Optimizing your Amazon Redshift and Tableau Deployment for Better Performance

Table of contents

Updated November 2020

Introduction

Amazon Redshift and Tableau Software are two powerful products in a modern analytics platform. With Amazon Redshift, you can create a massively scalable, cloud-based data warehouse in just a few clicks. Combined with the real-time responsiveness of Tableau, you can gain insights from that data just as easily. Tableau natively connects to Amazon Redshift for advanced speed, flexibility, and scalability, accelerating results from days to seconds.

From Box and Skyscanner to Pearson, thousands of customers are using Tableau on Amazon Redshift to analyze massive amounts of data everyday with speed and agility and get the answers they need to drive strategic action.

“

With Tableau, you just hook it up to the Redshift server, connect, run a query, and publish it to the Server and you're literally done in an hour. It's great—it feels like one product.

ABHISHEK GUPTA
SENIOR ANALYST, BOX

“

Deploying Tableau on Amazon Redshift was an enterprise-wide transformation exercise for Pearson. Tableau on Amazon Redshift provided next-generation data warehousing and analytics capabilities that allowed us to rapidly increase Tableau usage across the organization, resulting in operational efficiencies, more strategic programs and partnerships through evidence-based decision-making, and ultimately, driving a data-enabled culture.

JASON LOKKESMOE
AVP, BIG DATA & ANALYTICS BUSINESS DEVELOPMENT, PEARSON

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.

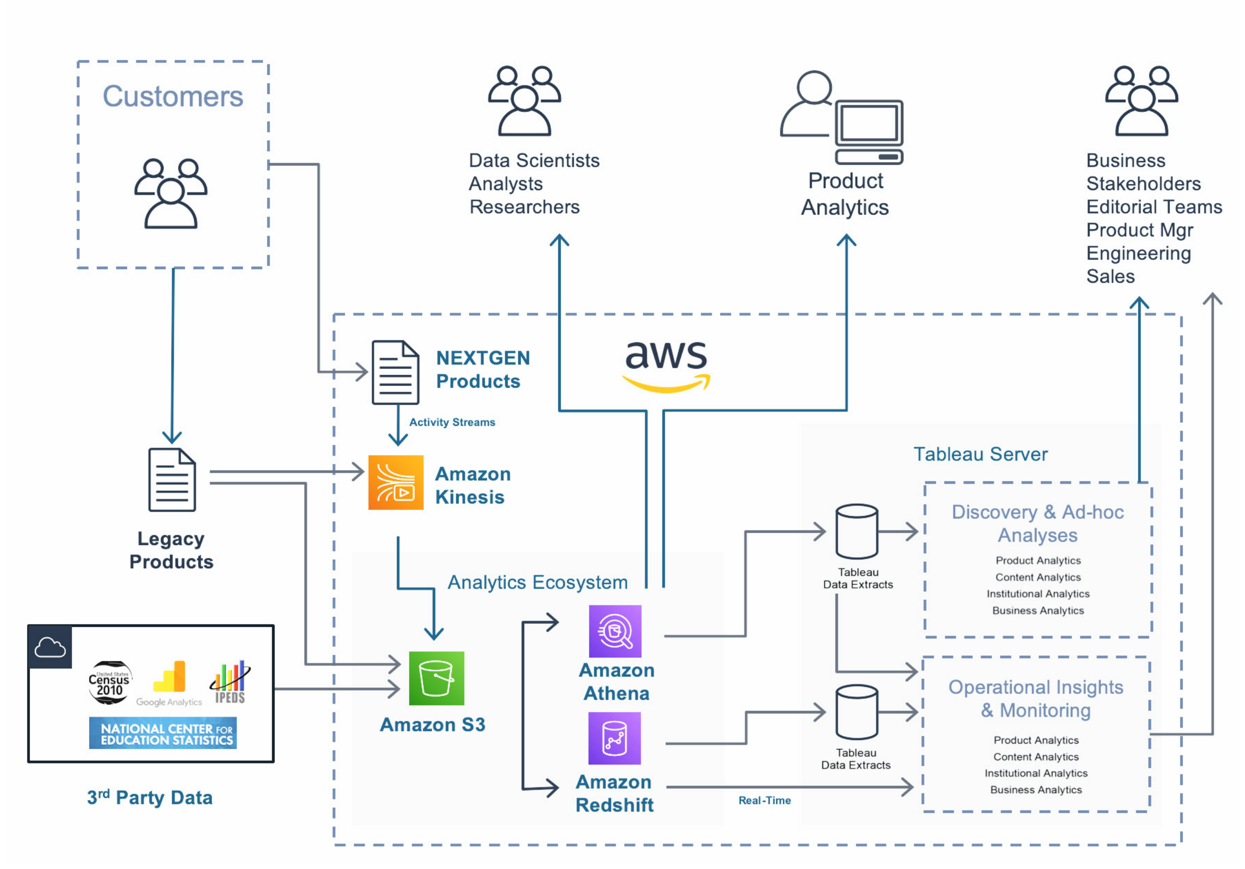
Tableau on Amazon Web Services

Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform, offering over 165 fully featured services from data centers globally. Millions of customers—including the fastest-growing startups, largest enterprises, and leading government agencies—trust AWS to power their infrastructure, become more agile, and lower costs. Tableau integrates with AWS services to empower enterprises to maximize the return on your organization's data and to leverage their existing technology investments. It all starts with direct connections to Amazon data sources including Amazon Redshift (including Redshift Spectrum), Amazon Aurora, Amazon Athena and Amazon EMR.

Thanks to these market leading integrations, Tableau is the natural choice of platform for analyzing the data stored in Amazon's data sources. Beyond this, Tableau provides the depth and breadth of capabilities to ensure that data can be confidently deployed across the entire enterprise. Tableau Server runs seamlessly in AWS's cloud infrastructure so organizations that prefer to deploy applications on Amazon Web Services have a complete solution offering from Tableau.

An example of a customer who has successfully implemented Amazon Redshift and Tableau is Pearson. Pearson offers educational courseware, assessments, and a range of teaching and learning services powered by technology. The global company works with other learning institutions, such as K-12 schools, community colleges, and businesses, as its clientele. At Pearson, data is spread across many sources and silos—yet uncovering that data is critical to the success of the organization and its clients. Pearson wanted to increase the likelihood of deriving insight from data, reduce time to insight, and simplify their architecture. At the most fundamental level, Pearson needed access to better data, faster.

Using Tableau and AWS, Pearson can better scale its data as its user base grows. Just a few years ago, Pearson had roughly 500 Tableau users; now, more than 7000 users across different departments including Legal, Facilities, Product, Marketing, HR, Supply Chain, Tech, Sales, and Product Analytics use the solution. Jason Lokkesmoe, AVP, Big Data & Analytics Business Development at Pearson, indicates that using Tableau and AWS, the company has saved an estimated 68K hours across 900 projects over the past three years—work that used to require four engineers up to two weeks on average to complete. For more on this use case watch this recorded webcast.



About Amazon Redshift

Amazon Redshift is the most popular, fast, scalable, and simple cloud data warehouse 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 analytics platform 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.

Drivers

Although Tableau desktop has the Redshift driver, we still encourage users to upgrade when new versions of the Redshift drivers are available. Running an application on the latest driver will ensure that the customers will get better performance, bugs recovery, and better security features. Visit the Drivers page of the Tableau website to get the latest driver version. J Just filter for Data Source "Amazon Redshift" and the desired OS.

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).

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 our Help article on Data Aggregation in Tableau 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 materialized view 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. So, it is recommended to try and avoid a lot of sheets on the same dashboard to get better performance for end users.

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.

So, our guidance is to err on the side of sending fewer queries to allow for better latency. It's generally recommended that you limit the number of items or quick filters on a dashboard to ensure that not too many queries are running concurrently on the cluster, potentially degrading the overall dashboard load times. Also, having a higher number of sheets on a dashboard can be difficult to see the important details if you 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. 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.

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 get 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 the Help article Creating Sets.

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.

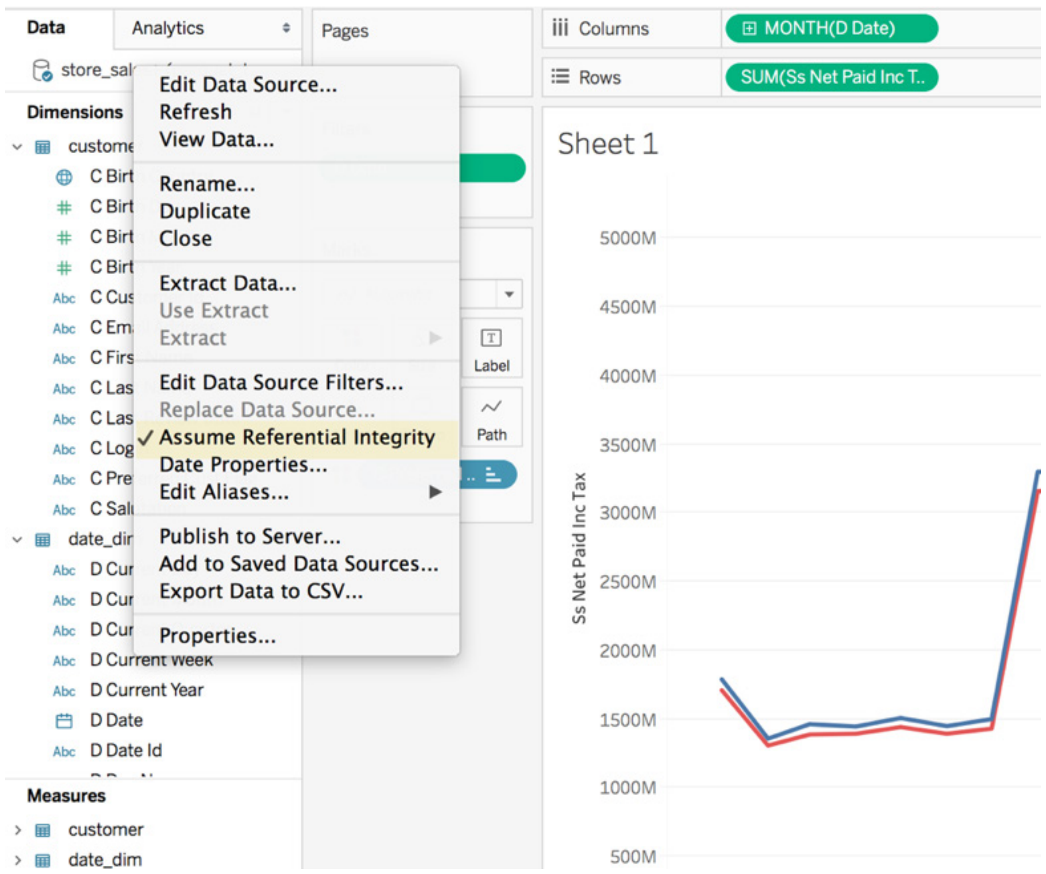
Tableau and joins in Amazon Redshift

Row-based databases require joins to accomplish data analytics, such that joins have become the norm in the business intelligence world. In column-based databases, fewer joins imply better performance. 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. Only do this if the relationships hold in your database, because if you give Tableau incorrect information, it may show incorrect results.



✓ Assume Referential Integrity

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 more complex, 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 by using the good data model and optimizing Amazon Redshift Cluster 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:

Optimizing the database and data model

Remove unused columns and opt for additional tables instead of egregiously wide ones. This is because Redshift is a columnar database and data is stored on disk column by column rather than row by row like you would in a traditional database like Postgres. This comes with a huge performance advantage as analytical queries only deal with a subset of columns. Therefore, we would only scan certain blocks for relevant columns and are able to reduce the amount of I/O that needs to happen. This also explains why "select *" type queries are inefficient and unnecessary as Redshift has to do additional work to access all columns causing heavy disk I/O.

You must also use the smallest possible column size. Why? It's because when executing queries, Redshift allots memory based on declared column width, not actual size of data in the column. Wider-than-necessary columns waste RAM and decrease the number of rows that can be loaded into memory. This results in a higher chance queries might spill to disk and slow down.

Here are some additional resources on performance tuning and database design:

- Best practices for designing tables
- Tuning query performance
- Loading tables with automatic compression
- Increasing the available memory
- Best Practices for loading data
- Working with Advisor
- Implementing workload management

Building out your Amazon Redshift cluster

When deploying Amazon Redshift, there are three sets of node types, each with two sizes, for a total of six 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 node types

RA3 nodes with managed storage enable you to optimize your data warehouse by scaling and paying for the compute and managed storage independently. With RA3, you choose the number of nodes based on your performance requirements and only pay for the managed storage that you use. Size your RA3 cluster based on the amount of data you process daily. You launch clusters that use the RA3 node types in a virtual private cloud (VPC)

The RA3 node type is based on AWS Nitro and includes support for Redshift managed storage. Amazon Redshift managed storage uses large, high-performance SSDs in each RA3 node for fast local storage and Amazon S3 for longer-term durable storage. Redshift managed storage automatically manages data placement across tiers of storage and caches the hottest data in high-performance SSD storage while automatically offloading colder data to Amazon Simple Storage Service (S3). Redshift managed storage uses advanced techniques such as block temperature, data blockage, and workload patterns to optimize performance

RA3 nodes with managed storage are a great fit for analytics workloads that require massive storage capacity. Also, it can be a good fit for workloads such as operational analytics, where the subset of data that is important and evolves constantly over time. In the past, there was pressure to offload or archive old data to other places because of fixed storage limits. This made maintaining the operational analytics data set and the larger historical dataset difficult to query when needed.

Consider choosing RA3 node types in these cases:

- You need the flexibility to scale and pay for compute separate from storage.
- Your data volume is growing rapidly or is expected to grow rapidly.
- You want the flexibility to size the cluster based only on your performance needs.

Older Generation Nodes:

- 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.

- 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.

For more information, see Clusters and Nodes in Amazon Redshift.

Amazon Redshift RA3 nodes with managed storage are generally available. RA3 nodes enable you to scale and pay for compute and storage independently allowing you to size your cluster based only on your compute needs. Now you can analyze even more data cost-effectively.

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.

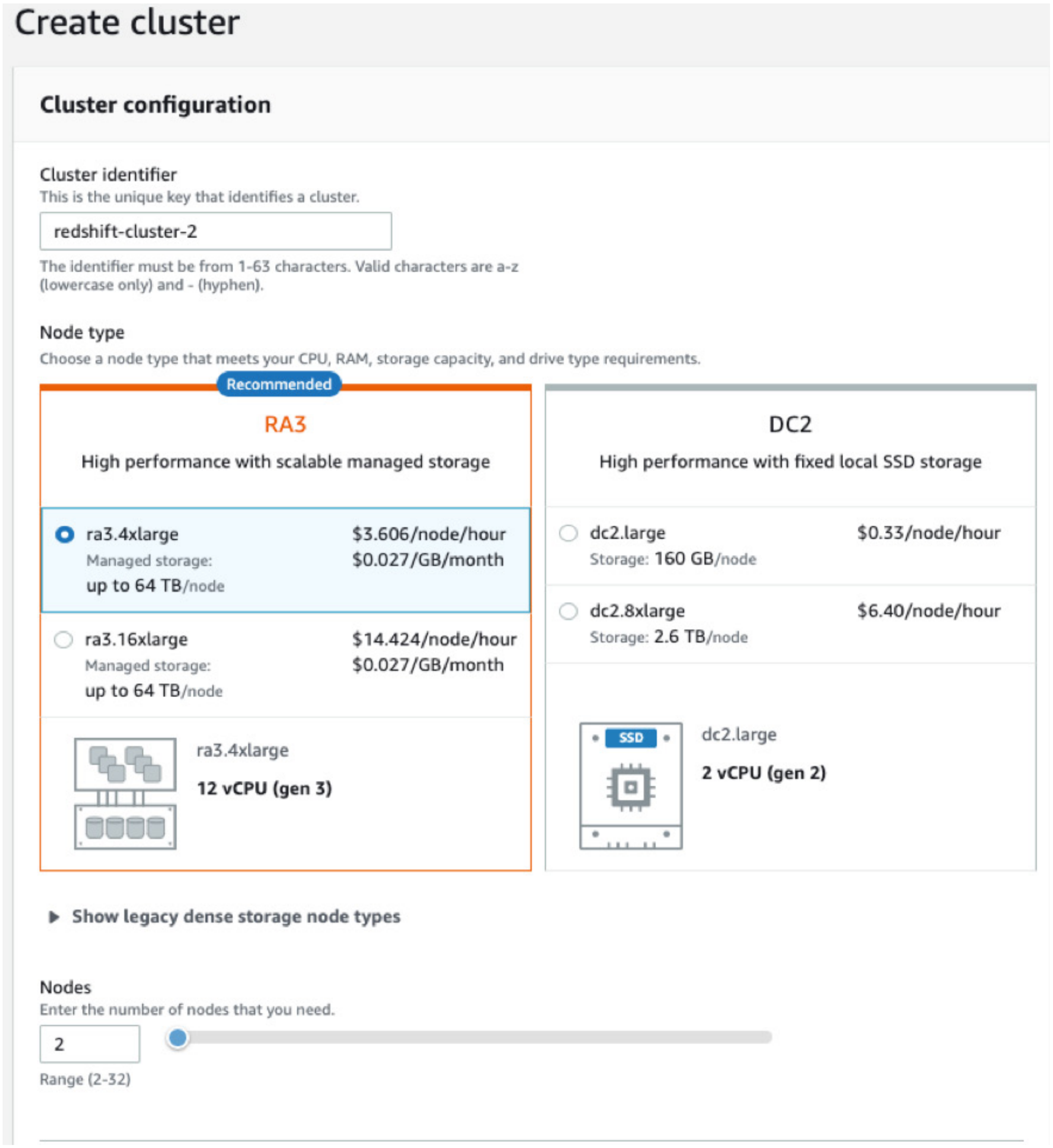
We recommend that you upgrade existing workloads running on DS2 node type clusters to RA3 node types to take advantage of improved performance and to get more storage capacity. RA3 nodes provide the following advantages:

- They are flexible to grow your compute capacity without increasing your storage costs. And they scale your storage without over-provisioning compute capacity.

- They use high performance SSDs for your hot data and Amazon S3 for cold data. Thus, they provide ease of use, cost-effective storage, and high query performance.

- They use high bandwidth networking built on the AWS Nitro System to further reduce the time taken for data to be offloaded to and retrieved from Amazon S3.

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



Create cluster

Cluster configuration

Cluster identifier
This is the unique key that identifies a cluster.
redshift-cluster-2
The identifier must be from 1-63 characters. Valid characters are a-z (lowercase only) and - (hyphen).

Node type
Choose a node type that meets your CPU, RAM, storage capacity, and drive type requirements.

Recommended

RA3
High performance with scalable managed storage

[✓]	ra3.4xlarge Managed storage: up to 64 TB/node	$3.606/node/hour $0.027/GB/month
○	ra3.16xlarge Managed storage: up to 64 TB/node	$14.424/node/hour $0.027/GB/month

ra3.4xlarge
12 vCPU (gen 3)

DC2
High performance with fixed local SSD storage

○	dc2.large Storage: 160 GB/node	$0.33/node/hour
○	dc2.8xlarge Storage: 2.6 TB/node	$6.40/node/hour

dc2.large
2 vCPU (gen 2)

▶ Show legacy dense storage node types

Nodes
Enter the number of nodes that you need.
2
Range (2-32)

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 Redshift, you can order data on multiple columns using the compound sort keys, which list the columns used in the sort key within a specific order. 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.

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. When using CREATE TABLE, if you are unsure what distribution style to pick, you can specify DISTSTYLE AUTO or skip the DISTSTYLE option and let Amazon Redshift choose the appropriate distribution style. To view the distribution style applied to a table, query the PG_CLASS system catalog table.Refer to Advance table design playbook or Choosing a data distribution method in the Amazon Redshift Developer Guide to decide which columns to designate for Distribution Keys"

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.

Refer to Advance table design playbook or Choosing a data distribution method in the Amazon Redshift Developer Guide to 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 there'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.

KEY Distribution: 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 Distribution: 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.

EVEN Distribution: 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.

AUTO Distribution: With AUTO option, Amazon Redshift assigns an optimal distribution style based on the size of the table data. For example, Amazon Redshift initially assigns ALL distribution to a small table, then changes to EVEN distribution when the table grows larger.

See Advance table design playbook to decide which columns in your tables you should designate for distribution key and sort key.

Also, Amazon Redshift launched a new capability called Redshift Advisor that will provide recommendations on distribution keys and sort keys based on the query patterns of your cluster.

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.

Amazon Redshift recently introduced latest compression mechanism called AZ64 to achieve a high compression ratio and improved query processing. AZ64 encoding has consistently better performance and compression than LZO. It has comparable compression with ZSTD but greatly better performance. With workloads we tested against, the following results were observed. We recommend that you evaluate the benefit for your workloads.

- Compared to RAW encoding, AZ64 consumed 60–70% less storage, and was 25–30% faster.
- Compared to LZO encoding, AZ64 consumed 35% less storage, and was 40% faster.
- Compared to ZSTD encoding, AZ64 consumed 5–10% less storage, and was 70% faster.Note that the above numbers are for a full workload and individual queries might get a much higher boost.

With your CREATE TABLE and ALTER TABLE statements, you can enable AZ64 encoding on columns with the SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIMESTAMP, TIMESTAMPTZ data types.

For more details about AZ64 encoding, see Compression Encodings in the Amazon Redshift Database Developer Guide.

Consider setting COMPUPDATE ON, when you are loading data to an empty table using COPY command. It ensures that optimal column encodings are applied to the table. For incremental loads to a Redshift table and when stage table has same encodings as final table, consider setting option COMPUPDATE OFF when using the COPY command. Data may change over a period of time, so the existing encoding may not be the best choice for your table. Analyze Compression to ensure that you still have optimal compression settings.

You can run Analyze Compression periodically or use the compression encoding recommendations generated by Redshift Advisor to ensure that you still have optimal compression settings to get better performance.

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.

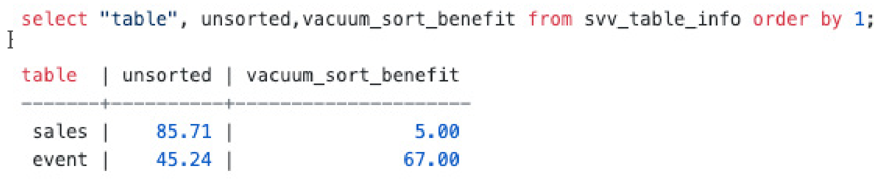
For more information on encryption, see Amazon Redshift Database Encryption in the Amazon Redshift Management Guide.

Vacuum and analyze your tables

The ANALYZE operation updates the statistical metadata of table so that the query planner can choose optimal plans. By default, Amazon Redshift continuously monitors your database and automatically performs analyze operations in the background. To minimize impact to your system performance, automatic analyze runs during periods when workloads are light.

When your are loading data using the COPY command, specifying the STATUPDATE ON option automatically runs analyze after the data finishes loading. If you run ANALYZE as part of your extract, transform, and load (ETL) workflow, automatic analyze skips tables that have current statistics. Similarly, an explicit ANALYZE skips tables with up-to-date table statistics. For more information on ANALYZE, read Analyzing tables in the Amazon Redshift Developer Guide.

Amazon Redshift introduced new feature, Auto Table Sort. This capability sorts data in the background to maintain table data in the order of its sort key. Amazon Redshift keeps track of your scan queries to determine which sections of the table will benefit from sorting. This will lessen the need to vacuum the table frequently. To determine whether your table will benefit by running VACUUM SORT, monitor the vacuum_sort_benefit column in SVV_TABLE_INFO.



```
select "table", unsorted,vacuum_sort_benefit from svv_table_info order by 1;

table  | unsorted | vacuum_sort_benefit
-------+----------+---------------------
 sales |    85.71 |                5.00
 event |    45.24 |               67.00
```

For the table "sales", even though the table is ~86% physically unsorted, the query performance impact from the table being 86% unsorted is only 5%. This might be either because only a small portion of the table is accessed by queries, or very few queries accessed the table. For the table "event", the table is ~45% physically unsorted. But the query performance impact of 67% indicates that either a larger portion of the table was accessed by queries, or the number of queries accessing the table was large. The table "event" can potentially benefit from running VACUUM SORT.

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.

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 20200810.csv, 20200811.csv, and 20200812.csv representing three different days of data, and the sort key is by datetime, then execute the COPY commands in the order 20200810.csv, 20200811.csv, and 20200812.csv to prevent the need to vacuum.

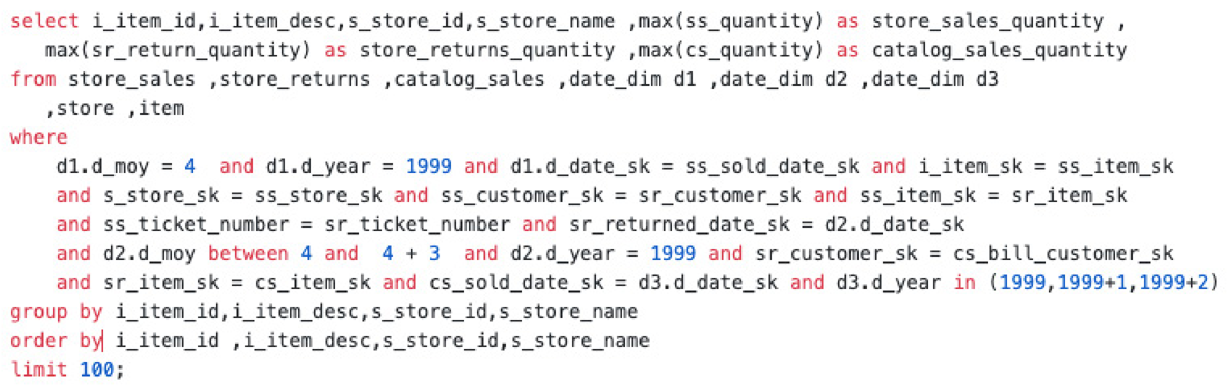
Materialized views

Amazon Redshift recently introduced support for materialized views. Often, performing large, complex queries over large tables are expensive because of system resources and time. Materialized views provide the functionality to store precomputed results set for those complex queries. Amazon Redshift returns the precomputed results from the materialized view, without having to access the base tables at all. From the user standpoint, the query results are returned much faster compared to when retrieving the same data from the base tables.

Materialized views are especially useful for speeding up queries that are predictable and repeated. Instead of performing resource-intensive queries against large database tables (such as aggregates or multiple-table joins), applications can query a materialized view and retrieve a precomputed result set. For example, consider the scenario where a set of queries is used to populate a collection of charts. This use case is ideal for a materialized view, because the queries are predictable and repeated over and over again. Please find more details here.

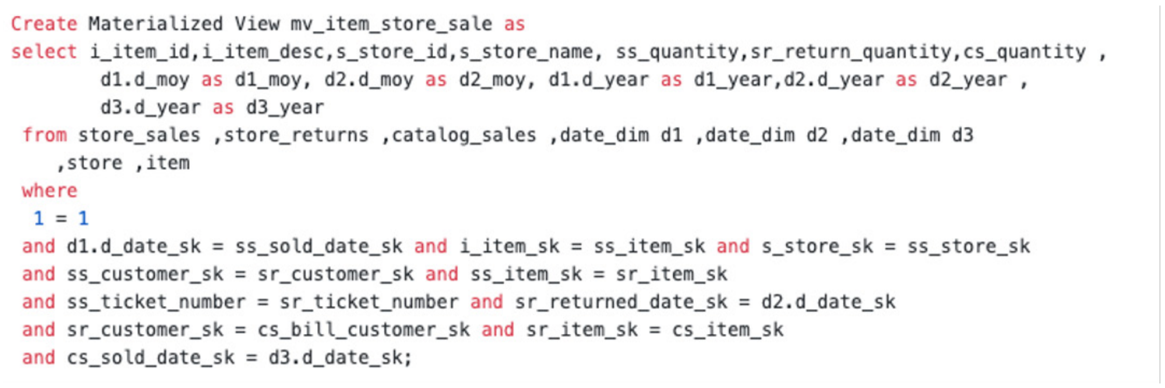
Whenever there is change in the base tables (data is inserted, deleted, or updated), the materialized views need to refresh to represent the current data. Refresh Materialized View is the command to keep the data updated. There are two ways to refresh the views. First is Incremental Refresh, in which Redshift quickly identifies changes in the base changes since the last refresh and updates the data. Second is the Full Refresh, which is done when the incremental refresh is not possible. This will load all the data in the materialized views. Amazon Redshift automatically picks the refresh strategy for a materialized view depending on the SELECT query used.

For example, let's consider the following complex query is a ran from a dashboard report which displays the top manufacturer by total profit across different stores. This dashboard query I took from TPC DS Benchmarking Document. For better understanding the query, following is the ER-Diagram represents the relation between the fact and dimension table. The main fact table store_sales has approximately 8.6 billion records. This is 3TB data set with distinct fact and dimension tables.



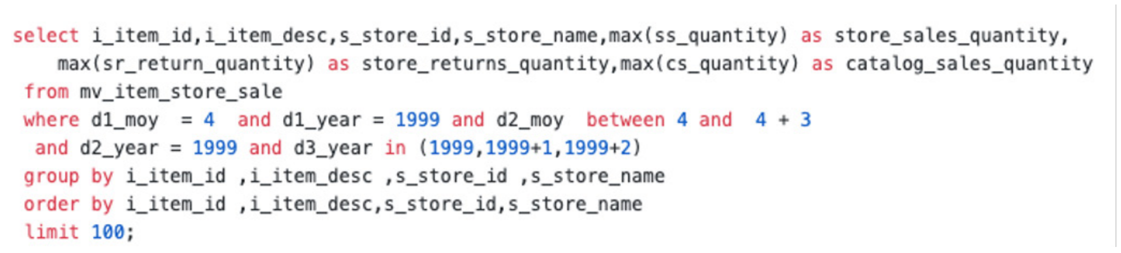
```
select i_item_id,i_item_desc,s_store_id,s_store_name ,max(ss_quantity) as store_sales_quantity ,
   max(sr_return_quantity) as store_returns_quantity ,max(cs_quantity) as catalog_sales_quantity
from store_sales ,store_returns ,catalog_sales ,date_dim d1 ,date_dim d2 ,date_dim d3
   ,store ,item
where
    d1.d_moy = 4  and d1.d_year = 1999 and d1.d_date_sk = ss_sold_date_sk and i_item_sk = ss_item_sk
    and s_store_sk = ss_store_sk and ss_customer_sk = sr_customer_sk and ss_item_sk = sr_item_sk
    and ss_ticket_number = sr_ticket_number and sr_returned_date_sk = d2.d_date_sk
    and d2.d_moy between 4 and  4 + 3  and d2.d_year = 1999 and sr_customer_sk = cs_bill_customer_sk
    and sr_item_sk = cs_item_sk and cs_sold_date_sk = d3.d_date_sk and d3.d_year in (1999,1999+1,1999+2)
group by i_item_id,i_item_desc,s_store_id,s_store_name
order by i_item_id ,i_item_desc,s_store_id,s_store_name
limit 100;
```

This preceding report took almost 18 seconds to execute. As more product gets sold and stores profit increases, this elapsed time gradually get longer. To speed up those reports, we can create a materialized view for this query. We will try to materialize the section of the query that will not only reduce the execution time for this query but also will be helpful for other queries having similar patterns. For example, the join among the store_sales, store_returns, catlog_sales, and date_dim tables are used by many other queries on the dashboard (use stl_query to get details). So, I created the materialized view that will have the join portion of the query. Below is the syntax to do so.



```
Create Materialized View mv_item_store_sale as
select i_item_id,i_item_desc,s_store_id,s_store_name, ss_quantity,sr_return_quantity,cs_quantity ,
        d1.d_moy as d1_moy, d2.d_moy as d2_moy, d1.d_year as d1_year,d2.d_year as d2_year ,
        d3.d_year as d3_year
 from store_sales ,store_returns ,catalog_sales ,date_dim d1 ,date_dim d2 ,date_dim d3
    ,store ,item
 where
  1 = 1
 and d1.d_date_sk = ss_sold_date_sk and i_item_sk = ss_item_sk and s_store_sk = ss_store_sk
 and ss_customer_sk = sr_customer_sk and ss_item_sk = sr_item_sk
 and ss_ticket_number = sr_ticket_number and sr_returned_date_sk = d2.d_date_sk
 and sr_customer_sk = cs_bill_customer_sk and sr_item_sk = cs_item_sk
 and cs_sold_date_sk = d3.d_date_sk;
```

The following query is for the same report which displays the top manufacturer by total profit across different stores using the materialized view.



```
select i_item_id,i_item_desc,s_store_id,s_store_name,max(ss_quantity) as store_sales_quantity,
    max(sr_return_quantity) as store_returns_quantity,max(cs_quantity) as catalog_sales_quantity
 from mv_item_store_sale
 where d1_moy  = 4  and d1_year = 1999 and d2_moy  between 4 and  4 + 3
  and d2_year = 1999 and d3_year in (1999,1999+1,1999+2)
 group by i_item_id ,i_item_desc ,s_store_id ,s_store_name
 order by i_item_id ,i_item_desc,s_store_id,s_store_name
 limit 100;
```

The same reports against the materialized views took almost 6 seconds to execute, the performance gain of 67%.

The advantage of using the Materialized View is two-fold:

1. Since the result are precomputed for the query, hence query will execute fast and in linear time even different users from multiple sessions running the same query.

2. This can reduce the load on the cluster since the refresh or incremental load of the Materialized view can we initiated when the cluster is comparatively ideal or has a low load.

Workload management

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 when you create custom WLM queue(s).

WLM can be configured in two modes: AUTO and MANUAL.

- Automatic WLM: Amazon Redshift manages query concurrency and memory allocation its own to deliver best throughput. It creates up to eight queues. Each queue is associated with a priority known as Query priority. This helps to assign priority to the queries. For example, consider a top management report query that has higher priority than an ETL job. So, you can assign the priority as HIGH for the report query and NORMAL priority to the ETL query. The priority is specified for a queue and inherited by all queries associated with the queue. You associate queries to a queue by mapping user groups and query groups to the queue. Here are more details on the Query Priority.

- Manual WLM: With Manual WLM you can manage the system performance. By modifying the WLM configuration to create separate queues for long running queries and short running queries. Each queue contains a number of query slots and also each queue is associated with a portion of available memory.

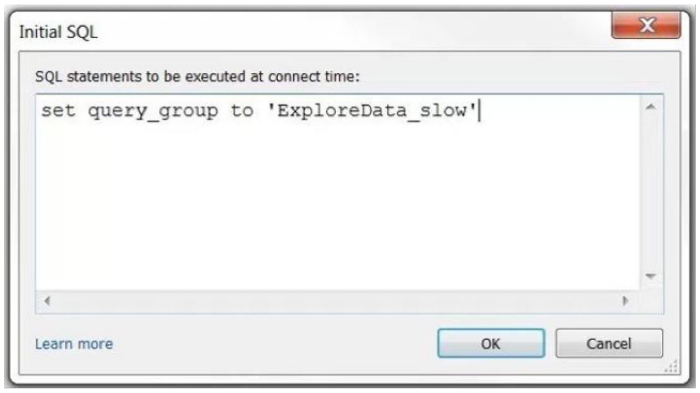
You can also use the Amazon Redshift query monitoring rules feature to set metrics-based performance boundaries for workload management (WLM) queues, and specify what action to take when a query goes beyond those boundaries. For example, for a queue that's dedicated to short running queries, you might 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 predefined rule templates in the Amazon Redshift management console to get you started.

Recommended configuration for WLM:

- Enable Auto WLM.
- Enable Concurrency Scaling to handle an increase in concurrent read queries, with consistent fast query performance.
- Enable Short Query Acceleration (SQA) so that short queries aren't forced to wait in queues behind longer queries.
- Create QMR rules to track and handle poorly written queries.

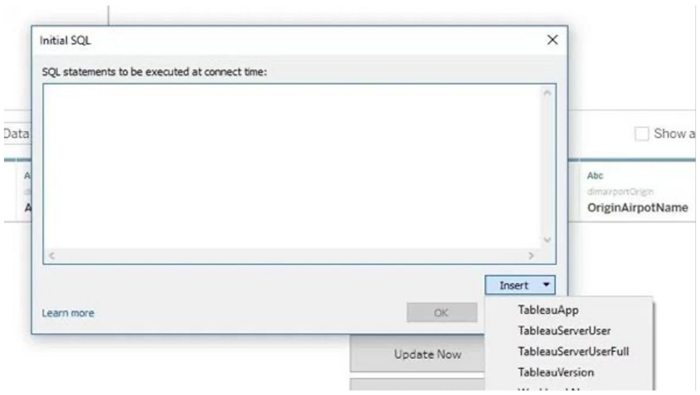
You can create additional WLM queues and pair them with Tableau's Initial SQL feature. Doing so will allow you to place specific queries into particular WLM queues, with simpler queries going to a "fast" queue with less dedicated RAM, and more complex queries going to a "slow" queue with more RAM.

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.



```
set query_group to 'ExploreData_slow'
```

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 the Help article Run Initial SQL.

Concurrency Scaling

With Concurrency scaling, you can configure Redshift to add more query processing power as-needed basis. When Concurrency Scaling is enabled, Amazon Redshift automatically adds additional transient clusters when you need it to process an increase in concurrent read queries. You manage which queries are sent to the concurrency scaling cluster by configuring the WLM queues. When concurrency scaling is enabled the eligible queries are passed to concurrency scaling cluster. This provides faster read even on heavy loaded cluster. This happens transparently and in a manner of seconds, and provides you with fast, consistent performance even as the workload grows to hundreds of concurrent queries. Additional processing power is ready in seconds and does not need to be pre-warmed or pre-provisioned. You pay only for what you use, with per-second billing and also accumulate one hour of concurrency scaling cluster credits every 24 hours while your main cluster is running. For more details on the concurrency scaling pricing is here.

The extra processing power is removed when it is no longer needed, making this a perfect way to address the burst use cases similar to the example stated above. You can enable the Concurrency Scaling for Tableau Dashboard queue and in case of heavy load on the cluster, the queries will be executed on a concurrent cluster hence improving the overall dashboard performance. To make concurrency scaling work with Tableau, it should use subqueries in place of temporary tables, which can be done by setting CAP_QUERY_SUBQUERY_QUERY_CONTEXT to "Yes."

In the example below, we will enable the concurrency scaling on the Redshift cluster for the Tableau Dashboard queries. We will create a user group in redshift and all the Dashboard queries will be allocated to this user group's queue. With concurrency scaling in place for the dashboard queries, there would be a significant reduction in query wait time.Please follow this blog post that has detailed steps how to enable concurrency scaling.

For this example, we created a workflow queue to run our dashboard queries. Now, we will add a rule for user group awsuser from which our tableau dashboard queries will be executed. This will enable the concurrent cluster to kick in if the Tableau dashboard queries ran from awsuser group are in the waiting queue.

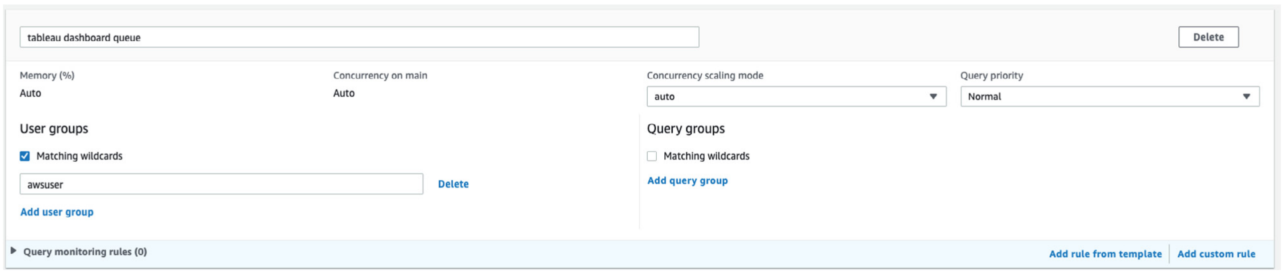


tableau dashboard queue

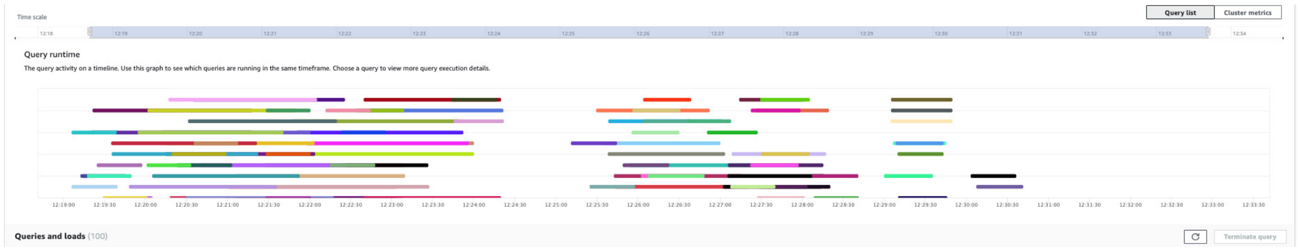
Memory (%): Auto
Concurrency on main: Auto
Concurrency scaling mode: auto
Query priority: Normal

User groups
☑ Matching wildcards
awsuser

Query groups
☐ Matching wildcards

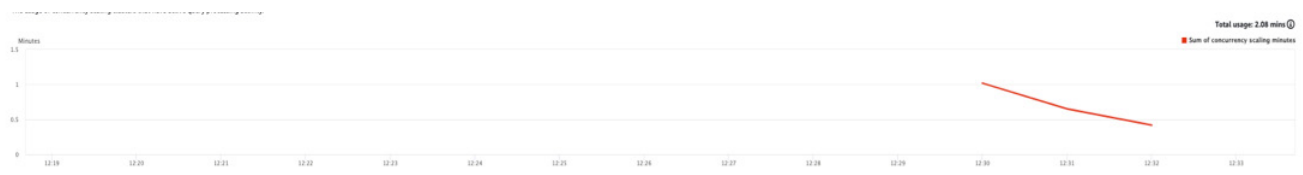
Query monitoring rules (0)

We ran several queries in parallel on the cluster, of which some of them are typical insert, copy queries like they are running from ETL jobs and some of them complex select queries such as dashboard queries. To determine the load on cluster we can use the Gantt chart provided on the Redshift console [as shown in below image]. You can use this chart to see how many queries running on which queue in the cluster and the execution time of the queries. This chart is available in the Query monitoring tab of the cluster in the Redshift console. For example, the below chart of the cluster that we used in example, you can see a long number of long running queries are executing in parallel.


Query list
Cluster metrics
Time scale
Query runtime
The query activity on a timeline. Use this graph to see which queries are running in the same timeframe. Choose a query to view more query execution details.
Queries and loads (100)
Terminate query

There are a number of queries are waiting in the WLM queue. This data is also available in both STV_ INFLIGHT system table or the Redshift AWS console. We observe the reduction in number of queries waiting in the WLM queue by 50%. This is because of the concurrency scaling has provided extra computational power and shared the load on the cluster.

In the below graph you can see how the concurrent cluster kicked in at 12:30. Due to that in the graph above you can see the wait time is reduced by half. Hence improving the Tableau dashboard query run time by reducing the wait time.



This example has demonstrated that the additional processing power come online as needed, and then go away when no longer needed, in the Workload Concurrency tab under Query Monitoring. You can also see that the queue time has reduced significantly.

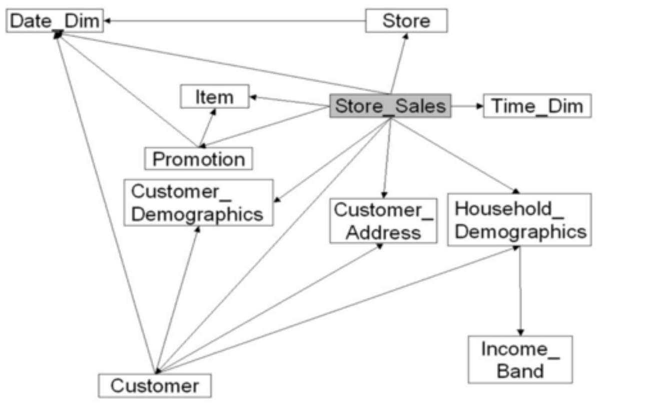
An Amazon Redshift optimization example

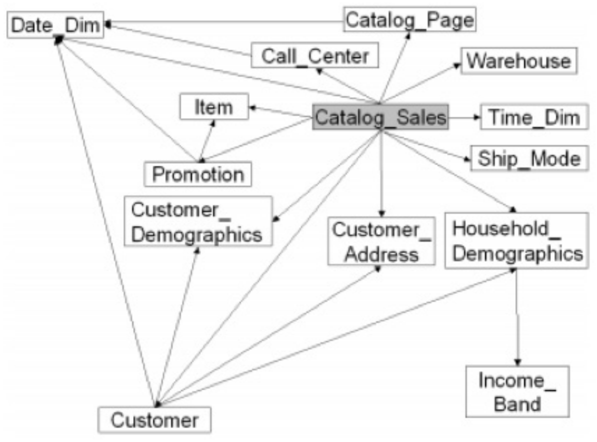
For this exercise we're using as the source the TPC-DS database.

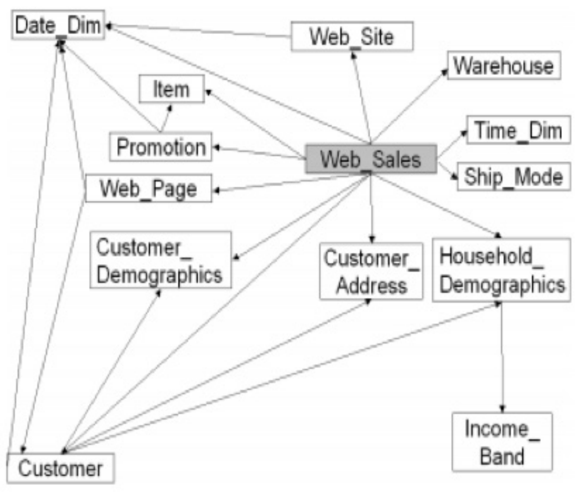
TPC-DS models the decision support functions of a retail product supplier selling through 3 channels (stores, web, and catalogs), while the data is sliced across 17 dimensions including Customer, Store, Time, Item, etc. The bulk of the data is contained in the large fact tables: Store Sales, Catalog Sales, Web Sales—representing daily transactions spanning 5 years. Read specs here.

It has been used extensively for testing performance, scalability and SQL compatibility across a range of Data Warehouse queries—from fast, interactive reports to complex analytics. In order to address the enormous range of query types and user behaviors encountered by a decision support system, TPC-DS utilizes a generalized query model. This model allows the benchmark to capture important aspects of the interactive, iterative nature of on-line analytical processing (OLAP) queries, the longer-running complex queries of data mining and knowledge discovery, and the more planned behavior of well known report queries.

Let load the TPC-DS data onto one of the clusters and start analyzing its performance. We will apply the optimization techniques that are discussed in this paper to improve the performance of the queries. The data model for the TPC-DS data is shown in the below image.







To ensure we are getting a true reflection of comparative Redshift performance between scenarios, let's turn off the Tableau cache.

You might also need to turn off cursors by altering your .TWB file (which is really just an xml file) and adding the following parameters to the odbc-connect-string-extras property.

- UseDeclareFetch=0;
- FETCH=0;

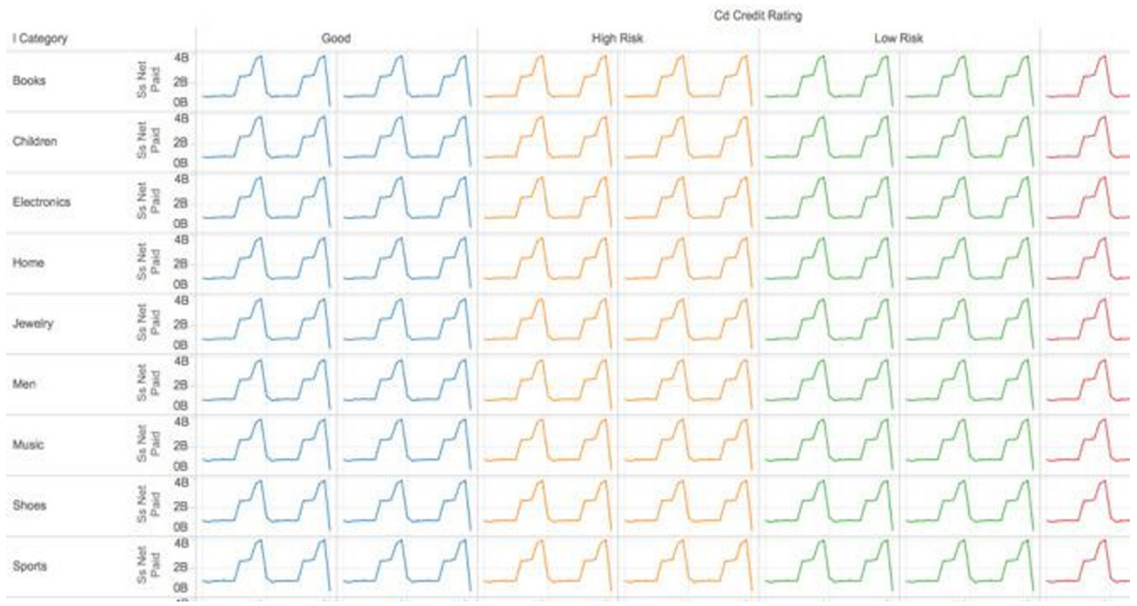
This will allow you to see what query is actually running under the hood in the Redshift console. This is explained further in the section below titled **Cursors and viewing query text data in the Amazon Redshift Console**.

You can also see the same by enabling the performance recorder feature in Tableau.

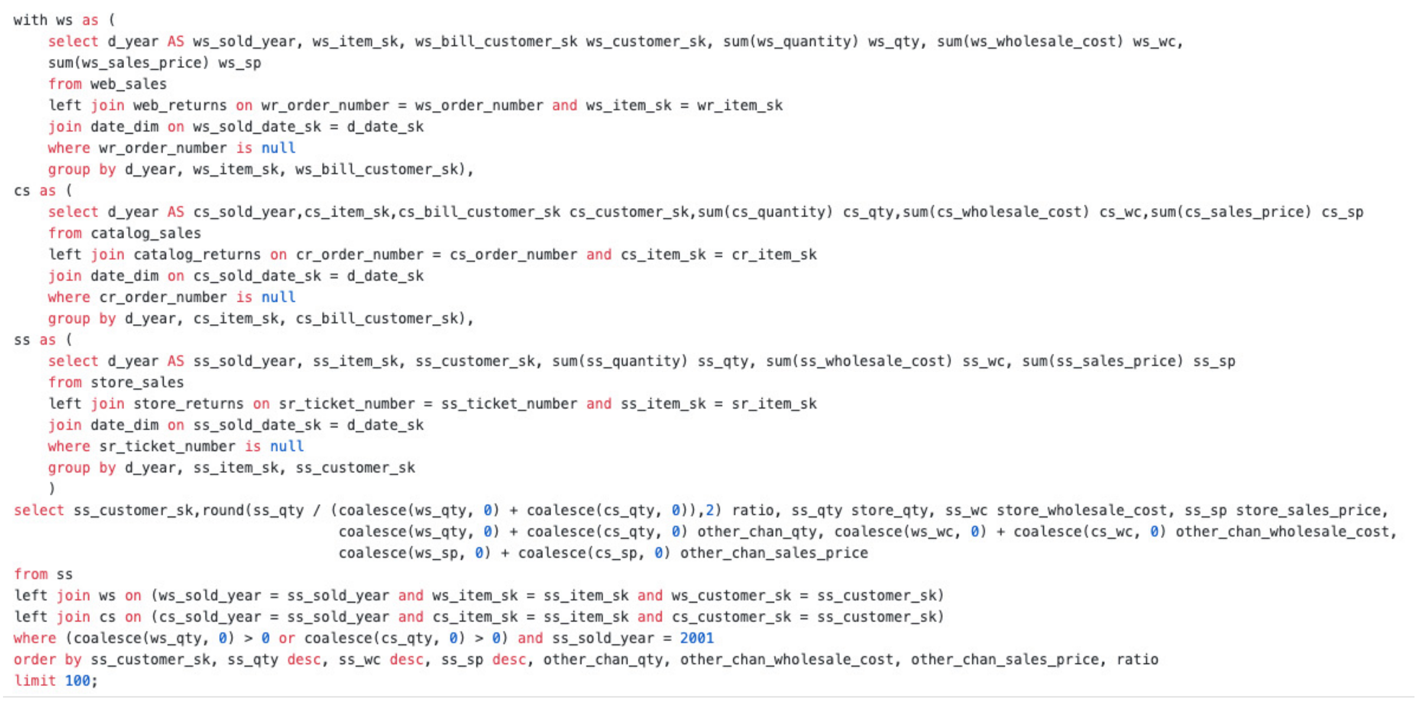
Let's also turn off the Redshift cache. You can do this by adding the command set enable_result_cache_for_session to off; in the Initial SQL dialog box on the Redshift connection window.

The goal after all is to force Redshift to work hard, so we don't want Tableau's or Redshift's cache getting in the way and making Redshift's life easy. The perceived performance results you'll see are therefore "worst case" because we always wait on an answer from Redshift before the user gets a result.

Now, let's generate the following visualization in Tableau, which attempts to explore whether credit ratings have any influence on how product sales trend over time across categories. The answer seems to be that it doesn't (not surprising, since it is synthetic benchmark data). We'll start by keeping the tables unoptimized.

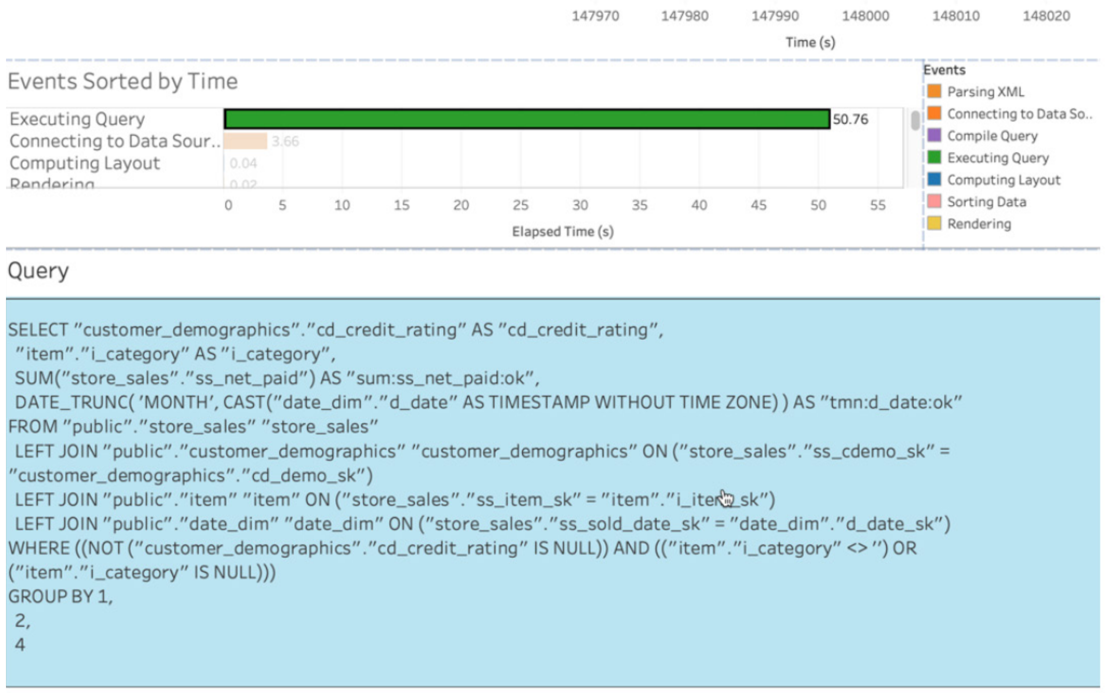


Following is one of the dashboard queries that takes around 6 min 40 seconds to run. We will now optimize the performance of the query. The optimizations techniques that we will apply will not only improve the performance of this query but also will improve the performance of other queries that are running on the cluster. Several factors can affect the performance of the query, like the number of nodes, Node types, Data distribution across the compute nodes, Data sort order, concurrent operations running on the clusters, query structure, etc. Here is the link that explains the query tuning in detail. We had implemented some of these techniques given in the link.



```
with ws as (
    select d_year AS ws_sold_year, ws_item_sk, ws_bill_customer_sk ws_customer_sk, sum(ws_quantity) ws_qty, sum(ws_wholesale_cost) ws_wc,
    sum(ws_sales_price) ws_sp
    from web_sales
    left join web_returns on wr_order_number = ws_order_number and ws_item_sk = wr_item_sk
    join date_dim on ws_sold_date_sk = d_date_sk
    where wr_order_number is null
    group by d_year, ws_item_sk, ws_bill_customer_sk),
cs as (
    select d_year AS cs_sold_year,cs_item_sk,cs_bill_customer_sk cs_customer_sk,sum(cs_quantity) cs_qty,sum(cs_wholesale_cost) cs_wc,sum(cs_sales_price) cs_sp
    from catalog_sales
    left join catalog_returns on cr_order_number = cs_order_number and cs_item_sk = cr_item_sk
    join date_dim on cs_sold_date_sk = d_date_sk
    where cr_order_number is null
    group by d_year, cs_item_sk, cs_bill_customer_sk),
ss as (
    select d_year AS ss_sold_year, ss_item_sk, ss_customer_sk, sum(ss_quantity) ss_qty, sum(ss_wholesale_cost) ss_wc, sum(ss_sales_price) ss_sp
    from store_sales
    left join store_returns on sr_ticket_number = ss_ticket_number and ss_item_sk = sr_item_sk
    join date_dim on ss_sold_date_sk = d_date_sk
    where sr_ticket_number is null
    group by d_year, ss_item_sk, ss_customer_sk
    )
select ss_customer_sk,round(ss_qty / (coalesce(ws_qty, 0) + coalesce(cs_qty, 0)),2) ratio, ss_qty store_qty, ss_wc store_wholesale_cost, ss_sp store_sales_price,
                                      coalesce(ws_qty, 0) + coalesce(cs_qty, 0) other_chan_qty, coalesce(ws_wc, 0) + coalesce(cs_wc, 0) other_chan_wholesale_cost,
                                      coalesce(ws_sp, 0) + coalesce(cs_sp, 0) other_chan_sales_price
from ss
left join ws on (ws_sold_year = ss_sold_year and ws_item_sk = ss_item_sk and ws_customer_sk = ss_customer_sk)
left join cs on (cs_sold_year = ss_sold_year and cs_item_sk = ss_item_sk and cs_customer_sk = ss_customer_sk)
where (coalesce(ws_qty, 0) > 0 or coalesce(cs_qty, 0) > 0) and ss_sold_year = 2001
order by ss_customer_sk, ss_qty desc, ss_wc desc, ss_sp desc, other_chan_qty, other_chan_wholesale_cost, other_chan_sales_price, ratio
limit 100;
```

As can be gleaned from the performance recorder snapshot for this query, the execution time for each query has been given at the end of the query.



```
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"."item" "item" ON ("store_sales"."ss_item_sk" = "item"."i_item_sk")
 LEFT JOIN "public"."date_dim" "date_dim" ON ("store_sales"."ss_sold_date_sk" = "date_dim"."d_date_sk")
WHERE ((NOT ("customer_demographics"."cd_credit_rating" IS NULL)) AND (("item"."i_category" <> '') OR
("item"."i_category" IS NULL)))
GROUP BY 1,
 2,
 4
```

As you can see, the query took almost 7 minutes to run. We will now progressively apply some optimization techniques to improve response times for this query.

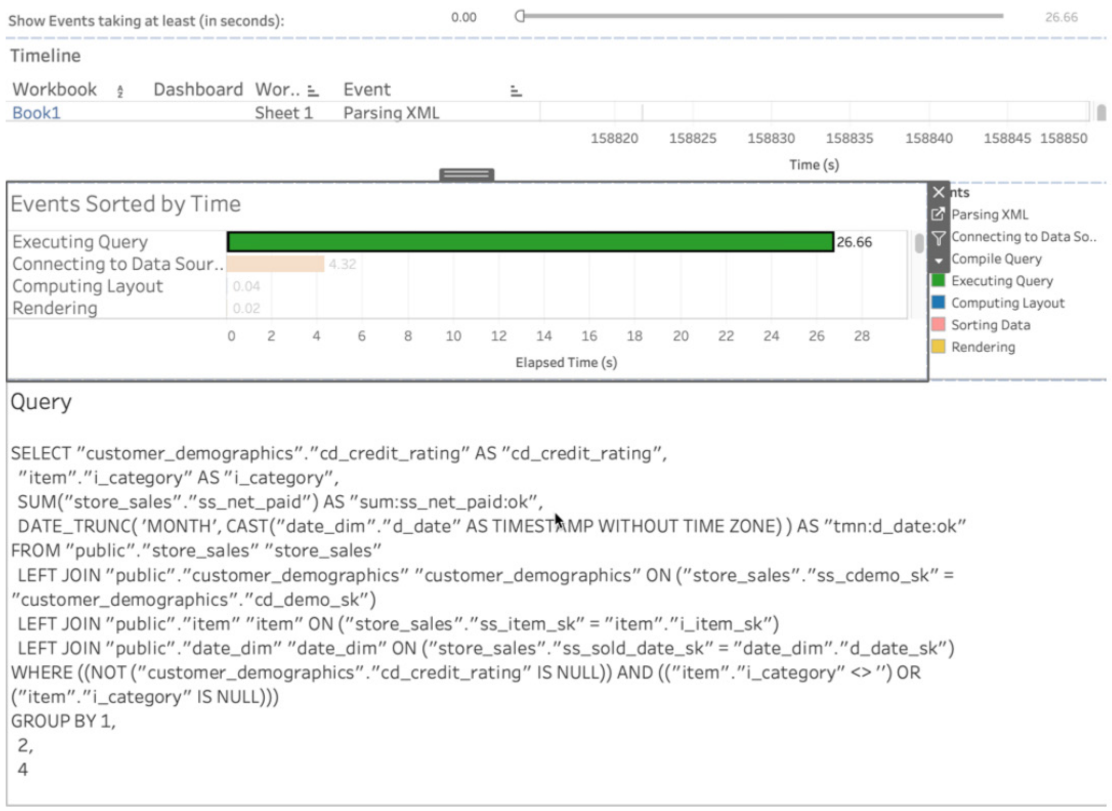
First, let's add some distribution keys and sort keys. The first step for optimization would be to ensure that data is distributed efficiently throughout the cluster for parallel processing. It will also minimize the cost of data movement necessary for query processing. Adding sort key will improve the performance since the data stored on the compute nodes are in the block structure. Having the sort key on a column will sorts the data within the blocks and that will help in reducing the number of blocks scanned by the query, hence better performance. So if your data is sorted by purchase time and your query asks for the revenue made in the last week, the query planner can quickly review the metadata of each block and see that all the records it needs to process are together in the most recent block (or blocks), saving it from having to process all the blocks to find the sales of interest. If you have five years of sales data and only need one week, Amazon Redshift will only have to scan less than half a percent of your table—a huge savings. In general, filtered columns make excellent sort key candidates.

The following statements add distribution keys to the tables being used in the query. In general, it's best practice to use the columns being joined on as distribution keys as you see in store_sales,customer_ demographics and item

For small dimension tables like date_dim, it makes sense to use a distribution style of ALL and store a full copy of the table on the first slice of each node. It'll take a little bit more storage but at least we're not broadcasting over the network across nodes any more—which is a huge win in terms of performance.

```
ALTER TABLE store_sales ALTER DISTKEY ss_item_sk, ALTER SORTKEY (ss_sold_date_sk);
ALTER TABLE store_returns ALTER DISTKEY sr_item_sk, ALTER SORTKEY (sr_returned_date_sk);
ALTER TABLE web_sales ALTER DISTKEY ws_item_sk, ALTER SORTKEY (ws_sold_date_sk);
ALTER TABLE web_returns ALTER DISTKEY wr_item_sk, ALTER SORTKEY (wr_returned_date_sk);
ALTER TABLE catalog_sales ALTER DISTKEY cs_item_sk, ALTER SORTKEY (cs_sold_date_sk);
ALTER TABLE catalog_returns ALTER DISTKEY cr_item_sk, ALTER SORTKEY (cr_returned_date_sk);
ALTER TABLE date_dim ALTER DISTSTYLE ALL;
```

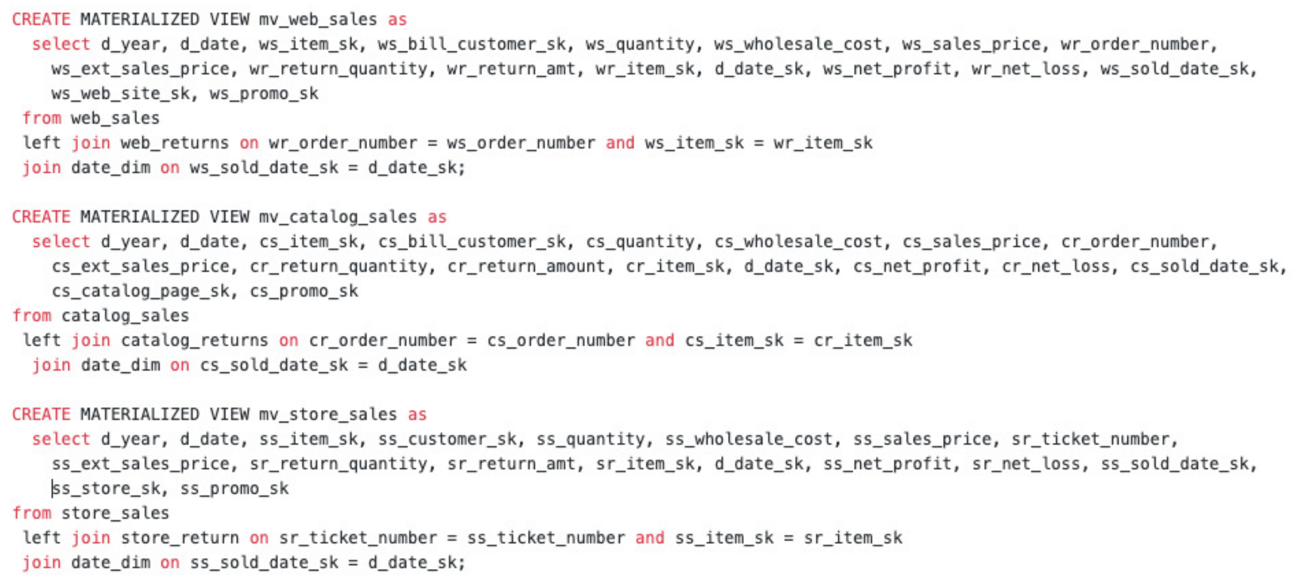
Refreshing the visualization after performing the tuning steps above cuts down the query response time. The query took from 06:40 min to 04:50 min to execute. That's a performance gain of 27%.



```
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"."item" "item" ON ("store_sales"."ss_item_sk" = "item"."i_item_sk")
 LEFT JOIN "public"."date_dim" "date_dim" ON ("store_sales"."ss_sold_date_sk" = "date_dim"."d_date_sk")
WHERE ((NOT ("customer_demographics"."cd_credit_rating" IS NULL)) AND (("item"."i_category" <> '') OR
("item"."i_category" IS NULL)))
GROUP BY 1,
 2,
 4
```

A methodology to guide you through the identification of optimal DISTSTYLEs and DISTKEYs for your unique workload is available in the Distribution Styles and Distribution Keys section of the Amazon Redshift Engineering's Advanced Table Design Playbook.

Next, Now, we will try to create Materialized View for this query. Looking at the workload and other queries partners, we will create three Materialized views. This will not only help for this query it will also improve the performance of other queries having the similar patterns running on the cluster.

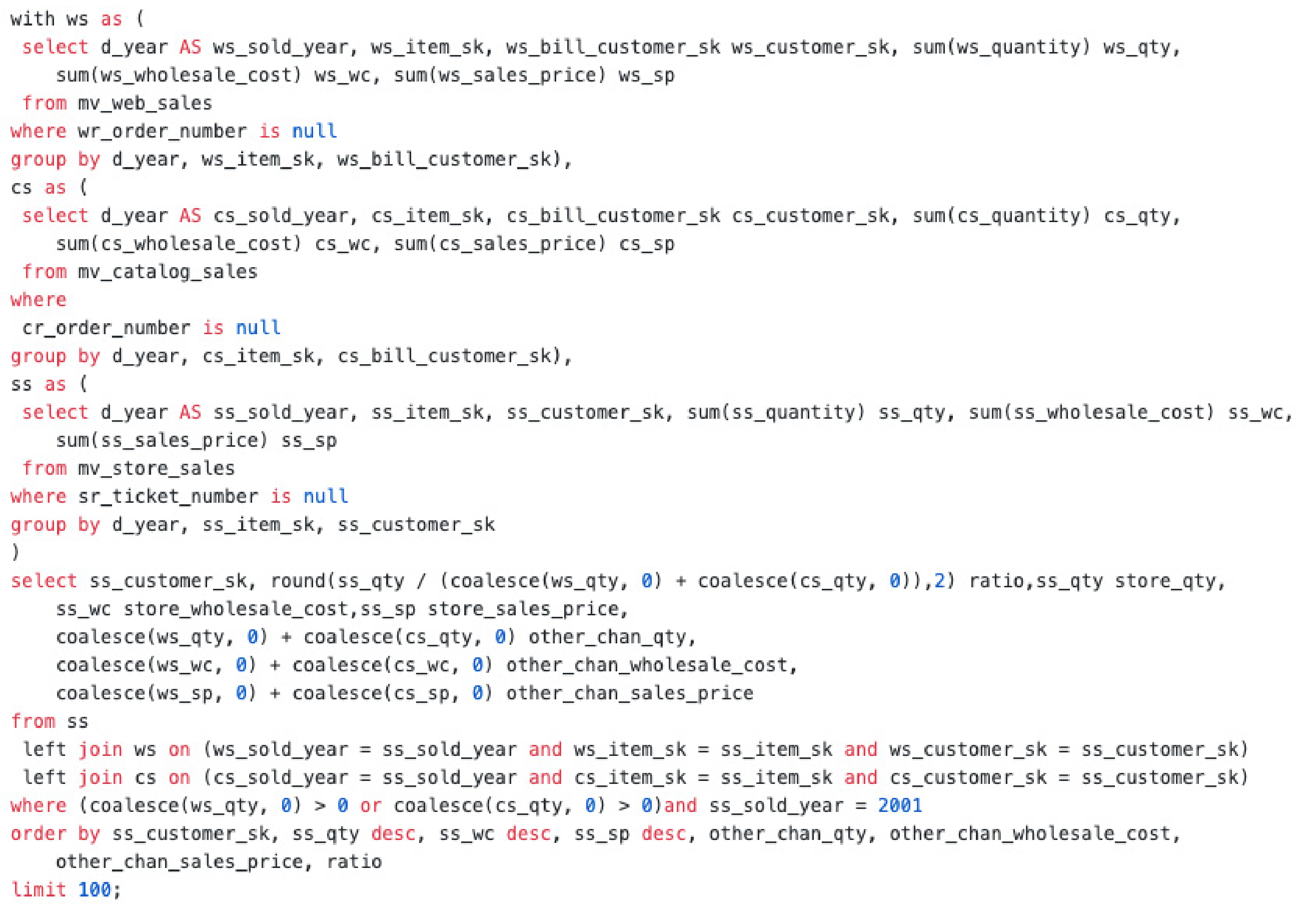


```
CREATE MATERIALIZED VIEW mv_web_sales as
  select d_year, d_date, ws_item_sk, ws_bill_customer_sk, ws_quantity, ws_wholesale_cost, ws_sales_price, wr_order_number,
    ws_ext_sales_price, wr_return_quantity, wr_return_amt, wr_item_sk, d_date_sk, ws_net_profit, wr_net_loss, ws_sold_date_sk,
    ws_web_site_sk, ws_promo_sk
 from web_sales
 left join web_returns on wr_order_number = ws_order_number and ws_item_sk = wr_item_sk
 join date_dim on ws_sold_date_sk = d_date_sk;

CREATE MATERIALIZED VIEW mv_catalog_sales as
  select d_year, d_date, cs_item_sk, cs_bill_customer_sk, cs_quantity, cs_wholesale_cost, cs_sales_price, cr_order_number,
    cs_ext_sales_price, cr_return_quantity, cr_return_amount, cr_item_sk, d_date_sk, cs_net_profit, cr_net_loss, cs_sold_date_sk,
    cs_catalog_page_sk, cs_promo_sk
from catalog_sales
 left join catalog_returns on cr_order_number = cs_order_number and cs_item_sk = cr_item_sk
  join date_dim on cs_sold_date_sk = d_date_sk

CREATE MATERIALIZED VIEW mv_store_sales as
  select d_year, d_date, ss_item_sk, ss_customer_sk, ss_quantity, ss_wholesale_cost, ss_sales_price, sr_ticket_number,
    ss_ext_sales_price, sr_return_quantity, sr_return_amt, sr_item_sk, d_date_sk, ss_net_profit, sr_net_loss, ss_sold_date_sk,
    ss_store_sk, ss_promo_sk
from store_sales
 left join store_return on sr_ticket_number = ss_ticket_number and ss_item_sk = sr_item_sk
 join date_dim on ss_sold_date_sk = d_date_sk;
```

After re-writing the query to use Materialized view it took 2 min 12 seconds from 4 min 50 seconds. That a gain of almost 55% from step 2 and 67% improvement from the step 1.



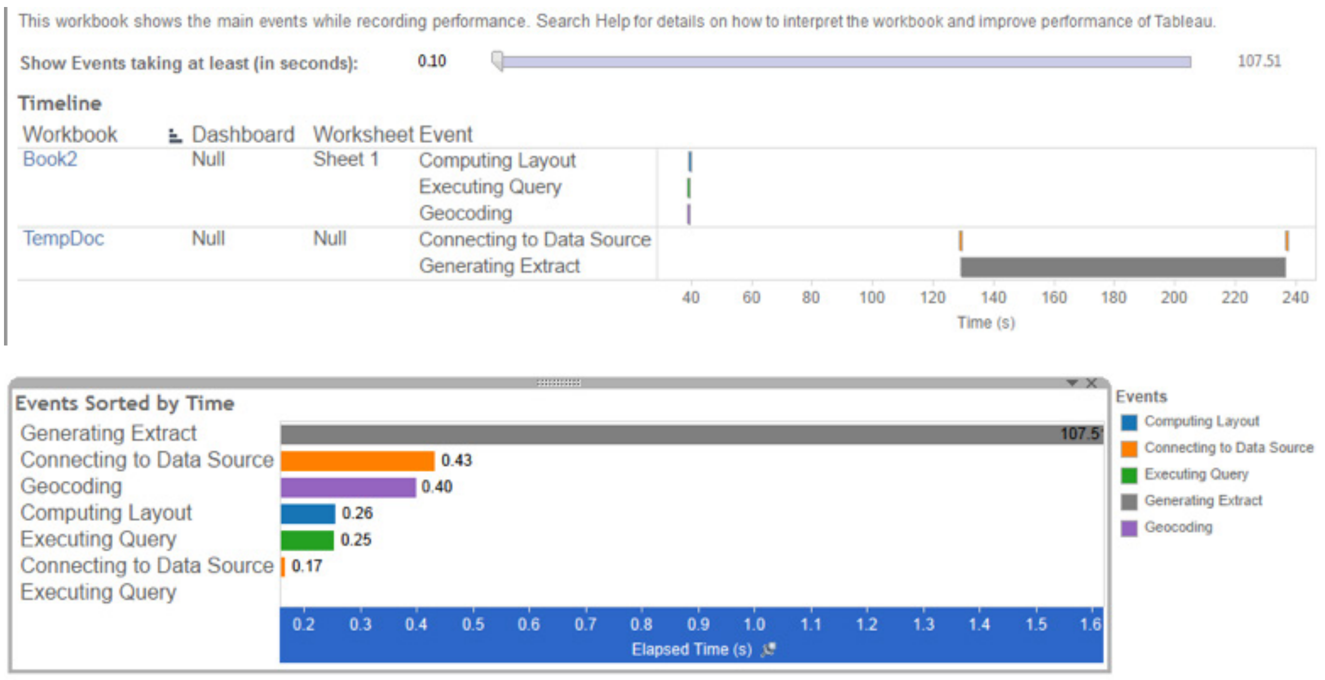
```
with ws as (
 select d_year AS ws_sold_year, ws_item_sk, ws_bill_customer_sk ws_customer_sk, sum(ws_quantity) ws_qty,
    sum(ws_wholesale_cost) ws_wc, sum(ws_sales_price) ws_sp
 from mv_web_sales
where wr_order_number is null
group by d_year, ws_item_sk, ws_bill_customer_sk),
cs as (
 select d_year AS cs_sold_year, cs_item_sk, cs_bill_customer_sk cs_customer_sk, sum(cs_quantity) cs_qty,
    sum(cs_wholesale_cost) cs_wc, sum(cs_sales_price) cs_sp
 from mv_catalog_sales
where
 cr_order_number is null
group by d_year, cs_item_sk, cs_bill_customer_sk),
ss as (
 select d_year AS ss_sold_year, ss_item_sk, ss_customer_sk, sum(ss_quantity) ss_qty, sum(ss_wholesale_cost) ss_wc,
    sum(ss_sales_price) ss_sp
 from mv_store_sales
where sr_ticket_number is null
group by d_year, ss_item_sk, ss_customer_sk
)
select ss_customer_sk, round(ss_qty / (coalesce(ws_qty, 0) + coalesce(cs_qty, 0)),2) ratio,ss_qty store_qty,
    ss_wc store_wholesale_cost,ss_sp store_sales_price,
    coalesce(ws_qty, 0) + coalesce(cs_qty, 0) other_chan_qty,
    coalesce(ws_wc, 0) + coalesce(cs_wc, 0) other_chan_wholesale_cost,
    coalesce(ws_sp, 0) + coalesce(cs_sp, 0) other_chan_sales_price
from ss
 left join ws on (ws_sold_year = ss_sold_year and ws_item_sk = ss_item_sk and ws_customer_sk = ss_customer_sk)
 left join cs on (cs_sold_year = ss_sold_year and cs_item_sk = ss_item_sk and cs_customer_sk = ss_customer_sk)
where (coalesce(ws_qty, 0) > 0 or coalesce(cs_qty, 0) > 0)and ss_sold_year = 2001
order by ss_customer_sk, ss_qty desc, ss_wc desc, ss_sp desc, other_chan_qty, other_chan_wholesale_cost,
    other_chan_sales_price, ratio
limit 100;
```

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.



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

Show Events taking at least (in seconds): 0.10

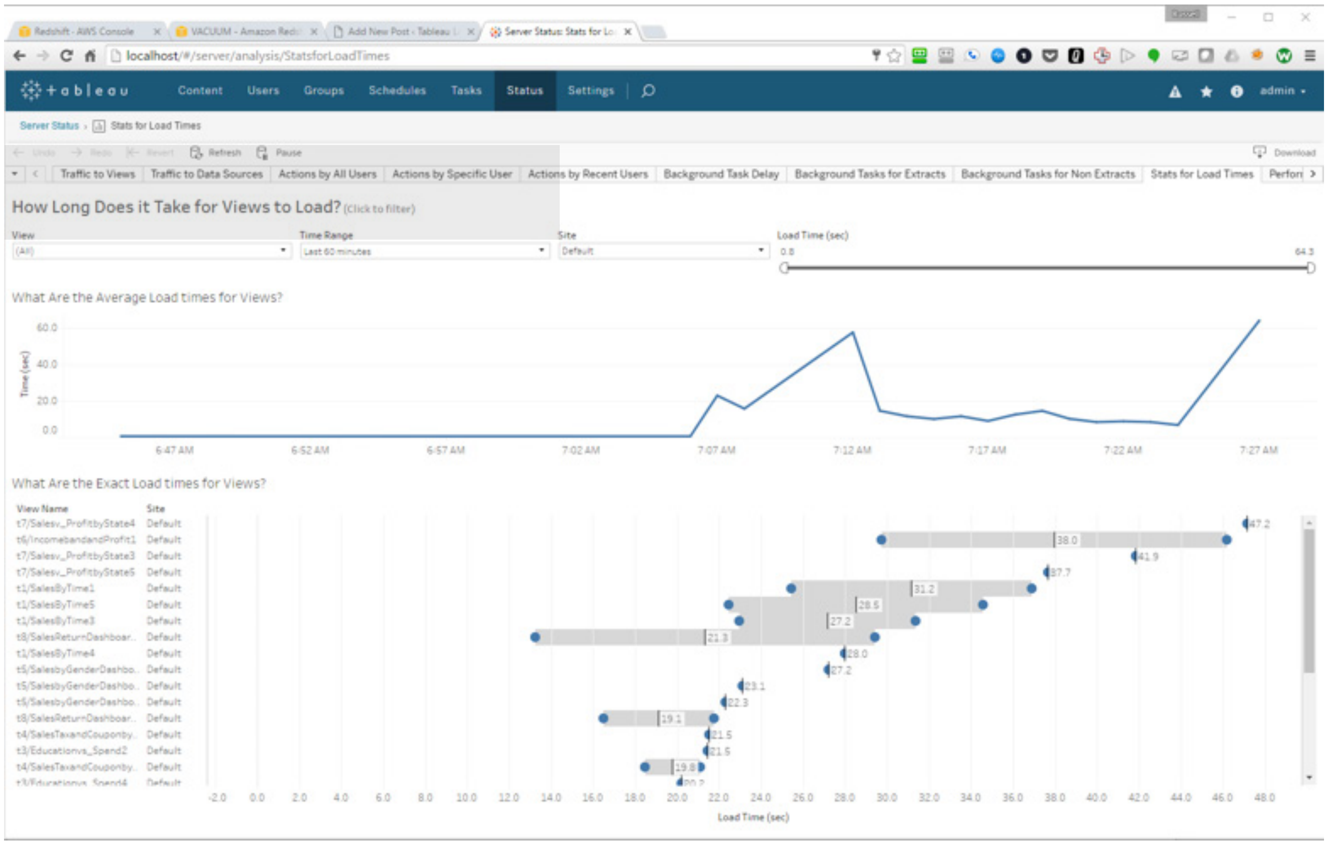
Workbook	Dashboard	Worksheet	Event
Book2	Null	Sheet 1	Computing Layout
			Executing Query
			Geocoding
TempDoc	Null	Null	Connecting to Data Source
			Generating Extract

Events Sorted by Time	Elapsed Time (s)
Generating Extract	107.5
Connecting to Data Source	0.43
Geocoding	0.40
Computing Layout	0.26
Executing Query	0.25
Connecting to Data Source	0.17
Executing Query	

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.”


localhost/#/server/analysis/StatsforLoadTimes
Content
Users
Groups
Schedules
Tasks
Status
Settings
admin
Server Status › Stats for Load Times
Traffic to Views
Traffic to Data Sources
Actions by All Users
Actions by Specific User
Actions by Recent Users
Background Task Delay
Background Tasks for Extracts
Background Tasks for Non Extracts
Stats for Load Times
How Long Does it Take for Views to Load? (Click to filter)
View
Time Range
Site
Load Time (sec)
What Are the Average Load times for Views?
Time (sec)
What Are the Exact Load times for Views?
View Name
Site
Load Time (sec)

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.

However, Tableau uses cursors when returning a result set from Redshift. The side effect of using a cursor is that you can’t see the actual plain-text SQL that Tableau fires inside the Redshift console. Instead, you’ll get a message like:

```
FETCH 10000 in “SQL_CUR03ART31”
```

which shows the executing cursor.

For more detail about how Tableau uses cursors in Amazon Redshift, please see the **More about cursors** within the **Additional considerations** section.

You 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. You also generally won't want to try an extract from Redshift with cursors turned off for the same reason.

You can read about leveraging a TDC here.

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.

Here is a basic customization which will turn cursors off (UseDelcareFetch=0):

```
<connection-customization class='redshift' version='9.0' enabled='true'>
  <vendor name='redshift' />
  <driver name='redshift' />
  <customizations>
        <customization name='odbc-connect-string-extras' value='UseDeclareFetch=0' />
  </customizations>
</connection-customization>
```

You can deploy a TDC using one of two approaches:

The customization above can be dropped into a text file with a .TDC extension and deposited into your Documents\My Tableau Repository\Datasources (Desktop) or Program Files\Tableau\Tableau Server\<version>\bin (Server) folder.

The customization can also be added directly to the XML of your data source. If you open your workbook and/or data source with a text editor, you might see something like this:

```
<named-connections>
  <named-connection caption='foo.foo.ap-southeast-1.redshift.amazonaws.com'
name='redshift.1foo'>
    <connection class='redshift' dbname='tpchdslitev1' odbc-connect-string-extras='' one-
time-sql='' port='5439' schema='public'
     server=foo.foo.ap-southeast-1.redshift.amazonaws.com' single-node='no' sslmode=''
username='foo' /
   </named-connection>
 </named-connections>
```

Change it to this:

```
<named-connections>


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

   <connection class='redshift' dbname='tpchdslitev1'

odbc-connect-string-extras='UseDeclareFetch=0' one-time-sql='' port='5439'

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

     <connection-customization class='redshift' enabled='true' version='10.1'>

        <vendor name='redshift' />

        <driver name='redshift' />

         <customizations>

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

         </customizations>

      </connection-customization>

    </connection>

  </named-connection>


</named-connections>
```

You can also simplify the named connection and simply use the odbc-connect-string by itself.

```
 <named-connections>

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

    <connection class='redshift' dbname='tpchdslitev1'

odbc-connect-string-extras='UseDeclareFetch=0' one-time-sql='' port='5439'

     schema='public' server=foo.foo.ap-southeast-1.redshift.amazonaws.com' single-
node='no' sslmode='' username='foo' /

   </named-connection>


 </named-connections>
```

You may want to go with the "inline data source" technique if you don't want the TDC file to apply globally. If you deploy a TDC for a vendor/driver to your Tableau Server, then ALL of the workbooks you deploy which use the same vendor/driver MUST use that TDC. If someone tries to publish or execute a "non-TDC-ed" workbook, you can expect to see an error message like this:

"Keychain authentication does not work because either the required TDC file is missing from Tableau Desktop, or the TDC file on Tableau Desktop differs from the TDC file on Tableau Server"

This will also break extract refreshes that used to work. Once you make these changes to remove cursors, you'll now see Tableau's queries in the AWS console, and can make the right adjustments accordingly.

Additional 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 Dense Compute node (DC2) 8XL multiple nodes cluster has a maximum result set of 3,200,000 MB. If you exceed this limit, resultsets will be written to disk, as needed.

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 Community Forums.

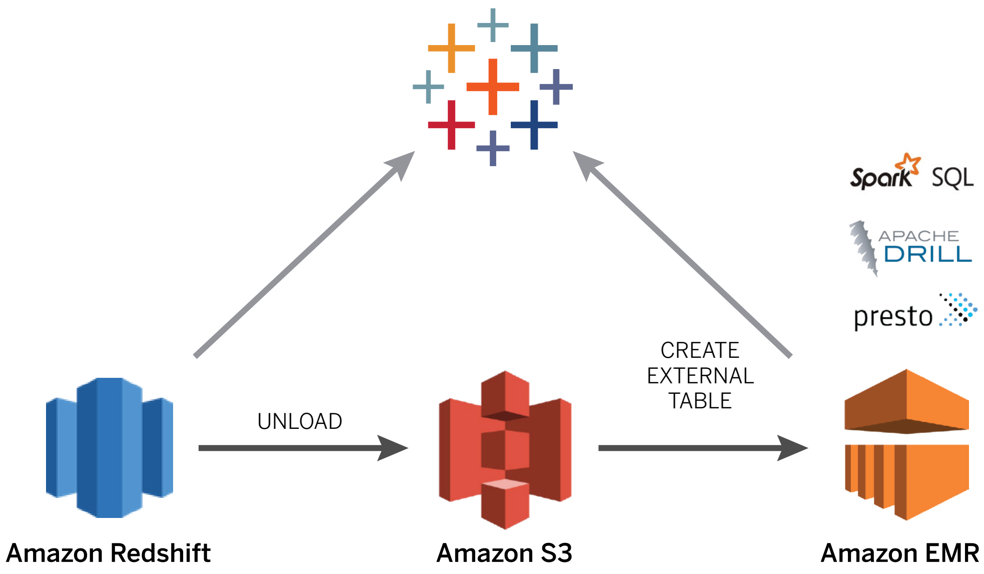
Amazon Redshift and Tableau Extracts

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 to 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.

If you absolutely need to extract the leaf level data, and cannot do with an aggregate, consider using an alternative that leverages a few other AWS services.



In the approach illustrated above, we're avoiding forcing Redshift to act as an Operational Data Store for Tableau, and instead doing an UNLOAD of the data into S3 (Redshift is super-fast at this). Then, we dynamically spin up an instance of EMR and use our favorite approach to ingest (Drill, Presto, or Spark) and make the data available. Tableau can extract directly from EMR, and the stand-up / extract / tear-down can be scripted and automated.

Amazon Redshift Spectrum

In Tableau, our Amazon Redshift connector includes support for Amazon Redshift Spectrum, so customers can connect directly to data in Amazon Redshift and analyze it in conjunction with data in Amazon Simple Storage Service (S3).

Many Tableau customers have large amount of data stored in Amazon S3. Amazon Redshift spectrum allows you to extend the analytic power of Amazon Redshift beyond data stored on local disks of Amazon Redshift cluster. Redshift Spectrum uses the AWS Glue catalog and provides the same view of data across Redshift Spectrum, Athena, and EMR.

With Redshift Spectrum, you can run Redshift SQL queries against exabytes of data in Amazon S3 “Data Lake” without loading data into Amazon Redshift and you pay only for the data you scanned. Like Amazon Redshift itself, you get the benefits of a sophisticated query optimizer, fast access to data on local disks, and scale out to thousands of nodes to scan and process exabytes of data sitting in Amazon S3—in few minutes.

Use cases that might benefit from the use of Amazon Redshift spectrum are:

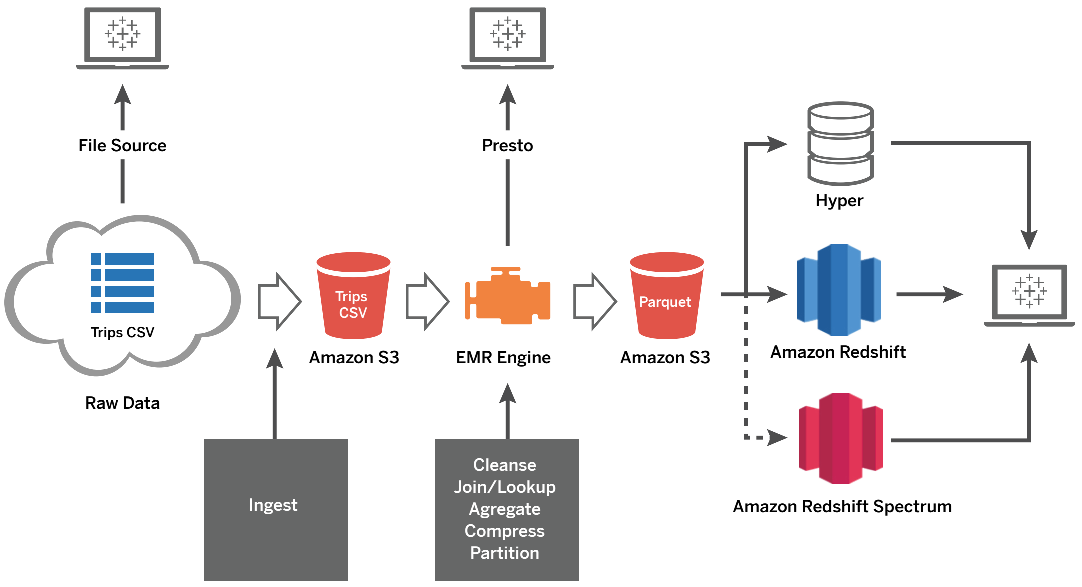
- Large volumes but less frequently accessed data
- Heavy scan- and aggregation-intensive queries
- Selective queries that can use partition pruning and predicate pushdown, so the output is fairly small

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 your Amazon S3 data lake.

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.

Also refer to best practices for Amazon Redshift Spectrum to optimize Redshift Spectrum query performance and cost.

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. 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.

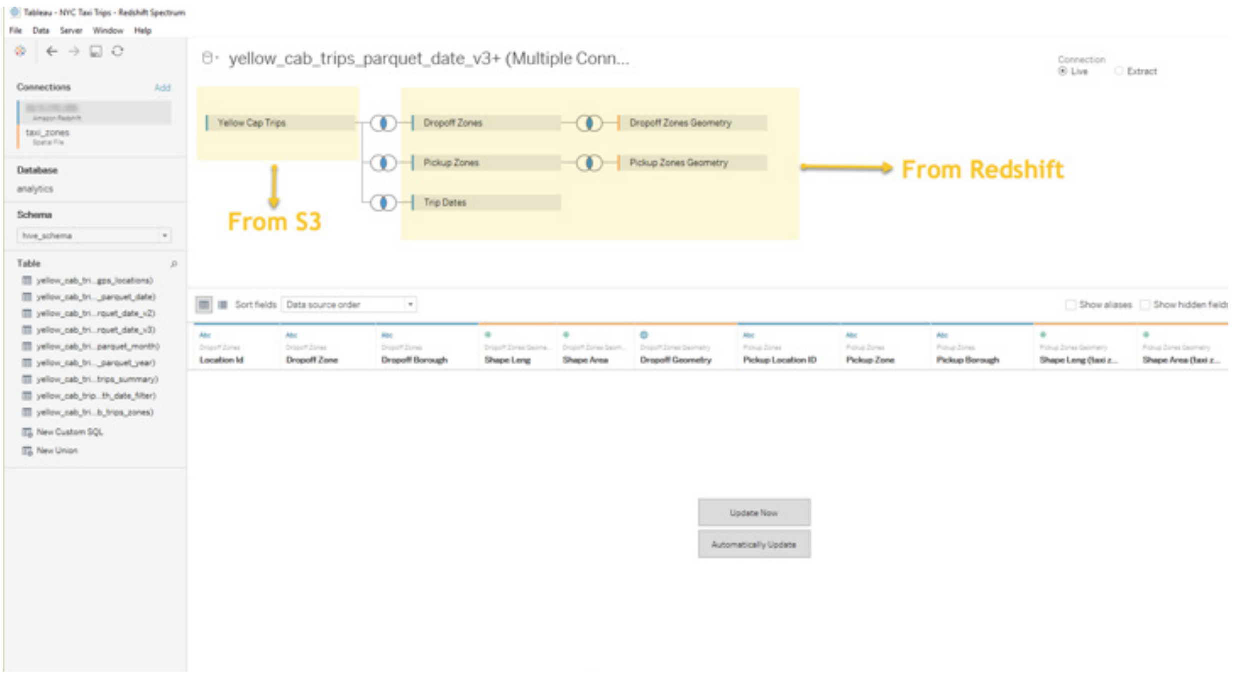
The data lands in Amazon S3. It is cleansed and partitioned via Amazon EMR and converted to an analytically optimized columnar Parquet format.

Note that you can point Tableau to the raw data in Amazon 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.

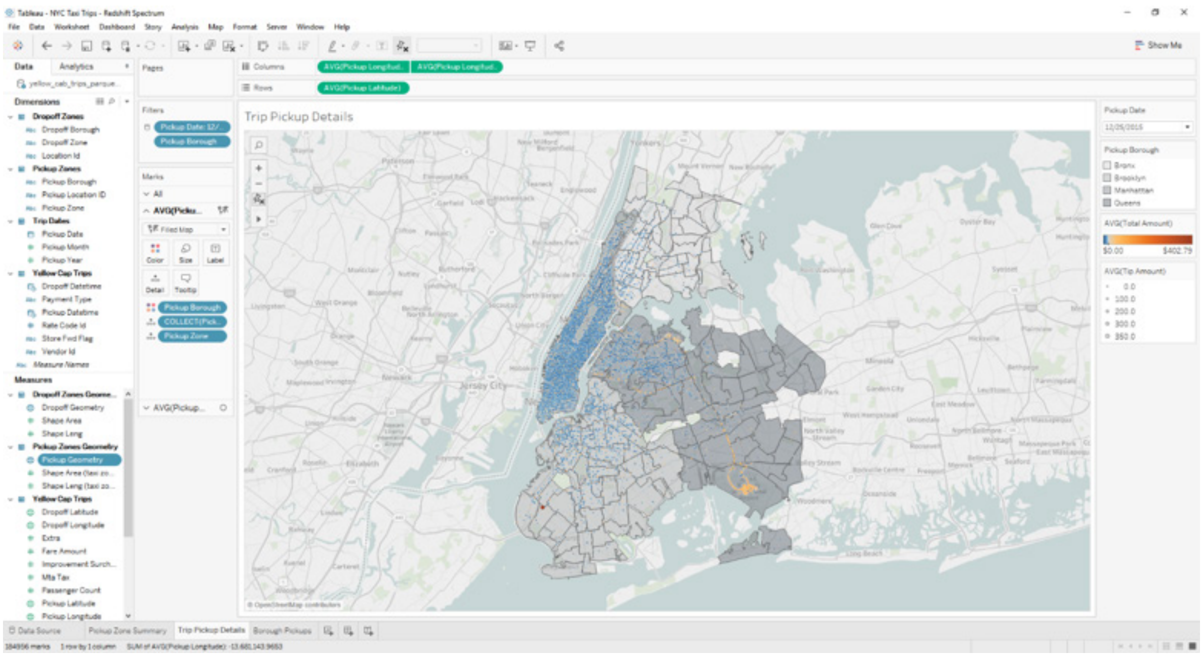
Once you discover those questions and determine if this sort of analysis has long-term advantages, you can automate and optimize that pipeline, adding new data as soon as it arrives so you can get it to the processes and people that need it. You may also want to provision this data to a highly performant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

As represented in the flow above, Amazon 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.

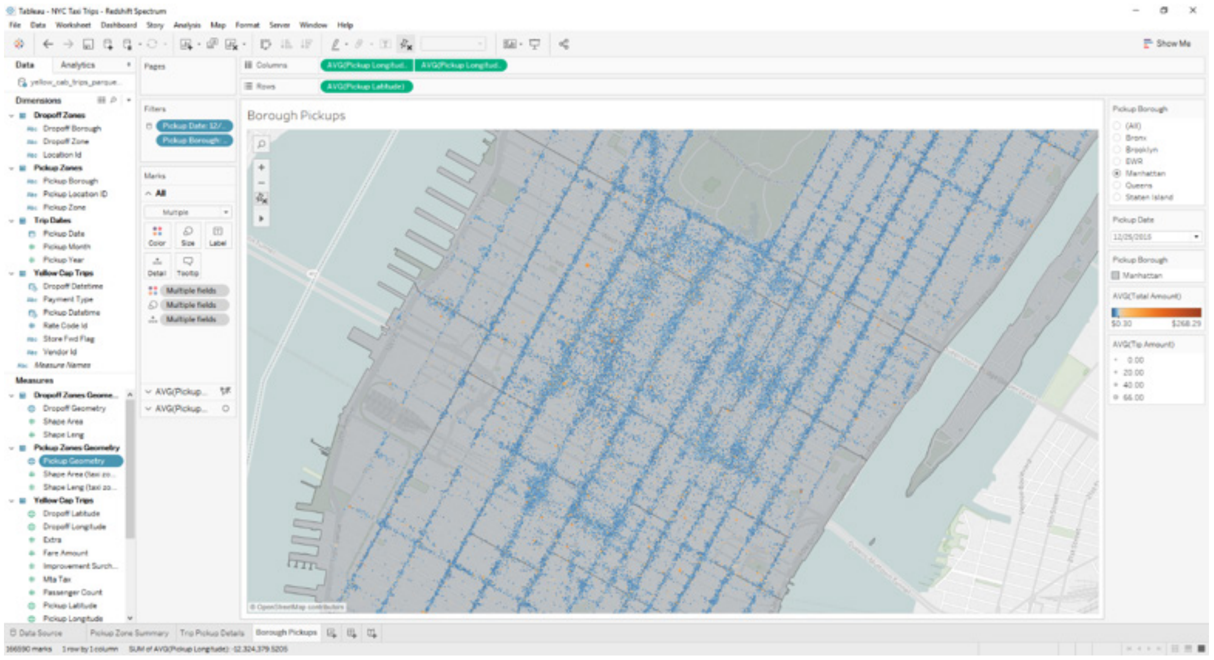
Now let's imagine that I'd like to know where and when taxi pickups happen on a certain date in a certain borough. With support for Amazon Redshift Spectrum, I can now join the Amazon S3 tables with the Amazon Redshift dimensions.



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 just look at Manhattan to identify pick-up "hotspots" where ride charges appear way higher than average.



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, 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.

In addition, please also check out the AWS blogs on best practices for Amazon Redshift, best practices for Amazon Redshift Spectrum, and best practices for designing ETL for 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.

Federated Queries

Amazon Redshift Federated Queries allow you to query and analyze data through operational databases, data warehouses, and data lakes. With the Federated Query feature, you can integrate queries from Amazon Redshift on live data in external databases with queries across your Amazon Redshift and Amazon S3 environments. Federated queries can work with PostgreSQL compatible external databases within Amazon like RDS and Aurora.

For example, to make data ingestion easier for Amazon Redshift, you could use federated queries to do the following.

- Create report directly on operational and analytical data.
- Apply transformation easily and quickly.

You can learn more about Federated Queries in the Amazon Database Developer Guide, or see this excellent blog that explains in steps how to use Federated Query with simplified ETL and Live data.

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 are powerful—and keeping in mind these performance tips and techniques will help you understand how to optimize the two 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 to reduce the queries that Tableau must execute.

Amazon Redshift allows anyone to deploy a data warehouse into the millions and billions of rows, all without procuring hardware and at a fraction of the cost of traditional database administration. 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.