommendations from older versions of Tableau have been optimized and are no longer of any significant impact. However, there are still a few things to consider:

Use aggregated values instead of row-level values when possible. It can often be a very small change with a significant performance difference. The difference comes down to order of operations. A row-level calculation containing a parameter requires the user input to be compared before aggregation on each and every row, while an aggregated calculation allows Tableau to aggregate the values before checking the user selection, this results in fewer conditional checks. Take for example a use case where you are allowing a user to select a measure to display. One way to write the calculation is:

The faster way to write the calculation is:

Notice the SUM aggregations used in the calculation. Since the conditional logic is working on summary-level data, things will be much faster. A bonus of doing it this way is that it allows you to do different types of aggregations based on user input if you desire. Just change the SUM to whichever aggregation makes sense.

Test the most frequent outcomes first

When Tableau processes your conditional statements, it stops as soon as it finds a match. To get the best performance, putting the most likely outcomes at the top will mean the majority of cases in your data will stop sooner. Consider the following example.
This logic:

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

Will be slower to evaluate than this:

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

This concept can also have some interesting implications. Often, it's worse for performance to have multiple IF statements nested together, but in cases where nested statements help the most frequent outcomes to be evaluated quickly, they might result in better processing.

ELSEIF > ELSE IF. In general, removing the space makes a conditional statement easier for Tableau and your data source to optimize. This is because ELSEIF statements are viewed as a single statement and can often be transformed into a CASE statement for the underlying query. Meanwhile, ELSE IF statements are considered 2 separate logical tests and result in nested logical statements in queries.

Tableau won't materialize a calculation with a parameter. This means that anything you've put in your calculated field will not gain the benefits of being precomputed and stored in your Tableau Extract. This may not matter if the conditional statement is simple, but sometimes you may have more complicated calculations happening within the larger calculation. Consider breaking the calculation apart so that Tableau can materialize pieces of it. Consider the following calculation that allows a user to provide a conversion rate:

We could break out the [Sales] – [Expenses] piece into its own calculation and it would be able to be materialized by Tableau. This is likely not going to matter on smaller datasets, but the larger the dataset and the more complicated the calculation, the more the potential benefit.

Level of Detail calculations

Level of Detail (LOD) calculations give a lot of power, enabling different levels of aggregation to be used in the same visualization. While Table Calculations are run by Tableau on the data already in the visualization, LOD calculations create additional queries that are processed by the underlying data source. With a live connection, this would be done by your original data source, while with an Extract it would be processed by Tableau's Hyper data engine. One consideration here is the speed and capabilities of the data source you're using, as these additional queries will need to be processed. These queries will attempt to run in parallel (i.e. at the same time), but remember the more queries generated by your dashboard, the more likely that some of your queries will have to wait for others to finish.

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

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

This means you have the choice between solving problems a few different ways. If you suspect the method you're currently using is slow, try replacing it. In general, LOD calculations will perform better on a well-designed dataset and using the logical relationship model. They also allow the data source to do the aggregation, meaning less data and processing in Tableau. Table calculations will work better when the data needed is already available within the view and will not require changing the level of detail to get the results you desire. Unlike LOD calculations, you absolutely need the lower level of detail in your view in order to get your result, so more data is being returned to Tableau from the data source.

Table calculations

Table calculations are a calculation that is always executed by Tableau, in memory, after the result of a query is retrieved. While this means more work for Tableau, it is generally happening on smaller result sets aggregated from the original data source. When used on well-designed visualizations, table calculations will generally perform quickly and efficiently.

If a table calculation is performing poorly, it is often because the query result being returned to Tableau is very large. In those cases, consider pushing back some aspects of the calculation to query or data source. One way to do this within Tableau is a level of detail (LOD) calculation.

It's also important to not use table calculations unnecessarily. For example, if we have a dataset of daily sales by store and we want to answer the question "what is the average sale per store". We could solve this with a table calculation:

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

This would require a visualization that showed sales for each store, so the table calculation can take the appropriate average. This could potentially result in a very large dataset and a lot of marks in our view.

Store	Sales	Avg. Sales	WINDOW_AVG(SUM(Sales))
79605	1	1	4,592
44035	2	2	4,592
33458	2	2	4,592
32503	2	2	4,592
32174	3	3	4,592
93405	4	4	4,592
98208	4	4	4,592
98002	4	4	4,592
72762	4	4	4,592
84041	5	5	4,592
76248	6	6	4,592
77489	6	6	4,592
77536	7	7	4,592
32127	8	8	4,592
59102	8	8	4,592
60441	9	9	4,592
83501	10	10	4,592

We have 2 more efficient options to consider:

1. Use a Tableau extract and aggregate sales to each Store. Then we can do a standard calculation of AVG([Sales])
2. Calculate the average in a standard calculation by taking a count of stores. SUM([Sales]) / COUNTD([Stores])

Sales	Avg. Sales	Distinct count of Store	SUM(Sales) / COUNTD([Store])
2,892,747	233	630	4,592

In many table calculations, you may find that the value that your aggregation is not changing over the partition. In these cases, use MIN() or MAX() instead of AVG() since they evaluate faster. Also be consistent in your choice if this applies to multiple calculations to increase your chance of a cache hit.

Advanced calculations

Tableau provides some additional ways for you to create more advanced calculations with the help of your external systems. The first are the RAWSQL expressions which allow you to write your own SQL queries to the underlying database. These work very much like **LOD expressions**, but give you the added capability to leverage additional database functions that may not be available in Tableau. There are some use cases of RAWSQL expressions that can be performance wins, but if native functions in Tableau will work leverage those.

The other set of advanced calculations are External services. These allow you to leverage things like Python, R, Matlab, and more to process your data and run models on it. These are very powerful functions, but can have significant performance impacts. In particular, you have to remember that you're adding an additional trip from Tableau to the external system and back again. This transmit and extra compute has to happen after the initial queries to the data from Tableau, but before the final rendering of your visualization.

Tuning of these systems and queries is beyond the scope of this document, but in many of the cases we've seen in production, these features can be replaced with other faster methods in production. These include using Level of Detail, Window Calculations, or pre-calculating the results in your data. For instance, you can run a clustering algorithm in Python using External Services. If that algorithm isn't using user inputs or changing based on the specific view, you can run the clustering algorithm on the dataset directly and store the cluster information.
One final thought, in cases where you're using these advanced functions to create an app-like experience with user inputs, there are ways to improve the user experience and the perception of performance. See the user experience section for more on that.

Filters & Controls

Sections:

Types of Filters

Filtering Dates

Designing Filter Actions

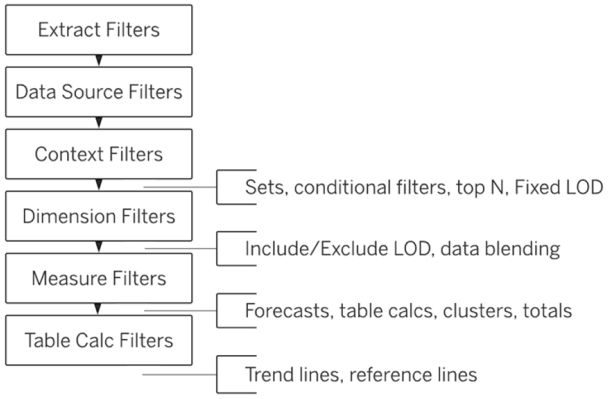
Filters and Controls

Filters are a vital tool that can be used for great impact in performance, both good and bad. They also can apply to multiple areas of Tableau, including the data source, the worksheet, and the dashboard. As such, many tips discussed in the making data sources section apply here, as limiting your data can help query performance. Despite that, inefficient filter design is often one of the most common causes of poorly performing workbooks.

Types of Filters

There are many levels and options with filters that can have unique performance impacts. These different types of filters apply at different times when a dashboard loads. This "order of operations" can impact performance as well as change the final result of your calculation. Here are the types of filters and the order in which they are applied:

1. **Extract filters** – applicable only when using Hyper data extracts and are applied on extract creation or refresh.
2. **Data source filters** – are the highest level of filter available on live connections. Data source filters apply to the entire data source for all worksheets and views. This is where user security and similar filters should be applied.
3. **Context filters** – Standard filters run without considering the context of your other filters. Setting a context filter allows you to apply that filter before your other filters are applied. For example, if you want the top 10 products for a selected region, you would apply the region as a context filter. Otherwise, Tableau will give you the top 10 products globally no matter what other normal filters are applied.
4. **Dimension / Set filters** - a standard filter using a dimension or set from your data.
5. **Measure filters** – a standard filter using a measure from your data.
6. **Table calculation filters** – a standard filter using a table or window calc in your visualizations.



To get the most out of your filters, keep these performances concepts in mind:

- **Reduce the number of filters in use.** Excessive filters on a view will create a more complex query, which takes longer to return results. Double-check your filters and remove any that aren't necessary. Also, if you have your filters applied to multiple worksheets, be aware that each change will trigger multiple queries as all the affected worksheets that are visible will update (non-visible worksheets are not run).

- **Limit use of "Show only relevant values"**. When you set filters to show only the relevant values, each filter control will apply the other filters to its list of values. This is great for users; they see what values apply to their current selections. However, to do this, Tableau must query the data each time a filter is applied to regenerate the list of values. This quickly adds up across multiple filters. If you do need to do this, consider using a Hyper Extract and embedding it into your published workbook, which will allow Tableau to optimize for your use case.
- **Use the apply button.** This won't make the filter faster, but it certainly helps your end-user get to their desired result faster, especially when they will be making multiple selections. Remove the need to wait for multiple refreshes.
- **Beware filters with many include or exclude options.** Filtering on a large list of items can result in a lot of extra information being passed to your data source or even in the creation of a temp table for very large filter sets. Use a higher-level dimension, like country instead of city, if possible.
- **Avoid filtering on the result of an aggregation.** Using filters that incorporate things like Top N results or Level of Detail can generate multiple queries. Say that you filter on the top 10 products by sales at your company. If you also have a filter for region, then Tableau first has to query the data to find the top 10 products, before running another query for the visualization filtered to region. The resulting 2 queries will often be more complicated for your underlying data source to optimize.
- **Including the filter dimensions in the visualization increases performance.** When you filter by a dimension that isn't in the view, this is called a slicing filter. These filters can be more expensive to evaluate, especially when they are complex calculations, and they limit Tableau's ability to do in-browser filtering. When you include the dimension filter in the view, Tableau has all the data already needed and can perform the filter without issuing another query.
- **Set your date filters to continuous.** Continuous date filters (relative and range-of-date filters) can take advantage of the indexing properties in your database and are faster than discrete date filters.
- **Ranges are better than itemized filters.** Selecting a range of values (such as Sales greater than 500 or Price between $10 and $20) is a much simpler query than a list of values. Consider ways that you might be able to leverage a range instead of proving a dropdown of include/exclude values.
- **Use action filters to reduce the query load and work across data sources.** Action filters benefit from some additional caching in the processing pipeline. Additionally they can often offer a great user experience and reduce duplication on your dashboard if you're already displaying a dimension in a chart.
- **Use parameters.** Parameters can be useful for a number of different use cases, including filtering. Parameters can be static or dynamic (i.e. loaded from your data). When static, they don't result in additional queries to the database at all. When dynamic, they will issue a single query when the workbook is open. Either case will result in a reduction in the number of queries and help performance. The downside to using parameters is that they are single value only.
- **User filters are effectively not cached.** Since these often relate to user security, Tableau cannot cache the results. Consider aggregating to a higher lever to eliminate the need for user filters.
- **Replace filters based on Level of Detail calculations with Sets based on formulas.**
- **Filters based on NOW() and TODAY() are not cached for long.** These types of filters are labeled "transient" by Tableau so that their cache expires faster than other types.
- **Follow data best practices**. These include proper indexing if you are using a live connection, using only the data you need, structuring the data properly, and materializing calculations.
- **Do not use context filters to improve query performance.** Long ago, there were versions of Tableau where context filters created a temporary table. This created common recommendations for (or against) context filters because of those temporary tables depending on circumstance. This is no longer the case and context filters are implemented in the query or locally in the data engine. Use context filters because you need them for your use case, not in hopes of improving performance.

- **Zooming does not filter**. When you zoom in on a visualization or map, Tableau does not filter the marks that are no longer on screen. All the marks are still there. Consider applying a filter for the area you want to see to speed up your view.

The type of filter control you use can also have an impact on performance. Some filter options do not require any queries at all to be rendered.

These will perform extremely fast:
- Custom Value List
- Wildcard Match
- Relative Date Filters
- Browse Period Date Filters

Two options that require the data to be queried in an efficient manner, just MIN and MAX values, to render the control:
- Measure Filters
- Ranged Date Filters

These final options require all dimensions members to be loaded before rendering and will take the longest of all the filter options:
- Multiple Value List
- Single Value List
- Compact List
- Slider

Filtering dates

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:
Continuous Range of Date Filters – Select on defined range of continuous dates. This is typically the fastest option.
Relative Date Filters – Select on a date range that is relative to a specific day
Discrete Date Filters – Select on individual dates that you've selected from a list.

For **discrete and continuous** date filters, one major consideration is data parts vs. date truncation. In most scenarios, truncation will be faster for your data source. Specific date parts to avoid are MY() and MDY() which can be costly to perform. For instance, in some databases the underlying table may be partitioned by date. If the specific DATEPART value your filtering isn't part of that key, the database will scan every partition, which is unnecessary and time consuming. One way to optimize here, besides using a truncation, is to materialize the DATEPART. If you're using a data extract, create a calculation that implements the DATEPART function and make sure it gets materialized.

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

Designing filter actions

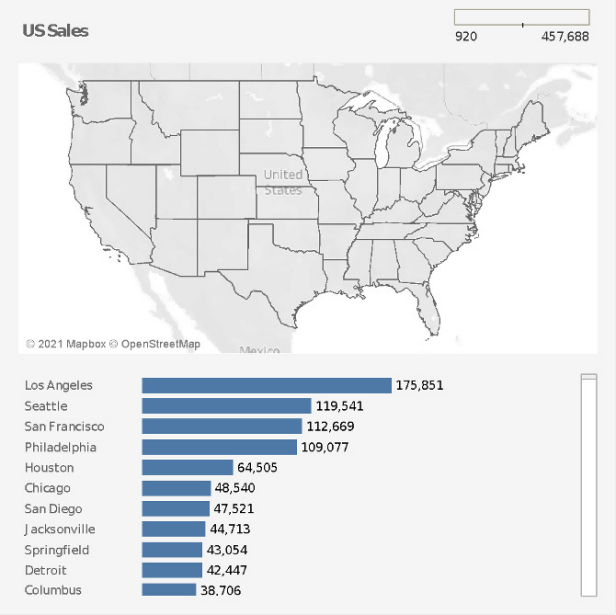
Filter actions are a fantastic way to add additional capability and exploration to your dashboard. Actions can replace the need for additional quick filter cards and create more efficient user drill-downs

Avoid filtering on too many values or dimensions

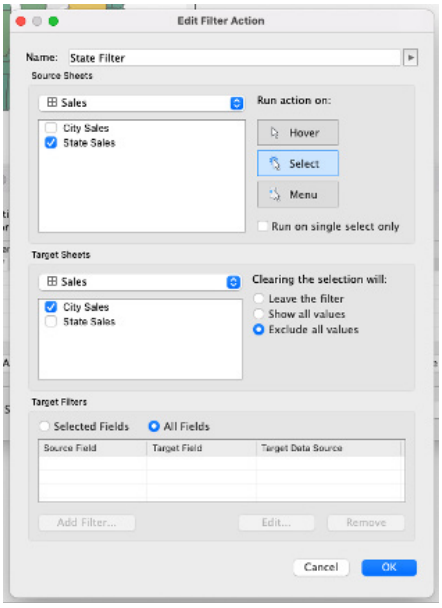
Just like quick filters, actions can cause complex queries if they filter on too many values or dimensions. In the case of values, these will be passed as a list (or temporary table) to the query, so a large number of values will be harder to process by the data source. If you are filtering on multiple dimensions, it'll be harder for the query to be optimized and you'll have a greater chance of missing an index even on a well-designed database.

Add targets to level of detail

If possible, add the Target Field of the action filter to your target view. When the field is used in an action filter, the information needed to do the filtering is available in the view and Tableau can do the filtering right in the browser. Consider the following dashboard:



If we create a filter action that targets the bar chart of cities from the map of States, we can speed up the performance by adding state to the level of detail of the bar chart. If State isn't part of the level of detail of the bar chart, then Tableau will have to query the data to find out which cities belong to the State you're filtering to.

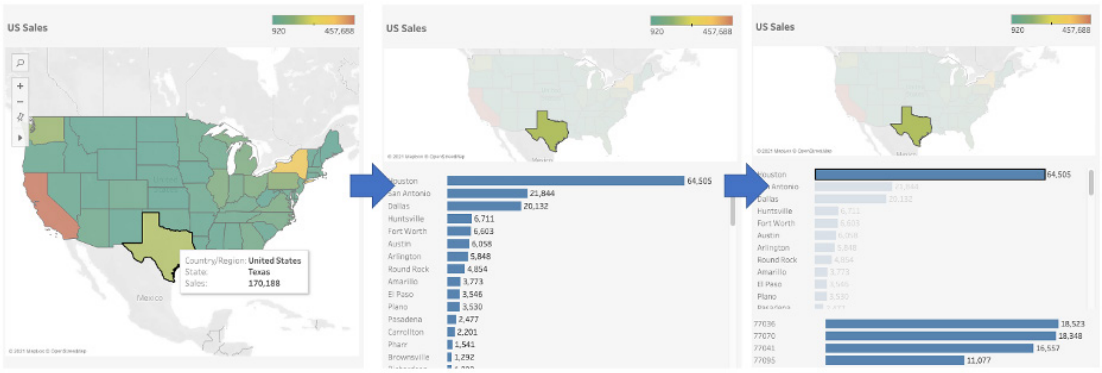


Leverage “exclude all values” on selection clear

When you create a filter, you can control what happens when the selection is cleared. By default, *Show all the values* is selected which removes the filter from the sheet, displaying all the previous data. Another option available to you is to *Exclude all values.* When cleared, this will cause the target sheets to go blank. To see this, once you’ve set it up, you’ll need to activate and clear the action at least once.

So why is this useful? It provides a very natural way to create a drilldown. Let’s look at our Sales dashboard again, but this time we’ll create a drilldown from State to City to Postal Code. With our actions set to Exclude all values on clear, you can give your users a State view, that only shows a city view once clicked. Then they can click a city, and a new Postal Code bar chart will appear below.

This is great for performance. On the initial dashboard load, only a single view is being loaded and a single query being executed. Further details are only requested once the user has made a decision. Providing these details only on demand keeps us from loading all the data on every initial load and instead delivering only what our users need as they need it.



Worksheets & Dashboards

Sections:

Maps

Tooltips

Drill-downs and Hidden Containers

Animations

Layers vs. multiple axis (“forklift..”)

Workbook-level performance factors

Worksheets and Dashboards

Tableau is at its best when it is an interactive, visual experience for your end-users. Many authors find themselves recreating legacy dashboards from other systems using old techniques. The results of these "lift-and-shift" style projects are dashboards that don't fully utilize the investment in Tableau while also providing inferior performance. To create an efficient Tableau dashboard, embrace the interactivity and stop viewing your work as developing static reports.

Much of the advice in this chapter will apply to both worksheets and dashboards, because Dashboards are just a collection of worksheets (plus a few additional elements). Filters, actions, animations, and more are all possible on both the worksheet and dashboard levels. The design of your worksheets will be the main driver of overall performance. Overall dashboard design can be leveraged to get the most out of your visualizations and help speed delivery of insight to your users, but it is important to note that performance on the dashboard level is really the aggregation of everything else. Think back to our discussion on **performance budget**, because dashboards are where this really comes into play. Many small items (filters, worksheets, containers, etc) can still add up to break your budget.

With that dashboard budget in mind, let's take a look at some specific types of worksheets/visualizations and their impact performance. For higher level advice see the earlier sections on **worksheet best practices** and **dashboard best practices**.

Maps

Maps in Tableau work very much like any other visualization and benefit from the same best practices. However, there are a few unique features to maps worth discussing..

Custom geocoding

When you import a custom geocoding role, it is written into Hyper. If you use custom geocoding in a workbook and save it as a packaged workbook, the entire database file is compressed into the TWBX file, which can substantially increase the overall size. The increased size has performance implications in opening the file and processing the larger set of geocoding data. A more efficient method is to add the geocoding info directly into your dataset via a join or blend.

Custom territories

In Tableau 10, the ability to combine areas into larger aggregated regions was added to Tableau Desktop. Be on the lookout for custom territories defined by many smaller regions. They can be slow until they are cached by the system. This is because Tableau has to first load the smaller regions and then resolve the shapes and borders into the larger territory.

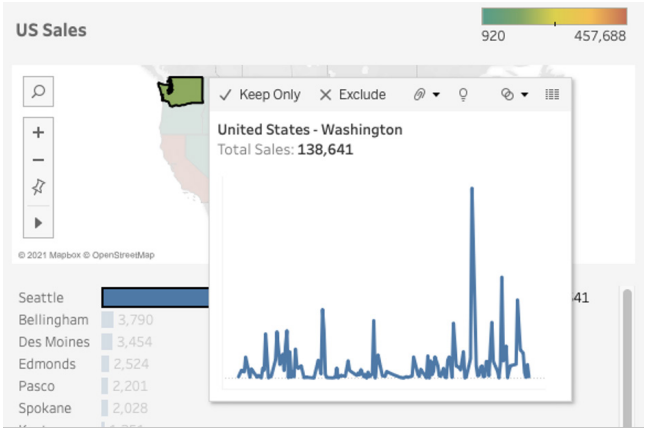
Filled maps vs. point maps

Filled map marks allow you to color an entire region (such as zip, county, state, or country) by providing a shape of that region. A point map simply graphs a single mark at the center of the region. When a user accesses your dashboard on Tableau Server or Tableau Online, parts of the visualization may be rendered within the user's browser (more on this in the server section). In these cases, sending over the region shapes is more expensive than a simple point map. If rendering seems slow, try using a point map instead.

Tooltips

Tooltips are a great way to provide details on demand, allowing the user to hover over a mark and pull up additional details. This streamlines your visualizations and means less to render on the initial visualization. Here are a few tips for getting the most out of your tooltips.

Keep Viz in Tooltips small and focused. Tableau allows you to embed visualizations inside your tooltip, giving you a fantastic way to drill down from a higher-level chart. The viz in tooltip is a static image from another view that is filtered to the mark you're hovering over and is generated when you hover. If your visualization is slow loading, there will be a noticeable lag before the tooltip appears, which can cause users to wonder if they are doing something wrong or miss that tooltips are available at. If you use viz in tooltips, stick to visuals with very quick loads, otherwise find another method to provide the drilldown view.



Consider using selected fields for your Viz in Tooltip. By default, Viz in Tooltip is filtered on All Fields, which considers all fields in the view when identifying matching records. You can change the level of detail for Viz in Tooltip by defining a filter on Selected Fields. You can do this by creating a Filter Action that targets the Viz in Tooltip worksheet directly. Note that this only works if your Viz in Tooltip is on the same data source as the source visualization.

Change ATTR to MIN or MAX. By default, when you drag a measure to the tooltip shelf, Tableau will add an ATTR aggregation around it. This performs both a MIN and MAX on the dimension to see if the values are the same, if they aren't it will display a *. If you know that the dimension doesn't contain multiple values, you'll get a modest performance boost from using MIN or MAX instead.

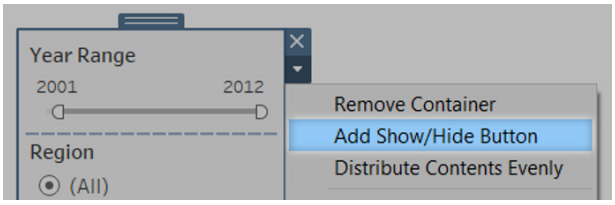
Leave tooltip formatting towards the end of development. This isn't about load times, but your personal development efficiency. A lot of time is wasted in formatting tooltips repeatedly. When you change the formatting in a Tableau tooltip, it stops automatically updating it as you add new dimensions and measures with changing design and user requests. It may seem like a small thing, but across a workbook with several dashboards the time can add up. Save the formatting until the end so you can tackle it once and for all (hopefully).

Drill-downs and Hidden Containers

This may seem obvious, but Tableau will only render and query what is needed to load the dashboard. This means that a good way to achieve performance is to limit the initial dashboard to summary information and only provide details when the user requests them. There are many different ways to achieve this, including filters, parameters, actions, and buttons.

One common way to provide a drill down into detail is to use action filters. See **designing filter actions** for tips on how to achieve this.

Another common way is to simply hide your detailed views in a container. When you float a container in Tableau, you have the ability to add a "show/hide" button. When the container is hidden, it's contents will not be rendered.



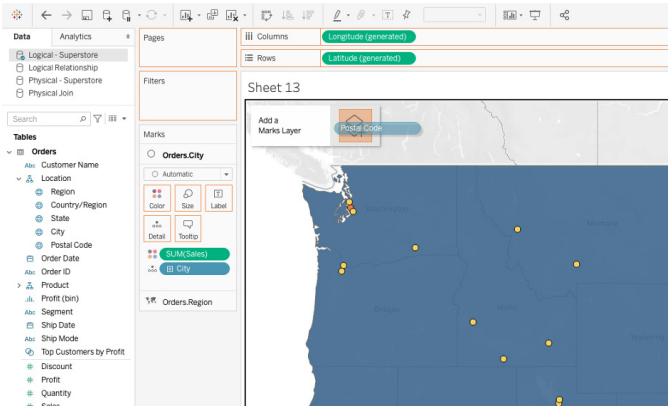
The Tableau community has used this button in a number of creative ways, including as a filter control panel, access to drill down info, and as a way to provide alternative views of the data. Keep in mind that queries may still be run when the container is hidden and the added items still increase the overall size and complexity of the dashboard. Putting things in a hidden container doesn't negate other best practices.

Animations

Animating your visualizations can help convey change to your user. As filters apply, they can follow along as the values resort or change value, helping them understand the relative impact. This is especially powerful for exploratory dashboards where users might be doing multiple comparisons across filters. From a performance perspective, there isn't a lot that needs to be considered beyond the more general guidance against too many marks. The queries on visualizations with animation are the same as those without, but Tableau will have the additional computation of the transition between marks. More marks mean more rendering time. On the user experience side, the animation will not begin until Tableau has gotten the results of the needed query, so if query performance is slow and you've chosen a slow animation duration, you'll extend the time the user has to wait to see the final visualization. Tune the duration of the animation for your use case and user expectations. Slow animations help illustrate the change and are easier to follow, faster animations appear more responsive. Animations aren't all or nothing, so target specific sheets for animations to be on (or off), to avoid problematic areas. In Tableau's 2021.2, animations will be on by default. Consider keeping off or turning off animation where there are a lot of marks. The result can be both overwhelming to the user and require significant rendering. With significant enough complexity, Tableau will switch to server-side rendering and animation won't play at all.

Layers vs. multiple axis ("forklifting")

In 2020.4, Tableau made layered marks possible. Previously, if you wanted to have multiple types of marks on a visualization, you had to use a multiple axis. The Tableau community has created all kinds of creative ways to use multiple axes to achieve impressive results, including common visualizations such as donut charts. For use cases as simple as a donut chart, layers will provide a cleaner and more flexible approach, but will provide similar rendering performance. If you're tackling a more complicated scenario, layering may provide a better solution that is easier for Tableau to render. I've seen many scenarios where complicated "forklifting" to create multi-format tables, for instance, has reduced performance significantly. Remember to be judicious in your application of these flexible features, it's easy to add a lot of marks.



Workbook-level performance factors

The performance of a dashboard within a workbook is typically the focus of any optimization, but there are a few things you can do on the workbook level to get the most out of Tableau. For best performance:

- Remove clutter from your workbooks, such as unused calculations, worksheets, dashboards, and data source. Large files take longer to load.
- Publish to the server without tabs. When tabs are enabled, Tableau will load elements of the other dashboards and worksheets.

As noted in the VizQL section, a Tableau workbook file (TWB) stores all the elements for every dashboard, worksheet, and feature used in your analysis. When this file is used, VizQL pulls the entire thing into memory and parses it before beginning the process of generating a dashboard for your user. The larger the file, the more time this takes on initial load. Removing unused calculations, worksheets, dashboards, and data sources will shrink the size of your file. If you have a large number of dashboards and/or worksheets in a single file, consider separating them into smaller groups.

When you publish a workbook, you have the choice between publishing with tabs and without tabs. Tabs provide a convenient way to navigate between dashboards and are required to leverage features such as actions that target other dashboards within the same workbook. One known issue in Tableau is that performance is less efficient when tabs are displayed, as Tableau is loading more information at once. Consider alternative forms of navigation such as navigation buttons or URL actions which do not require tabs to be present.

User experience and Tableau embedding

How a user perceives performance is different from actual performance. The human perception of time is not measured in seconds. Waiting in a long line can feel like an eternity, while watching our favorite show can seem to fly by. Our tools can use this to their advantage, making us feel that the performance is perhaps better than a stopwatch would tell us. In hotels, they frequently add mirrors to the lobby to improve guests' satisfaction with wait times for elevators. It doesn't take much to entertain your user and gain a few additional seconds of leeway.

In tech, when Google Chrome first entered the web browser market, one reason it won marketshare was for its fast performance. To be clear, it was faster on standard benchmarks, but it also changed elements of the experience, like intelligently loading pieces of the website, to make users feel it was even faster than it was. In Tableau, we don't have direct control over how things load in a dashboard, but we can use a few user experience tricks and some additional Tableau capabilities to smooth out the experience for our users, making it feel better and delivering a better, more enjoyable experience.

User experience researcher, Roma Shah said "to the typical user, speed doesn't only mean performance. Users' perception of your site's speed is heavily influenced by their overall experience, including how efficiently they can get what they want out of your site and how responsive your site feels." Pay attention to **good design**, users who find your dashboard attractive, easy to read, and simple to navigate will be more likely to rate its performance as high.

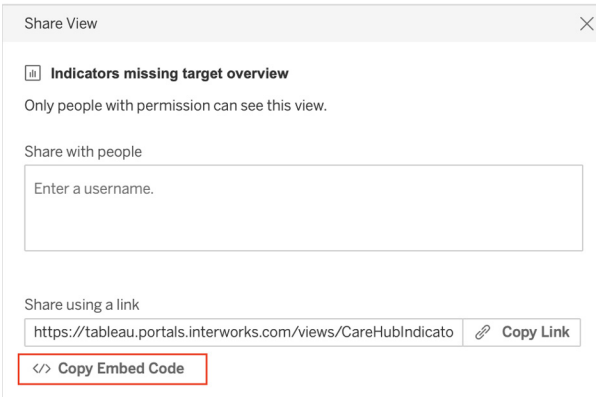
User experience tips and tricks in Tableau

Here are some quick tips and tricks when designing your dashboard:

- **Use the apply button on filters.** This saves users from multiple load times and makes the dashboard feel faster to use.
- **Break apart your dashboards into consumable chunks.** Multiple load times (within reason) are better than a single long load time. Thoughtfully group content and use drill-downs to quickly bring up additional details.
- **Use buttons for navigation.** Navigation buttons are fast and flexible. Use icons to help users quickly determine what buttons do and make it easier for them to find what they need.
- **Text should be simple and descriptive, especially in titles**.
- **Use font size and color intentionally and consistently**
- **Establish a hierarchy to your layout and make sure your key point or visualization stands out.**
- **Leverage embedding to control the experience even more.**

Embedding Tableau

You may have seen a Tableau Public dashboard embedded in a blog before, but did you know that you can do that with Tableau Server and Tableau Online? In fact, Tableau has APIs to help you craft entire new experiences that leverage Tableau. Companies are using this to create internal analytic portals, add dashboards to their intranets, or even build data products that they sell to their customers. Embedding exists in a broad range of use cases and complexity, from copying and pasting the embed code to deeply integrated systems using **Tableau's expansive APIs**.



One of the things about embedding Tableau into websites is the additional control you have over the user experience. Between being able to fully design the website around Tableau to the additional functionality that can be built on top of the Javascript API or the REST API, there are a lot of substantial customizations you can make for your use case. Here are a few we've found to be the most impactful:

- **Custom Navigation.** One of the biggest successes we see around embedding Tableau is to be able to introduce completely tailored navigation to the Tableau experience.
- **Adding custom Loading Screens.** These can range from custom graphics with your company's branding to an image of the dashboard to cover up the load-time before it becomes interactive.
- **Add a Tutorial or Pop-up notification**. On a website when a user navigates to a dashboard, you can bring up a tutorial, instructions, or other content to present to the user while the dashboard finishes loading. This both helps prepare users for using the dashboard, while also covering the loading time.
- **Preload additional dashboards.** If we know a user will likely follow a set path between dashboards, we can begin to load the next dashboard in the background for them, but only displaying it when they request. This makes the load almost instant.
- **Multiple dashboards on a single page.** We may not have control over how things load in a single dashboard, but we can control how multiple dashboards load on a webpage. If we break up dashboards into smaller sizes, we can code a page to intelligently load them
- **Filters improvements.** Tableau's Javascript API gives you access to filters in your dashboard. With that capability, you can build out enhancements for your dashboards such as remembering specific stats, adding an apply button for all your filters, or making filters global across workbooks or even Tableau Sites.
- **Substitute Parameters for Filters.** There are certain cases where filtering on parameter value may be faster than the filter itself, such as date logic that includes NOW or TODAY. When we embed Tableau, we can provide custom date controls, do any needed logic in the webpage, and pass Tableau a simple parameter value (or values) that could be easier to process.

These are just a few examples of benefits that embedding Tableau can provide. If you might benefit from increased flexibility in design, white labeling, navigation, or customization it's well worth your time to consider. Embedding is not a silver bullet to performance, but it does give you some additional tools in your arsenal for crafting an experience that users will love.

Performance On Tableau Server

Sections:

Caching

Warming the Cache

Client vs. Server-side Rendering

Leveraging the Tableau Ecosystem

Understanding Tableau Administration

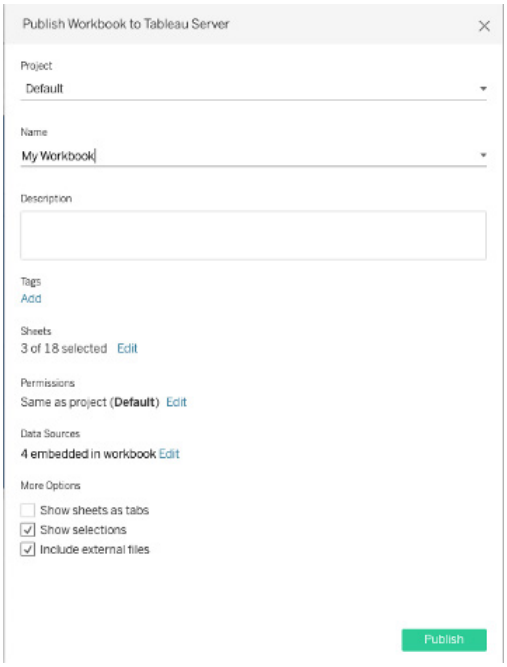
Performance on Tableau Server and Tableau Online

Desktop vs. Tableau Server/Online

Production Tableau dashboards are typically built to be published. Generally, a well performing dashboard on desktop should continue to perform well on Server, but certainly you may notice some differences. The Tableau Server or Online environment is different. It's more powerful hardware (hopefully), but it's not dedicated to you. Instead, it's serving the needs of many users across a variety of workbooks and workflows.

If your workbook is dramatically slower on Server, you may need to talk to your admins. Before you do that, however, there are a few things during publishing that can help with speed that have already been previously discussed. Try these things before reaching out to your Tableau Server admins:

- Disable Show Tabs
- Remove unneeded sheets and dashboards from your workbook file
- If you're using a hosted data extract, try embedding it into the workbook and republishing.
- Talk with other users and examine other workbooks to see if the issue is widespread.



If you've tried those methods and you've exhausted the other performance recommendations in this workbook, there's a few things you can try with the help of your administrator.

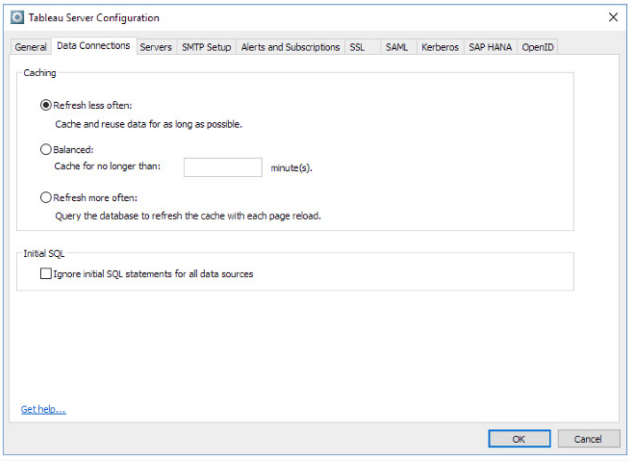
- **Upgrade Tableau Server.** The development team is constantly making improvements, upgrading to the latest release will sometimes yield improvements in performance and stability without further changes.
- **Test Tableau Desktop on the server.** Doing this can sometimes help identify if there's a configuration issue on the server, such as driver incompatibility or networking problem.
- **Check the admin views.** Tableau server comes with **several views for admins** to help monitor activity on Tableau Server. These are available on the server's status page. Good starting places are Stats for Load Times and Performance of Views.

Over the years, Tableau Server has grown in capability and complexity, so tuning the overall performance of your environment is more than we will go into here. If you're looking for information on configuring Tableau Server, the **baseline configuration** recommendation is a good place to start. Beyond that, it may be best to consider professional consulting services.

Caching

Tableau has many types and layers of cache and it is for the most part not exposed to your as the dashboard author. One thing is clear, you benefit greatly when cache is enabled and when your workbook design allows Tableau to use the cache.

On Tableau Server, you'll want to make sure cache is enabled. A Tableau Server administrator can do this in the Tableau Server Configuration utility or with a **tsm command.** For best performance you will want Tableau Server to have caching turned on to refresh less often. If you have the server capacity available, you can increase the size of the caches. There is no penalty to increasing these settings, so if your Tableau Server Admin has sufficient RAM they can increase the numbers significantly to avoid content being pushed from the cache.



Warming the cache

When a user first loads your workbook on Tableau Server, the cache will get populated. The next user will benefit from this with a faster loading workbook. As data refreshes or Tableau determines the cache needs to be refreshed, another user will typically have to experience that slower load time to repopulate the cache. What if we could step in and save that first user from having to load the workbook without the help of the cache? What if we could do that first load for our users? This concept is called cache warming.

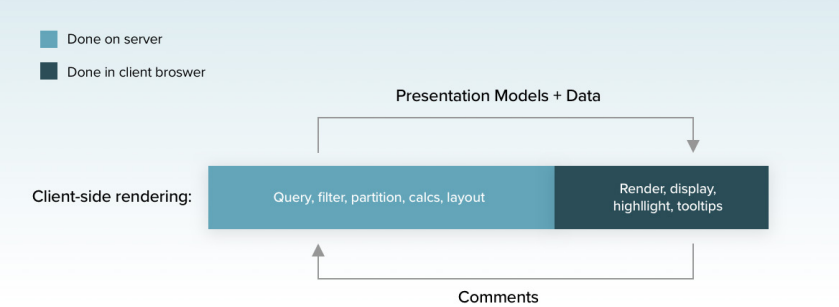
Cache warming gets a lot of attention and discussion in my experience. More than it probably deserves. Caching is great and can help slow loading workbooks run faster, but it isn't a magic bullet and shouldn't replace good design. On top of that, if you try to cache everything, you'll likely run out of space for caching and end up pushing other workbooks out of the cache, requiring them to do a fresh reload. If you're going to go down the route of cache warming, make it targeted, on high impact dashboards that serve a big audience or a key process. Also remember that if user security, user-specific elements, or transient functions (NOW and TODAY) are in your workbook, you will not be benefiting from cache regardless.

There are two ways we'll recommend for doing cache warming. The first is simply to add a **subscription** to your workbook. Subscriptions will load the dashboard to generate an image of the dashboard to be sent via email. This process will do that first load of the dashboard and populate the cache. Just make sure you set the subscription time to run before your users will view the dashboard and after any data refreshes.

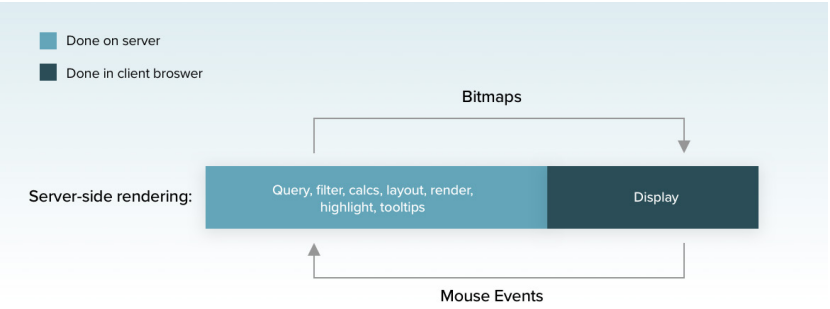
The second method is data acceleration. This will require administrator privileges and can be done by the REST API or by the accelerate_workbooks.py Python script available on **GitHub**. As the name implies, **Data Acceleration** works on your data and only populates the query cache. When it is set on, Tableau will precompute and fetch workbooks for the enabled workbooks. Workbooks may not appear to be faster if queries are already performant.

Client vs. server-side rendering

A visualization is rendered after the data has been queried and processed. That rendering can help on Tableau Server or in the end-user's web browser. By default, it happens in the web browser, this is called client-side rendering. Client-side rendering will often help limit the amount of data that needs to be transferred over the network from Tableau to the end-user, as well as any round-trip delays. Additionally, user interactions are often faster because they can be interpreted and rendered right in the browser without further network communication.



Server-side rendering is when views are rendered on Tableau Server. When this happens, the server renders the visualization and sends an image file to the user's browser instead of the data. This is ideally reserved for situations that benefit from the server's increased computing power and where transmitting the data to the client requires more bandwidth than the resulting image. There can be a large benefit for complex visualizations, but with the trade-off of reduced interactivity. Tooltips and highlighting will be slower with server-side rendering due to the required additional roundtrip to the server and back to process a new image.



Administrators can configure and fine-tune **client-side rendering**, which can be helpful for mobile devices. If you'd like to test which will give you better performance, you can use the URL parameter "render". Type ?:render=false at the end of your URL to get server-side rendering or ?:render=true for client-side rendering. The full URLs will look something like this:

```
Server-Side Rendering: http://localhost/views/Supplies/MyView?:render=false
Client-Side Rendering: http://localhost/views/Supplies/MyView?:render=true
```

You can also test the thresholds that Tableau uses to decide on individual views. You can provide a number to the URL parameter that allows you to compare a lower complexity (such as 80) to a higher complexity (such as 120) tipping points to see which results in more responsiveness to user interactions. For example:

```
http://localhost/views/Supplies/MyView?:render=80
```

Leveraging the Tableau ecosystem

Tableau workbooks are just one part of a much larger data ecosystem. Many workbooks and dashboard problems are the result of trying to solve one problem when other tools may be a better fit. Remember that creating a culture of data is more than just producing the perfect dashboard and can require enabling your users to leverage all the tools they have available to them.

Data exploration

Many slow dashboards exist because they attempt to give the end-user a way to explore a large dataset in an ad-hoc way by providing detailed views into the raw data with many, many filter options. Without a clearly defined purpose or message, the dashboard ends up trying to do too much and the performance suffers. If the main purpose is to give your users the ability to explore, helping them leverage Tableau's **Ask Data** or **Web Authoring** capabilities will result in a faster, more flexible experience. There may be some learning curve to picking up all the capabilities, but you'll empower your users to make better decisions with data if they aren't limited to the performance and features of a single workbook.

Data delivery

Similarly, many dashboards exist as a way for user's query data and pull it for some other use case. Tableau supports this with a wide variety of download options, but when the data is large or the queries can get complex, there may be better options. Of course we'd advocate for more users leveraging Tableau Desktop or Web Authoring to query the data with VizQL, but you can also process and deliver data with **Tableau Prep**.

Understanding Tableau administration

Tableau Server has four levels of administrators, which may be all one person or multiple depending on the size of your company. They are:

- Tableau Server administrators
- Tableau site administrators
- TSM administrators
- Tableau portal administrators

Tableau Server admins have access to administrative pages for creating and editing sites, adding users and setting roles, and many content-related tasks after the Tableau Server installation is complete. They can also create and manage other server and site administrators. This is an administrator who can help you with extract refreshes, schedules, and monitoring performance, along with other global settings.

Site administrators manage sites, user groups, and projects. They are a good resource for discussing permissions. They do not, however, have access to server-wide settings. When you have performance concerns, you will likely need to talk to your server administrator or TSM administrator

TSM administrators administer the Tableau installation and have a tool (TSM) that gives them command-line and web-based options for installing, upgrading, configuring, and maintaining Tableau Server. They are an administrator on the local computer that hosts Tableau Server, and they are responsible for installing the server and doing related administrative tasks like backing up data, restoring backups, creating log archives, and managing the cluster. This would be the administrator to talk to for bigger concerns around server performance, caching, and architecture.

Finally, Tableau portal administrators manage licensing and the associated keys for Tableau deployment via access to the Tableau customer portal. This administrator will help you with licensing for all Tableau products, including Desktop, Server, Prep Builder, etc.

Conclusion

Conclusion

Performance is a multifaceted problem and goes well beyond technical best practices. If there is one takeaway, it's to be thoughtful and purposeful with core user requirements and communicate the various tradeoffs in features vs. performance to build the right dashboards. Designing the right dashboards the right way will save you tons of time in troubleshooting.

When you find yourself figuring out how to improve an existing workbook, remember there can be many reasons for slow performance. Collect data on performance, identify the most expensive areas, and focus on them making your way down the list from biggest impact to smaller. Stop when it's fast enough or the effort to reward ratio no longer makes sense. There's a lot of art to the science of performance, you'll get better at identifying issues the more you work on it.

Remember that production dashboards should be:

- Built with a specific purpose
- Interactive and Visual
- Contain only the data you need, and that data should be well structured and clean
- Begin at an overview and drill into detail

Extracts are a great way to improve performance and we recommend for a wide range of production use-cases. Leverage them to do other best practices like materializing calculations and aggregating your data to your visualizations' required level of detail. It's easy to spend your performance budget on processing large data; smaller data means that you'll need to implement fewer performance recommendations elsewhere and can leverage more expensive features.

Remember that Tableau is continuing to improve and add features all the time. Look to the always creative and helpful Tableau community for the latest ways to improve performance or ways to create new and exciting ways to see and understand your data. Good luck, and may your production dashboards always be fast!

Appendix

Sections:

First Load Results

Impact of Interactions

Impact of Caching

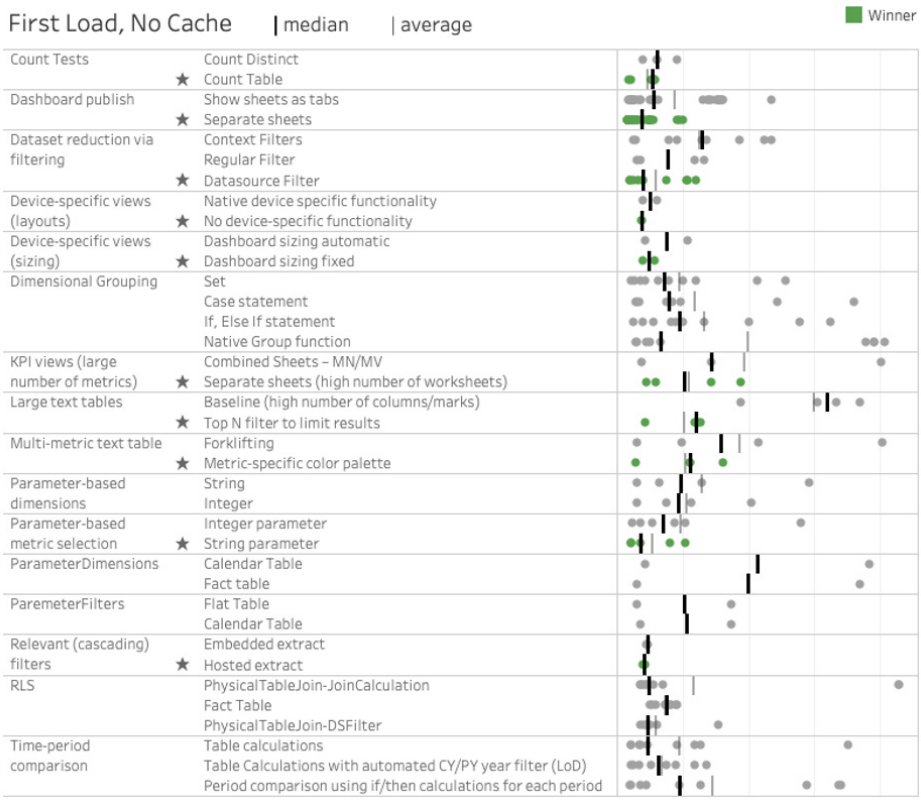
Distribution of Results

Basic Summary Stats

Appendix

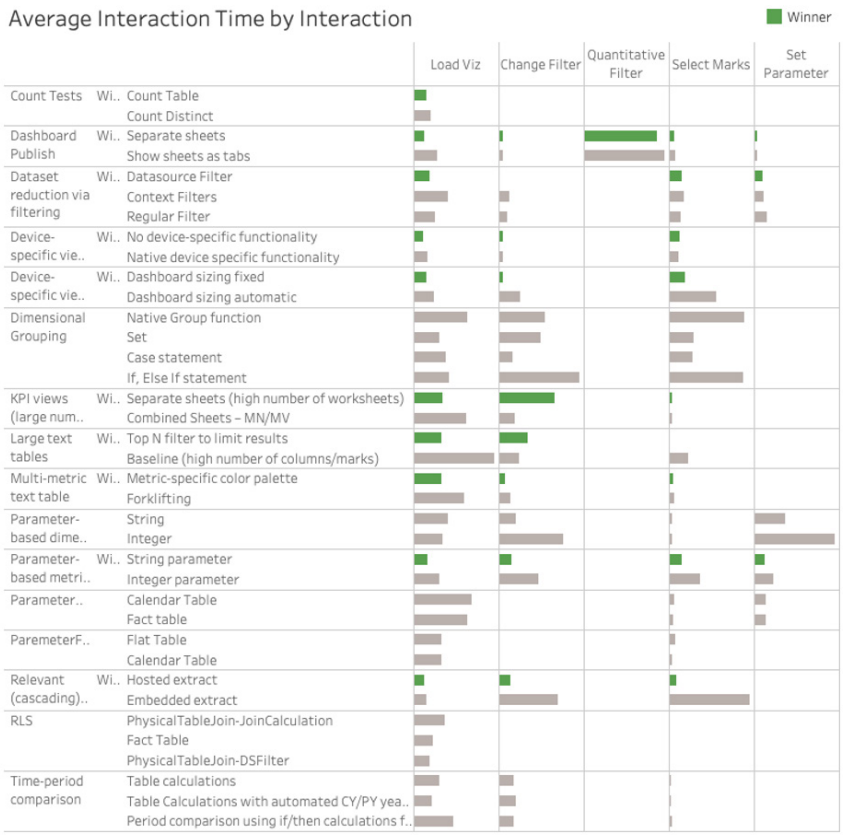
The recommendations on this paper are a combination of testing common situations and the experience of numerous Tableau experts working in real-life environments. We did testing on a dedicated Tableau Server running 2020.4 and using **Scout**. We grouped tests into Functions (the use-case) and Methodology (the way we achieved the use-case).

First load results



Throughout the tests, there were outliers, and performance sometimes varied substantially. Here's a comparison of both average and median results, with a star and green coloring denoting the methodologies that won in both average and median comparisons for the initial load without the benefit of cache.

Impact of interactions



In some instances, interactions can behave differently than the initial load. This is typically not significant, but using Embedded Extracts (vs. Hosted) performance gains for interactions can be substantial because of the optimizations. When you choose to embed your extract in a workbook, you sacrifice a small amount of performance on initial load (due to the increased data size of the extract from the optimizations) for much faster interaction times when filters change.

You see this tradeoff in other areas as well. **Applying a Top N Filter** is one way to boost initial load times by reducing the marks rendered. However, when filters are applied, we see the load time more than double due to more complex queries. In general, it would be better to find another way to limit your results.

Impact of caching

In our testing we compared our methodologies on cache. The above chart looks at the impact of cache on the average time. A blue arrow to the left shows cache improving performance. Often caching performed the best on the worst-performing techniques, helping to close the gap between methodologies. Suppose you are in a situation where you must use slow-performing techniques, like Native Grouping or IF ELSE statements. In that case, cache warming techniques like Data Acceleration and setting up subscriptions can provide substantial benefit.

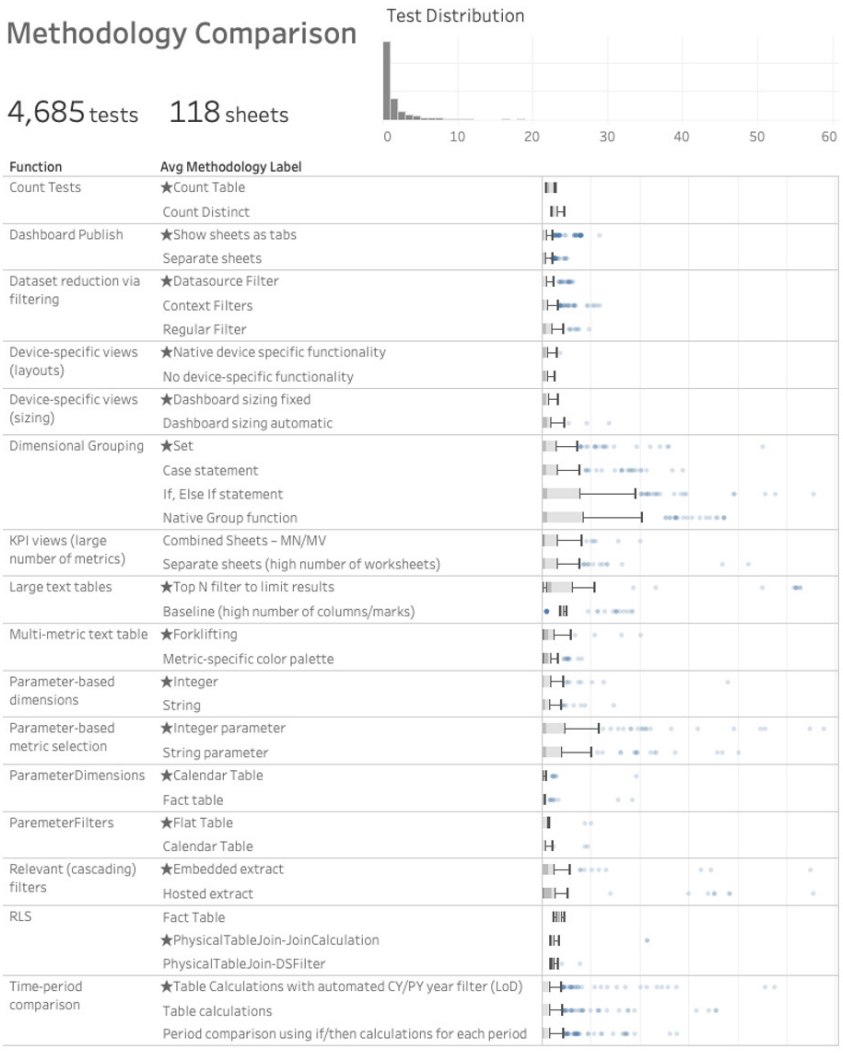
Using Scout, there is no clear way to see if our test successfully used the cache. What is being measured here is the difference between telling Tableau to attempt to use the cache and telling Tableau to refresh the cache (i.e. clear the existing cache). The times where cache resulted in longer interaction times may be due to the cost of a cache miss.

The new relationship model appears to benefit more from caching, with calculations COUNT on a table seeing a bigger speed boost then a similar Count Distinct.

Automatic sized dashboards also greatly benefited from caching, which is a change from previous behaviors in Tableau. Despite that, Fixed Sized Dashboard behaved more consistently throughout all testing and is still recommended.

Distribution of results

In the overall testing, with many interactions (including selection, filters, and parameters) we see a wide range of behaviors. This emphasizes that there isn't a "one-size fits all" approach to performance and many recommendations can depend on your use case, types of interactions, ability to cache, and complexity of filter combinations. One method may give you the best average time while another the best median time. In general, with our recommendations, we've sided with the average time because we think the outliers are part of the user experience, and consistency of results matter. When a complex query or a cache miss results in a very long wait time, a real user is waiting on that result, even if others got a slightly faster outcome.



Basic summary stats

The below table shows a more complete view of the raw test results:

Median Comparisons Table
(averaged across datasource scenarios)

		Median Interaction time	Avg. Interaction time	# of tests	Cache Difference (Median)	Refresh Cache Median	No Refresh Cache Median
Count Tests	Count Distinct	2.5	2.8	12.0	-0.67	3.0	2.4
★	Count Table	1.5	1.8	16.0	-1.48	2.7	1.2
Dashboard publish	Show sheets as tabs	0.5	1.0	525.0	0.00	0.5	0.5
★	Separate sheets	0.5	0.7	600.0	-0.02	0.5	0.5
Dataset reduction via filtering	Context Filters	0.5	1.3	234.0	0.00	0.5	0.5
	Regular Filter	0.9	1.5	96.0	0.05	0.8	0.9
★	Datasource Filter	0.3	0.9	198.0	0.00	0.3	0.3
Device-specific views (layouts)	Native device specific functionali..	0.9	1.0	48.0	0.01	0.9	0.9
★	No device-specific functionality	0.9	0.9	48.0	-0.01	0.9	0.9
Device-specific views (sizing)	Dashboard sizing automatic	1.1	1.6	48.0	-0.86	1.8	0.9
★	Dashboard sizing fixed	0.9	1.0	48.0	0.04	0.9	0.9
Dimensional Grouping	Set	0.8	2.8	239.0	0.03	0.8	0.8
	Case statement	0.8	2.9	219.0	0.03	0.8	0.8
	If, Else If statement	1.2	5.8	239.0	-0.82	1.8	1.0
	Native Group function	1.2	6.4	210.0	0.00	1.2	1.2
KPI views (large number of metrics)	Combined Sheets – MN/MV	0.7	2.6	84.0	-0.04	0.7	0.7
★	Separate sheets (high number of ..	0.7	3.3	96.0	-0.02	0.7	0.7
Large text tables	Baseline (high number of column..	4.2	5.6	70.0	0.12	4.1	4.2
★	Top N filter to limit results	1.9	10.1	50.0	0.06	1.8	1.9
Multi-metric text table	Forklifting	1.4	2.5	64.0	-0.10	1.4	1.3
★	Metric-specific color palette	1.4	1.8	64.0	-0.16	1.4	1.2
Parameter-based dimensions	Integer	0.4	1.8	96.0	-0.09	0.5	0.4
	String	0.6	1.3	96.0	0.13	0.4	0.6
Parameter-based metric selection	Integer parameter	0.9	5.2	149.0	0.40	0.8	1.2
★	String parameter	1.0	4.3	133.0	0.70	0.5	1.2
ParameterDimensions	Calendar Table	0.4	1.1	42.0	-0.01	0.4	0.4
	Fact table	0.5	1.3	48.0	0.01	0.5	0.5
ParemeterFilters	Flat Table	0.2	1.0	32.0	-0.01	0.2	0.2
	Calendar Table	0.2	1.0	32.0	0.00	0.2	0.2
Relevant (cascading) filters	Embedded extract	1.3	4.0	62.0	0.08	1.3	1.3
★	Hosted extract	2.0	6.7	51.0	0.25	1.7	2.0
RLS	Fact Table	3.7	3.6	19.0	-0.59	3.8	3.2
	PhysicalTableJoin-JoinCalculation	2.2	4.3	19.0	-0.29	2.4	2.2
	PhysicalTableJoin-DSFilter	2.4	2.8	17.0	0.13	2.4	2.5
Time-Period comparison	Table Calculations with automat..	0.2	2.5	220.0	-0.20	0.3	0.1
	Table calculations	0.3	2.2	228.0	0.02	0.3	0.3
	Period comparison using if/then ..	0.5	2.2	228.0	0.08	0.4	0.5

About the Authors

Sections:

Ben Bausili

Mat Hughes

About the Authors

Ben Bausili

Global Experience Practice Lead | InterWorks

Ben is on a mission to understand and create more understanding. Throughout his career, driven by a thirst for knowledge and desire to help, Ben has worn many hats; Database Administrator, Web Developer, Tableau Consultant, Data Scientist, and Principal Consultant. Based out of Tulsa, Oklahoma, he now shepherds InterWorks' Global Experience Practice leading his teams to create beautiful data experiences and products. This includes the analytics portal solution, **Curator by InterWorks**. Whatever the role, Ben loves bringing together knowledge and people to craft elegant solutions.

When not instigating positive changes for clients, Ben is most at home on a hiking trail with his family or geeking out about theme park design. His wife, Rachel, shares his love of adventure and travel. So, when they're not in a new city trying the local food or the great outdoors escaping it all, they are likely dreaming and planning their next family destination over (locally roasted) coffee.

Mat Hughes

Analytics Practice Lead, US | InterWorks

With over nine years of experience in the hotel industry, Mat's extensive hotel-operations experience eventually led him into the realm of revenue management and analysis, where he found his niche. Through helping others in the organization do the same thing, he became a believer in the value of explaining complex ideas in simple, elegant ways. Eventually, Mat soon sought another change in scenery. This time, he wanted to find a place that would really set his love of data free. He found that InterWorks was the perfect place, joining us as a business intelligence consultant. Mat can now apply his analytical skills and data knowledge to more than just hotels, although he still has a soft spot for them.

Mat loves his work, but he has a few more passions. He loves traveling with his wife, Kirby, and is willing to travel anywhere if it means a new experience can be found. He is particularly fond of traveling for concerts and music festivals. At any other time, whether it's between trips or during a few minutes of downtime, you can find him reading about whatever subject has caught his attention lately.

www.interworks.com

InterWorks is a people-focused tech consultancy delivering premier service and expertise in collaboration with strategic partners. Our clients trust us to guide them to approaching their unique challenges and we steward that responsibility seriously by bringing together the right people and technologies to position everyone for their greatest success. From the science of data to its storage, management, and visualization, we can support you in every aspect of your BI strategy. We value laying a strong foundation, and our relational approach leads us to customize solutions to meet your precise needs and empower you to do data your way.