Optimizing your Amazon Redshift and Tableau Software Deployment for Better Performance v2

By Russell Christopher and Vaidy Krishnan

Table of contents

Introduction

Amazon Redshift and Tableau Software are two powerful technologies in a modern analytics toolkit. Combined, they form a data warehousing and analysis solution that allows users to analyze datasets, running into the billions of rows, with speed and agility to drive better business outcomes. They are integrated out-of-the-box, so you can connect Tableau to your Amazon Redshift data warehouse with minimal time or effort. It's important to understand how to optimize each tool when integrating the two together, and doing so can yield considerable performance gains and ultimately shorten deployment cycles.

This paper introduces infrastructure advice as well as tips and hints to make the joint solution more efficient and performant. Some highlights to keep in mind are:

- On the Tableau side, the key is to *design* optimal views and dashboards that encourage guided analytical workflows, and not just open-ended data exploration. Always be on the lookout for ways to reduce the queries Tableau needs to send to Amazon Redshift, focusing on the most important questions at hand.

- Knowing how you use Tableau is important to understanding how Amazon Redshift can be optimized to return results fast. We will discuss several tools you can use to support the specific analytical workflows you build in Tableau. This includes cluster sizing, sort keys, and other optimizations to improve Amazon Redshift's efficiency

- Finally, we cannot overemphasize the importance of measuring and testing the performance of your integrated deployment. As you tune both Tableau and Amazon Redshift, you will make many changes; some that are big, many that are small. Just as analyzing data is core to the success of an overall business, measuring and analyzing the performance of your database and dashboards is core to the success of your deployment.

About Tableau Software

Tableau Software is a business intelligence solution that integrates data analysis and reports into a continuous visual analysis process, one that lets everyday business users quickly explore data and shift views on the fly to follow their train of thought. Tableau combines data exploration, visualization, reporting, and dashboarding into an application that is easy to learn and use. Anyone comfortable with Excel can create rich, interactive analyses and powerful dashboards in a drag and drop environment and share them securely across the enterprise. IT teams can manage data and metadata centrally, control permissions, and scale up to enterprise-wide deployments. With Tableau's release of Tableau Online, and Tableau Server deployable into public cloud platforms (including AWS) all front-end visualizations can also be published to the cloud, where the visualizations can directly query Amazon Redshift data warehouses.

About Amazon Redshift

Amazon Redshift is a fast, scalable, and simple data warehousing built to serve workloads at any scale. It allows you to run complex queries using sophisticated query optimization to deliver fast results to Tableau. It also allows you to extend your datasets to include vast amounts of unstructured data in your Amazon S3 "data lake" without having to load or transform any data. This opens up new possibilities for Tableau users to explore new datasets and gain deeper insights quickly. You can get started at $0.25 per hour, or with yearly costs below $1,000 per terabyte.

Optimizing Tableau

Tableau is a powerful tool that lets people ask even the most complex of questions without writing reams of code. As a popular saying goes though, *"With great power comes great responsibility"*.

To best leverage Amazon Redshift's resources such that they are utilized wisely, in workload throughput, it is important to carefully consider the design of your workbooks.

It's tempting to throw as many items on to a dashboard, or to display fancy bells and whistles just because you can. Instead, keep it simple. Display exactly what the most important data is, before allowing end users to explore additional parts of the data.

A few topical areas are listed below with advice on each. For an in-depth read on the best tips and tricks for the most performant workbooks, we recommend reading, Designing Efficient Workbooks.

Begin with summaries

Analysts are most successful with Tableau when they use it to create analytical workflows, and not just dashboards that display all the data possible. They build interactive applications that display summary data first, and then give viewers the control to dive deeper into the data that's most relevant to them.

Practically, this can mean filtering dashboards to only a few items, or displaying the data at an aggregate level (e.g. Country) with simple tools to dive deeper into data most relevant to the user (e.g. City). Smart design choices like these have concrete performance benefits (fewer rows of data are returned to Tableau), as well as perceived performance benefits (Tableau does not have to query data when it is impossible to display all the records on a dashboard without extensive scrolling).

A couple of examples:

Aggregate measures

When connecting Tableau to any database with billions of rows of data, Amazon Redshift included, make sure you're working with aggregated measures instead of disaggregated measures. Working with aggregated measures means that even if Amazon Redshift scans many billions of rows, it will aggregate the data and return fewer rows back to Tableau. This reduces both perceived query execution time, and the number of data points Tableau needs to actually render.

To do this, make sure the Aggregate Measures option on the Analysis menu is selected. For more information, see the post on Disaggregating and Aggregating Data in the Tableau Knowledge Base.

Aggregate dates

When working with dates, try aggregating the data to the hour or day. This is particularly useful for time series data that is generated at a high level of precision, but is not always analyzed at that level.

For example, website traffic data is typically machine-captured at the second, but is more often analyzed by the hour or day. Aggregating the data to hour or day can significantly decrease the time it takes to answer questions such as: What hour can I expect peak traffic to our website?

Create aggregate tables in Amazon Redshift

If you find that your dashboards primarily rely on aggregated data (such as aggregated dates by month or day), it may be worth creating new tables that pre-aggregate the data into separate summary tables in Amazon Redshift. Although this approach has some drawbacks, for summary dashboards that infrequently change but are frequently loaded, the performance benefits can be huge. Know also that you have to re-load with "re-aggregated" data each time new data is loaded into Amazon Redshift (or updated). The Tableau AWS Modern Data Warehouse Quickstart demonstrates this process and is an excellent guide.

Dashboards

In Tableau, workbooks are composed of individual sheets, each of which displays a visualization. You can combine multiple sheets to create dashboards and stories.

When connecting to an Amazon Redshift data source, however, it's best to limit the number of items on a dashboard. Displaying a dashboard requires Tableau to execute queries unless results are already cached. In general, each sheet in your dashboard will often execute at least one query. Put ten sheets on a dashboard, and you could have potentially ten queries targeting your Amazon Redshift instance at once.

Your sheets may also utilize Quick Filters, which are powerful tools to filter data. Quick filters often fire additional queries to determine "filterable values" to display to users. After combining several sheets with Quick Filters into your dashboard, it could end up executing more than a dozen queries. You could set up proper workload management queues and slots in Amazon Redshift to better service these queries, but these queries may still likely queue and therefore deliver poor perceived performance.

So, our guidance is to err on the side of sending fewer queries to allow for better latency. The rule here is, "less is more." This not only applies to the number of queries you need to send to Amazon Redshift, but also the human brain's ability to process information. It can be difficult to see the important details on a dashboard when there are 15 separate charts to look at, all at once; And it has the added benefit of allowing you to keep your Amazon Redshift design simple.

Filters

Be deliberate with filters. In general, columnar databases don't have indexes, and Amazon Redshift follows the norm, so it scans data when answering a question for Tableau. Creating filters in Tableau with supporting sort keys in Amazon Redshift can reduce the amount of data Amazon Redshift needs to scan (more on Sort Keys below).

On the Tableau side, authors often create interactive experiences which begin by returning "all" data. An author will craft a worksheet which shows "everything" and allow end users to narrow down what they want to see. When using Amazon Redshift as a data source, it's best practice to reverse this approach: Display the smallest meaningful subset of data as possible, but give the end user the option to discover additional useful data meaningful by "loosening" filters.

A couple examples:

Not all filters are equal

Quick filters are powerful tools, but most require a database query to populate the filter's values, and each query has varying levels of cost. For example, determining the values to display in a ranged date filter takes longer than for a relative date filter. In fact, relative date filters don't require a query at all, since their display values are constant (i.e., choose everything 30 days prior to today). On the other hand, ranged date filters require a query to pass the start and end date in order to return those values to the end user as a range of dates.

Sets can also be used as a cheaper alternative to some filters, particularly if you limit sets to a few items. They are less flexible, but therefore reduce query load time. For example, you can create sets that return the Top 50

items within a dimension, rather than having to query all the items and then filtering to the Top 50. For more information on Sets, see Creating and Using Sets in the Tableau online help.

Filters need design too.

Quick filters can be customized with a variety of options that can affect query execution time. "Show relevant values" is a powerful way to force filters to omit values that are no longer relevant based on other selections made in related filters. However, this process requires the execution of additional queries to determine whether or not filterable values are still relevant. Use this feature sparingly.

Similarly, the "multiple selection" filter with checkboxes will immediately run a query every time an item is selected or deselected, without waiting to see if a user intended to select other items as well. You can customize the filter to display an "Apply" button, which will wait to rerun the query until the "Apply" button is clicked.

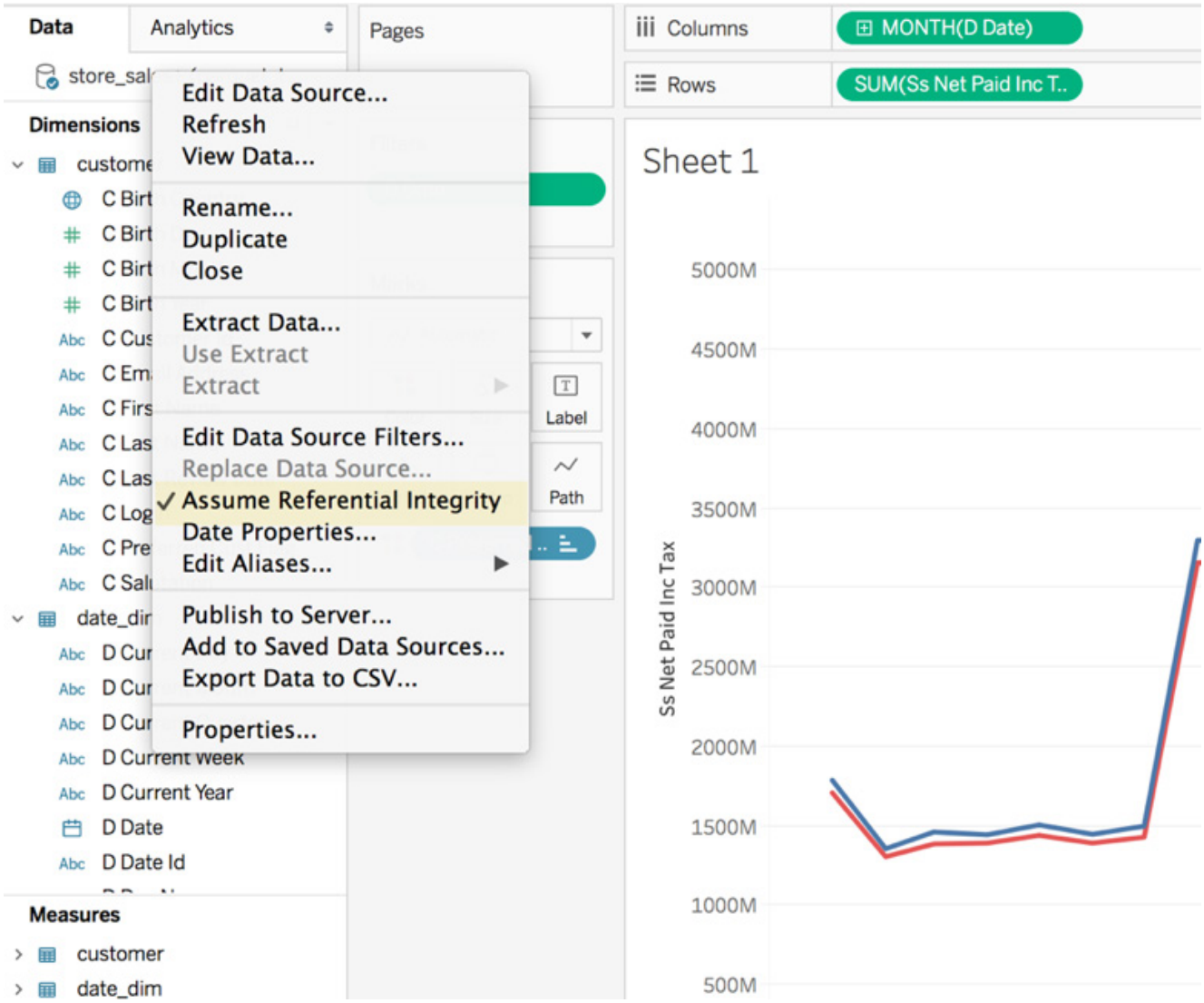
Tableau and joins in Amazon Redshift

Row based databases require joins heavily to accomplish data analytics, such that joins have become the norm in the business intelligence world. In Column based databases, less joins imply better performance, and as such you can get better throughput in Amazon Redshift with fewer joins, but in Tableau you will still need them.

Here are some tips to improve their performance.

In general, if you have related tables you want to join together, define those relationships with PRIMARY and FOREIGN KEYs in the database. Tableau can use this information to implement join culling, a process to simplify queries by using fewer JOINs, which allows Amazon Redshift to answer your questions faster.

Tableau users who are not database administrators can tell Tableau to "pretend" that the joins in the data source are backed up with Primary Keys and Foreign Keys. To do this, simply turn on "Assume Referential Integrity" in the Tableau data source.



For more on join culling and referential integrity, we recommend checking out 'Assuming Referential Integrity' in the online Tableau Help section.

Finally, when creating joins, make sure the columns you're joining are defined as NOT NULL in the Amazon Redshift table definition. If Tableau sees that a field used in a JOIN might contain nulls, Tableau will check the data for null values during the JOIN. This will cause Tableau to issue a query that is far more complex than necessary, and it will likely take longer to complete.

About Level of Detail Expressions/Calculations

Level of Detail Expressions are modifications to calculated fields within Tableau, and are often used to aggregate data a second time (e.g. taking the average of an average). These are extremely powerful calculations. However, they sometimes result in the generation of cross joins, which can negatively impact query performance in most databases, much less Amazon Redshift.

When using level of detail expressions, create a sort key on the dimension being used in the calculated field to improve their performance. You may need to view the actual underlying query to pinpoint the exact dimension (see the Optimizing Amazon Redshift, and Measuring Performance sections for more information).

Optimizing Amazon Redshift

There are two general areas to focus on when optimizing the Amazon Redshift side of your Amazon Redshift-Tableau deployment: Simplifying the database itself, and optimizing Amazon Redshift specifically for how you'll integrate it with Tableau.

For Amazon Redshift or any other columnar database for that matter, this means fewer joins, denormalizing table schemas, merging dimension tables into fact tables, and keeping columns sizes in tables as narrow as possible. All of these will improve query performance.

For the best integrations with Tableau, this means learning what analytical workflows you plan on promoting. Not all columns of data are of equal importance, and knowing which fields are used most often and are most critical will allow you to optimize Amazon Redshift to support prioritizing those fields through Sort Keys, Distribution Keys, and more.

If you're new to Amazon Redshift, we recommend reading the system overview, which explains many of the below concepts and considerations in great detail:

Here are some additional sections on performance tuning:

- Best practices for designing tables
- Tuning Query Performance
- Loading tables with automatic compression
- Increasing the available memory
- Best Practices for loading data
- Implementing workload management

Building out your Amazon Redshift cluster

When deploying Amazon Redshift, you have two sets of node types, each with two sizes, for a total of four cluster options. Each option will influence the capabilities of your cluster, specifically regarding compute power, memory, and storage. In turn, each capability influences the speed of your queries and the time it takes to display your Tableau dashboards.

Amazon Redshift supports:

- Dense storage nodes (DS2), which are useful for creating very large data warehouses using hard disk drives (HDDs). They are the most cost-effective and highest performance option for customers with tons of data that won't fit on DCs. Customers for whom performance isn't as critical or whose priority is reducing costs further can use the larger dense storage nodes and scale up to a petabyte or more of compressed user data for under $1,000/TB/year (3-year Reserved Instance pricing). Dense storage nodes come in two types: xlarge and 8xlarge.

- Dense compute nodes (DC2), which are useful for creating very high-performance data warehouses using fast CPUs, large amounts of RAM and solid-state disks (SSDs). They provide the highest ratio of CPU, memory and I/O to storage for customers whose primary focus is performance. DC2 is designed for demanding data warehousing workloads that require low latency and high throughput. Amazon Redshift has been tuned to leverage the better CPU, network, and disk on DC2 nodes, providing up to twice the performance of the first generation DC1 at the same price. Dense compute nodes come in two types: large and 8xlarge.

If you have a DC1.large Amazon Redshift cluster, you can simply restore to a new DC2.large cluster using an existing snapshot. To migrate from DS2.xlarge, DS2.8xlarge, or DC1.8xlarge Amazon Redshift clusters, you can use the resize operation to move data to your new DC2 cluster. For more information, see Clusters and Nodes in Amazon Redshift.

The DC2 family has a cost-effective smaller node for data warehouses less than 1TB where you would only require 1 node. The same query against the same data will perform significantly faster when using the dense compute class of nodes, resulting in more responsive Tableau workbook performance.

You can increase the capabilities of your cluster by scaling out the cluster and increasing the number of nodes, or by scaling up and changing the node size to a larger node. Your cluster sizing decisions will be influenced by how fast you want dashboards to load and how complex those dashboards are (i.e. how many queries need to be executed). Scaling your cluster will, of course, improve performance, but it must be the start of where

you spend time optimizing Amazon Redshift; without the additional tuning and optimizations below, you'll miss out on many additional performance improvements.

As a rule of thumb, we recommend Dense Compute Nodes (DC2) for their ability to run complex queries (e.g. queries with many joins) in an efficient manner across many nodes. Consider scaling out before scaling up.

The following image shows the Amazon Redshift console for launching a cluster:

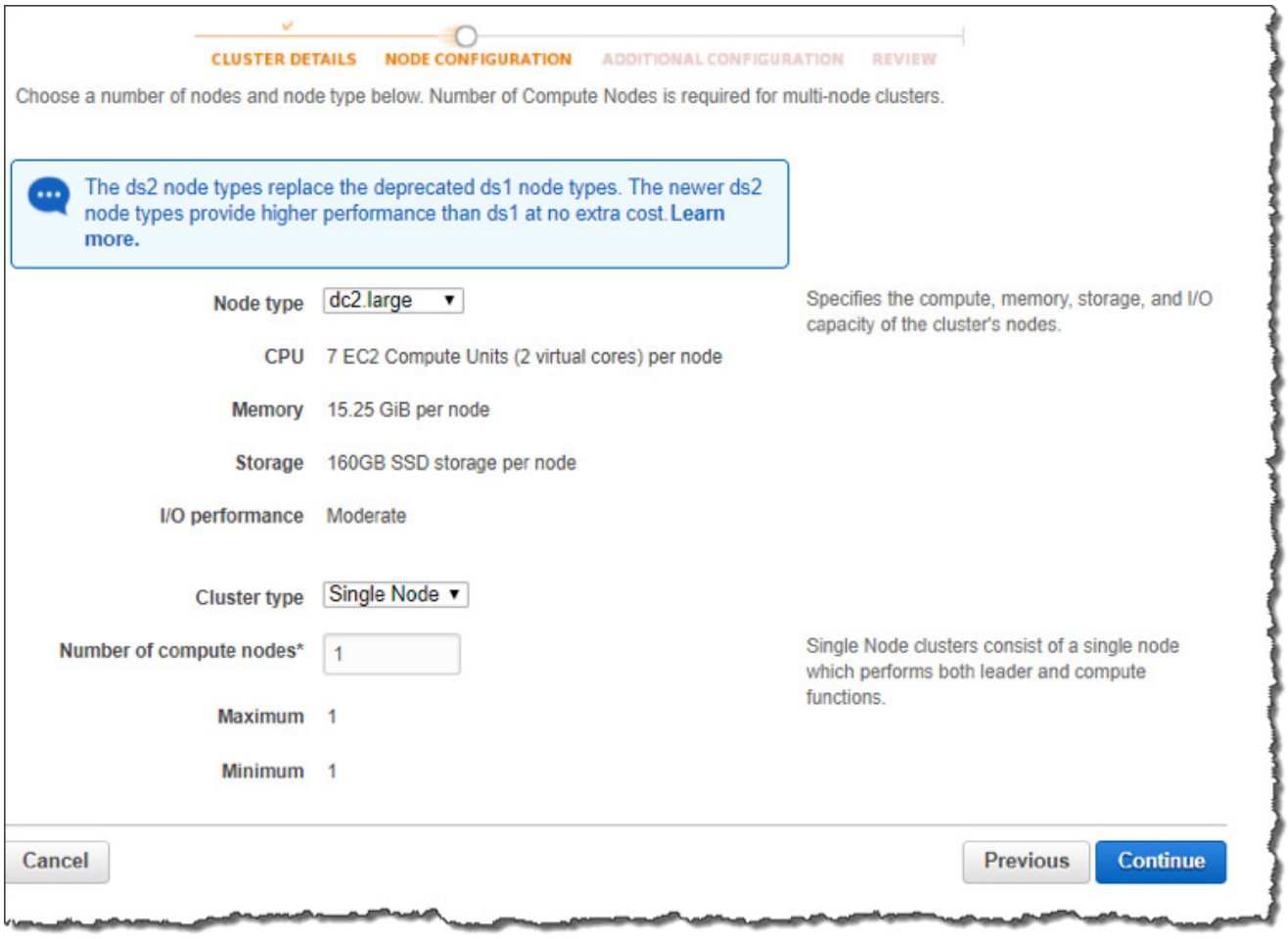


Figure 2: Improve query performance by adding a higher capacity node type.

For more information on node types and sizing options, see Amazon Redshift Clusters in the Amazon Redshift Management Guide.

Sort keys, distribution keys and compression

If there's only one thing you do after building your Amazon Redshift cluster, it should be tuning your Amazon Redshift tables with sort keys, distribution keys, and compression. We recommend the following two guides:

- Tutorial: Tuning Table Design
- Advanced Table Design Playbook

Sort keys

A sort key defines the order in which data is stored on disk. When data is sorted in a way that supports querying, Amazon Redshift scans less data and filters efficiently on the query predicates (your "WHERE" clause).

Sort keys are critical to an optimized Amazon Redshift deployment with Tableau. For example, if you have ten years of hospital patient visit data, every datetime-related query must scan all data to return the results. Placing a sort key on that date column allows Amazon Redshift to order the data by date, meaning time-related queries can skip dates that aren't relevant.

In Amazon Redshift, you can order data by multiple columns using two different methods:

1. Compound sort keys, which list columns used in the sort key within a specific order.
2. Interleaved sort keys, which give equal weight to all of the columns used in the sort key.

If you follow the above guidelines for optimizing Tableau, it should come as no surprise that we recommend using compound sort keys. One of Tableau's greatest strengths is building dashboards that promote an analytical workflow, answering one question and then showing more data to answer the next. Compound sort keys are perfect for this, in that they first sort the data according to one column, followed by subsequent columns within the sort key.

For example, take two fields in a sales dataset: Color and Product Type. There are likely only a handful of colors, but potentially thousands of product types. If you design a dashboard to encourage users to analyze sales first by Color and then by Product Type, you can easily create a compound sort key, choosing first the column Color, followed by Product Type. The result is drastically improved queries that filter the data to a specific color, and also queries that filter the data to a specific Color and Product Type. However, this sort key will have no effect if a query filters only on Product Type.

When a column specified for your sort key is highly selective (often called high cardinality), adding more columns to your Sort Keys provides little benefit and has a maintenance cost, so only add columns if selectivity is low. In this example, we can assume that Color has low cardinality so it's likely that adding Product Type will have a benefit. If you were to sort first by Product Type, it's unlikely that adding Color to the sort key will produce any performance gains.

Though they should be used sparingly, the benefit of interleaved sort keys is all columns in the key are given equal weight. Queries can choose to filter on any column, in any order, and potentially see a performance gain for doing so, at the cost of increased load and vacuum times. In our Color and Product Type example, an interleaved sort key allows queries to filter on either of the two columns, or any additional ones added to the sort key, though again, at an increased maintenance cost. In most analytical workflows, however, you almost always know which fields are most important and likely to be used first (e.g. Date or Product Type, over Product Model Number).

See Choosing Sort Keys in the Amazon Redshift Developer Guide for information that will help you decide which columns to designate for Sort Keys and Distribution Keys.

Here are a few tips:

- Put sort keys on the columns which are used as quick filters
- If dashboards query "recent data" more often, consider using the timestamp column as the lead column in a compound sort key.

Distribution keys

Choosing a distribution style and optimal distribution keys for your tables can significantly improve the performance of joins.

Amazon Redshift handles large amounts of data by parallelizing operations across multiple nodes, known as massively parallel processing. You can influence how this parallelization is implemented through three distribution styles (EVEN, KEY, or ALL) that define how data for a table is spread across the whole cluster. Tables can only have a single distribution key.

Combined with Sort Keys, carefully-planned Distribution Keys can lead to huge performance gains. For example, if you frequently join a table, specify the join column as both the sort key and the distribution key. This enables the query optimizer to choose a sort merge join instead of a slower hash join. Because the data is already sorted on the join key, the query optimizer can bypass the sort phase of the sort merge join, allowing Amazon Redshift to scan less data for each distinct item in the column.

See Choosing a data distribution method in the Amazon Redshift Developer Guide for information that will help you decide which columns to designate for Distribution Keys.

As their guide states, you should have two goals when distributing data:

1. Minimize the movement of data across nodes. If two tables are often going to be frequently joined together, load corresponding join data on the same node (i.e. basically distribute on the join keys) to reduce query time.

2. Evenly distribute data tables. Uneven distribution, also known as data distribution skew, means here's more data on one node than another, forcing some nodes in your cluster to do more work. This negatively impacts query performance. One easy way to test for data distribution skew is to visualize it in Tableau itself; bucket data in Amazon Redshift based on a potential distribution key field, and connect to it in Tableau.

A common distribution style for large tables is KEY. You specify one column in the table to be the KEY when you create the table. All the rows with the same key value always go to the same node. If two tables use the KEY distribution style, the rows from both tables with the same key go to the same node. This means that if you have two tables that are commonly joined, and the columns used in the join are the distribution keys, then joined rows will be collocated on the same physical node. This makes queries perform faster since there is less data movement between nodes. If a table, such as a fact table, joins with multiple other tables, distribute on the foreign key of the largest dimension that the table joins with. Remember to make sure that the distribution key results in relatively even distribution of table data.

ALL, the second distribution style, also promotes co-location of data on a join. This style distributes all the data for a table to all the nodes in the cluster. Replicating the data to each node has a storage cost and increases load time, but the tradeoff is that tables will always be local for any joins, which improves query performance. Good candidates for ALL distribution are any small dimension table, and specifically any slowly changing dimension tables in a star schema that don't share the same distribution key as the fact table.

If you do not choose a distribution style of KEY (by specifying DISTKEY when creating your table) or ALL (by specifying diststyle ALL when creating your table), then your table data is evenly distributed across the cluster. This is known as the EVEN distribution style.

See Choosing a data distribution method in the Amazon Redshift Developer Guide for information that will help you decide which columns in your tables you should designate for sort and distribution keys.

Compression

Compression settings can also play a big role when it comes to query performance in Amazon Redshift. Amazon Redshift optimizes data I/O and space requirements using columnar compression. It does this by analyzing the first 100,000 rows of data to determine the compression settings to use for each column when copying data into an empty table.

Most often you will want to rely upon the Amazon Redshift logic to automatically choose the compression type for you (strongly recommended). Advanced users can override these settings by specifying the compression scheme for each column when creating a table. Ensuring your columns are appropriately compressed leads to faster queries, because more data can be transferred with each read, and lower costs, because you may be able to house your data in a smaller cluster. See Choosing a column compression type and Loading tables with automatic compression in the Amazon Redshift Developer Guide for additional details on loading data with and controlling compression options.

While generally not recommended, you do have the option to turn off compression when you load data from Amazon Simple Storage Service into your tables by setting the COMPUPDATE option to OFF when using the COPY command.17. If the data stored in your tables varies over time as you incrementally load, then you can also periodically run Analyze Compression to ensure that you still have optimal compression settings.

Here are a few tips:

- Don't compress the first column in a compound sort key. You might end up scanning more rows than you have to as a result.
- Don't compress a column if it will act as a "single column sort key" (for the same reasons above).

Encryption

Certain types of applications with sensitive data require encryption of data stored on disk. Amazon Redshift has an encryption option that uses hardware-accelerated AES-256 encryption and supports user-controlled key rotation. Using encryption helps customers meet regulatory requirements and protects highly sensitive data at rest. Amazon Redshift has several layers of security isolation between end users and the nodes with the stored data. For example, end users cannot directly access nodes in an Amazon Redshift cluster where the data is stored. But even with hardware acceleration, encryption is an expensive operation that slows down performance by an average of 20%, with a peak overhead of as much as 40%.

Carefully determine if your security requirements require encryption beyond the isolation Amazon Redshift

provides, and only encrypt data if your needs require it. We also recommend testing perceived performance in Tableau with using encryption and without, and comparing the results.

For more information on encryption, see Amazon Redshift Database Encryption in the Amazon Redshift Management Guide.

Vacuum and Analyze Your Tables

After loading data that causes a significant number of additions, updates, or deletes, you should consider running a VACUUM command to optimize performance. VACUUM reclaims space for deleted items and ensures that the data is correctly sorted on disk (assuming your table defines a sort key). Vacuuming after load operations results in faster query execution. Note that the COPY command will sort the data when loading a table so you do not need to vacuum on initial load or sort the data in the load files.

Vacuuming can be an expensive operation, so you will want to run the command during off-peak times and increase the amount of memory available to the VACUUM command for faster execution. For more on VACUUM see Vacuuming tables in the Amazon Redshift Developer Guide.

In addition, analyzing tables after large data loads will increase query performance as it updates the statistical information that the query planner uses when building and optimizing a query plan. When loading data using the COPY command, specifying the STATUPDATE ON option automatically runs analyze after the data finishes loading.

Like VACUUM, the ANALYZE command is also resource intensive, so run it during off-peak times. If you run both VACUUM and ANALYZE, run VACUUM first because it affects the statistics generated by ANALYZE. For more information see Analyzing tables in the Amazon Redshift Developer Guide.

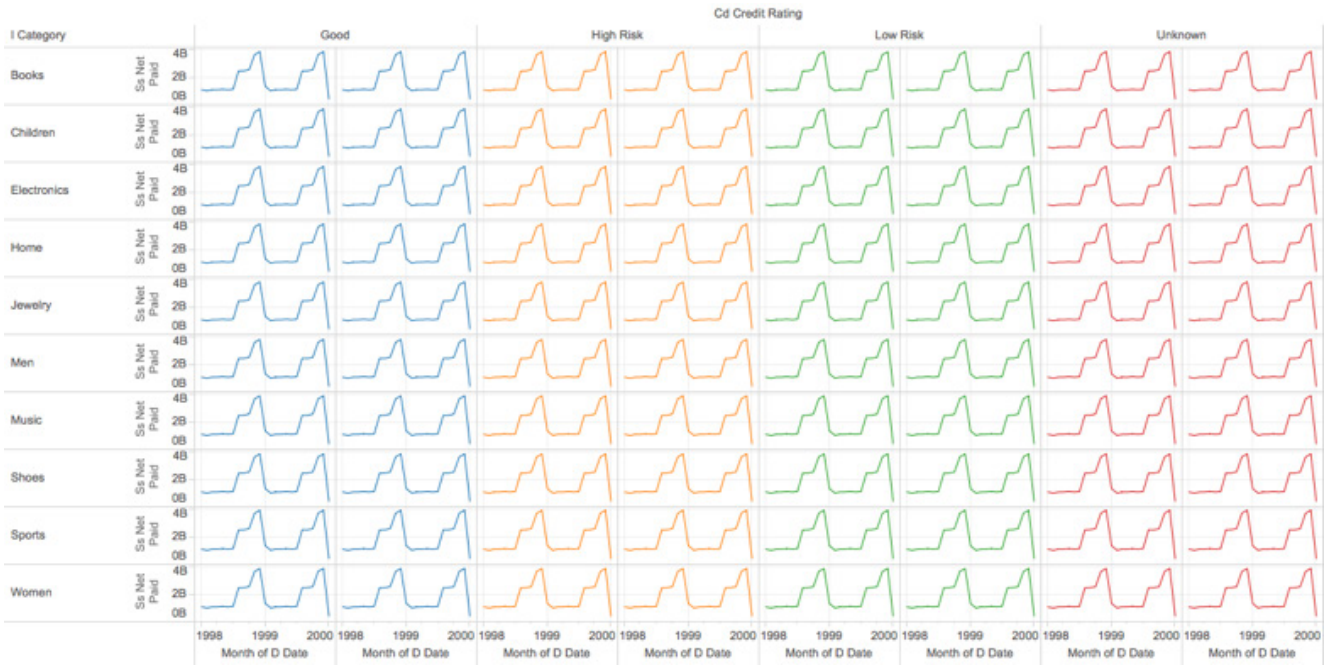
If you are loading multiple files into a table, and the files follow the ordering of the sort key, then you should execute COPY commands for the files in that order. For example, if you have 20130810.csv, 20130811.csv, and 20130812.csv representing three different days of data, and the sort key is by datetime, then execute the COPY commands in the order 20130810.csv, 20130811.csv, and 20130812.csv to prevent the need to vacuum.

An Amazon Redshift optimization example

Let's take two important optimization tips listed above, and compare query times for an identical visualization. Specifically, we'll look at:

- Simplifying the database schema
- Applying sort and distribution keys

Using a 3-billion row database in Amazon Redshift, let's generate the following visualization in Tableau.



You can use a number of different Amazon Redshift configurations, for the same data, to generate the above visualization. However, flattening the database schema to reduce the number of necessary joins significantly decreases on the query execution time. The entire query for both schemas is listed below.

- Complex schema: 205 seconds
- Simplified, flattened schema: 55 seconds

Additionally, we can apply sort and distribution keys on fields in the 'WHERE' clause (see full query below), spreading out and reducing the amount of work Amazon Redshift must accomplish. Adding sort keys and distribution keys "on top" of both our complex and simplified schema yields the following query execution times:

- Complex schema: 44 seconds
- Simplified, flattened schema: 37 seconds

In other words, optimizing your Amazon Redshift deployment can lead to exponential decreases in query execution time, which in turn "make Tableau fast".

Complex schema

(query time of 205 seconds, 44 seconds with sort and distribution keys)

```
SELECT
   "store_sales"."cd_credit_rating" AS "cd_credit_rating _store_sales_",
   "store_sales"."i_category" AS "i_category _store_sales_",
   SUM("store_sales"."ss_net_paid") AS "sum_ss_net_paid_ok",
   DATE_TRUNC( 'MONTH',
   CAST("date_dim"."d_date" AS TIMESTAMP WITHOUT TIME ZONE) ) AS "tmn_d_date_ok"
FROM
   "public"."store_sales" "store_sales"
LEFT JOIN
   "public"."date_dim" "date_dim"
      ON (
         "store_sales"."ss_sold_date_sk" = "date_dim"."d_date_sk"
      )
WHERE
   (
      ("store_sales"."cd_credit_rating" >= 'Good')
      AND (
         "store_sales"."cd_credit_rating" <= 'Unknown'
      )
      AND (
         (CASE "store_sales"."cd_education_status"
            WHEN '4 yr Degree' THEN '4 yr Degree'
            WHEN 'Advanced Degree' THEN '4 yr Degree'
```

```
            WHEN 'Primary' THEN 'Primary'
            WHEN 'Secondary' THEN 'Primary'
            WHEN '2 yr Degree' THEN 'Some College'
            WHEN 'College' THEN 'Some College'
            ELSE "store_sales"."cd_education_status"
        END) IN (
            '4 yr Degree', 'Primary', 'Unknown'
        )
    )
    AND (
        "store_sales"."i_category" >= 'Electronics'
    )
    AND (
        "store_sales"."i_category" <= 'Sports'
    )
    AND (
        DATE_TRUNC( 'MONTH', CAST("date_dim"."d_date" AS TIMESTAMP WITHOUT TIME ZONE) ) >=
(TIMESTAMP '2000-04-01 00:00:00.000')
    )
    AND (
        DATE_TRUNC( 'MONTH', CAST("date_dim"."d_date" AS TIMESTAMP WITHOUT TIME ZONE) ) <
(TIMESTAMP '2001-04-01 00:00:00.000')
    )
)
GROUP BY
    1,
    2,
    4
```

Simple schema

(query time of 55 seconds, 37 seconds with sort and distribution keys)

```
SELECT
   “customer_demographics”.”cd_credit_rating” AS “cd_credit_rating”,
   “item”.”i_category” AS “i_category”,
   SUM(“store_sales”.”ss_net_paid”) AS “sum_ss_net_paid_ok”,
   DATE_TRUNC( ‘MONTH’,
   CAST(“date_dim”.”d_date” AS TIMESTAMP WITHOUT TIME ZONE) ) AS “tmn_d_date_ok”
FROM
   “public”.”store_sales” “store_sales”
LEFT JOIN
   “public”.”customer_demographics” “customer_demographics”
      ON (
         “store_sales”.”ss_cdemo_sk” = “customer_demographics”.”cd_demo_sk”
      )
LEFT JOIN
   “public”.”date_dim” “date_dim”
      ON (
         “store_sales”.”ss_sold_date_sk” = “date_dim”.”d_date_sk”
      )
LEFT JOIN
   “public”.”item” “item”
      ON (
         “store_sales”.”ss_item_sk” = “item”.”i_item_sk”
      )
WHERE
   (
```

```
((CASE "customer_demographics"."cd_education_status"
    WHEN '4 yr Degree' THEN '4 yr Degree'
    WHEN 'Advanced Degree' THEN '4 yr Degree'
    WHEN 'Primary' THEN 'Primary'
    WHEN 'Secondary' THEN 'Primary'
    WHEN '2 yr Degree' THEN 'Some College'
    WHEN 'College' THEN 'Some College'
    ELSE "customer_demographics"."cd_education_status"
END) IN ('4 yr Degree', 'Primary', 'Unknown'))
AND (
    "customer_demographics"."cd_credit_rating" IN (
        'Good ', 'High Risk ', 'Low Risk ', 'Unknown '
    )
)
AND (
    "item"."i_category" >= 'Electronics'
)
AND (
    "item"."i_category" <= 'Sports'
)
AND (
 DATE_TRUNC( 'MONTH', CAST("date_dim"."d_date" AS TIMESTAMP WITHOUT TIME ZONE) ) >=
(TIMESTAMP '2000-04-01 00:00:00.000')
)
AND (
    DATE_TRUNC( 'MONTH', CAST("date_dim"."d_date" AS TIMESTAMP WITHOUT TIME ZONE) ) <
(TIMESTAMP '2001-04-01 00:00:00.000')
)
```

)

GROUP BY

1,

2,

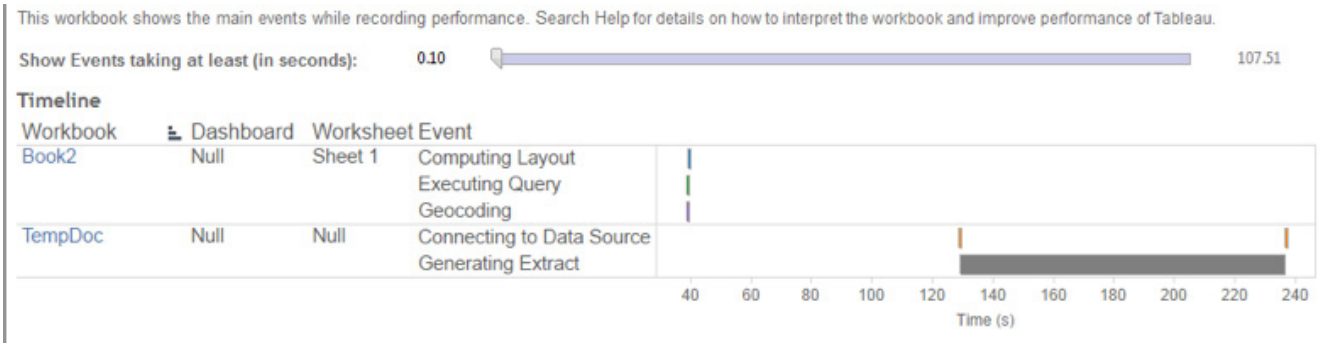
4

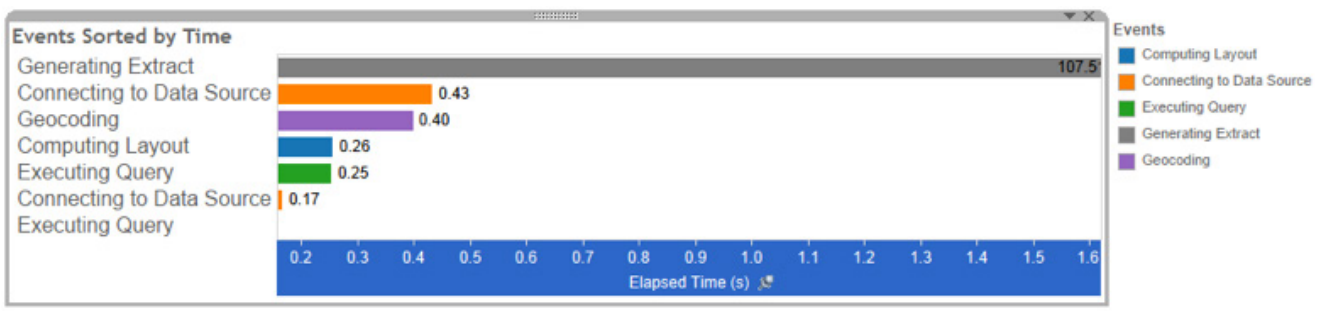
Measuring performance between Amazon Redshift and Tableau

The best way to successfully deploy Amazon Redshift and Tableau together is to measure the performance of your deployment—query and dashboard load times—and track the impact of each and any modifications you make. Tableau provides two simple options to understand how your workbooks are performing and where they take the longest amount of time to load.

Tableau Performance Recorder

Tableau comes with a built-in performance recorder, which reports various details about the sheets within a workbook, including total load time and how much of it was spent executing queries versus computing the correct display layout.

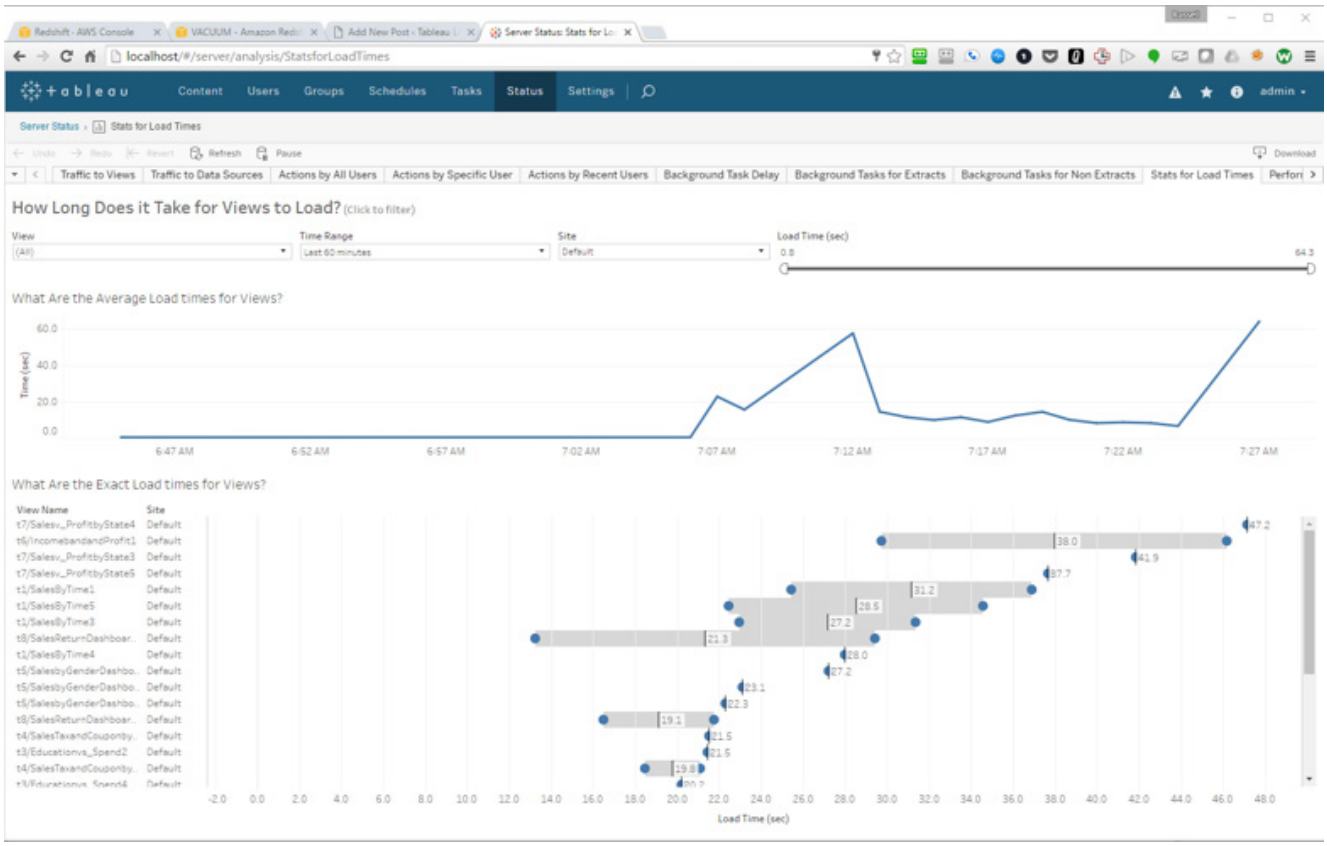




For more detailed documentation on Tableau performance recorder, visit the Performance section of Tableau's Help page.

Tableau Server Admin Views

Every instance of Tableau Server comes with built-in administrative dashboards that, among several metrics, report the load times for each view over time. Specifically, you can use the "Stats for Load Times" view under "Status."



Cursors and viewing query text data in the Amazon Redshift Console

If you're looking to go even deeper in measuring performance, Amazon Redshift comes with a console that allows you to view the actual queries a client sends to Amazon Redshift. However, since Tableau uses cursors with Amazon Redshift, you will only see the cursor executing – not the plain-text SQL. For more detail about how Tableau uses cursors in Amazon Redshift, please see the 'More about Cursors' section in 'Other Tips.'

You can turn cursors off by using a Tableau Data Customization (TDC), but this will cause ALL rows to be delivered to Tableau simultaneously, potentially maxing out the RAM on your machine.

To create a TDC and turn off cursors, open the XML of your data source by opening the workbook or data source using a text editor. You'll see a section that looks like:

```
<named-connections>

  <named-connection caption='foo.foo.ap-southeast-1.Amazon Redshift.amazonaws.com' name='Amazon Redshift.1foo'>

    <connection class='Amazon Redshift' dbname='tpchdslitev1' odbc-connect-string-extras='' one-time-sql='' port='5439' schema='public'

     server=foo.foo.ap-southeast-1.Amazon Redshift.amazonaws.com' single-node='no' sslmode='' username='foo' /

   </named-connection>

 </named-connections>
```

Which you should alter by adding in **the following**:

```
<named-connections>

  <named-connection caption='foo.foo.ap-southeast-1.Amazon Redshift.amazonaws.com' name='Amazon Redshift.lfoo'>

    <connection class='Amazon Redshift' dbname='tpchdslitev1' odbc-connect-string-extras='UseDeclareFetch=0' one-time-sql='' port='5439'

    schema='public' server='foo.foo.ap-southeast-1.Amazon Redshift.amazonaws.com' single-node='no' sslmode='' username='root'>

      <connection-customization class='Amazon Redshift' enabled='false' version='10.1'>

       <vendor name='Amazon Redshift' />

       <driver name='Amazon Redshift' />

        <customizations>

          <customization name='odbc-connect-string-extras' value='UseDeclareFetch=0' />

        </customizations>

      </connection-customization>

    </connection>

  </named-connection>

</named-connections>
```

Once you make that change to remove cursors, you'll now see Tableau's queries in the AWS console, and can make the right adjustments accordingly.

A few other considerations

More about cursors

Tableau uses cursors when returning queried data from Amazon Redshift. Using cursors lets Tableau retrieve large data sets efficiently by retrieving results a chunk at a time rather than all at once, reducing the amount of memory consumed.

Despite allowing you to retrieve more data than would otherwise be possible, cursors do come with some performance side effects. Cursors force all data to be streamed to the Leader Node before returning data to Tableau, potentially leading to slower response times.

Amazon Redshift also sets a limit on the space allocated to cursors on each node depending on the node type. For example, a single Dense Compute node (DC1) has a maximum result set of 16000 MB. If you exceed this limit, the query will fail, and it's possible to cause any other queries running on the same node to fail as well.

You can turn cursors off, but doing this will cause Amazon Redshift to try to return all the data at once, which could cause potential out of memory errors on the client-side. It's not uncommon for datasets in Amazon Redshift to be too big for you to retrieve without the use of a cursor.

Amazon Redshift places limits on the total size of the returned results per cluster since open cursors consume resources on the cluster. The maximum result size limits depend upon the type of the cluster node, and whether the cluster is a single node or multiple nodes. You can tradeoff between the maximum total result size allowed per cursor and the number of concurrent cursors permitted by modifying the maximum result size per cursor. For example, by default a dw2.large cluster running with 2 or more nodes allows a total maximum result size of 384 GB per cluster, which permits two cursors, with a maximum result set size of 192 GB per cursor. If you decide that the maximum result size required is 32 GB, then you can increase the number of concurrent cursors available by setting max_cursor_result_set_size to 32000 (MB) (384000/32000=12). The number of concurrent cursors cannot exceed the maximum number of concurrent queries.

Amazon Redshift limits the concurrent queries and adjusts the max_cursor_result_set_size accordingly if the value you set causes the concurrent cursor number to be higher.

See Cursor constraints in the Amazon Redshift Developer Guide for more information and limits.

For more information on working with cursors and Tableau, see Working with Amazon Redshift Concurrent Cursor Limit in the Tableau Knowledge Base.

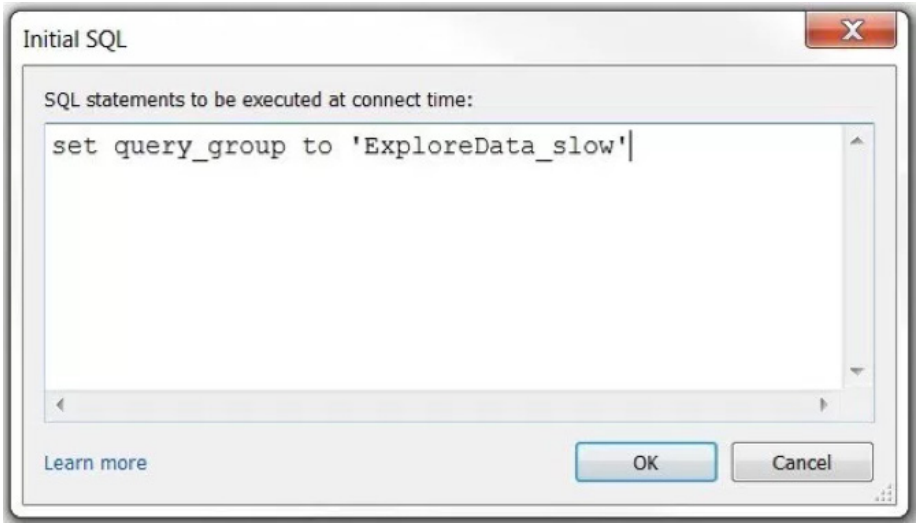
Workload Management Queues

Amazon Redshift allows you to manage query execution via Workload Management (WLM) queues. WLM queues manage how many concurrent queries are executed and how much of your cluster's RAM a query can consume. By default, each Amazon Redshift cluster has a single WLM queue which allows a maximum of five concurrent queries to run. This number can be modified, though Amazon Redshift currently advises that the maximum be set to no more than 15 concurrent queries for the entire cluster. Each WLM queue can also be configured to consume a set amount of your cluster's RAM.

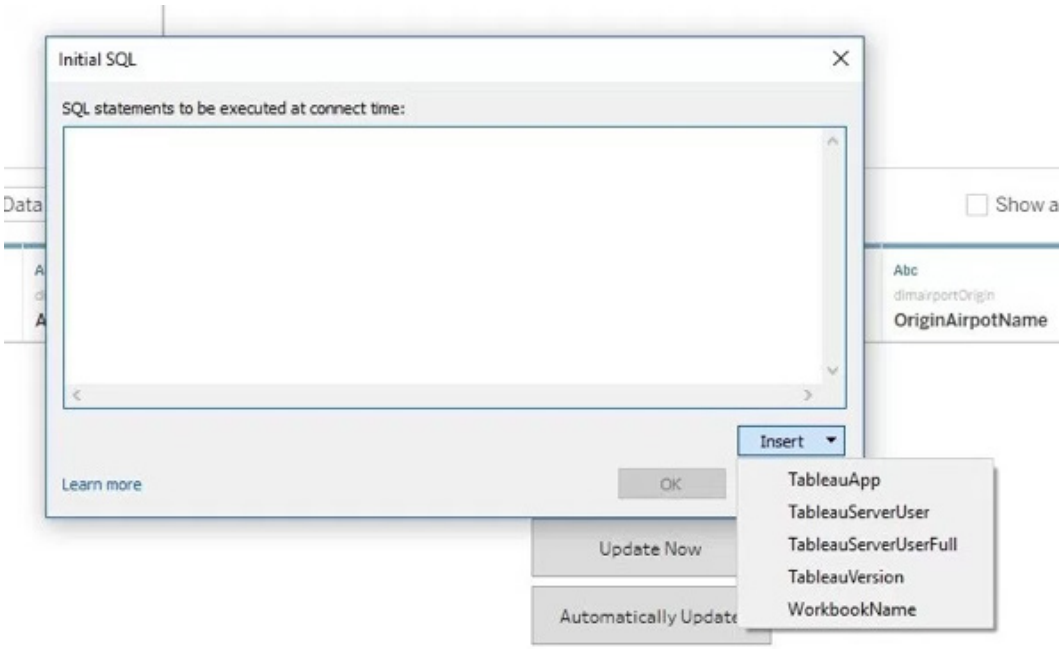
You can also use the new Amazon Redshift query monitoring rules feature to set metrics-based performance boundaries for workload management (WLM) queues, and specify what action to take when a query goes beyond those boundaries. For example, for a queue that's dedicated to short running queries, you might create a rule that aborts queries that run for more than 60 seconds. To track poorly designed queries, you might have another rule that logs queries that contain nested loops. AWS also provides pre-defined rule templates in the Amazon Redshift management console to get you started.

You can create additional WLM queues and pair them with Tableau's Initial SQL feature. Doing so will allow you to place specific queries into particular WLM queues, with simpler queries going to a "fast" queue with less dedicated RAM, and more complex queries going to a "slow" queue with more RAM.

Once you've created and configured different WLM queues, create multiple data connections in Tableau to the same Amazon Redshift database. When connecting, use the Initial SQL feature to execute a **SET query_group** statement This tells Amazon Redshift which WLM queue to send the query to:



Initial SQL also supports the use of parameters, including variables such as the name of a Tableau Workbook, or the name of a Tableau Server user. You can effectively use these parameters to force the work of certain Tableau Workbooks into a "high-priority" (or "low-priority") WLM queue. You can also direct queries run by *certain users* to a specific WLM queue.



For more on Amazon Redshift Workload Management Queues, see Tutorial: Configuring Workload Management (WLM) Queues to Improve Query Processing

For more on Tableau and Initial SQL, see Run Initial SQL in Tableau's Online Help section.

Amazon Redshift and Tableau Extracts

While querying, Tableau can leverage live connections against Amazon Redshift or use Tableau Data Extracts.

Data extracts can help you avoid challenges around concurrency in Amazon Redshift. They also allow you perform pre-aggregation, which is valuable. Consider and test benefits of the TDE for your deployment: We recommend you only extract small portions of your Amazon Redshift database. Choose "slices" of data that are relevant to most of your users.

Here are some things to consider if you pursue this path:

- Aggregate extracts if you can. Tableau provides a number of ways to do this, including removing fields that aren't used within a workbook, or rolling date fields up to a higher-level-of-detail (e.g. months, instead of seconds).

- Avoid scheduling multiple extract refreshes in parallel to avoid hitting the open cursor size limit.
- Check the size of the data you will extract in advance. Assuming you leverage cursors, Amazon Redshift results are materialized on the Leader node of your cluster. This will slow down the extract process. There is a maximum amount of data that can be materialized on the Leader node, depending on the size and type of nodes in your cluster.
- Consider other users of your Amazon Redshift cluster. If you use all your cursor space on the Amazon Redshift cluster and cause an error, anyone else currently using cursors on the same cluster will encounter errors as well.
- As mentioned above, large node sizes provide more cursor space. Consider moving from large to 8xlarge nodes.

Amazon Redshift Spectrum

We announced an update to our Amazon Redshift connector with support for Amazon Redshift Spectrum (external S3 tables) in Nov 2017. This feature was released as part of Tableau 10.3.3 and will be available broadly in Tableau 10.4.1. In Tableau, customers can now connect directly to data in Amazon Redshift and analyze it in conjunction with data in Amazon Simple Storage Service (S3).

How does support for Amazon Redshift Spectrum help customers?

Many Tableau customers have large buckets of data stored in Amazon S3. If this data needs to be accessed frequently and stored in a consistent, highly structured format, then you could provision it to a data warehouse like Amazon Redshift. If you want to explore this S3 data on an ad hoc basis—to determine whether or not to provision it and where—you could use Amazon Athena, a serverless interactive query service from AWS that requires no infrastructure setup and management.

But what if you want to analyze both the frequently-accessed data stored locally in Amazon Redshift AND your full data sets stored in Amazon S3?

What if you want the throughput of disk and sophisticated query optimization of Amazon Redshift AND a service that combines a serverless scale-out processing capability with the massively reliable and scalable S3 infrastructure?

What if you want the super fast performance of Amazon Redshift AND support for open storage formats (e.g. Parquet, ORC) in S3?

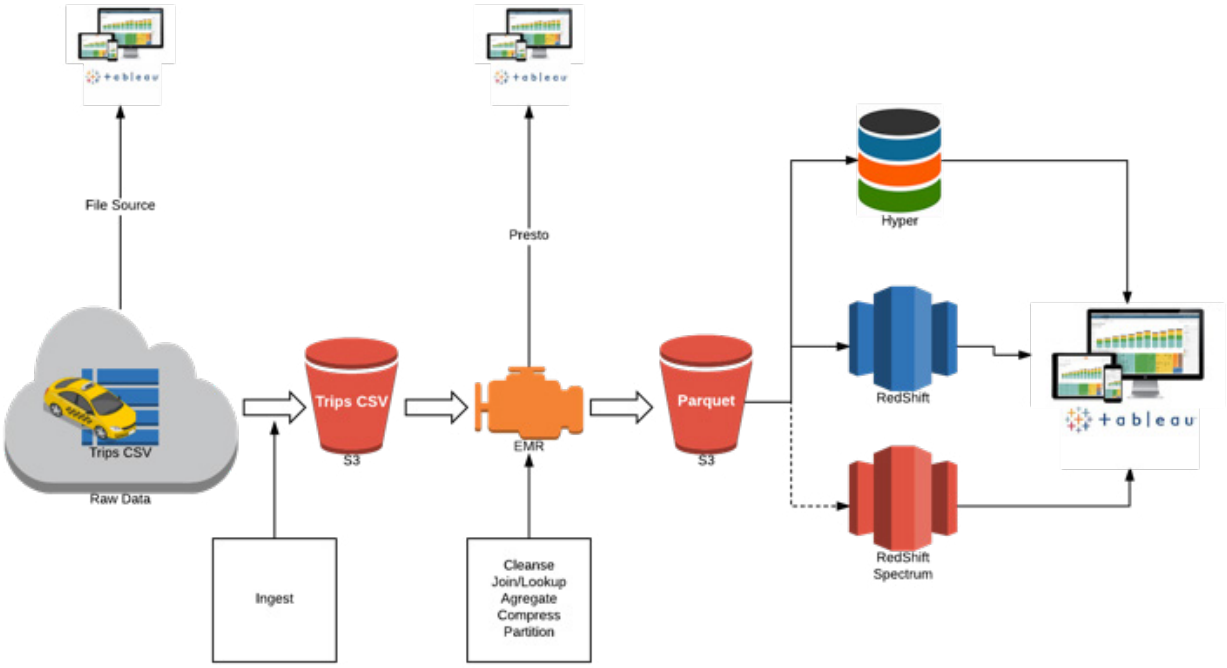
To enable these "ANDs" and resolve the tyranny of OR's, AWS launched Amazon Redshift Spectrum in early 2017.

Amazon Redshift Spectrum provides the freedom to store data where you want, in the format you want, and have it available for processing when you need it. Since the Amazon Redshift Spectrum launch, Tableau has worked tirelessly to provide best-in-class support for this new service, allowing customers to extend their Amazon Redshift analyses out to the entire universe of data in their S3 data lakes

Let's explore how Tableau works with Amazon Redshift Spectrum. In this example, I'll also show you how and why you might want to connect to your AWS data in different ways, depending on your use case.

The experiment

I'm using the following pipeline to ingest, process, and analyze data with Tableau on an AWS stack.



In this example, I'll use the New York City Taxi data set as the source data. The data set has nine years' worth of taxi rides activity—including pick-up and drop-off location, amount paid, payment type—captured in 1.2 billion records.

The data lands in S3. It is cleansed and partitioned via Amazon EMR and converted to an analytically optimized columnar Parquet format.

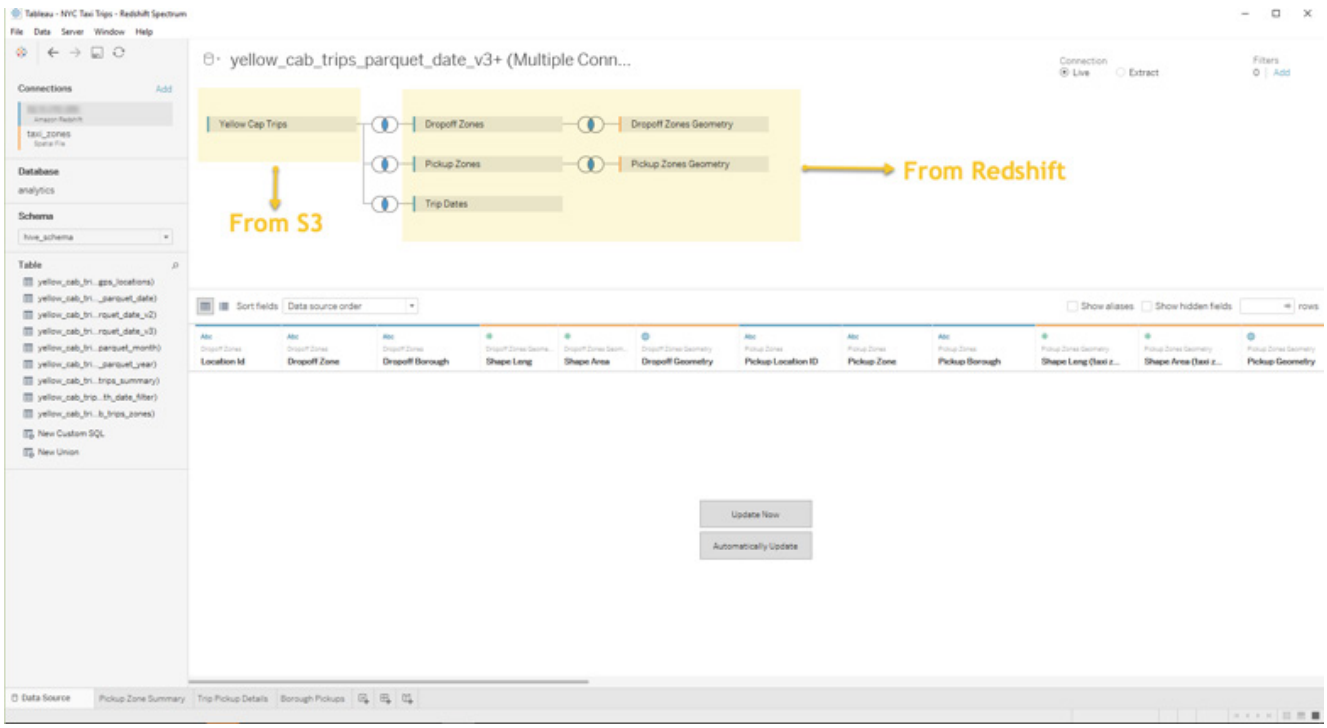
Note that you can point Tableau to the raw data in S3 (via Amazon Athena) as well as access the cleansed data with Tableau using Presto via your Amazon EMR cluster. Why might you want to use Tableau this early in the

pipeline? Because sometimes you want to discover what's out there and understand some questions worth asking before you even start the analysis.

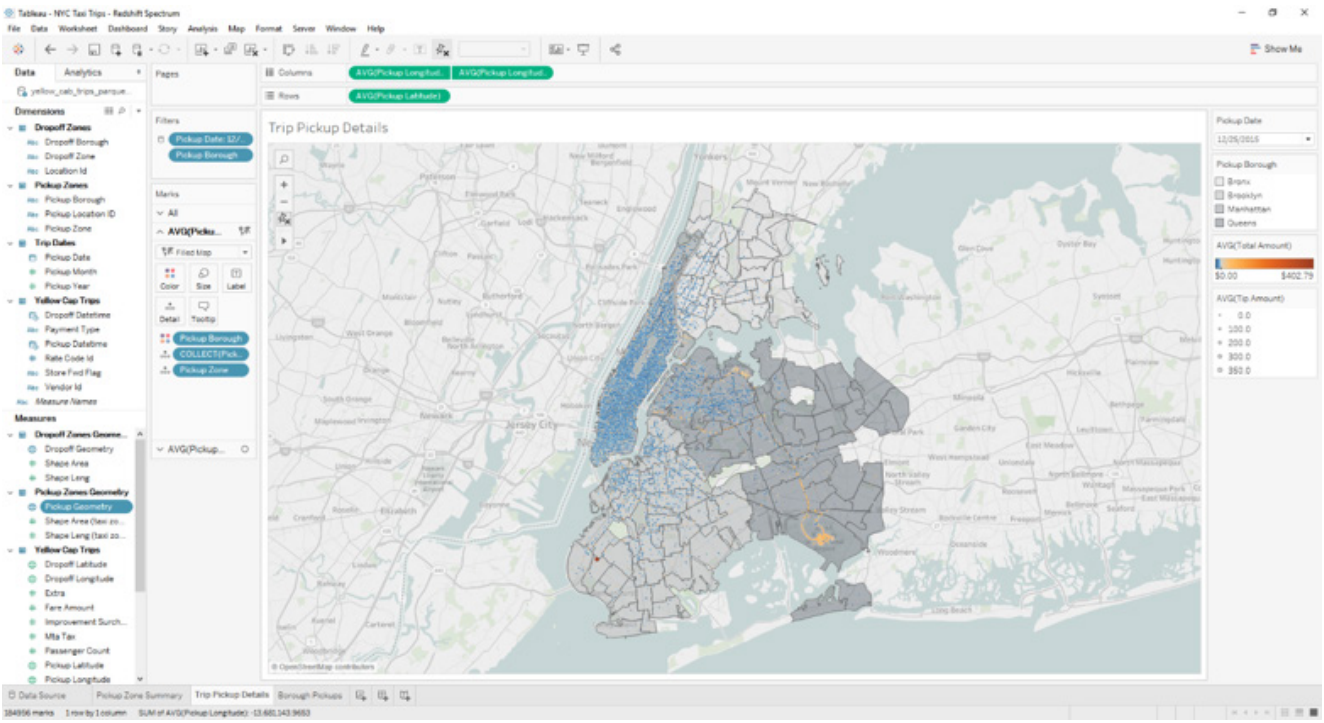
Once you discover those questions and determine if this sort of analysis has long-term advantages, you can automate and optimize that pipeline, adding new data as soon as it arrives so you can get it to the processes and people that need it. You may also want to provision this data to a highly performant "Hotter" layer (Amazon Redshift or a Tableau extract) for repeated access.

As represented in the flow above, S3 contains the raw, denormalized taxi ride data at the timestamp level of granularity. This is the fact table. Amazon Redshift has the time dimensions broken out by date, month, and year, along with the taxi zone information.

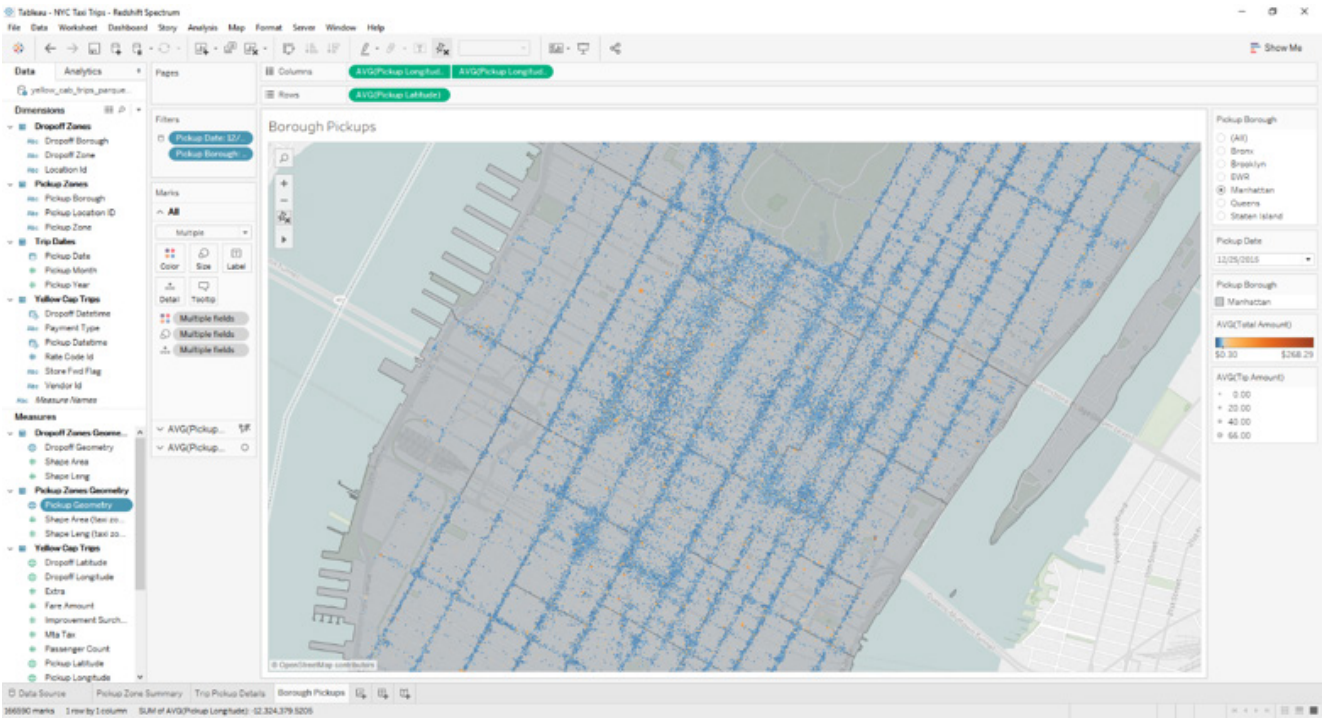
Now let's imagine that I'd like to know where and when taxi pickups happen on a certain date in a certain borough. With support for Amazon Redshift Spectrum, I can now join the S3 tables with the Amazon Redshift dimensions.



I can then analyze the data in Tableau to produce a borough-by-borough view of NYC ride density on Christmas Day 2015.



I could also hone in on just Manhattan to identify pick up "hotspots" where ride charges appear way higher than average.



With Amazon Redshift Spectrum, you now have a fast, cost-effective engine that minimizes data processed with dynamic partition pruning. Further improve query performance by reducing the data scanned. You could do this by partitioning and compressing data and by using a columnar format for storage.

At the end of the day, your choice of data source that you connect to in Tableau should be based

on what variable you want to optimize for. For example, you may choose to connect live to Amazon Athena, Amazon Redshift, Amazon Redshift Spectrum, or bring a subset of your data into a Tableau extract.

Start by considering:

- Cost: Are you comfortable with the serverless cost model of Amazon Athena and potential full scans vs. the advantages of no set up?
- Performance: Do you want the throughput of local disk?
- Setup effort and time: Are you okay with the lead time of an Amazon Redshift cluster setup vs. just bringing everything into a Tableau extract?

To meet the many needs of our customers, Tableau's approach is simple: it's all about choice. This includes how you choose to connect to and analyze your data.

For more on how to approach data architecture decisions for the enterprise, watch this Big Data Strategy session my friend Robin Cottiss and I delivered at Tableau Conference 2017. We share several examples of companies leveraging the Tableau on AWS platform and provide a detailed run-through of the aforementioned demonstration.

In addition, please also check out the ten best practices for Amazon Spectrum guidance from AWS. Among other things this provides recommendations to improve scan-intensive concurrent workloads, optimize storage, and configure your cluster, all with an eye to improving performance.

Conclusion

Tableau Software and Amazon Redshift are two technologies that can provide a business intelligence platform for today's business users, users who demand responsive and visually compelling solutions. Both tools are powerful, and keeping in mind these performance tips and techniques will help you understand how to optimize the two tools together.

Tableau can turn any business user into a self-driven, question-and-answer superhero. Remember to design dashboards with analytical workflows that focus on the right questions, and reduce the queries that Tableau must execute.

Amazon Redshift allows anyone to deploy a database into the millions and billions of rows, all without procuring hardware and at a fraction of the cost of traditional database administration. Remember that it is not a high-concurrency database; simplifying the schema and optimizing tables for the workflows you created in Tableau will make Amazon Redshift more efficient.

Above all, test and measure the performance of the two technologies together. Make changes to each, and test and measure them again. That's how you will deliver a great user experience when using Tableau Software with Amazon Redshift.