Tuning your Amazon Redshift and Tableau Software Deployment for Better Performance

April 2014

Abstract

Amazon Redshift and Tableau Software's ability to connect directly provides business users the power and agility to analyze and gain insights from data sets running into the billions of rows. Understanding how to optimize each of these technologies as they work together can yield considerable performance gains and ultimately shorten deployment cycles. This paper addresses issues relating to query patterns, data modeling, and workbook construction in an effort to achieve optimal responsiveness.

Introduction

Amazon Redshift is a fully managed, petabyte-scale data warehouse service. It is optimized for data sets ranging from a few hundred gigabytes to over a petabyte and costs less than $1,000 per terabyte per year.

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, all of the front-end visualizations can also be published to the cloud, where the visualizations can directly query Amazon Redshift data warehouses.

Tableau software with Amazon Redshift provides a powerful, attractive, and easy to manage warehousing and analysis solution.

Improving Performance with Amazon Redshift and Tableau

You will want to follow good design and query practices to provide the best user experience possible when analyzing large data sets using Tableau. Improving responsiveness involves making solid database design decisions, performing regular data warehouse maintenance and following good practices when constructing Tableau visualizations.

Monitoring Query Performance

Pinpointing poor query performance can often be cumbersome, but Tableau's built-in performance recorder[1] helps diagnose a slow or inconsistent report quickly. After successfully identifying poor query performance in a Tableau analysis of Amazon Redshift, you can often resolve the issue by reviewing your tables and query design, tuning your queries, and making adjustments to your workbook.

[1] Performance recorder is currently only available on Tableau Server, not Tableau Online.

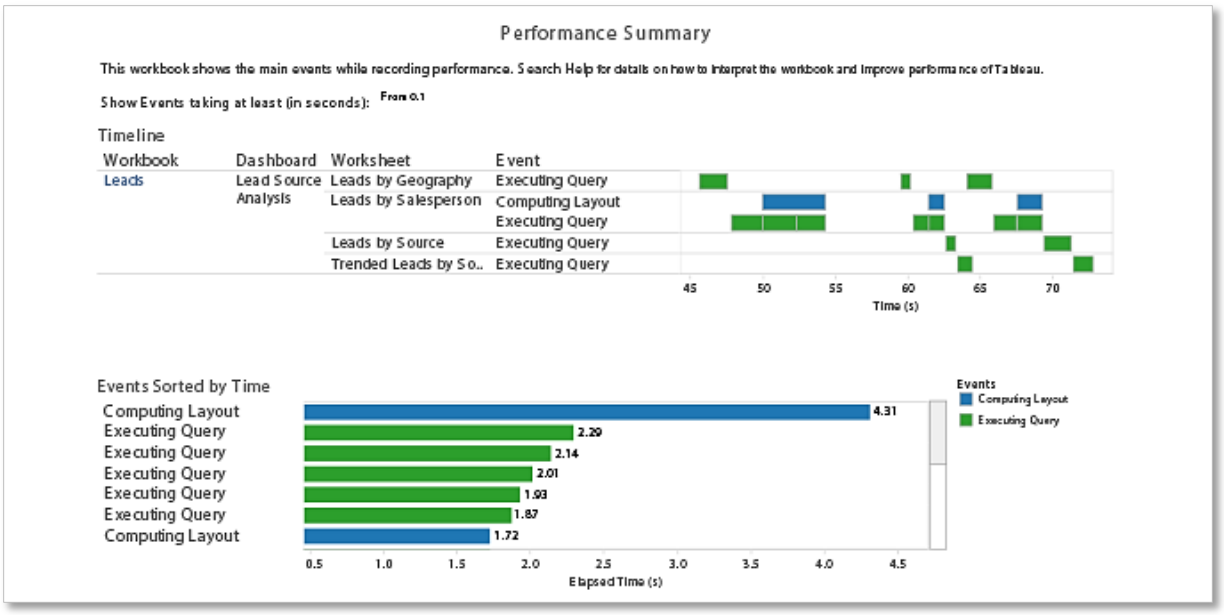


Figure 1: Use Tableau's performance recorder to diagnose slow performance.

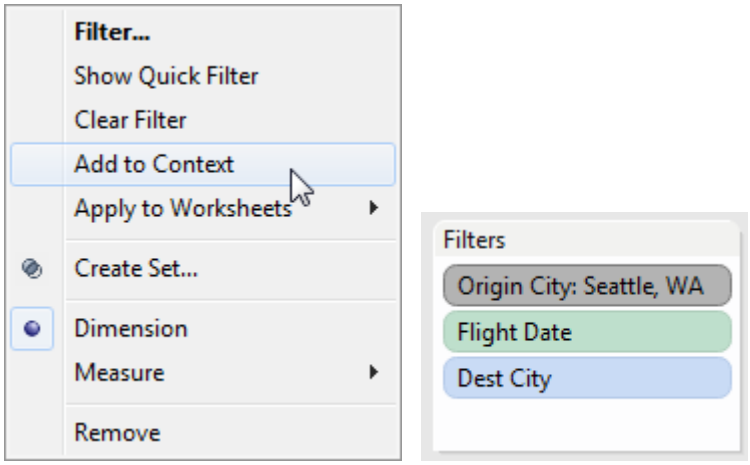
*For more detailed documentation on Tableau performance recorder, go to the Performance section of Tableau's Help page.[2]

Optimizing Tableau Software for Query Performance

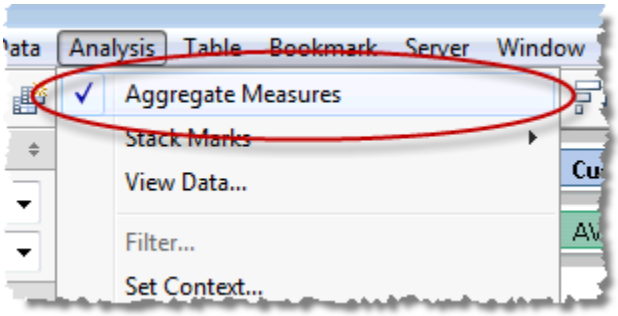
When designing reports and dashboards, you can take advantage of the following Tableau Desktop features to ensure responsiveness by minimizing query complexity, the number of queries, and the size of the results returned from Amazon Redshift:

[2] http://onlinehelp.tableausoftware.com/current/pro/online/en-us/performance_tips.html

- **Context filters** – If you are setting filters that significantly reduce the data set size and that will be used for more than several data views, you can set those filters as context filters. Refer to Filtering in the Tableau online help for how to create context filters.[3] For more information about performance improvement with context filters, see Speeding up Context Filters in the Tableau online help.[4]

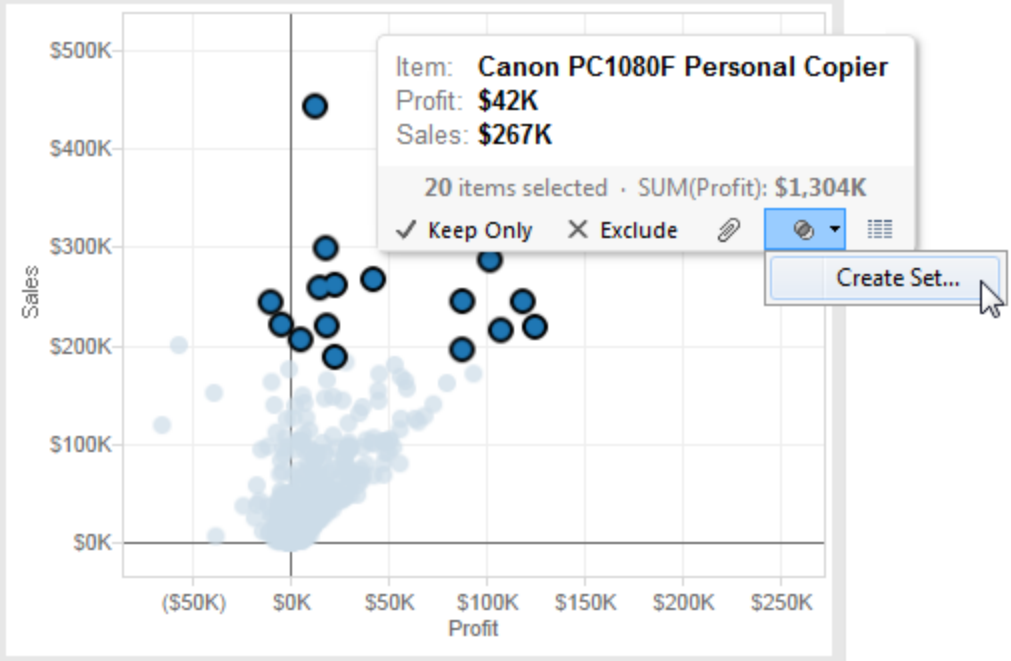


- **Aggregate measures** – If the views you create are slow, make sure you are working with aggregated measures rather than disaggregated measures. Slow views usually mean you are trying to view many rows of data at once. You can reduce the number of rows by aggregating the data. To do this, make sure the **Aggregate Measures** option on the **Analysis** menu is selected. For more information, see Disaggretating and Aggregating Data in the Tableau Knowledge Base.[5]



- **Sets** – If you want to filter a dimension to remove members based on a range of measure values, you can create a set rather than using a quantitative filter. For instance, you can create a set that only returns the **Top 50** items in a dimension, rather than all of the items in a dimension. For more information, see Creating and Using Sets in the Tableau online help.[6]

[3] http://onlinehelp.tableausoftware.com/current/pro/online/en-us/filtering.html
[4] http://onlinehelp.tableausoftware.com/current/pro/online/en-us/performance_contextfilters.html
[5] http://kb.tableausoftware.com/articles/knowledgebase/disaggregating-and-aggregating-data
[6] http://onlinehelp.tableausoftware.com/current/pro/online/en-us/help.htm#sortgroup_sets.html?Highlight=sets



- **Include only columns you need** – When creating a group from a selection as described in Sorting, Grouping, and Sets in the Tableau online help, make sure you've included only the columns of interest.[7] Each additional column included in the set will result in decreased performance.
- **Add filters first** – If you are working with a large data source and have automatic updates turned off, you may be creating an expensive query when adding filters to the view. Rather than building the view and then specifying filters, you should first specify the filters and then drag fields to the view. That way, when you run the update or turn on automatic updates, the filters will be evaluated first.

For more details on optimizing Tableau's performance when designing a fast performing dashboard, see the following articles in the Tableau Knowledge Base and online tutorials, respectively:

- Optimizing Tableau Server Performance[8]
- Authoring for Performance on Tableau Server[9]

[7] http://onlinehelp.tableausoftware.com/current/pro/online/en-us/sorting_groups.html
[8] http://kb.tableausoftware.com/articles/knowledgebase/optimizing-tableau-server-performance
[9] http://www.tableausoftware.com/learn/tutorials/on-demand/tableau-server-authoring-performance

Optimizing Amazon Redshift for Query Performance

Many factors can affect Amazon Redshift querying performance:

- Cluster Size
- Data encryption
- Distribution style and key
- Sort key
- Compression settings
- Vacuuming
- Analyzing
- Relationships
- Data size
- The complexity of your queries

Appropriately Sized Cluster Capacity

The type and number of compute nodes chosen determines the total amount of compute, memory, and storage available to the Amazon Redshift cluster. Compute, memory, and storage influences the speed of your queries, the amount of query concurrency that can be effectively achieved, and the amount of data the cluster can store. Amazon Redshift supports two types of nodes: dense storage and dense compute.

- **Dense storage** nodes (DW1) are useful for creating very large data warehouses using hard disk drives (HDDs) for a very low price point. They are the most cost-effective and highest performance option for customers with more than 1TB of data in their data warehouses (or 500 GB in a single node cluster). 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 compute** nodes (DW2) 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. The DW2 family has a cost-effective smaller node for data warehouses less than 1TB. 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.

Each node class has two sizes for a total of four different node types. You add compute, storage and memory capacity by increasing the number of nodes or changing the node size to a larger node. You can use either command line tools or the Amazon Redshift console to change node types or number of nodes, including changing from dense storage nodes to dense compute nodes.

Having a larger node size supports a higher level of concurrency in refreshing workbooks. This is because larger nodes support more cursors. If you are using the direct connect capability in your workbooks, then ensuring that you have adequate compute and memory capacity in addition to sufficient storage results in better query performance and a snappier end user experience. The speed at which you require your workbooks to be refreshed and the amount of direct connect queries you are pushing down to the data warehouse will influence your cluster sizing decisions. Consider a higher capacity if you have many workbooks being refreshed concurrently, or if you expect a lot of users running direct connect queries concurrently.

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

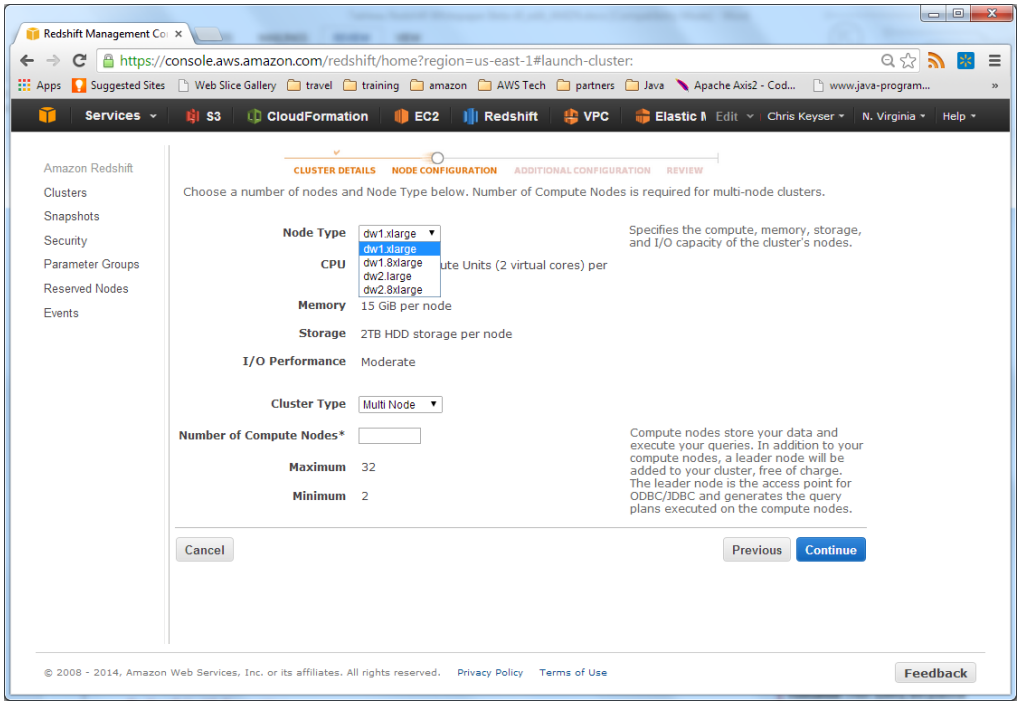


Figure 2: Improve query performance by adding a higher capacity node type.

For more information on node types and sizing options, see Amazon Redshift Clusters in the *Amazon Redshift Management Guide*.[10]

Be Careful with 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. Amazon Redshift also supports Hardware Security Modules (HSMs) on premises and using AWS CloudHSM for direct control of encryption-key generation and management by customers. Using encryption helps customers meet regulatory requirements and protects highly sensitive data at rest. 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.

[10] http://docs.aws.amazon.com/redshift/latest/mgmt/working-with-clusters.html

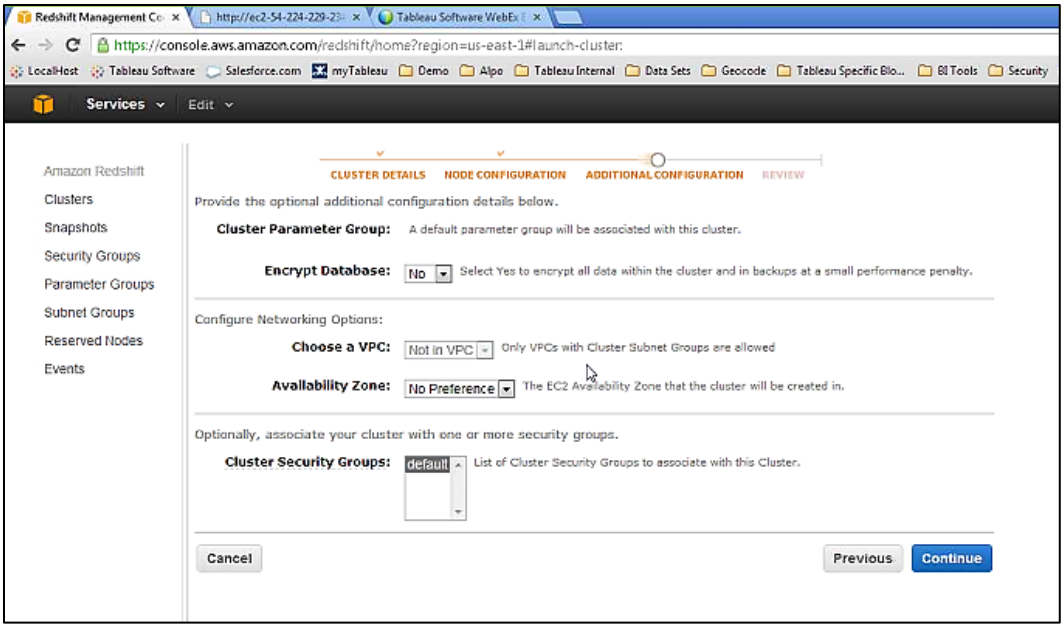


Figure 3: Amazon Redshift supports encryption options.

*For more information on encryption, see Amazon Redshift Database Encryption in the *Amazon Redshift Management Guide*.[11]

Choose the Right Distribution Style

Choosing a distribution style and optimal distribution key for your tables can significantly improve the performance of joins. Amazon Redshift scales out by parallelizing operations across multiple nodes. The distribution style (EVEN, KEY, or ALL) defines how data for a table is spread across the cluster.

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 of the rows with the same key value always go to the same node. If two different tables use the distribution style of KEY, the rows for 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. Always make sure that the distribution key results in relatively even distribution of table data.

A second distribution style, ALL, also promotes co-location of data on a join. This style distributes all of the data for a table to all of the nodes in the cluster. Replicating the data to each node has a storage cost and increases load time. The tables will always be local for any joins, which improves query

[11] http://docs.aws.amazon.com/redshift/latest/mgmt/working-with-db-encryption.html

performance. Good candidates for *ALL* distribution are 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.

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.[12]

Choosing a Sort Key

A sort key defines the order the data is stored on disk for a table. This ordering helps Amazon Redshift filter efficiently on the query conditions, or predicates, in your "*where*" clause. If you frequently join a table, such as a dimension table, specifying the same column for your distribution and sort key often enables Amazon Redshift to perform highly optimized joins on that column. If you have to choose between one field that allows for more efficient joins and a second that is used for filtering, then choose the commonly filtered column. You can define multiple columns for a sort key that will be applied in order. For example, if you chose the Color column followed by the Product Type column for your sort key, then the data is sorted on disk first by color then by product type. For queries that filter on both color and product type, specifying multiple columns for your sort key will help. If a query is filtered by only product type, the sort key will have no impact, since the product type ordering is just within a color. If a query is filtered only on color, the sort key will apply since color is the leading column in the sort key. When a column specified for your sort key is highly selective (often called high cardinality), adding more columns 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. See Choosing a data distribution method[13] and Choosing sort keys[14] in the *Amazon Redshift Developer Guide* for information that will help you decide which columns to designate for sort and distribution keys.

Use 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 through the use of 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. Very 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[15] and Loading tables with

[12] http://docs.aws.amazon.com/redshift/latest/dg/t_Distributing_data.html
[13] http://docs.aws.amazon.com/redshift/latest/dg/t_Distributing_data.html
[14] http://docs.aws.amazon.com/redshift/latest/dg/t_Sorting_data.html
[15] http://docs.aws.amazon.com/redshift/latest/dg/t_Compressing_data_on_disk.html

automatic compression[16] in the *Amazon Redshift Developer Guide* for additional details on loading data with compression 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.

Vacuum Your Tables

After loading data that causes a significant number of additions, updates, or deletes, you should run a VACUUM command to optimize performance. VACUUM reclaims space for deleted items and ensures that the data is 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.

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 date-time, then execute the COPY commands in the order 20130810.csv, 20130811.csv, and 20130812.csv to prevent the need to vacuum.

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*.[18]

Analyze Your Tables

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*.[19]

[16] http://docs.aws.amazon.com/redshift/latest/dg/c_Loading_tables_auto_compress.html
[17] http://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html
[18] http://docs.aws.amazon.com/redshift/latest/dg/t_Reclaiming_storage_space202.html
[19] http://docs.aws.amazon.com/redshift/latest/dg/t_Analyzing_tables.html

Explore Key Relationships

You can define primary key and foreign key relationships in your data model as well. Amazon Redshift can use these relationships as hints when creating query plans as a way to optimize the query plan. Tableau uses primary and foreign key relationships to detect and remove redundant queries. You should define relationships, but only if you know that the source system for the data is enforcing the relationships (i.e., the system from which the data is being loaded into Amazon Redshift). Amazon Redshift does not enforce these relationship constraints and may return unexpected results for queries if a relationship is defined but integrity is not enforced. For more information see Defining constraints in the *Amazon Redshift Developer Guide*.[20]

Learn Other Amazon Redshift Performance Tips

The Amazon Redshift System Overview in the *Amazon Redshift Developer Guide* explains many of these concepts and considerations in great detail.[21] Amazon Redshift documentation has specific sections on performance tuning, including the following:

- Best practices for designing tables[22]
- Tuning Query Performance[23]
- Loading tables with automatic compression[24]
- Increasing the available memory[25]
- Best Practices for loading data[26]
- Implementing workload management

In addition, see the article Optimizing for Star Schemas on Amazon Redshift in the Articles & Tutorials section of the Amazon Web Services website.[27]

Considerations When Using Cursors

When returning queried data from Amazon Redshift, Tableau uses cursors. 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. Typically the size of data sets are much larger than you can 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 trade off between the maximum total result size allowed per cursor and the number of concurrent

[20] http://docs.aws.amazon.com/redshift/latest/dg/t_Defining_constraints.html
[21] http://docs.aws.amazon.com/redshift/latest/dg/c_redshift_system_overview.html
[22] http://docs.aws.amazon.com/redshift/latest/dg/c_designing-tables-best-practices.html
[23] http://docs.aws.amazon.com/redshift/latest/dg/c-optimizing-query-performance.html
[24] http://docs.aws.amazon.com/redshift/latest/dg/c_Loading_tables_auto_compress.html
[25] http://docs.aws.amazon.com/redshift/latest/dg/c_best-practices-increase-slot-count.html
[26] http://docs.aws.amazon.com/redshift/latest/dg/c_loading-data-best-practices.html
[27] http://aws.amazon.com/articles/8341516668711341

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.[28] Though we recommend querying live to gain the full benefits of using Tableau with Amazon Redshift, there may be cases in which you choose to use extracts. To avoid hitting the open cursor limit when dealing with large extracts in Tableau, it is best to avoid scheduling the refreshes in parallel. For more information, see Working with Amazon Redshift Concurrent Cursor Limit in the Tableau Knowledge Base.[29]

Conclusion

Business users today demand responsive and visually compelling solutions. By keeping in mind these performance tips and techniques, you can ensure that you deliver a great user experience when using Tableau Software with Amazon Redshift.

[28] http://docs.aws.amazon.com/redshift/latest/dg/declare.html
[29] http://kb.tableausoftware.com/articles/knowledgebase/redshift-concurrent-cursor-limit