BEST PRACTICES FOR USING TABLEAU WITH SNOWFLAKE

BY ALAN ELDRIDGE, ET. AL.

snowflake

WHITE PAPER

TABLE OF CONTENTS

1 Introduction

2 What Is Tableau?

2 What Is Snowflake?
2 Platform as a cloud service
3 Snowflake architecture

4 What You DON'T Have to Worry About with Snowflake

4 Creating Efficient Tableau Workbooks

5 Connecting Tableau to Snowflake
5 Use the right connector
6 Live connections versus Tableau extracts
7 Relationships versus joins
7 Assume Referential Integrity
8 Custom SQL
12 Initial SQL
14 Views
15 Materialized Views

16 Working with Semi-Structured Data
16 The VARIANT data type
18 Accessing semi-structured data from Tableau
20 RAWSQL
22 ELT

23 **Working with Snowflake Time Travel**
23 Accessing historical data
23 Accessing Snowflake Time Travel data from Tableau
25 Working With Snowflake Clones

26 **Working with Snowflake Secure Data Sharing**

27 **Implementing Role-Based Security**
27 Setting up the data access rules
28 Passing in the user context
28 Column-level security solution for any data tool
28 Secure user defined functions (secure UDFS)
29 Tableau-only solution
31 Solution for any data tool

33 **Using Custom Aggregations**

34 **Scaling Snowflake Warehouses**
34 Resizing a warehouse to improve performance
36 Adding warehouses to improve concurrency
38 Scale Across

39 **Caching**
39 Tableau caching
39 Presentation layer
39 Client-side rendering in the browser
40 Server-side rendering with tiles
40 More about view models and bootstrap responses
41 Analytics layer
41 Data layer
42 Snowflake caching
42 Result caching
42 Warehouse caching

43 **Other Performance Considerations**
43 Constraints
43 Temp tables

46 **Measuring Performance**
46 In Tableau
46 Performance recorder
47 Desktop logs
49 Server logs
50 Server performance views
50 Resource Monitoring Tool
51 TabJolt
51 In Snowflake
51 The Snowflake database
51 Snowflake Information Schema
51 Differences between account usage and Information Schema
52 Snowflake Query History
54 Snowflake Query Profile
55 Execution time
55 Statistics
55 "Exploding" joins
56 Queries too large to fit in memory
56 Inefficient pruning
57 Clustering
57 Linking performance data between Tableau and Snowflake

59 **Conclusion**

60 **About Snowflake**

Introduction

Data is increasingly important for organizations. Greater volumes of data are being generated and captured across systems at increasing levels of granularity. At the same time, more users are demanding access to this data to answer questions and gain business insight.

Snowflake and Tableau are leading technologies addressing the challenges of increasing data and demand. Snowflake provides a near limitless platform for data storage and processing, and Tableau provides a highly intuitive, self-service data analytics platform.

The objective of this white paper is to help you make best use of features from these highly complementary products. It is intended for Tableau users who are new to Snowflake, Snowflake users who are new to Tableau, and any users who are new to both.

This white paper describes the best ways to work with key Snowflake and Tableau features including:

- Tips for creating Tableau workbooks
- Important things to know about connecting Tableau and Snowflake, including effective use of relationships and joins, and information about when and how to use custom SQL
- Best practices for using semi-structured data
- Optimal ways to use Snowflake features such as Time Travel, Snowflake Secure Data Sharing, and scaling
- When and how to use custom aggregations
- Information about using role-based security
- Methods for using caches to improve performance
- Techniques to monitor performance

WHAT IS TABLEAU?

Tableau Software is a business intelligence solution that integrates data analysis and reporting into a continuous visual analysis cycle that lets everyday business users quickly explore data through charts and shift views on the fly. Tableau combines data exploration, visualization, reporting, and dashboarding into an application that is easy to learn and use.

Tableau's solution set consists of three main products:

- Tableau Desktop is the end-user tool for data analysis and dashboard building. It can be used on its own or with Tableau Server and Tableau Online.
- Tableau Prep is the data prep tool for cleaning, combining, and reshaping data before analysis and visualization.
- Tableau Server is the platform that provides services for collaboration, governance, administration, and content sharing. This can be deployed on premises or in the cloud (on AWS, Microsoft Azure, or GCP). Tableau Online is a software-as-a-service version of Tableau Server.

Either working standalone with Tableau Desktop, or by publishing content to Tableau Server or Tableau Online, you can directly work with data stored in Snowflake's cloud data platform.

WHAT IS SNOWFLAKE?

Snowflake's Data Cloud is a global network where thousands of organizations mobilize data with near-unlimited scale, concurrency, and performance. Inside the Data Cloud, organizations have a single unified view of data so they can easily discover and securely share governed data, and execute diverse analytics workloads. Snowflake provides a tightly integrated analytic data warehouse as a service, billed based on consumption. It is faster, easier to use, and far more flexible than traditional data warehouse offerings.

Snowflake uses a SQL database engine and a unique architecture designed specifically for the cloud.

Platform as a cloud service

Snowflake is a true SaaS offering. There is no hardware (virtual or physical) or software for you to select, install, configure, or manage. In addition, ongoing maintenance, management, and tuning are handled by Snowflake.

All components of Snowflake's service (other than an optional command-line client) run in a secure public or government cloud infrastructure.

Snowflake is cloud agnostic and uses virtual compute instances from each cloud provider (AWS EC2, Azure VM, Google Compute Engine). In addition, it uses object or file storage from AWS S3, Azure Blob Storage, or Google Cloud Storage for persistent storage of data. Due to Snowflake's unique architecture and cloud independence, you can seamlessly replicate data and operate from any of the clouds simultaneously.

For more information about Snowflake, visit the Snowflake website.

Snowflake architecture

Snowflake's architecture is a hybrid of traditional shared-disk database architectures and shared-nothing database architectures. Similar to shared-disk architectures, Snowflake uses a central data repository for persisted data that is accessible from all compute nodes in the data platform. But similar to shared-nothing architectures, Snowflake processes queries using massively parallel processing (MPP) compute clusters where each node in the cluster stores a portion of the entire data set locally. This approach offers the data management simplicity of a shared-disk architecture, but with the performance and scale-out benefits of a shared-nothing architecture.

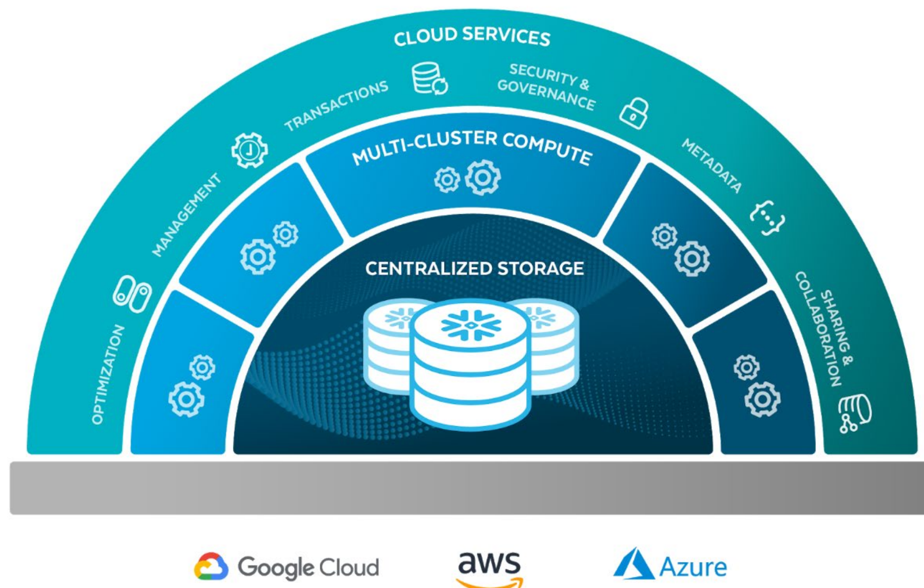


Figure 1

Snowflake's unique architecture consists of three layers built upon a public cloud infrastructure:

- **Cloud services:** Services that coordinate activities across Snowflake, processing user requests from login to query dispatch. This layer provides optimization, management, security, sharing, and other features.
- **Multi-cluster compute:** Snowflake processes queries using virtual warehouses. Each virtual warehouse is an MPP compute cluster composed of multiple compute nodes allocated by Snowflake from Amazon EC2, Azure VM, or Google Cloud Compute. Each virtual warehouse has independent compute resources, so high demand in one virtual warehouse has no impact on the performance of other virtual warehouses. For more information, see Virtual Warehouses in the Snowflake online documentation.
- **Centralized Storage:** Snowflake uses AWS S3, Azure Blob Storage, or Google Cloud Storage to store data into its internal optimized, compressed, columnar format using micro-partitions. Snowflake manages the data organization, file size, structure, compression, metadata, statistics, and replication. Data objects stored by Snowflake are not directly visible by customers, but they are accessible through SQL query operations run using Snowflake.

WHAT YOU DON'T HAVE TO WORRY ABOUT WITH SNOWFLAKE

Because Snowflake is a cross-cloud platform offered as a service, there are lots of things you don't need to worry about compared to a traditional on-premises solution:

- **Installing, provisioning, and maintaining hardware and software:** All you need to do is create an account and load your data. You can then immediately connect from Tableau and start querying.
- **Determining the capacity of your data warehouse:** Snowflake has scalable compute and storage, so it can accommodate all of your data and all of your users. You can adjust the count and size of your virtual warehouses to handle peaks and lulls in your data usage. You can even turn your warehouses completely off to stop incurring costs when you are not using them.
- **Learning new tools and expanded SQL capabilities:** Snowflake is fully ANSI-SQL compliant, so you can use the skills and tools you already have (like Tableau). Snowflake provides connectors for ODBC, JDBC, Python, Spark, and Node.js, as well as web and command-line interfaces.
- **Siloed structured and semi-structured data:** Business users increasingly need to work with both traditionally structured data (for example, data in VARCHAR, INT, and DATE columns in tables) as well as semi-structured data in formats like XML, JSON ,and Parquet. Snowflake provides a special data type called VARIANT that enables you to load your semi-structured data in natively and then query it with SQL.
- **Optimizing and maintaining your data:** You can run analytic queries quickly and easily without worrying about managing how your data is indexed or distributed across partitions. Snowflake also provides built-in data protection capabilities, so you don't need to worry about snapshots, backups, or other administrative tasks like running VACUUM jobs.
- **Securing your data and complying with international privacy regulations:** All data is encrypted when it is loaded into Snowflake, and it is kept encrypted at all times when at rest and in transit. If your business requirements include working with data that requires HIPAA, PII, PCI, FEDRamp compliance, and more, Snowflake can support these validations with the Business Critical and higher editions.
- **Sharing data securely:** Snowflake Secure Data Sharing enables you to share near real-time data internally and externally between Snowflake accounts without copying and moving data sets. Data providers provide secure data shares to their data consumers, who can view and seamlessly combine it with their own data sources. Snowflake Data Marketplace includes many data sets that you can incorporate into your existing business data, such as weather, demographics, or traffic, for greater data-driven insights.

CREATING EFFICIENT TABLEAU WORKBOOKS

To create efficient Tableau workbooks, follow the guidelines in the Designing Efficient Workbooks white paper.

The key points are:

- **Keep it simple:** Most performance problems are caused by inefficient workbook design. Allow your users to incrementally drill down to details by filtering, rather than trying to show everything at once.
- **Less is more:** The fewer rows and columns you work with, the faster your queries will execute. Also, the fewer marks you draw, the faster your workbooks will render.
- **Trust your tools:** The query generator in Tableau is one of the most efficient on the market, so trust it to create the queries for you. The less you customize queries, the better they will be.

CONNECTING TABLEAU TO SNOWFLAKE

This section describes several important considerations for connecting Tableau to Snowflake, including selecting the right connector, when to use relationships, when to use initial SQL, and how to use views.

Use the right connector

To connect to Tableau to Snowflake, use the native connector option, Snowflake, as shown below. This ensures that Tableau generates SQL optimized for running on Snowflake.

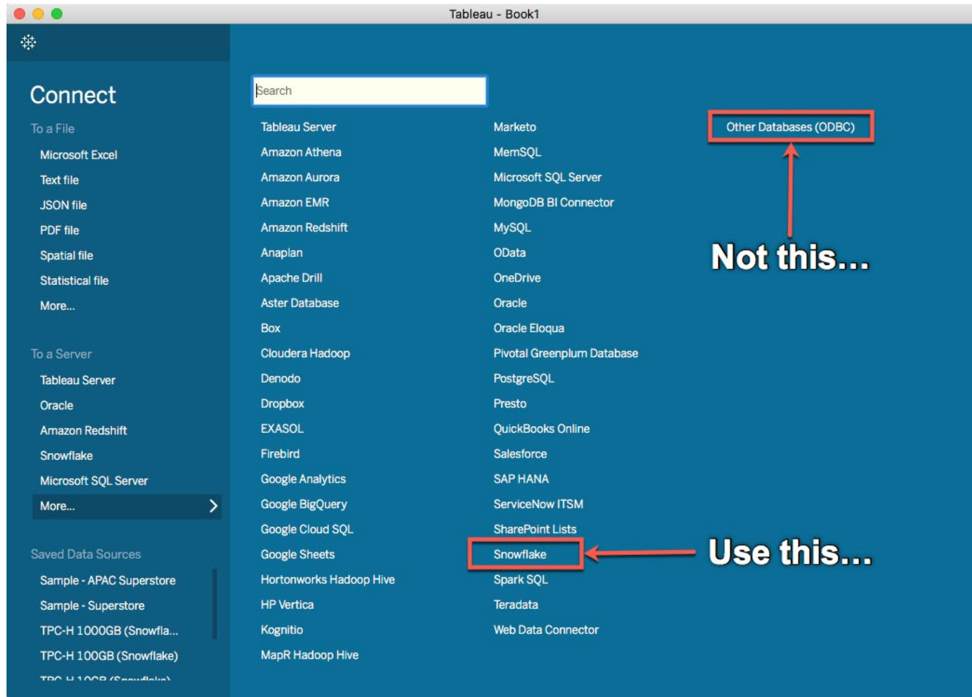


Figure 2

For details on connection information, see the Tableau documentation.

After you select the native connector, you see the following dialog box:

Snowflake
Server:
Role: Optional
Enter information to sign in to the server:
Authentication: Username and Password
Username:
Password:
SAML IdP(Okta):
Enter custom driver parameters:
Initial SQL...
Sign In

Figure 3

Enter the information as prompted. Here are some notes to keep in mind:

- **Role:** The role determines which warehouses, databases, schemas, and tables are accessible. If you leave this blank, the system defaults to the Snowflake default role.
- **SAML IdP (Okta):** If you are using SAML authentication, enter **externalbrowser**. This will open a web browser to your SAML provider and enable you to authenticate through the browser.

After you sign in, you see the data source panel. This dialog box lets you select the virtual warehouse, database, and schema. It also lets you set up relationships and joins.

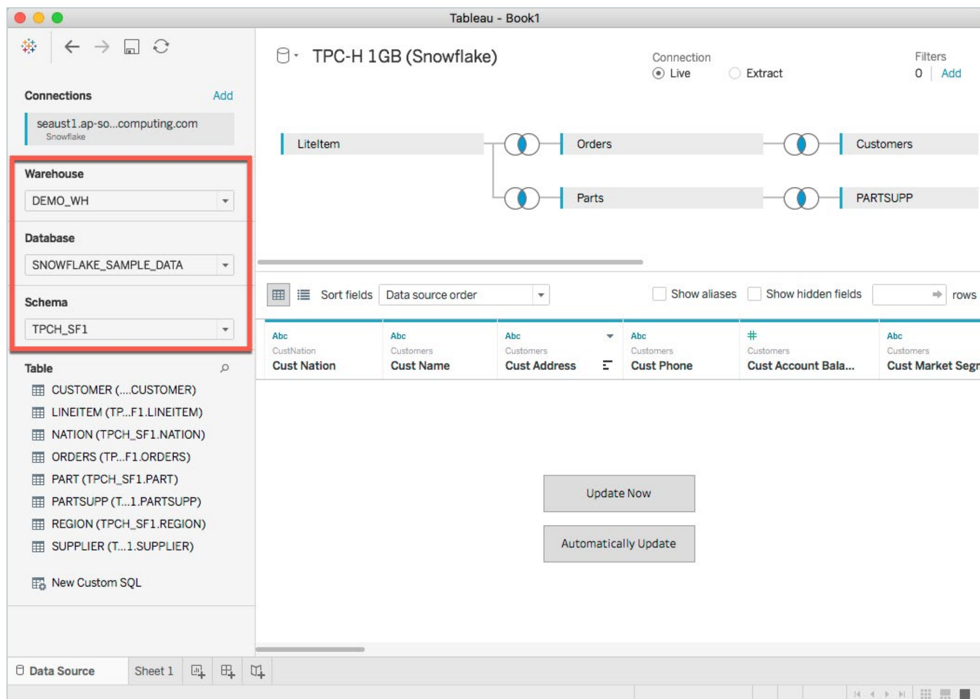


Figure 4

Live connections versus Tableau extracts

The data source panel enables you to select either a live connection or an extract (upper right corner). To take advantage of Snowflake's high performance data warehouse, select Live.

However, you may want to use a Tableau extract (a compressed snapshot of data loaded into memory) for any of the following situations:

- Users require an offline data cache that can be used without a connection to Snowflake.
- Users are joining Snowflake with other data sources that are slow. Creating an extract will pull data from both sources and remove the performance bottleneck from the additional source.
- Users create aggregated extracts to act as summarized data caches. This can be an effective approach to working with large, slow data lakes. However, because Snowflake can provide fast query results when processing large volumes of structured and semi-structured data, this may be unnecessary.

Note: To create or modify an extract connection, you need a live connection. (You cannot use webedit.)

The following sections describe alternatives for connecting Snowflake and Tableau, including relationships, custom SQL, initial SQL, and views. All of the approaches are valid if implemented correctly. You should select the most appropriate solution based on your particular needs of performance, data freshness, maintainability, and reusability.

Relationships versus joins

Tableau 2020.2 introduced new data modeling capabilities that enable users to create relationships between tables rather than specifying the join type and key. Relationships are a dynamic, flexible way to combine data from multiple tables for analysis. We recommend using relationships as your first approach to combining your data because it makes data preparation and analysis easier and more intuitive. Use joins only when you absolutely need to.

Here are some advantages to using relationships to combine tables:

- Make your data source easier to define, change, and reuse.
- Make it easier to analyze data across multiple tables at the correct level of detail (LOD).
- Do not require the use of LOD expressions or LOD calculations for analysis at different levels of detail.
- Only query data from tables with fields used in the current viz.

The Tableau query engine produces the following optimal query that joins only the tables needed and returns just the displayed columns:

```
SELECT "CUSTOMER"."C_MKTSEGMENT" AS "C_MKTSEGMENT",
    COUNT(DISTINCT "ORDERS"."O_ORDERKEY") AS "ctd:O_ORDERKEY:ok",
    DATE_TRUNC('MONTH',"ORDERS"."O_ORDERDATE") AS "tmn:O_ORDERDATE:ok"
FROM "TPCH_SF1"."ORDERS" "ORDERS"
    LEFT JOIN "TPCH_SF1"."CUSTOMER" "CUSTOMER" ON ("ORDERS"."O_CUSTKEY" =
"CUSTOMER"."C_CUSTKEY")
GROUP BY 1,
    3
```

Based on the actions in the drag and drop, Tableau will create the optimal SQL with the necessary joins.

More information on relationships can be found here:

Assume Referential Integrity

In some cases, you can improve query performance for joins by selecting the option to **Assume Referential Integrity** from the **Data** menu. When you use this option, Tableau will include the joined table in the query only if it is specifically referenced by fields in the view.

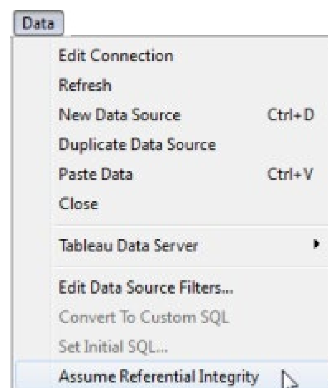


Figure 5

Using this setting is appropriate when you know that your data has referential integrity but your database is not enforcing or cannot enforce referential integrity (note: Snowflake does not enforce referential integrity). The **Assume Referential Integrity** option in Tableau can only affect performance on Tableau's end. If your data does not have referential integrity and you turn on this setting, query results may not be reliable.

Custom SQL

Tableau can generate efficient queries if you define the relationships between the tables and let the query engine write SQL specific to the view being created. However, sometimes specifying relationships in the data connection window does not offer all the flexibility you need.

Although creating a data connection using a custom SQL statement can be useful, it can reduce performance. This is because, in contrast to defining relationships, custom SQL is never deconstructed and is always executed atomically. This means no join culling occurs, and the whole query may have to be processed, possibly multiple times.

If you need to use custom SQL, for example to analyze semi-structured data or use Snowflake Time Travel, there are best practices you can follow to mitigate some of the issues. These include *Initial SQL* and *Views*.

Example

Consider the example used in the *Relationships versus joins* section. It shows the number of records in the TPCH_SF1 sample schema for each month, broken down by the market segment:

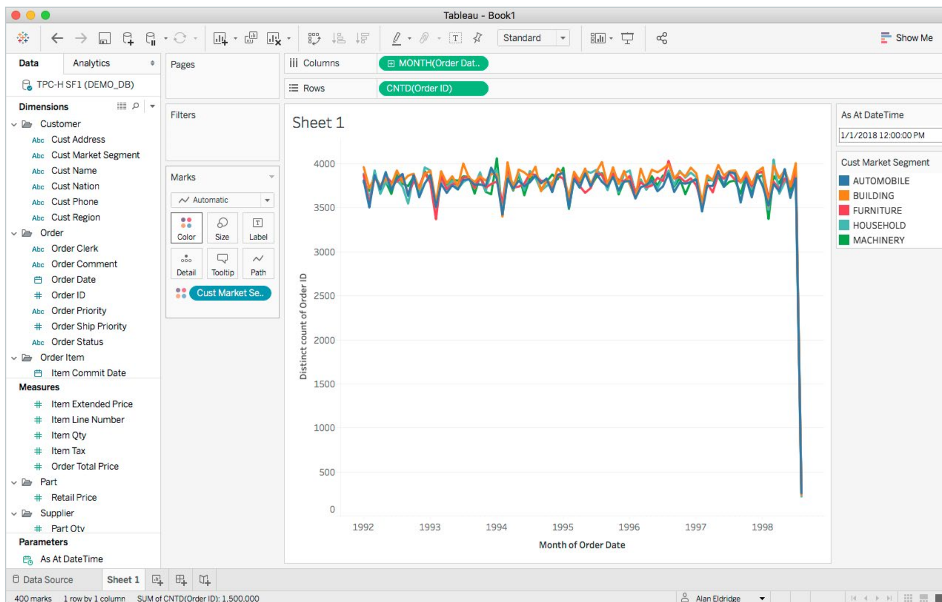


Figure 6

If the underlying data model uses the recommended approach of relating the tables, the resulting query joins only the tables needed and returns just the columns being displayed:

```
SELECT “CUSTOMER”.”C_MKTSEGMENT” AS “C_MKTSEGMENT”,
    COUNT(DISTINCT “ORDERS”.”O_ORDERKEY”) AS “ctd:O_ORDERKEY:ok”,
    DATE_TRUNC(‘MONTH’,”ORDERS”.”O_ORDERDATE”) AS “tmn:O_ORDERDATE:ok”
FROM “TPCH_SF1”.”ORDERS” “ORDERS”
    LEFT JOIN “TPCH_SF1”.”CUSTOMER” “CUSTOMER” ON (“ORDERS”.”O_CUSTKEY” =
“CUSTOMER”.”C_CUSTKEY”)
GROUP BY 1,
    3
```

This results in the following optimal query plan:

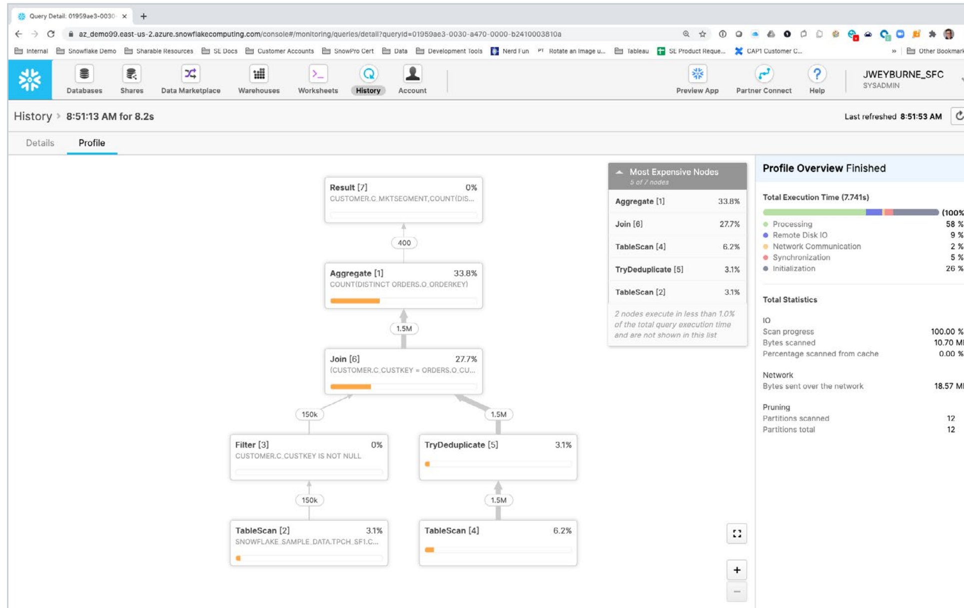


Figure 7

What happens, though, if business logic is embedded in the data model? Using custom SQL, there are two approaches. You can isolate the custom SQL to the affected part of the model, keeping the rest of the schema as join definitions, or you can encapsulate the entire data model (with all table joins) into a single, monolithic custom SQL statement.

To demonstrate the first approach, the following example replaces the ORDERS table with a custom SQL statement:

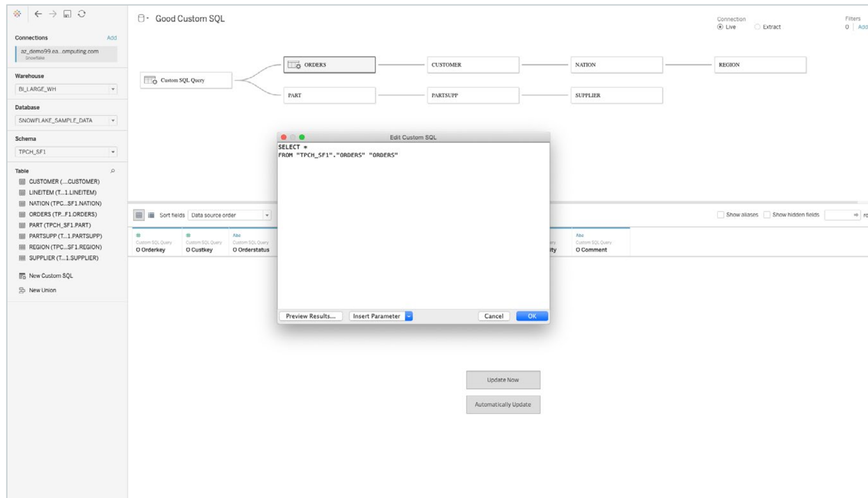


Figure 8

Tableau generates the following query (the custom SQL is highlighted):

```
SELECT "CUSTOMER"."C_MKTSEGMENT" AS "C_MKTSEGMENT",
    COUNT(DISTINCT "Custom SQL Query"."O_ORDERKEY") AS "ctd:O_ORDERKEY:ok",
    DATE_TRUNC('MONTH',"Custom SQL Query"."O_ORDERDATE") AS "tmn:O_ORDERDATE:ok"
FROM (
    SELECT *
    FROM "TPCH_SF1"."ORDERS" "ORDERS"
) "Custom SQL Query"
    LEFT JOIN "TPCH_SF1"."CUSTOMER" "CUSTOMER" ON (TRUNC("Custom SQL Query"."O_CUSTKEY") =
"CUSTOMER"."C_CUSTKEY")
GROUP BY 1,
    3
```

The custom SQL is not decomposed, but because its scope is just for the ORDERS table, Tableau can cull (eliminate) the joins to the unneeded tables. The Snowflake optimizer then parses this into an optimal query plan, identical to the initial example:

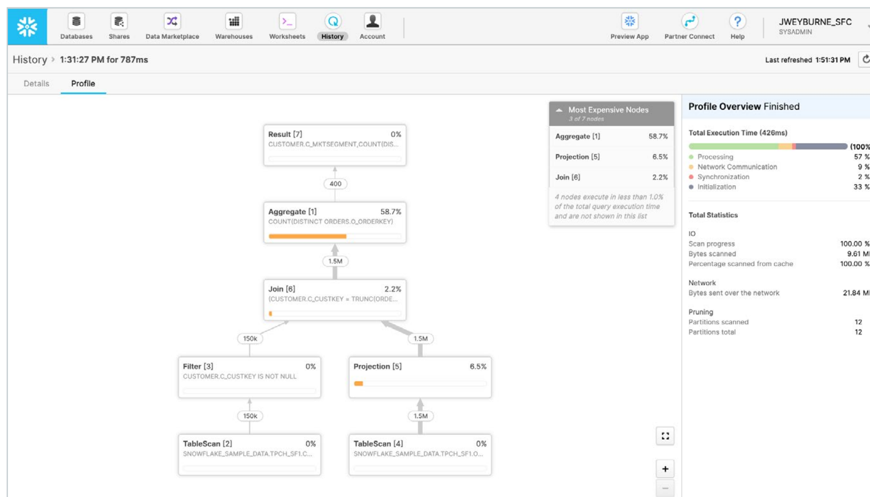


Figure 9

New Tableau users might try using the second approach and encapsulate the entire data model (with all table joins) into a single, monolithic custom SQL statement.

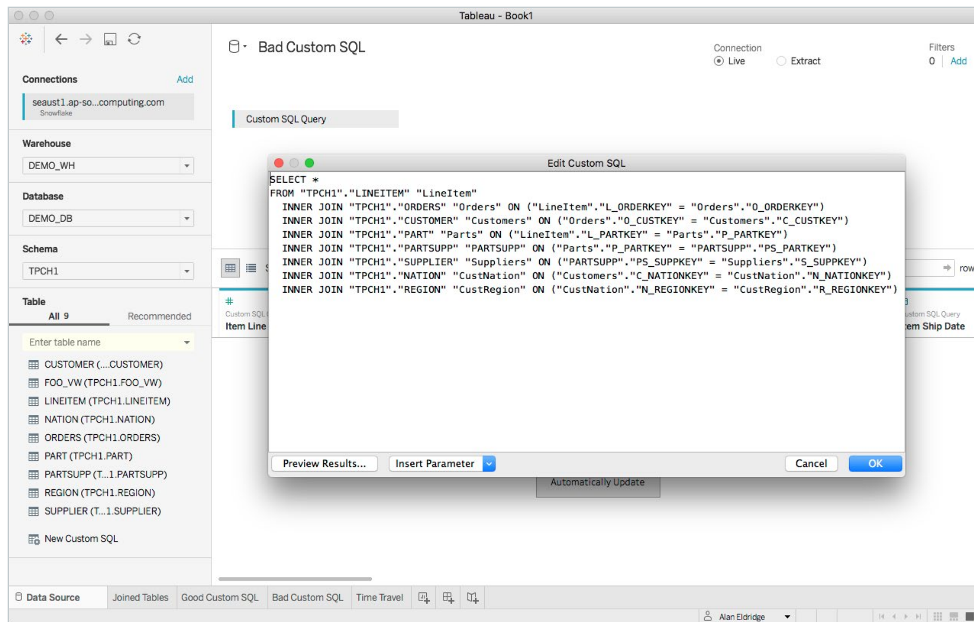


Figure 10

Again, the custom SQL is not decomposed so the Tableau query engine wraps the custom SQL in a surrounding SELECT statement. This means there is no join culling and Snowflake is required to join all the tables together before the required subset of data is selected (the custom SQL is highlighted):

```
SELECT "Custom SQL Query"."C_MKTSEGMENT" AS "C_MKTSEGMENT",
    COUNT(DISTINCT "Custom SQL Query"."O_ORDERKEY") AS "ctd:O_ORDERKEY:ok",
    DATE_TRUNC('MONTH',"Custom SQL Query"."O_ORDERDATE") AS "tmn:O_ORDERDATE:ok"
FROM (
    SELECT *
    FROM "TPCH_SF1"."LINEITEM" "LineItem"
    INNER JOIN "TPCH_SF1"."ORDERS" "Orders" ON ("LineItem"."L_ORDERKEY" =
"Orders"."O_ORDERKEY")
    INNER JOIN "TPCH_SF1"."CUSTOMER" "Customers" ON ("Orders"."O_CUSTKEY" =
"Customers"."C_CUSTKEY")
    INNER JOIN "TPCH_SF1"."PART" "Parts" ON ("LineItem"."L_PARTKEY" = "Parts"."P_PARTKEY")
    INNER JOIN "TPCH_SF1"."PARTSUPP" "PARTSUPP" ON ("Parts"."P_PARTKEY" =
"PARTSUPP"."PS_PARTKEY")
    INNER JOIN "TPCH_SF1"."SUPPLIER" "Suppliers" ON ("PARTSUPP"."PS_SUPPKEY" =
"Suppliers"."S_SUPPKEY")
    INNER JOIN "TPCH_SF1"."NATION" "CustNation" ON ("Customers"."C_NATIONKEY" =
"CustNation"."N_NATIONKEY")
    INNER JOIN "TPCH_SF1"."REGION" "CustRegion" ON ("CustNation"."N_REGIONKEY" =
"CustRegion"."R_REGIONKEY")
) "Custom SQL Query"
GROUP BY 1,
    3
```

This results in a less efficient query:

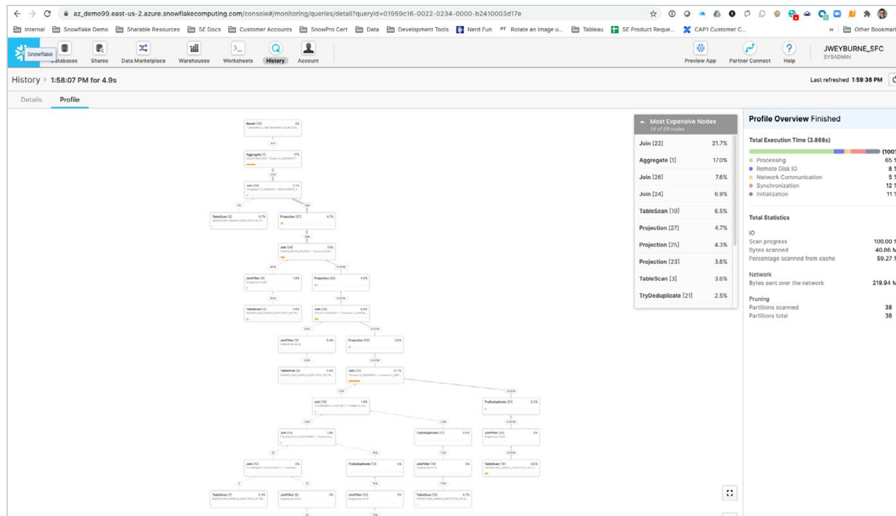


Figure 11

This approach is inefficient as this entire query plan needs to be run for potentially every query in the dashboard.

Initial SQL

If, for some reason, you need to use custom SQL, you can avoid repeated runs by using initial SQL to create a temporary table to be the selected table in your query. Because initial SQL is executed only once when the workbook is opened (as opposed to every time the visualization is changed for custom SQL), this could significantly improve performance, but the data populated into the temp table will be static for the duration of the session, even if the data in the underlying tables changes.

You can also use initial SQL to set additional session context within Snowflake. Commands such as USE WAREHOUSE, USE ROLE, and USE DATABASE can be leveraged to alter the Tableau Server user's operational context in Snowflake from the original author. Note the Tableau Server user must have rights in Snowflake to change to the new context. Further, the Tableau Server user's context can be sent into initial SQL to a Table Value Function (TVF) to pull back a highly targeted data set for that user.

Other variables from Tableau can also be passed into Snowflake to address additional use cases in the integration. These variables are listed below (see Tableau Documentation for more details):

PARAMETER	DESCRIPTION	EXAMPLE VALUE
TableauServerUser	The user name of the current server user. Use when setting up impersonation on the server. Returns an empty string if the user is not signed in to Tableau Server.	jsmith
TableauServerUserFull	The user name and domain of the current server user. Use when setting up impersonation on the server. Returns an empty string if the user is not signed in to Tableau Server.	domain.lan\jsmith
TableauApp	The name of the Tableau application.	Low
TableauVersion	The version of the Tableau application.	Low
WorkbookName	The name of the Tableau workbook. Use only in workbooks with an embedded data source.	Med

Example

Using the example above, instead of placing the entire query in a custom SQL statement, you could use it in an initial SQL block and instantiate a temporary table:

```
CREATE OR REPLACE TEMP TABLE TPCH1.FOO AS
SELECT *
FROM "TPCH1"."LINEITEM" "LineItem"
    INNER JOIN "TPCH1"."ORDERS" "Orders" ON ("LineItem"."L_ORDERKEY" =
"Orders"."O_ORDERKEY")
    INNER JOIN "TPCH1"."CUSTOMER" "Customers" ON ("Orders"."O_CUSTKEY" =
"Customers"."C_CUSTKEY")
    INNER JOIN "TPCH1"."PART" "Parts" ON ("LineItem"."L_PARTKEY" = "Parts"."P_PARTKEY")
    INNER JOIN "TPCH1"."PARTSUPP" "PARTSUPP" ON ("Parts"."P_PARTKEY" =
"PARTSUPP"."PS_PARTKEY")
    INNER JOIN "TPCH1"."SUPPLIER" "Suppliers" ON ("PARTSUPP"."PS_SUPPKEY" =
"Suppliers"."S_SUPPKEY")
    INNER JOIN "TPCH1"."NATION" "CustNation" ON ("Customers"."C_NATIONKEY" =
"CustNation"."N_NATIONKEY")
    INNER JOIN "TPCH1"."REGION" "CustRegion" ON ("CustNation"."N_REGIONKEY" =
"CustRegion"."R_REGIONKEY");
```

The FOO table is selected as the data source:

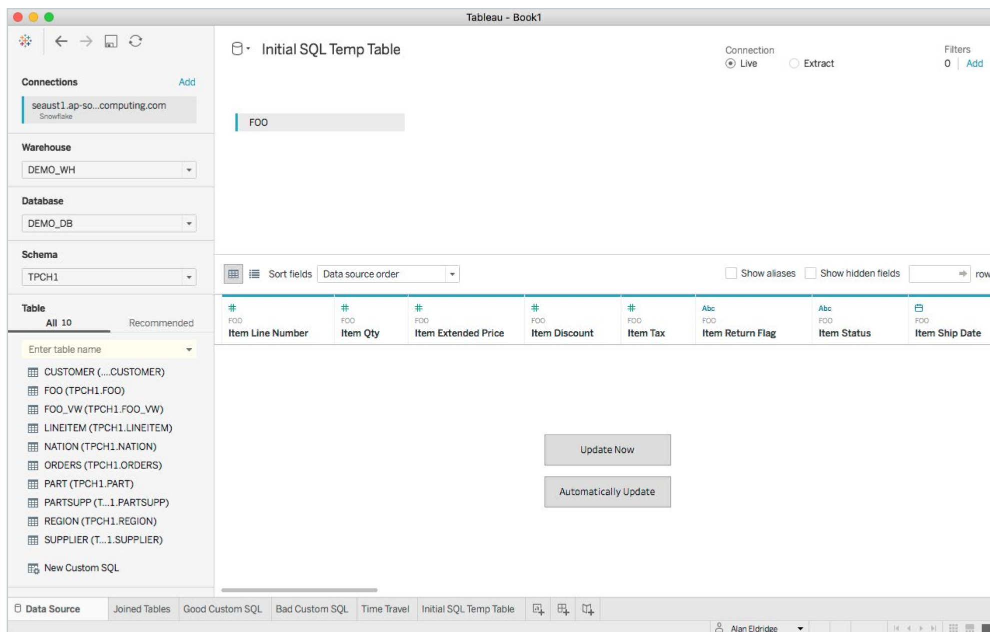


Figure 12

And Tableau generates the following query:

```
SELECT "FOO_VW"."C_MKTSEGMENT" AS "C_MKTSEGMENT",
    COUNT(DISTINCT "FOO_VW"."O_ORDERKEY") AS "ctd:O_ORDERKEY:ok",
    DATE_TRUNC('MONTH',"FOO_VW"."O_ORDERDATE") AS "tmn:O_ORDERDATE:ok"
FROM "TPCH1"."FOO" "FOO"
GROUP BY 1,
    3
```

This has a very simple query plan and fast execution time, but the data returned will not reflect changes to the underlying fact tables until a new session is started and the temp table is recreated.

Note: If you plan to publish your workbook to share with others, be sure that the administrator has not restricted initial SQL from running. Also, note that temp tables take up additional space in Snowflake that will contribute to the account storage charges, but because they are ephemeral this is generally not significant.

Views

Views are another alternative to using custom SQL. Unlike initial SQL, views ensure that your results contain current data. Here is an example of a view:

```
CREATE OR REPLACE VIEW FOO_VW AS
    SELECT *
    FROM "TPCH1"."LINEITEM" "LineItem"
        INNER JOIN "TPCH1"."ORDERS" "Orders" ON ("LineItem"."L_ORDERKEY" =
    "Orders"."O_ORDERKEY")
        INNER JOIN "TPCH1"."CUSTOMER" "Customers" ON ("Orders"."O_CUSTKEY" =
    "Customers"."C_CUSTKEY")
        INNER JOIN "TPCH1"."PART" "Parts" ON ("LineItem"."L_PARTKEY" = "Parts"."P_PARTKEY")
        INNER JOIN "TPCH1"."PARTSUPP" "PARTSUPP" ON ("Parts"."P_PARTKEY" =
    "PARTSUPP"."PS_PARTKEY")
        INNER JOIN "TPCH1"."SUPPLIER" "Suppliers" ON ("PARTSUPP"."PS_SUPPKEY" =
    "Suppliers"."S_SUPPKEY")
        INNER JOIN "TPCH1"."NATION" "CustNation" ON ("Customers"."C_NATIONKEY" =
    "CustNation"."N_NATIONKEY")
        INNER JOIN "TPCH1"."REGION" "CustRegion" ON ("CustNation"."N_REGIONKEY" =
        "CustRegion"."R_REGIONKEY")
```

In Tableau, use the view as the data source:

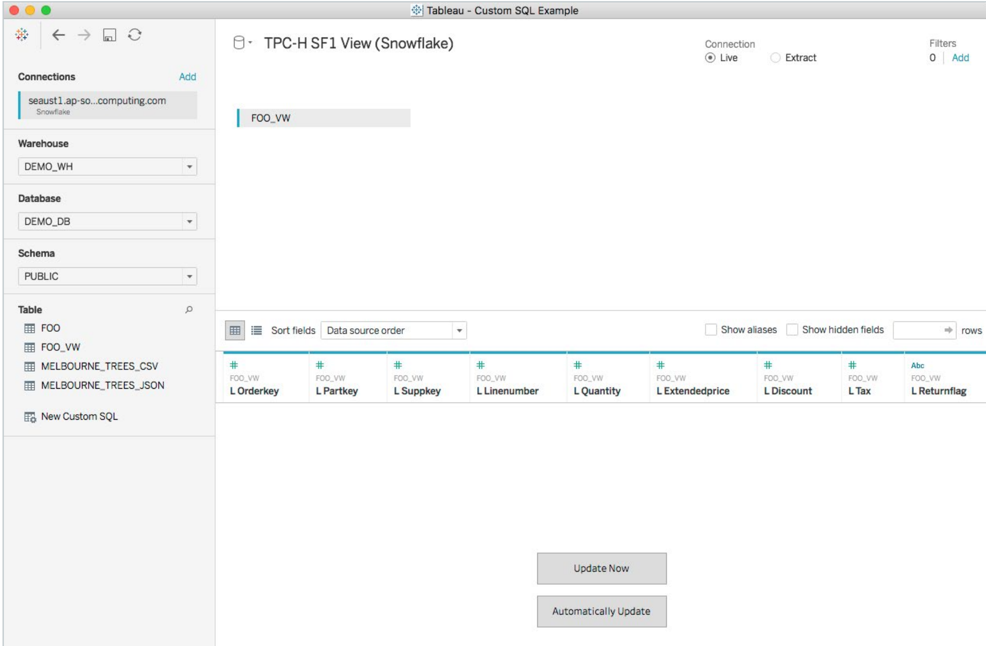


Figure 13

The query generated by Tableau is simple:

```
SELECT "FOO_VW"."C_MKTSEGMENT" AS "C_MKTSEGMENT",
    COUNT(DISTINCT "FOO_VW"."O_ORDERKEY") AS "ctd:O_ORDERKEY:ok",
    DATE_TRUNC('MONTH',"FOO_VW"."O_ORDERDATE") AS "tmn:O_ORDERDATE:ok"
FROM "TPCH1"."FOO_VW" "FOO_VW"
GROUP BY 1,
    3
```

However, this takes longer to run and has a less efficient query plan as the view needs to be evaluated at query time. The benefit is that you will always see up-to-date data in your results:

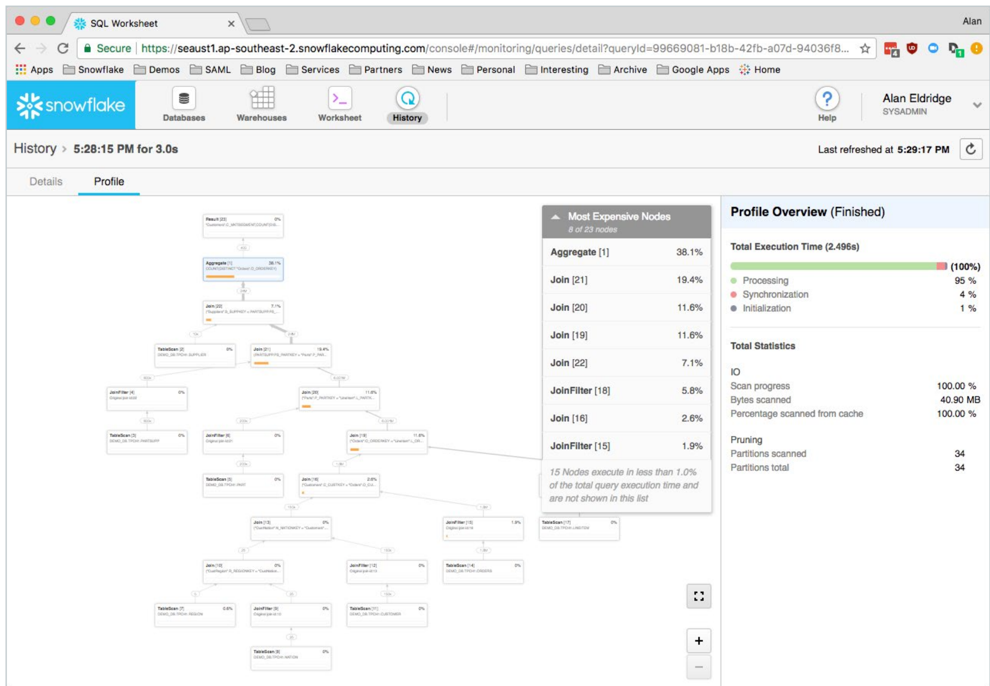


Figure 14

Materialized Views

In scenarios where view performance against a single table is less than desirable, consider using Materialized Views (MVs). Materialized views are automatically and transparently maintained by Snowflake. A background service updates the materialized view after changes are made to the base table. This is more efficient and less error-prone than manually maintaining the equivalent of a materialized view at the application level. Data accessed through materialized views is always current, regardless of the amount of DML that has been performed on the base table.

It's important to note that users don't need to specify a materialized view in a SQL statement in order for the view to be used. The query optimizer can automatically rewrite queries against the base table or regular views to use the materialized view instead.

For example, suppose that a materialized view contains all of the rows and columns that are needed by a query against a base table. The optimizer can decide to rewrite the query to use the materialized view, rather than the base table. This can dramatically speed up a query, especially if the base table contains a large amount of historical data.

WORKING WITH SEMI-STRUCTURED DATA

Today, business users work with data in multiple forms from numerous sources, including an ever-expanding amount of machine-generated data from applications, sensors, and mobile devices. Increasingly, this data is provided in semi-structured data formats such as JSON, Avro, ORC, Parquet, and XML that have flexible schemas. These semi-structured data formats do not conform to the standards of traditionally structured data, but instead contain tags or other types of markup that identify individual, distinct elements within the data:


```
{ "city": {
        "coord": { "lat": -37.813999, "lon": 144.963318 },
        "country": "AU",
        "findname": "MELBOURNE",
        "id": 2158177,
        "name": "Melbourne",
        "zoom": 5 },
    "clouds": {
        "all": 88 },
    "main": {
        "humidity": 31, "pressure": 1010, "temp": 303.4,
        "temp_max": 305.15, "temp_min": 301.15 },
    "rain": {
        "3h": 0.57 },
    "time": 1514426634,
    "weather": [ {
        "description": "light rain", "icon": "10d",
        "id": 500, "main": "Rain" } ],
    "wind": {
        "deg": 350, "speed": 4.1 }
}
```


Two of the key attributes that distinguish semi-structured data from structured data are nested data structures and the lack of a fixed schema:

- Unlike structured data, which represents data as a flat table, semi-structured data can contain multiple-level hierarchies of nested information.
- Structured data requires a fixed schema that is defined before the data can be loaded and queried in a relational database system. Semi-structured data does not require a prior schema definition, and the schema can constantly evolve so new attributes can be added at any time.

Tableau 10.1 introduced support for directly reading JSON data files. However, you can achieve broader support for semi-structured data by first loading the data into Snowflake. Snowflake provides native support for semi-structured data, including:

- Flexible schema data types for loading semi-structured data without transformation
- Direct ingestion of JSON, Avro, ORC, Parquet, and XML file formats
- Automatic conversion of data to Snowflake's optimized internal storage format
- Database optimization for fast and efficient querying of semi-structured data

The VARIANT data type

Rather than requiring semi-structured data to be parsed and transformed into a traditional schema of single-value columns, Snowflake stores semi-structured data in a single column of a special type: VARIANT. Each VARIANT column can contain an entire semi-structured object consisting of multiple key-value pairs. For example:

```
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.WEATHER.WEATHER_14_TOTAL
LIMIT 2;
```

V::VARIANT	T::TIMESTAMP
{ "city": { "coord": { "lat": 27.716667, "lon": 85.316666 }, "country": "NP", "findname": "KATHMANDU", "id": 1283240, "name": "Kathmandu", "zoom": 7 }, "clouds": { "all": 75 }, "main": { "humidity": 65, "pressure": 1009, "temp": 300.15, "temp_max": 300.15, "temp_min": 300.15 }, "time": 1504263774, "weather": [{ "description": "broken clouds", "icon": "04d", "id": 803, "main": "Clouds" }], "wind": { "deg": 290, "speed": 2.6 } }	1.Sep.2017 04:02:54
{ "city": { "coord": { "lat": 8.598333, "lon": -71.144997 }, "country": "VE", "findname": "MERIDA", "id": 3632308, "name": "Merida", "zoom": 8 }, "clouds": { "all": 12 }, "main": { "grnd_level": 819.46, "hu-midity": 90, "pressure": 819.46, "sea_level": 1027.57, "temp": 287.006, "temp_max": 287.006, "temp_min": 287.006 }, "time": 1504263774, "weather": [{ "description": "few clouds", "icon": "02d", "id": 801, "main": "Clouds" }], "wind": { "deg": 122.002, "speed": 0.75 } }	1.Sep.2017 04:02:54

The VARIANT type stores the individual keys and their values in a columnar format, just like normal columns in a relational table. This means that storage and query performance for operations on data in a VARIANT column are very similar to storage and query performance for data in a normal relational column.[1]

Note that the maximum number of key-value pairs for a single VARIANT column is 1,000. If your semi-structured data has more than 1,000 key-value pairs, you may benefit from spreading the data across multiple VARIANT columns. Additionally, each VARIANT entry is limited to a maximum size of 16 MB of compressed data.

You can query individual key-value pairs directly from VARIANT columns with a minor extension to traditional SQL syntax:

```
SELECT V:TIME::TIMESTAMP TIME,
    V:CITY:NAME::VARCHAR CITY,
    V:CITY.COUNTRY::VARCHAR COUNTRY,
    (V:MAIN.TEMP_MAX - 273.15)::FLOAT AS TEMP_MAX,
    (V:MAIN.TEMP_MIN - 273.15)::FLOAT AS TEMP_MIN,
    V:WEATHER[0].MAIN::VARCHAR AS WEATHER_MAIN
FROM SNOWFLAKE_SAMPLE_DATA.WEATHER.WEATHER_14_TOTAL;
```

TIME	CITY	COUNTRY	TEMP_MAX	TEMP_MIN	WEATHER_MAIN
8-Jan-2018 1:05 am	Melbourne	AU	20	19	Rain
8-Jan-2018 12:02 am	Melbourne	AU	19	18	Rain
7-Jan-2018 11:05 pm	Melbourne	AU	19	18	Clouds

Detailed information about working with semi-structured data is in the Snowflake online documentation.

[1] For non-array data that uses only native JSON types (strings and numbers, not timestamps). Non-native values such as dates and timestamps are stored as strings when loaded into a VARIANT column, so operations on these values could be slower and consume more space than when stored in a relational column with the corresponding data type. For more information, visit the Snowflake Documentation.

Accessing semi-structured data from Tableau

Tableau does not recognize the VARIANT data type so it doesn't automatically create queries with the SQL extensions outlined above. This means you need to manually create the SQL necessary for accessing the data in these columns.

One way to access semi-structured data is to use custom SQL. Be sure to follow the best practices for using custom SQL described earlier. Specifically, don't use a monolithic statement that joins across multiple tables. Instead, use a discrete statement to reference the key-value pairs from the VARIANT column and then join that custom SQL "table" with the other regular tables. Also remember to select **Assume Referential Integrity** in Tableau so the query generator can cull tables when they are not required.

Example

To use the WEATHER_14_TOTAL table in the sample WEATHER schema, create a custom SQL "table" in Tableau using the following query:

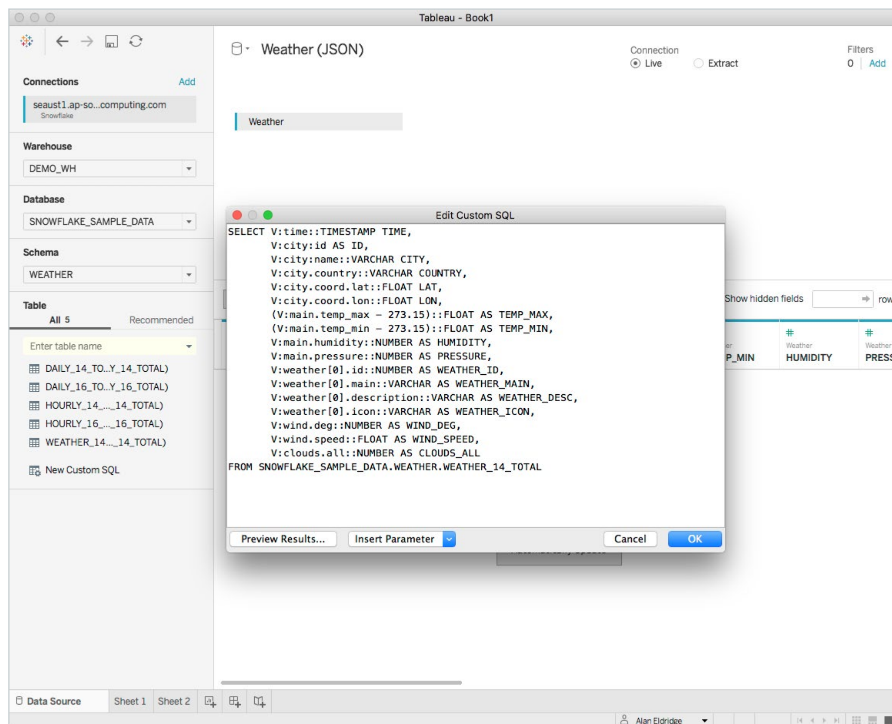


Figure 15

This data source can then be used to create vizzes and dashboards as if the user were connected to a traditional structured schema, with similar performance:

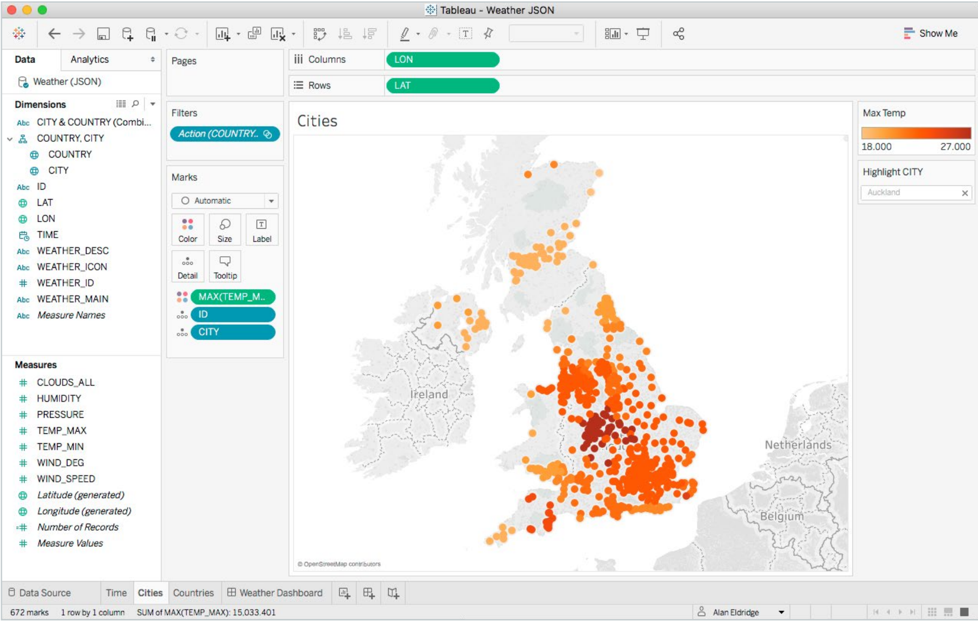


Figure 16

In this example, the WEATHER_14_TOTAL table has about 66 million records, and response times in Tableau using an XS warehouse were acceptable.

Of course, as outlined earlier, this SQL could also be used in a view or an initial SQL statement to create a temp table. The specific approach you use should be dictated by your needs. Alternatively, as a way to ensure governance and a consistent view of the JSON data, you can always publish the data source to Tableau Server (or Tableau Online) for reuse across multiple users and workbooks:

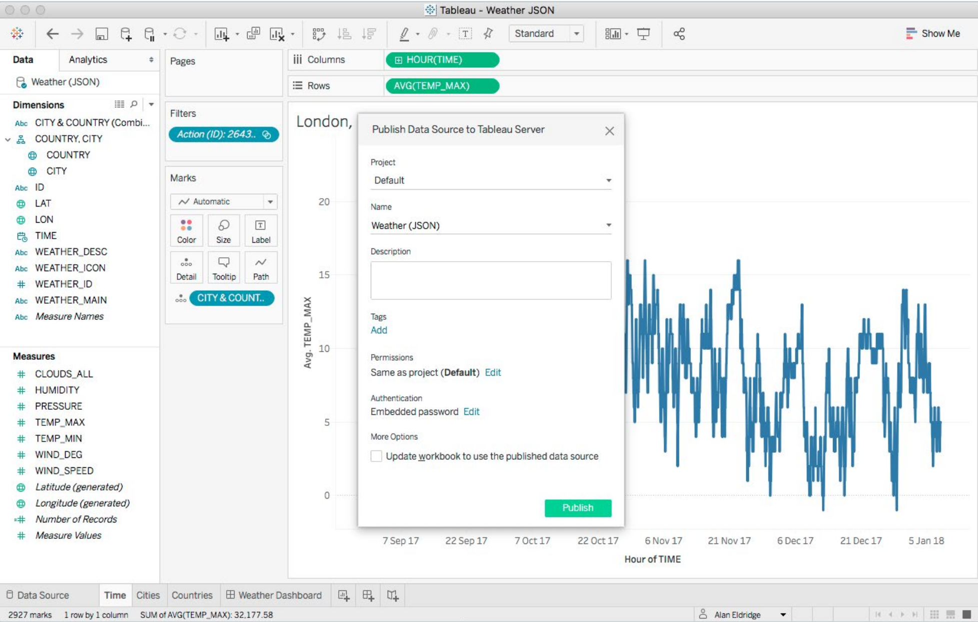


Figure 17

RAWSQL

Tableau provides several functions, called RAWSQL functions, that enable you to pass SQL fragments directly to the underlying data source for processing. Use this to access additional database functions such as HLL or even to access user-defined functions in Snowflake. For more information about RAWSQL functions, see the Tableau documentation.

Example

For example, these RAWSQL functions dynamically extract values from a VARIANT field:

Figure 18

This SQL fragment is passed through to Snowflake without alteration.

Here is a query from a calculation based on RAWSQL statements (the fragments are highlighted):

```
SELECT (V:city.country::string) AS "ID",
    AVG(((V:main.temp_max - 273.15)::float)) AS "avg:temp_max:ok"
FROM "WEATHER"."WEATHER_14_TOTAL" "WEATHER_14_TOTAL" GROUP BY 1
```

The advantage of this approach is that you can extract specific VARIANT values without needing to create an entire custom SQL table statement.

Example

This example uses a Snowflake table, TBL_Holidays, that is defined and populated with dates. First, create a UDF file in Snowflake to look at the number of holidays between two dates:

```
--Create a UDF to Find Holiday Counts
CREATE OR REPLACE FUNCTION UDF_HOLIDAYS(DATESTART Date, DATEEND Date)
    RETURNS INTEGER
    AS
    $$
        SELECT COUNT(*) FROM TBL_HOLIDAYS WHERE DATE_HOLIDAY BETWEEN DATESTART AND DATEEND
    $$;
```

Now in Tableau the Snowflake UDF can be addressed with RAWSQL:

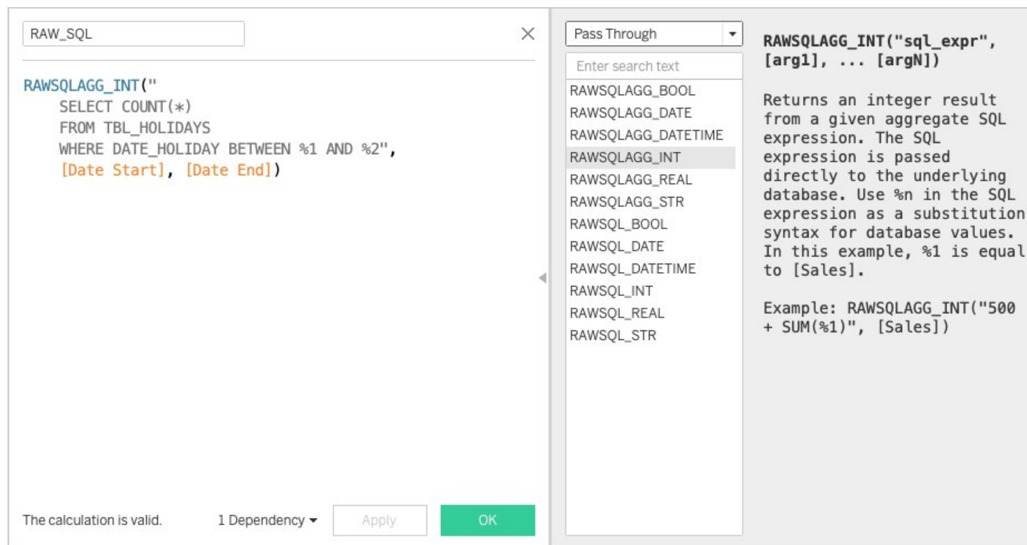


Figure 19

```
RAWSQLAGG_INT("
    SELECT COUNT(*)
    FROM TBL_HOLIDAYS
    WHERE DATE_HOLIDAY BETWEEN %1 AND %2",
    [Date Start], [Date End])
```

This will now return the number of custom-defined holidays between two dates in Tableau from a table populated in Snowflake.

You can extend these types of functions to custom aggregations or you can use an external function to perform additional data-related tasks on the aggregated data before visualization in Tableau.

ELT

Although there are benefits to keeping your semi-structured data in VARIANT data types, if your data schema is well defined and static, you may benefit from performing ELT transformations to convert your data into a traditional data schema.

There are multiple ways to do this:

- Use SQL statements directly in Snowflake (for more details about the supported semi-structured data functions, see the Snowflake documentation.) If you need to conditionally separate the data into multiple tables you can use Snowflake's multi-table insert statement to improve performance, as shown here. Two features that Snowflake offers to assist in the building of extensible ELT pipelines are Streams and Tasks. A stream object records data manipulation language (DML) changes made to tables, including inserts, updates, and deletes, as well as metadata about each change. This allows you to take actions on the changed data, such as separating semi-structured data into multiple tables. Tasks are used to define these actions and schedule their execution.
- Use third-party ETL/ELT tools from Snowflake partners such as Informatica, Matillion, Fivetran, Alteryx, and others. Note that if you are dealing with large volumes of data, use tools that can take advantage of in-DB processing. This means that the data processing will be pushed down into Snowflake, which greatly increases workflow performance by eliminating the need to transfer massive amounts of data out of Snowflake to the ETL tool, manipulate it locally, and then push it back into Snowflake.

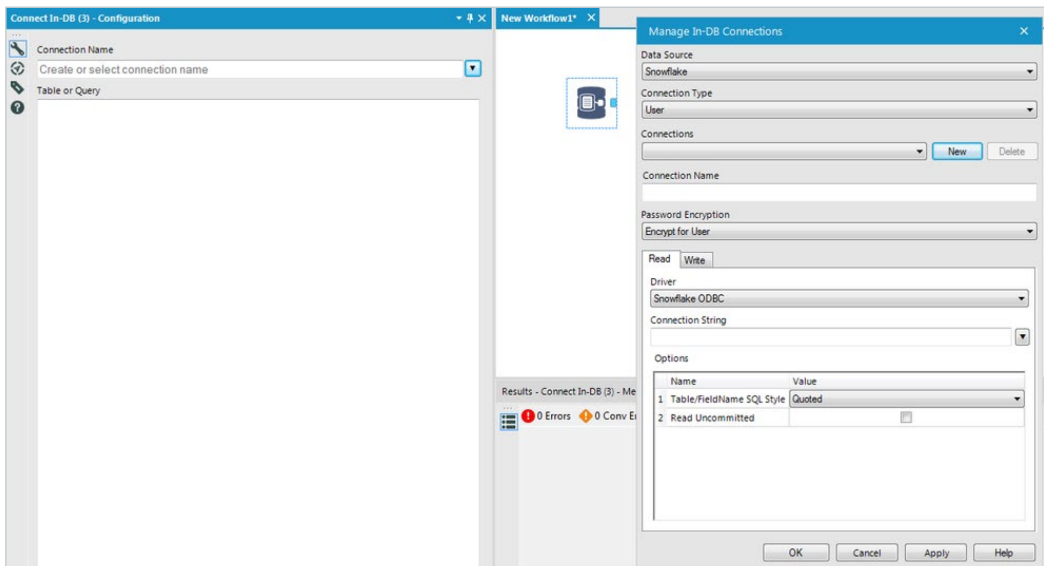


Figure 20

- Two features that Snowflake offers to assist in the building of extensible ELT pipelines are Streams and Tasks. A stream object records data manipulation language (DML) changes made to tables, including inserts, updates, and deletes, as well as metadata about each change. This allows you to take actions on the changed data, such as separating semi-structured data into multiple tables. Tasks are used to define these actions and schedule their execution.

WORKING WITH SNOWFLAKE TIME TRAVEL

Snowflake Time Travel is a powerful feature that lets you access historical data. For Tableau, it enables users to query data from the past that has since been updated or deleted.

When any data manipulation operations are performed on a table (such as INSERT, UPDATE, or DELETE) Snowflake retains previous versions of the table data for a defined period of time. This enables you to query earlier versions of the data using an AT or BEFORE clause.

Accessing historical data

The following query selects historical data from a table as of the date and time represented by the specified timestamp:

```
SELECT *
FROM my_table AT(timestamp => 'Mon, 01 May 2015 16:20:00 -0700'::timestamp);
```

The following query selects historical data from a table as of five minutes ago:

```
SELECT *
FROM my_table AT(offset => -60*5);
```

The following query selects historical data from a table up to, but not including, any changes made by the specified statement:

```
SELECT *
FROM my_table BEFORE(statement => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726');
```

Time travel features are standard for all accounts for up to one day. Extended time travel (up to 90 days) requires Snowflake Enterprise Edition or higher. In addition, time travel requires additional data storage, which has associated fees.

Accessing Snowflake Time Travel data from Tableau

As with semi-structured data, Tableau does not automatically understand how to create time travel queries in Snowflake, so you must use custom SQL. Be sure to use best practices by limiting the scope of your custom SQL to the tables affected.

Example

The following Tableau data source shows a copy of the TPCH_SF1 schema, with time travel enabled. To query the LineItem table and use the data it contained at an arbitrary point in the past, use a custom SQL statement for just the LineItem table. In this case, set an AT clause and use a parameter to pass in the time value:

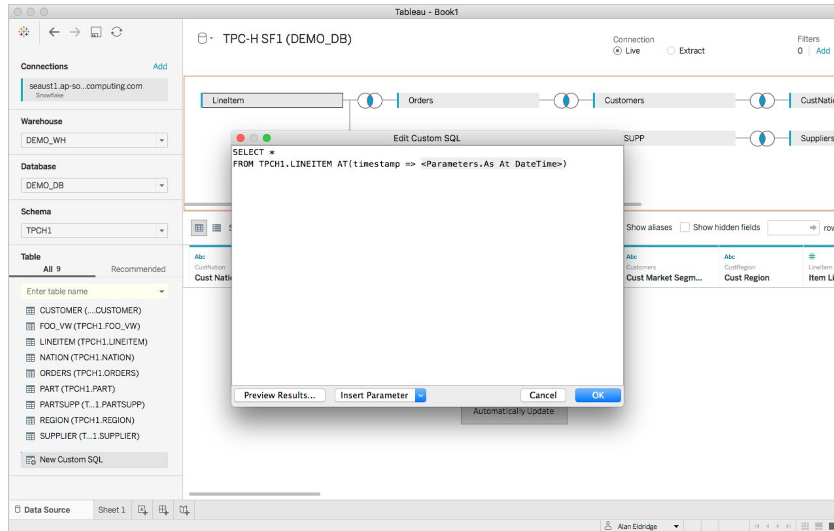


Figure 21

Using this query in the following worksheet:

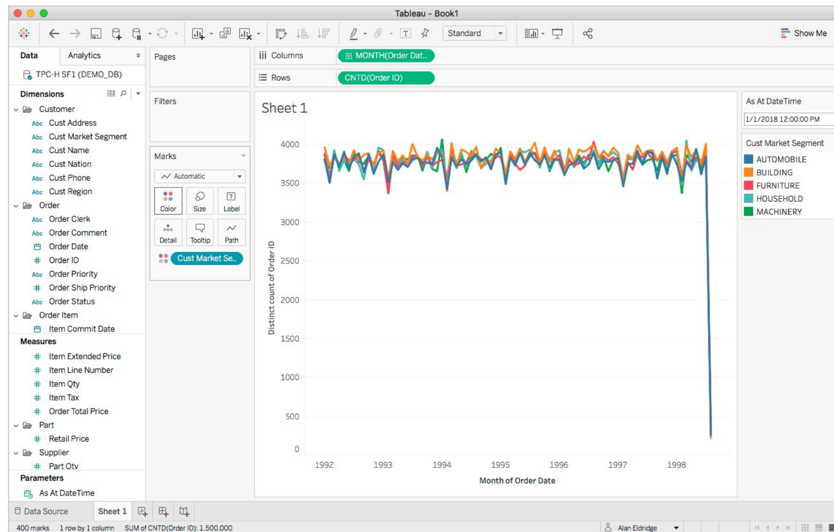


Figure 22

Results in the following query running in Snowflake (the custom SQL is highlighted):

```
SELECT "Customers"."C_MKTSEGMENT" AS "C_MKTSEGMENT",
    COUNT(DISTINCT "Orders"."O_ORDERKEY") AS "ctd:O_ORDERKEY:ok",
    DATE_TRUNC('MONTH',"Orders"."O_ORDERDATE") AS "tmn:O_ORDERDATE:ok"
FROM (
    SELECT *
    FROM TPCH1.LINEITEM AT(timestamp => '2018-01-01 12:00:00.967'::TIMESTAMP_NTZ)
) "LineItem"
    INNER JOIN "TPCH1"."ORDERS" "Orders" ON ("LineItem"."L_ORDERKEY" = "Orders"."O_ORDERKEY")
    INNER JOIN "TPCH1"."CUSTOMER" "Customers" ON ("Orders"."O_CUSTKEY" =
"Customers"."C_CUSTKEY")
GROUP BY 1,
    3
```

The parameter makes it easy for you to select different as-at times. Again, this SQL could also be used in a view or in an initial SQL statement to create a temp table, depending on your needs.

Working With Snowflake Clones

Another important feature that Snowflake offers are clones, sometimes referred to as "zero-copy clones" since they take up no storage when they're created. Clones can be used in conjunction with Time Travel to create point-in-time snapshots of a database, schema, or table. They can also be used to create dev and test environments, as well as personal sandboxes. Clones are mutable, meaning that data within a clone can be modified.

An additional benefit of clones is that users need not be concerned with AT or BEFORE clauses to access data. Clones, by definition, represent data at a particular point in time.

WORKING WITH SNOWFLAKE SECURE DATA SHARING

Snowflake Secure Data Sharing makes it possible to directly share data in near real time and in a secure, governed and scalable way from Snowflake's platform. Organizations can use it to easily share data internally across lines of business, or even externally with customers and partners. Because no data is transmitted, it significantly reduces the traditional pain points of storage duplication and latency. Instead, Snowflake Secure Data Sharing enables data consumers to directly access read-only copies of live data in a data provider's account.

The obvious advantage of this for Tableau users is the ability to query data shared by a provider and to know it is always up-to-date. No ongoing data administration is required by the consumer.

Example

To access shared data, you first view the available inbound shares:

```
SHOW SHARES;
```

You can then create a database from the inbound share and apply appropriate privileges to the necessary roles:

```
//CREATE A DATABASE FROM THE SHARE
CREATE OR REPLACE DATABASE SNOWFLAKE_SAMPLE_DATA
FROM SHARE SNOWFLAKE.SHARED_SAMPLES;

//GRANT PERMISSIONS TO OTHERS
GRANT IMPORTED PRIVILEGES ON DATABASE SNOWFLAKE_SAMPLE_DATA TO ROLE PUBLIC;
```

Users can then access the database objects as if they were local. The tables and views appear to Tableau the same as any other:

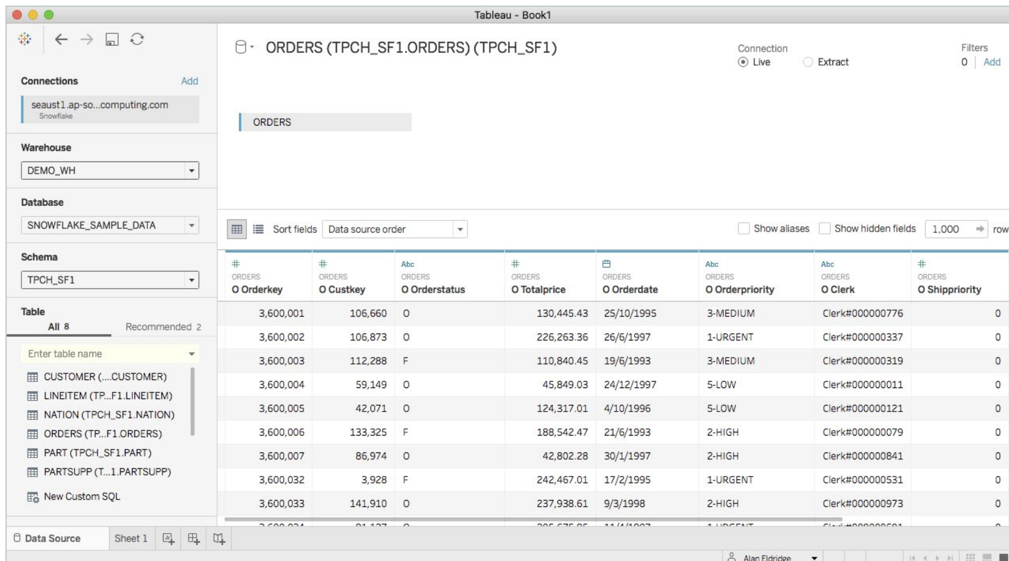


Figure 23

IMPLEMENTING ROLE-BASED SECURITY

A common data access requirement is role-based security where the data returned in a Tableau viz is restricted by row, column, or both. There are multiple ways to achieve this in Tableau and Snowflake. The method you choose depends on how reusable you want the data to be.

This section provides general guidelines for setting up data access rules. The guidelines are followed by information about passing in user content, for both Tableau-only and generic solutions.

Setting up the data access rules

To restrict the data available to a user, you need to define records in a table for each user context and the data elements you want to make accessible.

Example

Using the TPCH_SF1 schema in the sample data, you can create a simple REGION_SECURITY table as follows:

RS_REGIONKEY	RS_USER
1	alan
2	alan
3	alan
4	alan
5	alan
1	clive
2	clive
3	kelly
4	kelly

Link this table via the REGION and NATION tables to the CUSTOMER table:

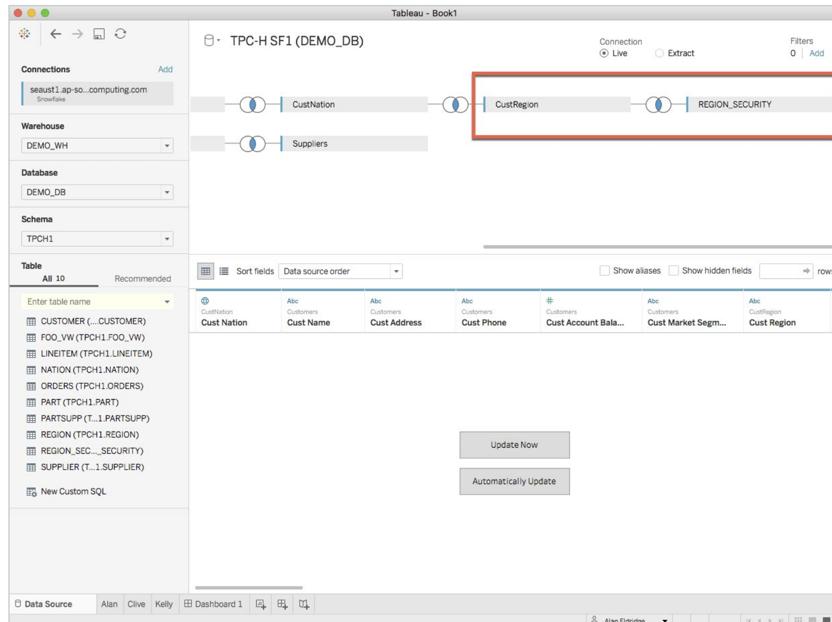


Figure 24

Tableau returns different result sets for each of the three RS_USER values:

Figure 25

To create access rules across multiple dimensions of the data (for example, to restrict access by region and product brand), create multiple access rule tables and join them into the data schema.

Passing in the user context

To make the above model secure, enforce the restriction that a viewing user can see only the result set permitted for their user context, specifically where the RS_USER field equals the viewer's username, and pass this username from Tableau into Snowflake. The procedure depends on whether you are creating a Tableau-only solution or something that is generic and will work for any data tool.

COLUMN-LEVEL SECURITY SOLUTION FOR ANY DATA TOOL

To restrict access for users connecting via any tool, set up restrictions in Snowflake. The process depends on whether each user has a unique Snowflake login or whether all queries are sent to Snowflake via a common login.

In addition to row-level security via secure views, Snowflake also offers Dynamic Data Masking. Dynamic Data Masking is a Column-level Security feature that uses masking policies to selectively mask plain-text data in table and view columns at query time.

At query runtime, the masking policy is applied to the column at every location where the column appears. Depending on the masking policy conditions, the SQL execution context, and role hierarchy, Snowflake query operators may see the plain-text value, a partially masked value, or a fully masked value.

For more details about how masking policies work, including the query runtime behavior, creating a policy, usage with tables and views, and management approaches using masking policies, see: Understanding Column-level Security.

SECURE USER DEFINED FUNCTIONS (SECURE UDFS)

Some of the internal optimizations for SQL UDFs require access to the underlying data in the base tables. This access might allow data that is hidden from users of the UDF to be exposed indirectly through programmatic methods.

In addition, the SQL expression or JavaScript code used to create a UDF, also known as the UDF definition or text, is visible to users in the following commands and interfaces:

- SHOW FUNCTIONS and SHOW USER FUNCTIONS commands
- GET_DDL utility function
- FUNCTIONS Information Schema view
- Query Profile (in the web interface)

For security or privacy reasons, you might not wish to expose the underlying tables or algorithmic details for a UDF. With secure UDFs, the definition and details are visible only to authorized users (i.e. users who are granted the role that owns the UDF).

More information about Secure UDFs can be found here.

TABLEAU-ONLY SOLUTION

If you are building a Tableau-only solution, you can define the security logic in Tableau. This can be useful when the viz author does not have permission to modify the database schema or add new objects (for example, views) which would happen if you were consuming a data share from another account. It is also useful if the users in Tableau (specifically, users accessing the viz via Tableau Server, as opposed to the author creating it in Tableau Desktop) do not use individual logins for Snowflake.

Example

To enforce the restriction, create a data source filter in Tableau that restricts the result set to where the RS_USER

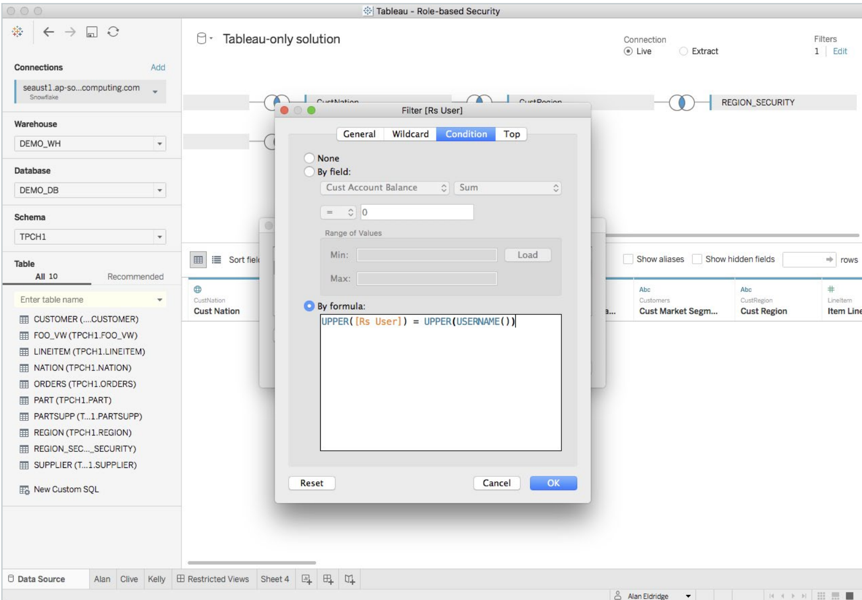


Figure 26

field matches the Tableau function USERNAME() which returns the login of the current user:

The result shows only the data permitted for the viewing user (note that in the following screenshot there are no

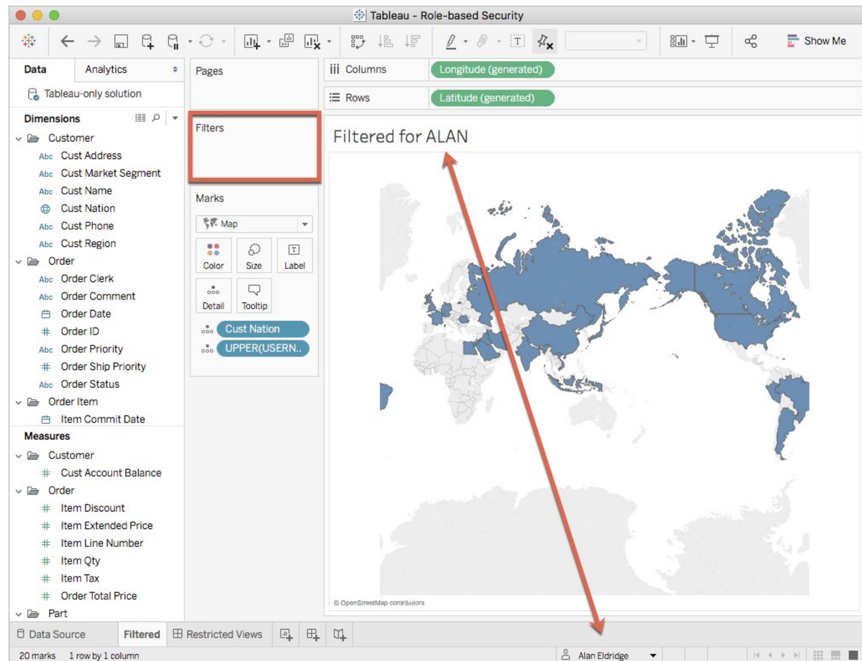


Figure 27

worksheet-level filters as the filter is enforced on the data source):

The USERNAME() function is evaluated in Tableau and then pushed through to Snowflake as a literal, as you can see in the resulting query:

```
SELECT ‘ALAN’ AS “Calculation_3881117741409583110”,
    “CustNation”.”N_NAME” AS “N_NAME”
FROM “TPCH1”.”NATION” “CustNation”
    INNER JOIN “TPCH1”.”REGION” “CustRegion” ON (“CustNation”.”N_REGIONKEY” =
“CustRegion”.”R_REGIONKEY”)
    INNER JOIN “TPCH1”.”REGION_SECURITY” “REGION_SECURITY” ON (“CustRegion”.”R_REGIONKEY” =
“REGION_SECURITY”.”RS_REGIONKEY”)
WHERE (UPPER(“REGION_SECURITY”.”RS_USER”) = ‘ALAN’)
GROUP BY 2
```

To enforce this filter and prevent a workbook author from editing or removing it, you should publish the data source

to Tableau Server and create the workbook using the published data source.

SOLUTION FOR ANY DATA TOOL

To restrict access for users connecting via any tool, set up restrictions in Snowflake. The process depends on whether each user has a unique Snowflake login or whether all queries are sent to Snowflake via a common login.

Example

Continuing the example above, rather than using a data source filter in Tableau, you can create a view in Snowflake:

```
create or replace secure view "TPCH1"."SECURE_REGION_VW" as
    select R_REGIONKEY, R_NAME, R_COMMENT, RS_USER
    from "TPCH1"."REGION" "CustRegion"
    inner join "TPCH1"."REGION_SECURITY" "REGION_SECURITY"
    on ("CustRegion"."R_REGIONKEY" = "REGION_SECURITY"."RS_REGIONKEY")
WHERE (UPPER("REGION_SECURITY"."RS_USER") = UPPER(CURRENT_USER));
```

This example uses the Snowflake variable CURRENT_USER, but you could use CURRENT_ROLE if your solution needed to be more scalable. This requires each viewing user to be logged in to Snowflake with their own credentials. You can enforce this when publishing the workbook to Tableau Server by setting the authentication type to **Prompt user:**

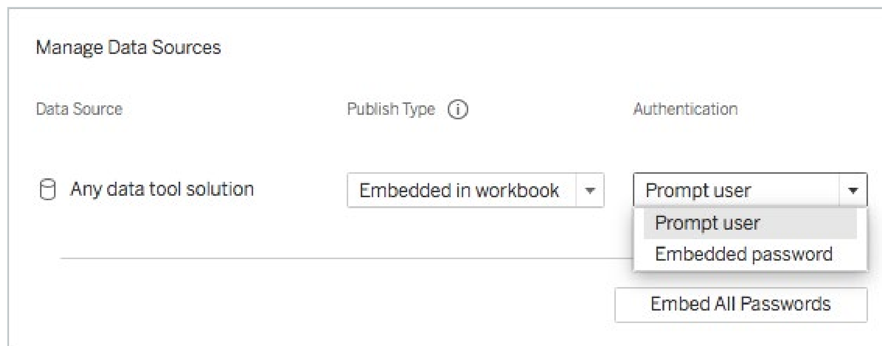


Figure 28

If you use **Embedded password**, CURRENT_USER and CURRENT_ROLE will be the same for all user sessions, and you need to pass in the viewing user name via an initial SQL block:

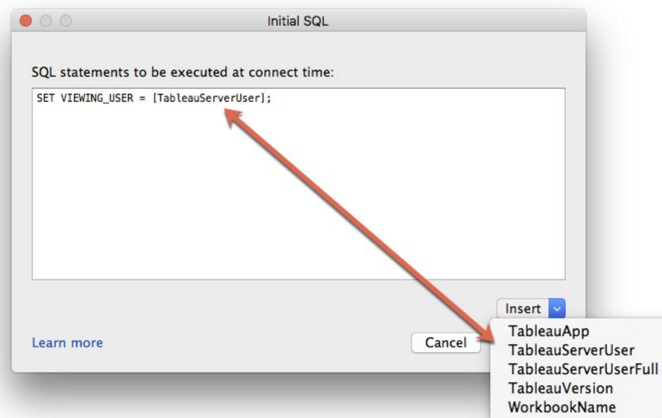


Figure 29

The view would then reference this variable:

```
create or replace secure view "TPCH1"."SECURE_REGION_VW" as
    select R_REGIONKEY, R_NAME, R_COMMENT, RS_USER
    from "TPCH1"."REGION" "CustRegion"
    inner join "TPCH1"."REGION_SECURITY" "REGION_SECURITY"
    on ("CustRegion"."R_REGIONKEY" = "REGION_SECURITY"."RS_REGIONKEY")
WHERE (UPPER("REGION_SECURITY"."RS_USER") = UPPER($VIEWING_USER));
```

The final step to enforce the security rules specified in the SECURE_REGION_VW view is to enforce referential integrity in the schema. Without this, if you don't use the SECURE_REGION_VW in your query, then join culling could drop this table from the query and security would be bypassed.

If your data permits, you can create constraints between the tables in Snowflake (see Snowflake's documentation for details) or you can simply uncheck **Assume Referential Integrity** in Tableau:

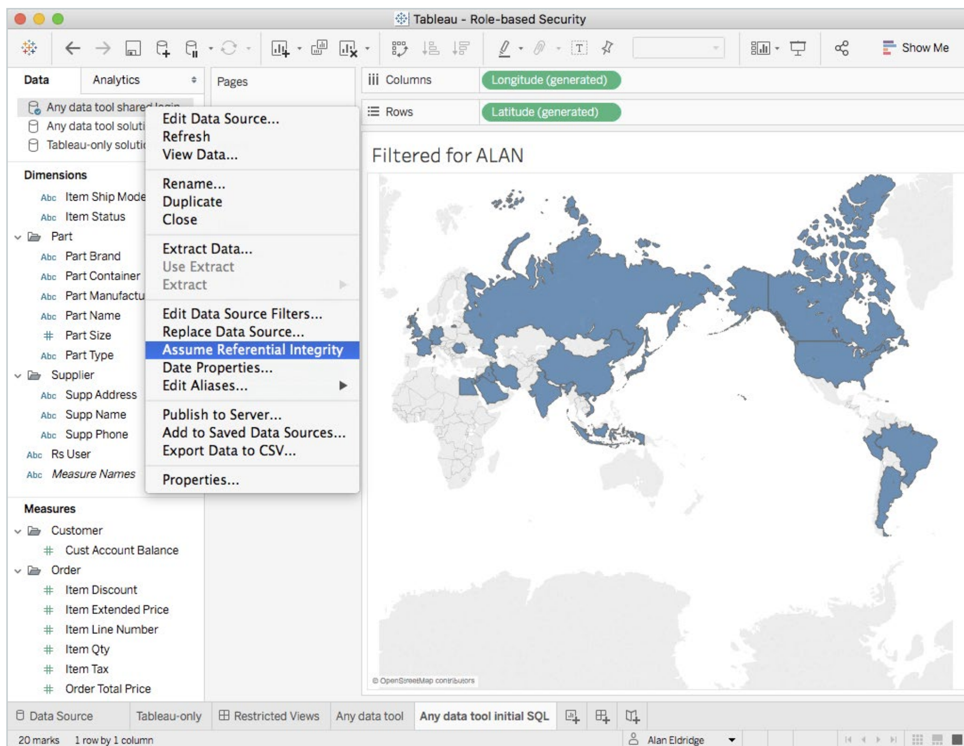


Figure 30

USING CUSTOM AGGREGATIONS

Snowflake provides many custom aggregation functions outside the ANSI SQL specification. Some of these are specific to working with semi-structured data, while others are useful for approximating results (such as cardinality, similarity, and frequency) when you are working over very large data volumes. A full list of the available aggregation functions is in the Snowflake online documentation.

To use these functions in Tableau, leverage the pass-through functions in Tableau.

Example

Snowflake provides a custom aggregation function APPROX_COUNT_DISTINCT which uses HyperLogLog to return an approximation of the distinct cardinality of a field. To use this function, create a calculated field that leverages the appropriate RAWSQL function. This example shows using the calculation in a Tableau viz:

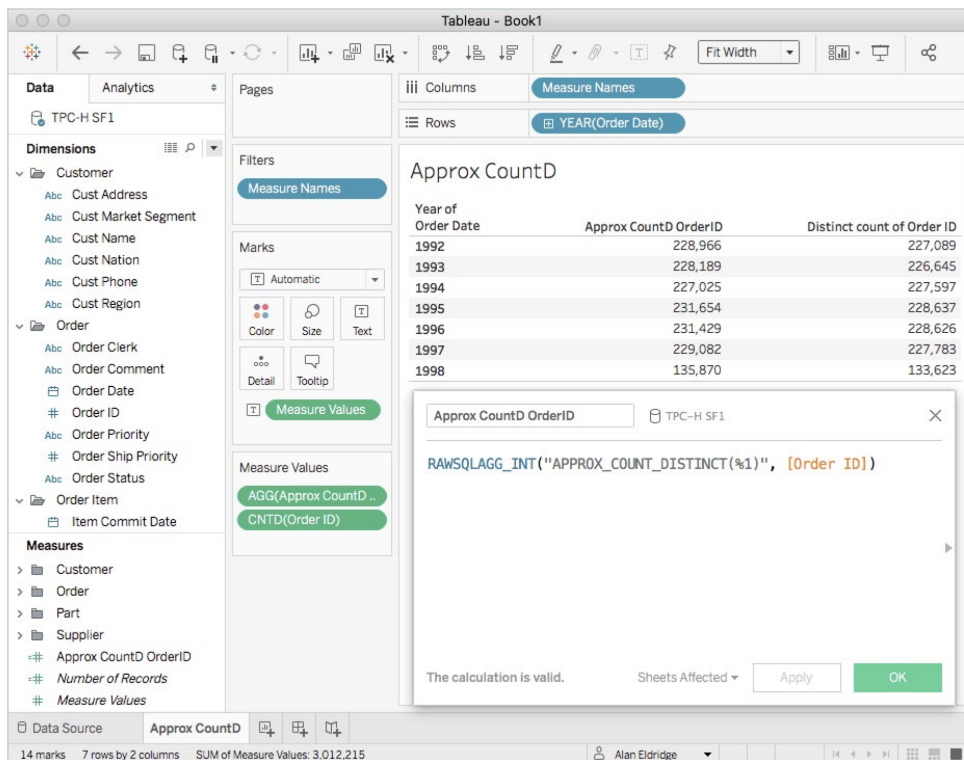


Figure 31

Tableau produces the following query in Snowflake (the pass-through function is highlighted):

```
SELECT COUNT(DISTINCT “Orders”.”O_ORDERKEY”) AS “ctd:O_ORDERKEY:ok”,
    (APPROX_COUNT_DISTINCT(“Orders”.”O_ORDERKEY”)) AS
“usr:Calculation_1944218056180731904:ok”,
    DATE_PART(‘YEAR’,”Orders”.”O_ORDERDATE”) AS “yr:O_ORDERDATE:ok”
FROM “TPCH_SF1”.”LINEITEM” “LineItem”
    INNER JOIN “TPCH_SF1”.”ORDERS” “Orders” ON (“LineItem”.”L_ORDERKEY” =
“Orders”.”O_ORDERKEY”) GROUP BY 3
```

SCALING SNOWFLAKE WAREHOUSES

Snowflake supports two ways to scale warehouses:

- Scale up by resizing a warehouse
- Scale out by adding clusters to a warehouse (requires Snowflake Enterprise Edition or higher)

Resizing a warehouse to improve performance

Resizing a warehouse generally improves query performance, particularly for larger, more complex queries. It can also help reduce the queuing that occurs if a warehouse does not have enough servers to process multiple concurrent queries. (But warehouse resizing is not intended to handle concurrency issues; instead, use additional warehouses or a multi-cluster warehouse if this feature is available for your account).

The number of servers required to process a query depends on the size and complexity of the query. For the most part, query processing scales linearly with warehouse size, particularly for larger, more complex queries:

- The overall data size of the tables being queried has more impact than the number of rows.
- Filtering in a query using predicates and the number of joins and tables in the query also affects processing.

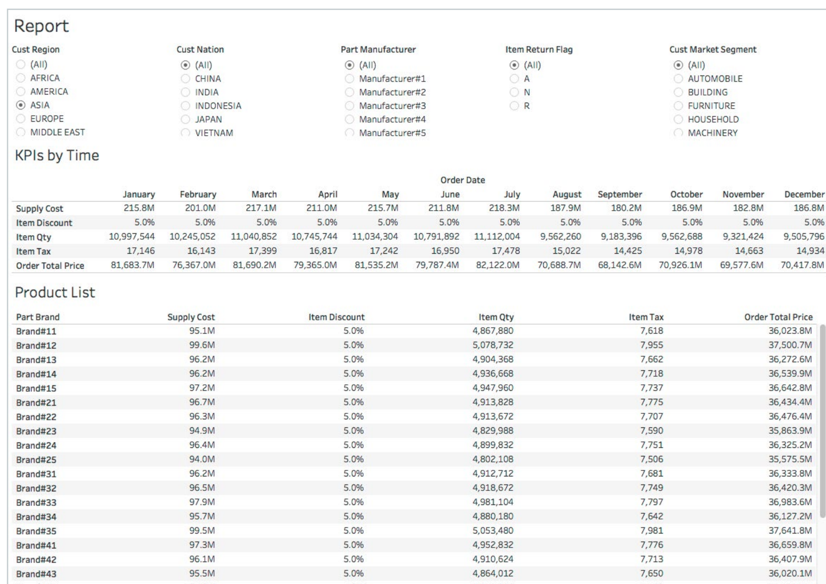
Snowflake supports resizing a warehouse at any time, even while it is running. However, note the following:

- Larger warehouses do not necessarily make a query run faster. For smaller, basic queries that are already executing quickly, you may not see any significant improvement after resizing.
- Resizing a running warehouse does not impact queries that are already being processed by the warehouse; the additional servers are only used for queued and new queries.

Decreasing the size of a running warehouse removes servers from the warehouse. When the servers are removed, the cache associated with the servers is dropped, which can impact performance. Keep this in mind when choosing whether to decrease the size of a running warehouse or keep it at the current size.

Example

To demonstrate the scale-up capability of Snowflake, the following dashboard was created against the TPCH_SF10 sample data schema and published to Tableau Server. Note that this dashboard was intentionally designed to ensure multiple queries would be initiated in the underlying DBMS:



Report

Cust Region	Cust Nation	Part Manufacturer	Item Return Flag	Cust Market Segment
(All)	(All)	(All)	(All)	(All)
AFRICA	CHINA	Manufacturer#1	A	AUTOMOBILE
AMERICA	INDIA	Manufacturer#2	N	BUILDING
ASIA	INDONESIA	Manufacturer#3	R	FURNITURE
EUROPE	JAPAN	Manufacturer#4		HOUSEHOLD
MIDDLE EAST	VIETNAM	Manufacturer#5		MACHINERY

KPIs by Time

	Order Date											
	January	February	March	April	May	June	July	August	September	October	November	December
Supply Cost	215.8M	201.0M	217.1M	211.0M	215.7M	211.8M	218.3M	187.9M	180.2M	186.9M	182.8M	186.8M
Item Discount	5.0%	5.0%	5.0%	5.0%	5.0%	5.0%	5.0%	5.0%	5.0%	5.0%	5.0%	5.0%
Item Qty	10,997,544	10,245,052	11,040,852	10,745,744	11,034,304	10,791,892	11,112,004	9,562,260	9,183,396	9,562,688	9,321,424	9,505,796
Item Tax	17,146	16,143	17,399	16,817	17,242	16,950	17,478	15,022	14,425	14,978	14,663	14,934
Order Total Price	81,683.7M	76,367.0M	81,690.2M	79,365.0M	81,535.2M	79,787.4M	82,122.0M	70,688.7M	68,142.6M	70,926.1M	69,577.6M	70,417.8M

Product List

Part Brand	Supply Cost	Item Discount	Item Qty	Item Tax	Order Total Price
Brand#11	95.1M	5.0%	4,867,880	7,618	36,023.8M
Brand#12	99.6M	5.0%	5,078,732	7,955	37,500.7M
Brand#13	96.2M	5.0%	4,904,368	7,662	36,272.6M
Brand#14	96.2M	5.0%	4,936,668	7,718	36,539.9M
Brand#15	97.2M	5.0%	4,947,960	7,737	36,642.8M
Brand#21	96.7M	5.0%	4,913,828	7,775	36,434.4M
Brand#22	96.3M	5.0%	4,913,672	7,707	36,476.4M
Brand#23	94.9M	5.0%	4,829,988	7,590	35,863.9M
Brand#24	96.4M	5.0%	4,899,832	7,751	36,325.2M
Brand#25	94.0M	5.0%	4,802,108	7,506	35,575.5M
Brand#31	96.2M	5.0%	4,912,712	7,681	36,333.8M
Brand#32	96.5M	5.0%	4,918,672	7,749	36,420.3M
Brand#33	97.9M	5.0%	4,981,104	7,797	36,983.6M
Brand#34	95.7M	5.0%	4,880,180	7,642	36,127.2M
Brand#35	99.5M	5.0%	5,053,480	7,981	37,641.8M
Brand#41	97.3M	5.0%	4,952,832	7,776	36,659.8M
Brand#42	96.1M	5.0%	4,910,624	7,713	36,407.9M
Brand#43	95.5M	5.0%	4,864,012	7,650	36,020.1M

Figure 32

The data source for the workbook was also configured with an initial SQL statement to disable query result caching within Snowflake. (Result caching, covered in the next section of this white paper, would make subsequent queries extremely fast because the query results do not need to be recomputed and are read from cache in less than a second.)

TabJolt was then installed on a Windows PC to act as a load generator, invoking the dashboard with a :refresh=y parameter to prevent caching within Tableau Server. A series of tests ran the dashboard against different Snowflake warehouse sizes from XS to XL. All tests were run for five minutes with a single simulated user using the InteractVizLoadTest script. The results are as follows:

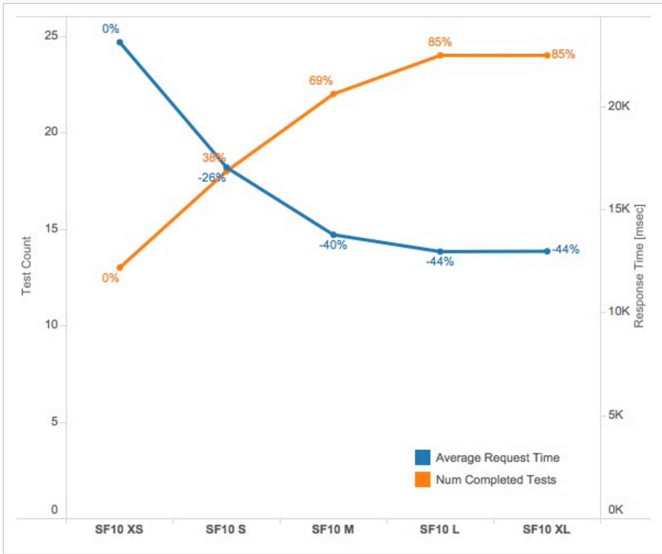


Figure 33

As you can see, an increase in the size of the warehouse (from XS up to XL) results in decreased query times because there are more compute resources available. The improvement levels off from L to XL sized warehouses as compute resources are no longer a bottleneck.

You can also see in the first run of the query at each warehouse size (the orange dot in the following chart), data must be read from the underlying S3 table storage into the warehouse cache. Subsequent runs for each warehouse size can then read data entirely from the warehouse cache, resulting in improved performance.

Figure 34

To achieve the best results, execute relatively homogeneous queries (for example, similar size, complexity, and data sets) on the same warehouse. Executing queries of widely varying size or complexity on the same warehouse makes it more difficult to analyze warehouse load, which can make it more difficult to select the best size to match the size, composition, and number of queries in your workload.

Adding warehouses to improve concurrency

By default, a warehouse consists of a single cluster of servers that determine the total resources available for executing queries. As queries are submitted to a warehouse, the warehouse allocates server resources to each query and begins executing the queries. If resources are insufficient to execute all the queries submitted, Snowflake queues the additional queries until the necessary resources become available.

But with multi-cluster warehouses, Snowflake supports allocating, either statically or dynamically, a larger pool of resources to each warehouse. Multi-cluster warehouses enable you to scale warehouse resources to manage your user and query concurrency needs as they change, such as during peak and off hours.

When deciding whether to use multi-cluster warehouses and choosing the number of clusters to use per warehouse, consider the following:

- Unless you have a specific requirement for running in maximized mode, multi-cluster warehouses should be configured to run in auto-scale mode, which enables Snowflake to automatically start and stop clusters as needed. A new cluster is started when queries are queuing for more than 30 seconds, and a cluster is stopped if it has not run a query for 30 minutes.
- Set the minimum number of clusters to the default value of 1 to ensure that additional clusters are started only as needed. However, if high availability is a concern, set the value higher than 1. This helps ensure warehouse availability and continuity in the unlikely event that a cluster fails.
- Set the maximum value to be as large as possible, while being mindful of the warehouse size and corresponding credit costs. For example, an XL warehouse (16 servers) with the maximum of 10 clusters will consume 160 credits in an hour if all 10 clusters run continuously for the hour.

Multi-cluster warehouses are a feature of Snowflake Enterprise Edition or higher.

Example

To demonstrate the scale-out capability of Snowflake, the same dashboard as above was used against the TPCH_SF1 sample data schema. It was published to Tableau Server, and then, using TabJolt, a series of test runs were performed, running the above dashboard against different Snowflake warehouse configurations:

WAREHOUSE SIZE	# WAREHOUSE CLUSTERS	TOTAL # SERVERS
XS	1, 2, 3, 4	1, 2, 3, 4
S	1, 2, 3, 4	2, 4, 6, 8
M	1	4

All tests were run for five minutes with 20 concurrent threads (simulated users) using the InteractVizLoadTest script, and the total number of completed tests and average test response times were recorded. The results are as follows:

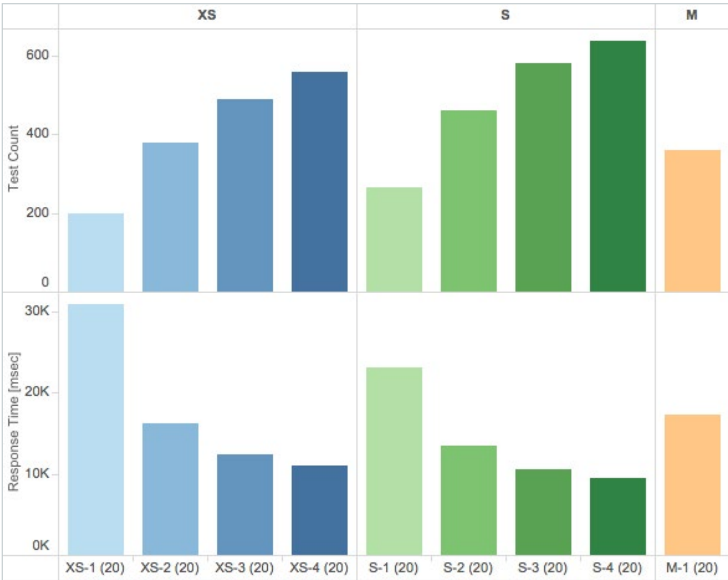


Figure 35

As the size of the warehouse increased (scaling up in a single cluster from XS to S to M) the total number of tests completed increased by 81% and the average test response time decreased by 44%. This is unsurprising as the number of servers increased from one to two, then from two to four. Notice the following chart on the left:

Figure 36

Figure 37

However, the above chart on the right shows that with a constant number of servers, (four XS clusters, two S clusters, or a single M cluster, each with four servers) the multi-cluster warehouse approach has significantly better scalability characteristics with 55% more completed tests and an average response time improvement of 36%.

Scale Across

While having the ability to scale warehouses up and out for Tableau usage provides numerous benefits, it's important to remember that Snowflake allows you to create separate warehouses for each of your workloads. For example, you may determine that a Small warehouse with a minimum of 1 cluster and maximum of 4 clusters is right for your Tableau environment. In addition to the Tableau workload, it's very likely that data is being loaded into Snowflake and transformed, data scientists are performing their tasks (data prep, feature engineering, etc), QA is performing its tests, and so on. And all of these workloads could be running simultaneously.

The diversity of these workloads can easily be handled by Snowflake. Separate warehouses can be created for each workload, and sized for the unique complexity and concurrency requirements of those workloads.

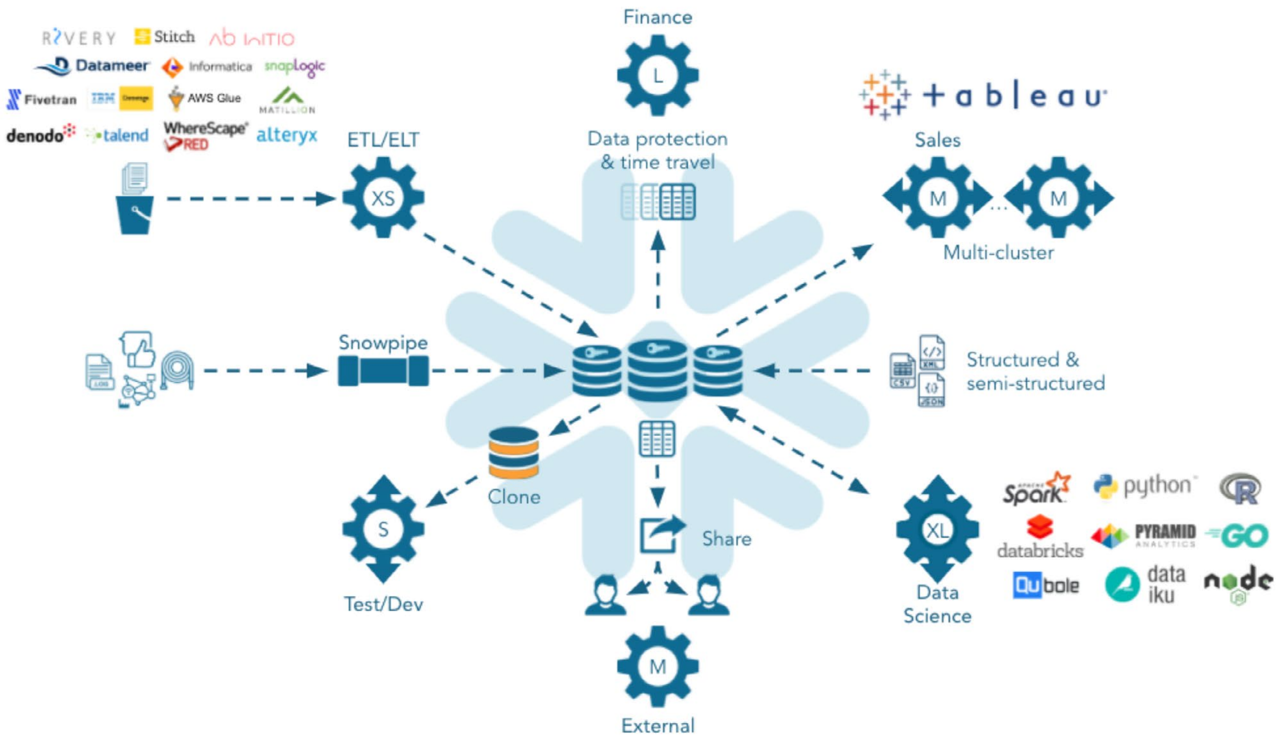


Figure 38

CACHING

The combination of Tableau and Snowflake enables caching at multiple levels. The more effective you can make your caching, the more efficient your environment will be. Queries and workbooks will return results faster and less load will be applied to the server layers, leading to greater scalability.

Tableau caching

Tableau has caching in both Tableau Desktop and Tableau Server which can significantly reduce the amount of rendering, calculation, and querying needed to display views to users. Caching can occur in three layers:

- The presentation layer (Tableau Server only)
- The analytics layer
- The data layer

For Tableau Server in particular, caching is critical for achieving concurrent user scalability.

PRESENTATION LAYER

Caching at the presentation layer is relevant for Tableau Server only. In Tableau Server, multiple end users view and interact with views via a browser or mobile device. Depending on the capability of the browser and the complexity of the view, rendering will be done either on the client or the server.

Client-side rendering in the browser

During client-side rendering, the client browser downloads a JavaScript, *viz client*, that can render a Tableau viz. It then requests the initial data package, called the *bootstrap response*, for the view. The bootstrap response package includes the *view model*, which includes the view layout and the data for the marks to be displayed. With this data, the viz client can then draw the view locally in the client's browser.

As the user explores the view, simple interactions (such as tooltips, highlighting, and selecting marks) are handled locally by the viz client using the local data cache. This means no communication with the server is required, and the result is a fast screen update. This also reduces the compute load on the server which helps with Tableau Server scalability. Visit Tableau Documentation to learn more.

Figure 39

Server-side rendering with tiles

Not all vizzes can be rendered client-side. More complex interactions (for example, changing a parameter or filter) cannot be handled locally, so the update is sent to the server, and the new view model is sent to update the local data cache.

In server-side rendering, the view is rendered as a set of static image tiles. These tiles are sent to the browser, and a much simpler viz client assembles them into the final view. The viz client monitors the session for a user interaction (such as hovering for a tooltip, selecting marks, highlighting, or interacting with filters), and if anything needs to be drawn, it sends a request to the server for new tiles. No data is stored on the client, so all interactions require a server round-trip.

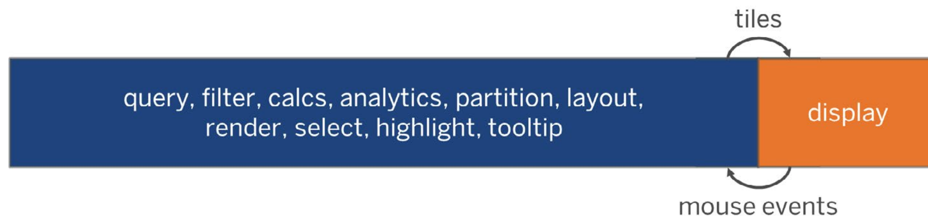


Figure 40

To help with performance, the tile images are persisted on Tableau Server in the tile cache. If the same tile is requested, it can be served from the cache instead of being re-rendered.

Note that the tile cache can be used only if the subsequent request is for the exact same view. This means requests for the same view from different sized browser windows can result in different tiles being rendered. But if you set your dashboards to fixed size instead of automatic, the view requests will always be the same size, irrespective of the browser window dimensions, allowing a better hit rate for the tile cache.

More about view models and bootstrap responses

Tableau needs the view model to render a viz. The VizQL Server generates the model based on several factors including the requesting user credentials and the size of the viewing window. It computes the view of the viz (including things like the relevant header and axis layouts, label positions, legends, filters, and marks) and then sends it to the renderer (either the browser for client-side rendering or the VizQL Server for server-side rendering).

The VizQL Server on Tableau Server maintains a copy of view models for each user session. This is initially the default view of the viz, but it is updated to reflect user interactions with the view, including any highlights, filters, or selections. A dashboard can reference one view model for each worksheet. It includes the results from local calculations (such as table calculations, reference lines, forecasts, and clustering) and the visual layout (how many rows and columns to display for small multiples and crosstabs, the number and interval of axis ticks and grid lines to draw, the number and location of mark labels to be shown, and other elements).

Avoid generating view models, which can be computationally intensive. The initial bootstrap response is persisted on Tableau Server by the Cache Server. When a request comes in for a view, if the bootstrap response already exists, it can be served without recomputing and reserializing it.

Not all views can use the bootstrap response cache, though. If a view uses any relative date filters, user-specific filters, or published data server connections, it will prevent the bootstrap response from being cached.

Like the tile cache, the bootstrap response is specific to the view size, so requests for the same view from browser windows with different sizes will produce different bootstrap responses. To prevent this, set your dashboards to be fixed size instead of automatic.

Maintaining the view model means the VizQL doesn't need to recompute the view state with every user interaction. The models are created and cached in-memory by the VizQL Server, which shares results across user sessions where possible. To ensure a visual model can be shared, do the following:

- **Make sure the size of the viz display area is the same:** Setting your dashboards to have a fixed size will allow greater reuse and lower the workload on the server.
- **Make sure the selections and filters match:** To increase the likelihood that different sessions will match, avoid publishing workbooks with **Show Selections** checked.
- **Ensure the same credentials are used to connect to the data source:** The model can be shared only across sessions where users have the same credentials. Avoid prompting for credentials to connect to the data source.
- **Avoid user filtering:** If the workbook contains user filters or has calculations containing functions such as USERNAME() or ISMEMBEROF(), the model is not shared with any other user sessions. Use these functions with caution as they can significantly reduce the effectiveness of the model cache.

The session models are not persistent. By default, they expire after 30 minutes of inactivity to recycle memory on the server. If your sessions expire before you are finished with a view, consider increasing the VizQL session timeout setting.

ANALYTICS LAYER

The analytics layer includes the data manipulations and calculations performed on the underlying data. To avoid unnecessary computations and reduce queries, this layer uses an abstract query cache.

A user can generate many physical queries when interacting with a view. Rather than executing each query, Tableau groups the queries into a batch and decompiles them to find optimizations. This could involve removing duplicate queries, combining multiple similar queries into a single statement, or even eliminating queries where the results of one can be derived from the results of another. Tableau checks this cache, indexed by the logical structure of the query, before executing the native query.

Tableau does **not** cache the results of queries that:

- Return large result sets (which are too big for the cache)
- Execute quickly (when it is faster to run the query than check the cache)
- Have user filters
- Use relative date filters

Caching in this layer applies to both Tableau Desktop and Tableau Server; however, there are differences in the caching mechanisms between the two tools.

DATA LAYER

The data layer addresses the native connection between Tableau and the data sources. Caching at this level persists the results of queries for reuse. It also determines the nature of the connection to the data source, whether you are using live connections or the Tableau data engine (replaced with Hyper in Tableau 10.5 and later). It uses a native query cache.

The native query cache is similar to the abstract query cache, but instead of being indexed by the logical query structure, it is keyed by the actual query statement. Multiple abstract queries can resolve to the same native query.

Tableau does not cache the results of queries that:

- Return large result sets (which are too big for the cache)
- Execute quickly (when it is faster to run the query than check the cache)
- Have user filters
- Use relative date filters

Visit Tableau Documentation on configuring the data cache and configuring workbook performance after a scheduled refresh to learn more.

Snowflake caching

Snowflake also has caching features, including persisting query results and caching table data at the warehouse level.

RESULT CACHING

Snowflake persists query results for 24 hours before purging them. A persisted result is available for reuse by another query, as long as the user executing the query has the necessary access privileges and all of the following conditions have been met:

- The new query syntactically matches the previously executed query.
- The table data contributing to the query result has not changed.
- The persisted result for the previous query is still available.
- Configuration options that affect how the result was produced have not changed.
- The query does not include functions that must be evaluated at execution time (such as CURRENT_TIMESTAMP)

Reusing cached results can substantially reduce query time because Snowflake bypasses query execution and, instead, retrieves the result directly from the cache. Each time the persisted result for a query is reused, Snowflake resets the 24-hour retention period, up to a maximum of 31 days from when the query was first executed. After 31 days, the result is purged, and the next time the query is submitted, a new result is returned and persisted.

Result reuse is controlled by the session parameter USE_CACHED_RESULT. By default, the parameter is enabled but can be overridden at the account, user, and session level if desired.

Note that the result cache in Snowflake will contain very similar data as the native query cache in Tableau; however, they operate independently.

WAREHOUSE CACHING

Each running warehouse maintains a cache of table data accessed during query processing. This enables improved performance for subsequent queries if they are able to read from the cache instead of from the tables in the query. The size of the cache is determined by the number of servers in the warehouse. The more servers in the warehouse, the larger the cache.

This cache is dropped when the warehouse is suspended, which may result in slower initial performance for some queries after the warehouse is resumed. As the resumed warehouse runs and processes more queries, the cache is rebuilt, and queries that are able to take advantage of the cache will experience improved performance.

Keep this in mind when deciding whether to suspend a warehouse or leave it running. In other words, consider the trade-off between saving credits by suspending a warehouse versus maintaining the cache of data from previous queries to help with performance.

OTHER PERFORMANCE CONSIDERATIONS

Besides caching, you can use constraints and temp tables to improve performance, as described in this section.

Constraints

Constraints define integrity and consistency rules for data stored in tables. Snowflake provides support for constraints as defined in the ANSI SQL standard, as well as some extensions for compatibility with other databases, such as Oracle. Constraints are provided primarily for data modeling and support compatibility with other databases and client tools.

To increase performance, define constraints between tables you intend to use in Tableau. Tableau uses constraints to perform join culling (join elimination), which can improve the performance of generated queries. If you cannot define constraints, be sure to set Assume Referential Integrity in the Tableau data source to allow the query generator to cull unneeded joins.

Temp tables

Tableau users create temp tables in multiple situations (for example, with complex filters, actions, or sets). When Tableau users are unable to create temporary tables, Tableau tries to use alternate query structures, but these can be less efficient and, in extreme cases, can cause errors.

Example

The following visualization is a scatter plot showing Order Total Price vs. Supply Cost for each Customer Name in the TPCH_SF1 sample data set. In this example, several of the points have been lassoed to create a set which is then

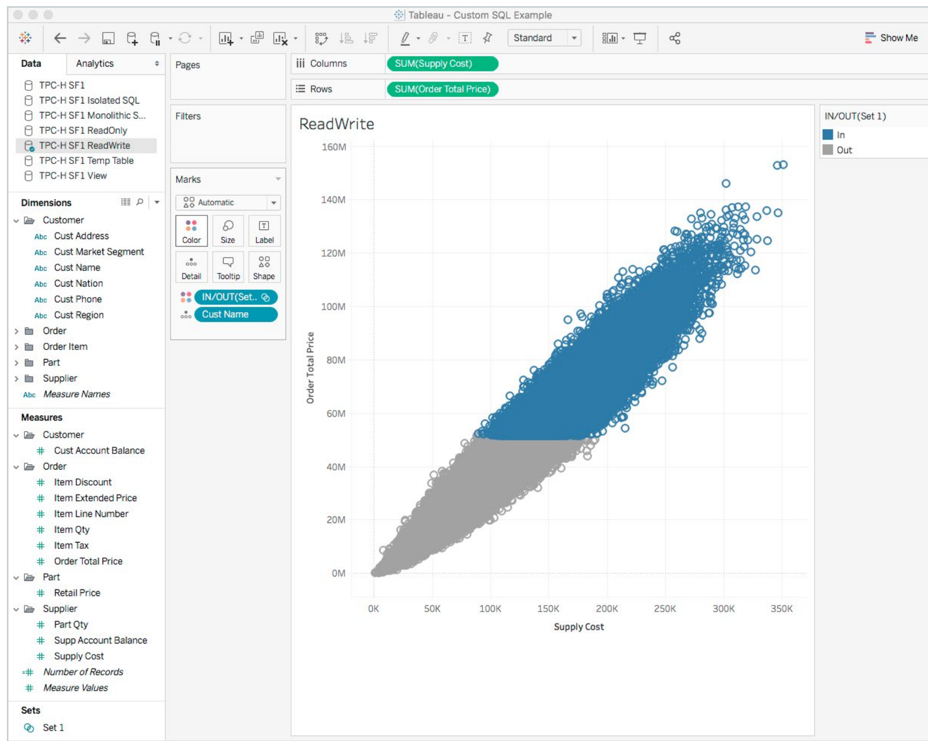


Figure 41

used on the color shelf to show IN/OUT: The USERNAME() function is evaluated in Tableau and then pushed through At set creation, Tableau created a temp table in the background to hold the list of dimensions specified in the set (in this case, the customer name):

```
CREATE LOCAL TEMPORARY TABLE "#Tableau_37_2_Filter" (
    "X_C_NAME" VARCHAR(18) NOT NULL,
    "X__Tableau_join_flag" BIGINT NOT NULL
    ) ON COMMIT PRESERVE ROWS
```

The temp table is then populated with the selected values:

```
INSERT INTO "#Tableau_37_2_Filter" ("X_C_NAME", "X__Tableau_join_flag")
    VALUES (?, ?)
```

Finally, the following query is then executed to generate the required result set:

```
SELECT "Customers"."C_NAME" AS "C_NAME",
    (CASE WHEN ((CASE WHEN (NOT ("Filter_1"."X__Tableau_join_flag" IS NULL)) THEN 1 ELSE 0
END) = 1) THEN 1 ELSE 0 END) AS "io:Set 1:nk",
    SUM("Orders"."O_TOTALPRICE") AS "sum:O_TOTALPRICE:ok",
    SUM("PARTSUPP"."PS_SUPPLYCOST") AS "sum:PS_SUPPLYCOST:ok"
FROM "TPCH1"."LINEITEM" "LineItem"
    INNER JOIN "TPCH1"."ORDERS" "Orders" ON ("LineItem"."L_ORDERKEY" = "Orders"."O_ORDERKEY")
    INNER JOIN "TPCH1"."CUSTOMER" "Customers" ON ("Orders"."O_CUSTKEY" =
"Customers"."C_CUSTKEY")
    INNER JOIN "TPCH1"."PART" "Parts" ON ("LineItem"."L_PARTKEY" = "Parts"."P_PARTKEY")
    INNER JOIN "TPCH1"."PARTSUPP" "PARTSUPP" ON ("Parts"."P_PARTKEY" =
"PARTSUPP"."PS_PARTKEY")
LEFT JOIN "#Tableau_37_2_Filter" "Filter_1" ON ("Customers"."C_NAME" =
    "Filter_1"."X_C_NAME")
GROUP BY 1,
    2
```

If a user does not have the right to create temp tables in the target database or schema, Tableau will create WHERE IN clauses in the query to specify the set members:

```
SELECT "Customers"."C_NAME" AS "C_NAME"
FROM "TPCH_SF1"."CUSTOMER" "Customers"
WHERE ("Customers"."C_NAME" IN ('Customer#000000388', 'Customer#000000412', 'Customer#000000679', 'Customer#000001522',
'Customer#000001948', 'Customer#000001993', 'Customer#000002266', 'Customer#000002548', 'Customer#000003019',
'Customer#000003433', 'Customer#000003451',
...
'Customer#000149362', 'Customer#000149548'))
GROUP BY 1
ORDER BY 1 ASC
```

However, this approach is subject to the limitations of the query parser, and you will encounter errors if you try to create a set with more than 16,384 marks:

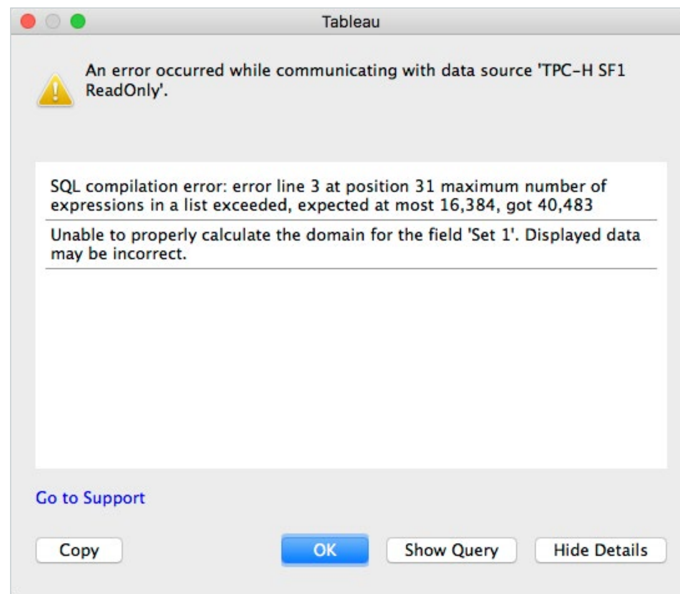


Figure 42

If the data volumes are not massive, you can overcome this error by converting the data connection from a live connection to an extracted connection.

MEASURING PERFORMANCE

Both Tableau and Snowflake offer several mechanisms for analyzing performance.

In Tableau, you can use the following:

- Performance recorder
- Resource monitoring tool
- Server performance views (Tableau Server)

Snowflake offers several built-in performance analysis features, including:

- Information schema
- Query history
- Query profiles

In Tableau

Several tools, including open-source tools, are available to help you analyze query performance.

PERFORMANCE RECORDER

The first place you should look for performance information is the Performance Recording feature of Tableau Desktop and Server. You enable this feature under the **Help** menu:

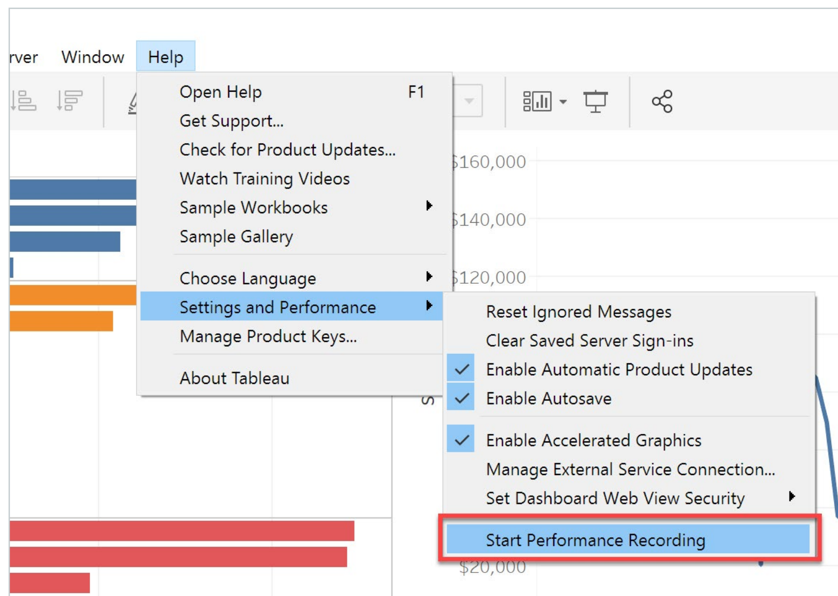


Figure 43

Start performance recording, then open your workbook. Interact with it as if you were an end user, and when you feel you have gathered enough data, go back to the **Help** menu and stop recording. Another Tableau Desktop window opens to show the data captured:

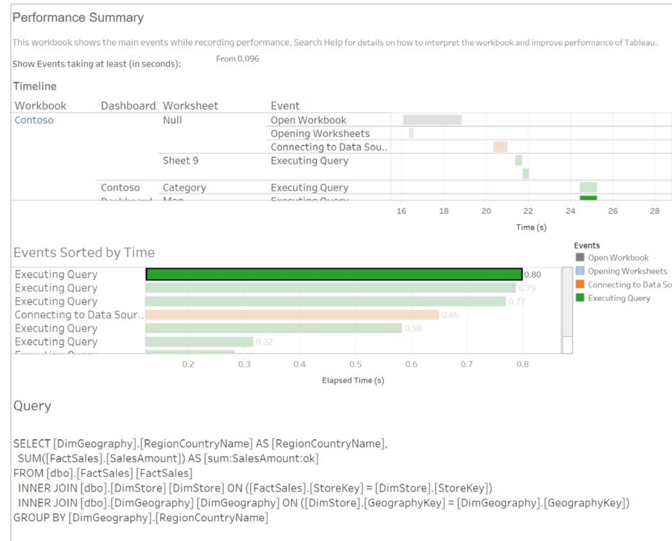


Figure 44

You can now identify the actions in the workbook that take the most time. For example, in the above image, the selected query takes 0.8 seconds to complete. Clicking on the bar shows the text of the query being executed. As the output of the performance recorder is a Tableau Workbook, you can create additional views to explore this information in other ways.

Use this information to identify which sections of a workbook to review, so you can focus on where you can get the best improvement for the time you spend. For more information on interpreting these recordings, see the Tableau documentation.

DESKTOP LOGS

Log files can provide detailed information on what Tableau is doing. The default location for Desktop is C:\Users\<username>\Documents\My Tableau Repository\Logs\log.txt in Windows and //Users/<username>/Documents/My Tableau Repository for MacOS. This file contains a lot of information written as JSON encoded text, but if you search for begin-query or end-query, you can find the query string being passed to the data source. Looking at the end-query log record also shows the time the query took to run and the number of records that were returned to Tableau:


```
{"ts":"2020-05-24T12:25:41.226","pid":6460,"tid":"1674","sev":"info","req":"-","sess":"-","
site":"-","user":"-","k":"end—query", "v":{"protocol":"4308fb0","cols":4,"query":"SELECT
[DimProductCategory].[ProductCategoryName] AS [none:ProductCategoryName:nk],\n
[DimProductSubcategory].[ProductSubcategoryName] AS [none:ProductSubcategoryName:nk],\n
SUM(CAST(([FactSales].[ReturnQuantity]) as BIGINT)) AS [sum:ReturnQuantity:ok],\n
SUM([FactSales].[SalesAmount]) AS [sum:SalesAmount:ok]\nFROM [dbo].[FactSales]
[FactSales]\n  INNER JOIN [dbo].[DimProduct] [DimProduct] ON ([FactSales].[ProductKey] =
[DimProduct].[ProductKey])\n  INNER JOIN [dbo].[DimProductSubcategory]
[DimProductSubcategory] ON ([DimProduct].[ProductSubcategoryKey] =
[DimProductSubcategory].[ProductSubcategoryKey])\n  INNER JOIN [dbo].[DimProductCategory]
[DimProductCategory] ON ([DimProductSubcategory].[ProductCategoryKey] =
[DimProductCategory].[ProductCategoryKey])\nGROUP BY
[DimProductCategory].[ProductCategoryName],\n
[DimProductSubcategory].[ProductSubcategoryName]","rows":32,"elapsed":0.951}}
```

Since Tableau 10.1, Tableau supports JSON files as a data source, so you can also open the log files with Tableau for easier analysis. You can analyze the time it takes for each event, what queries are being run, and how much data they are returning:

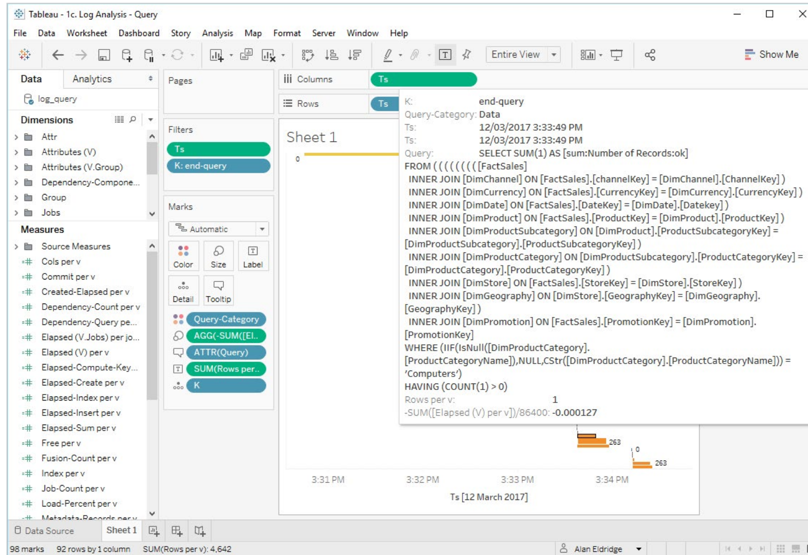


Figure 45

Another useful tool for analyzing Tableau Desktop performance is the Tableau Log Viewer (TLV). This cross-platform tool enables you to easily view, filter, and search the records from Tableau's log file. A powerful feature of TLV is that it supports live monitoring of Tableau Desktop, so you can see in real time the log entries created by your actions in Tableau.

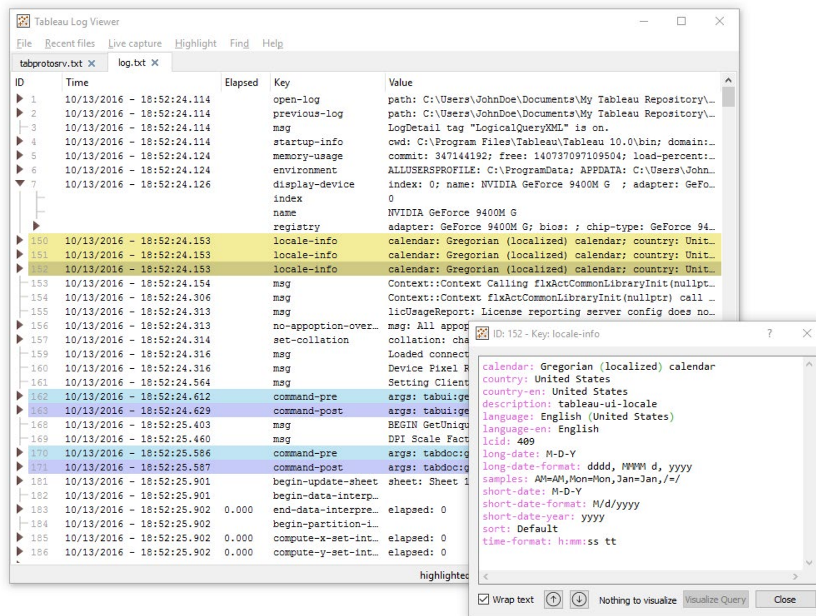


Figure 46

This tool is made available as-is by Tableau and can be downloaded from GitHub.

Additionally, you could load the JSON logs into Snowflake and take advantage of the VARIANT data type to help with your analysis. Loading the Tableau JSON logs to Snowflake allows quick and easy access to a variety of Snowflake analytical functions on JSON as well as full SQL support through object dot notation and nested array flattening.

```
select * from stg_Logs where log_json:k::string = 'qp-batch-summary' or log_json:k::string = 'end-query';
```

SERVER LOGS

On Tableau Server, the logs are in C:\ProgramData\Tableau\Tableau Server\data\tabsvc\<service name>\Logs. In most cases, you should focus on the VizQL Server log files. If you do not have console access to the Tableau Server machine, you can download a ZIP archive of the log files from the Tableau Server status page:

Log Files

Date generated	Size	Status
18 Jun 2017 2:08 pm	39.2 MB	Snapshot ready to download.

Generate Snapshot... | Download Snapshot | Delete Snapshot

Figure 47

Just like the Tableau Desktop log files, these are JSON-formatted text files, so you can open them in Tableau Desktop or read them in their raw form. However, because the information about user sessions and actions is spread across multiple files (corresponding to the various services of Tableau Server), you may prefer to use a powerful tool called Logshark to analyze server logs. Logshark is a command-line utility that you can run against Tableau logs to generate a set of workbooks that provide insights into system performance, content usage, and error investigation. You can use Logshark to visualize, investigate, and solve issues with Tableau.

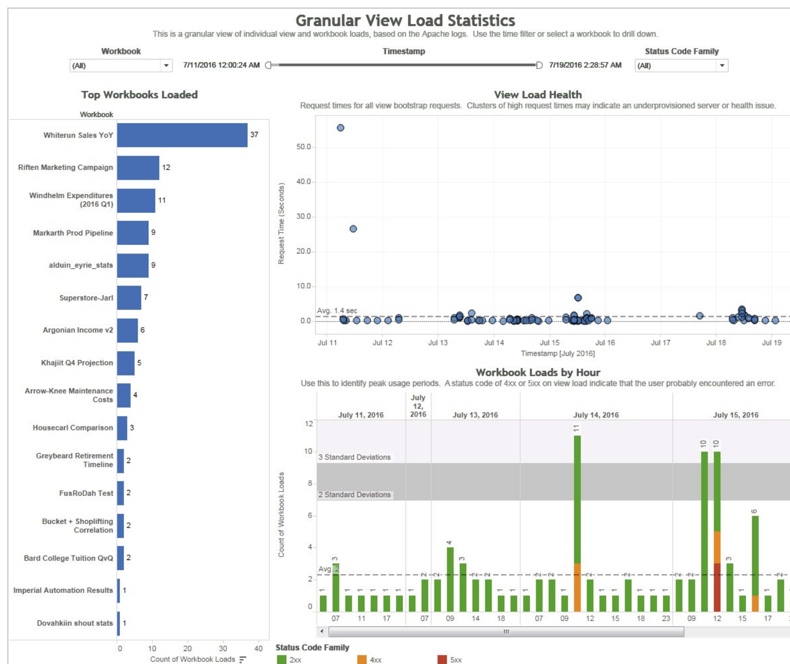


Figure 48

You can find out more about Logshark here, and you can download it from GitHub.

Additionally, you could load the JSON logs into Snowflake and take advantage of the VARIANT data type to help with your analysis.

SERVER PERFORMANCE VIEWS

Tableau Server comes with several views to help administrators monitor activity on Tableau Server. The views are located in the Analysis table on the server's Maintenance page. They can provide valuable information on the performance of individual workbooks, helping you focus your attention on those that are performing poorly:

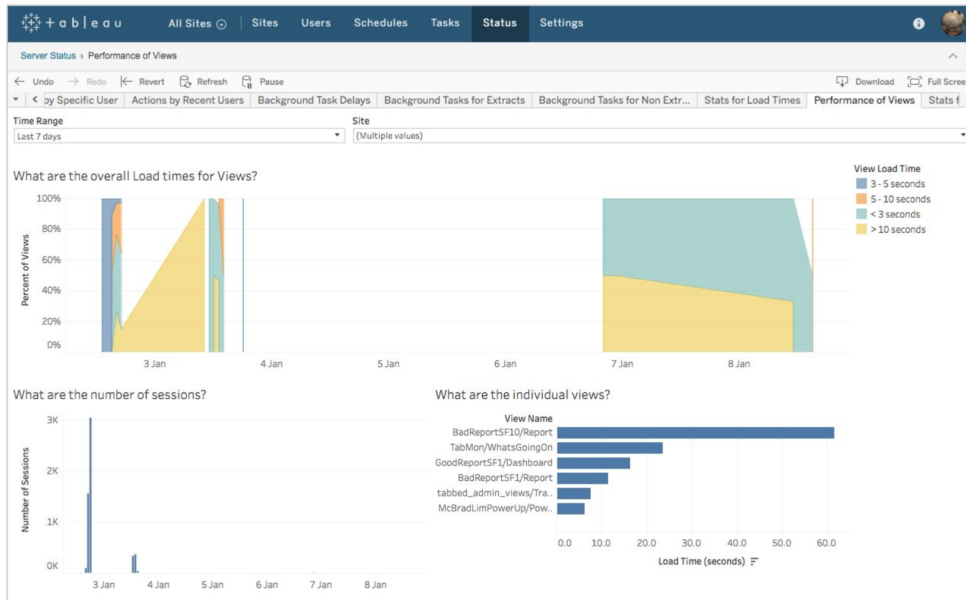


Figure 49

More information on these views can be found here.

Additionally, you can create custom administrative views by connecting to the PostgreSQL database that makes up part of the Tableau repository. Instructions can be found here.

RESOURCE MONITORING TOOL

The Resource Monitoring Tool is a tool that you can use to monitor the health and performance of your Tableau Server. It gathers data from your Tableau Server and provides a comprehensive look at the health of Tableau Server. Using this tool, you can identify what is causing slow load times, extract failures, and other critical issues. More information about the Resource Monitoring Tool can be found in Tableau Documentation.

TABJOLT

Use TabJolt to determine if platform capacity is causing issues. TabJolt is particularly useful for testing Snowflake multi-cluster warehouse scenarios when you want to simulate a load generated by multiple users.

TabJolt is a "point-and-run" load and performance testing tool specifically designed to work easily with Tableau Server. Unlike traditional load-testing tools, TabJolt can automatically simulate loads on your Tableau Server without script development or maintenance. Because TabJolt is compatible with Tableau's presentation model, it can automatically load visualizations and interpret possible interactions during test execution.

This enables you to just point TabJolt to one or more workbooks on your server and automatically load and execute interactions on the Tableau views. TabJolt also collects key metrics including average response time, throughput, and 95th percentile response time. In addition, it captures Windows performance metrics for correlation.

Of course, even with TabJolt, you should understand Tableau Server's architecture. Treating Tableau Server as a black box for load testing is not recommended, and it will likely yield results that aren't in line with your expectations.

You can find more information on TabJolt here.

In Snowflake

Snowflake has several built-in features that enable you to monitor performance of the queries generated by Tableau and to link them back to specific workbooks, data connections, and users.

THE SNOWFLAKE DATABASE

SNOWFLAKE is a system-defined, read-only shared database, provided by Snowflake. The database is automatically imported into each account from a share named ACCOUNT_USAGE. The SNOWFLAKE database is an example of Snowflake utilizing Secure Data Sharing to provide object metadata and other usage metrics for your account.

SNOWFLAKE INFORMATION SCHEMA

The Snowflake Information Schema (Data Dictionary) is a set of system-defined views and functions that provide extensive information about the objects created in your account. The Snowflake Information Schema is based on the SQL-92 ANSI Standard Information Schema, but also includes views and functions that are specific to Snowflake.

DIFFERENCES BETWEEN ACCOUNT USAGE AND INFORMATION SCHEMA

The account usage views and the corresponding views (or table functions) in the Information Schema utilize identical structures and naming conventions, but with some key differences, as described in this section:

Difference	Account Usage	Information Schema
Includes dropped objects	Yes	No
Latency of data	From 45 minutes to 3 hours (varies by view)	None
Retention of historical data	1 Year	From 7 days to 6 months (varies by view/table function)

Figure 50

A Tableau dashboard that leverages the account usage data to provide insights about your Snowflake account can be found here. Please also review the Account Usage documentation.

SNOWFLAKE QUERY HISTORY

Within the Information Schema is a set of tables and table functions that can be used to retrieve information on queries that have been run during the past seven days.

If you have AccountAdmin access to the Snowflake account usage data, you can view the full account-level query history for the last 12 months. This can be a useful tool for analyzing overall query volume and performance. For more information, see the Snowflake documentation.

The query history can provide detailed information on the profile execution timing and profile of queries, which you can use to determine if your warehouse sizes are appropriate. You can see information on the start time, end time, query duration, number of rows returned, and the data volume scanned. You can also see the amount of time queries spent queued versus executing, which is useful information when you are considering adding multi-cluster scaling to your warehouses.

You can access this information through the query_history table functions in Tableau:

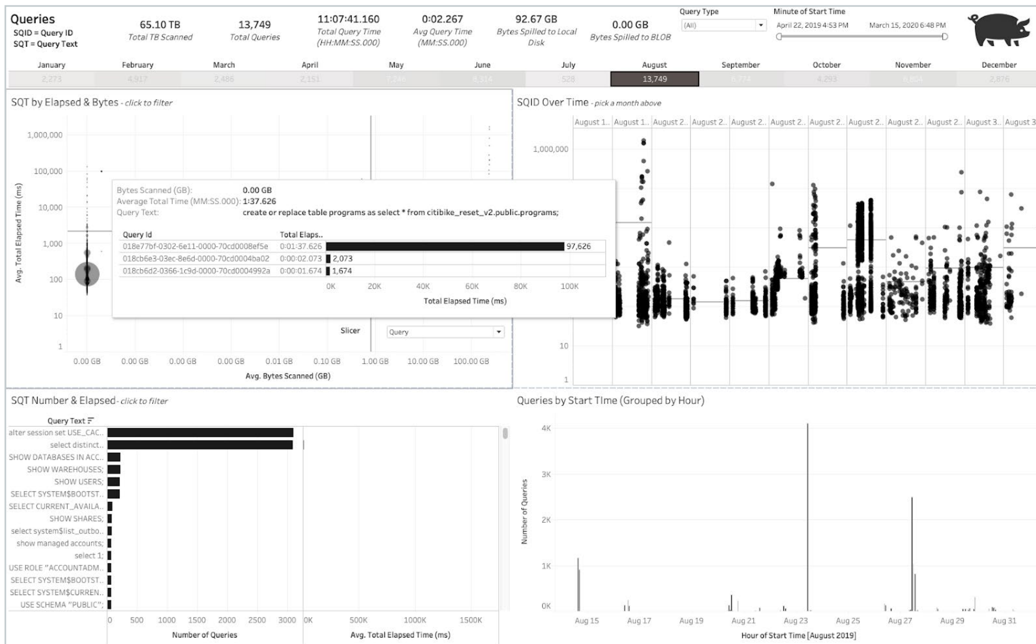


Figure 51

In addition, you can access the query history for an account from the Snowflake web interface:

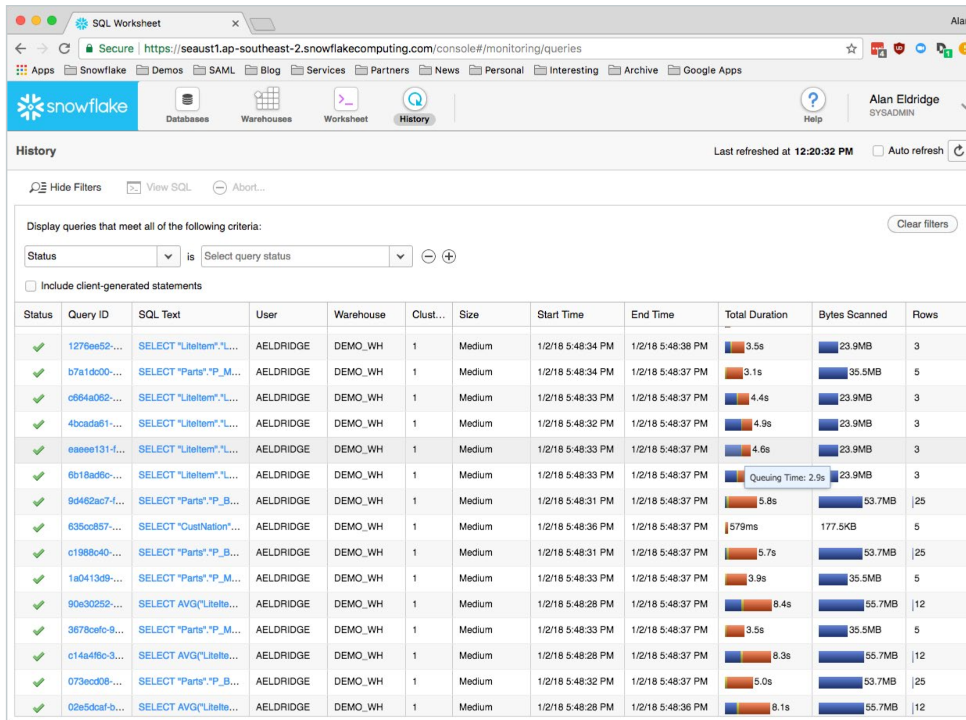


Figure 52

This page displays all queries executed in the last 14 days, including queries executed by you and other users. This view shows query duration broken down by queueing, compilation, and execution. It is also useful for identifying queries that are resolved through the result cache.

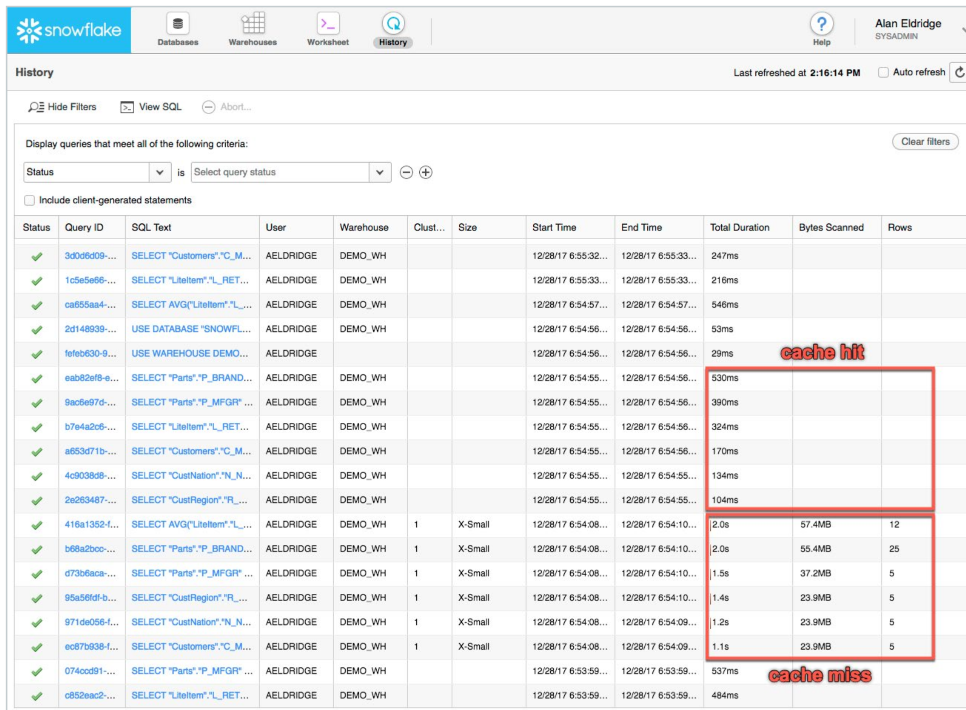


Figure 53

By clicking on the hyperlink in the Query ID column, you can drill through to more detailed statistics for the selected query:

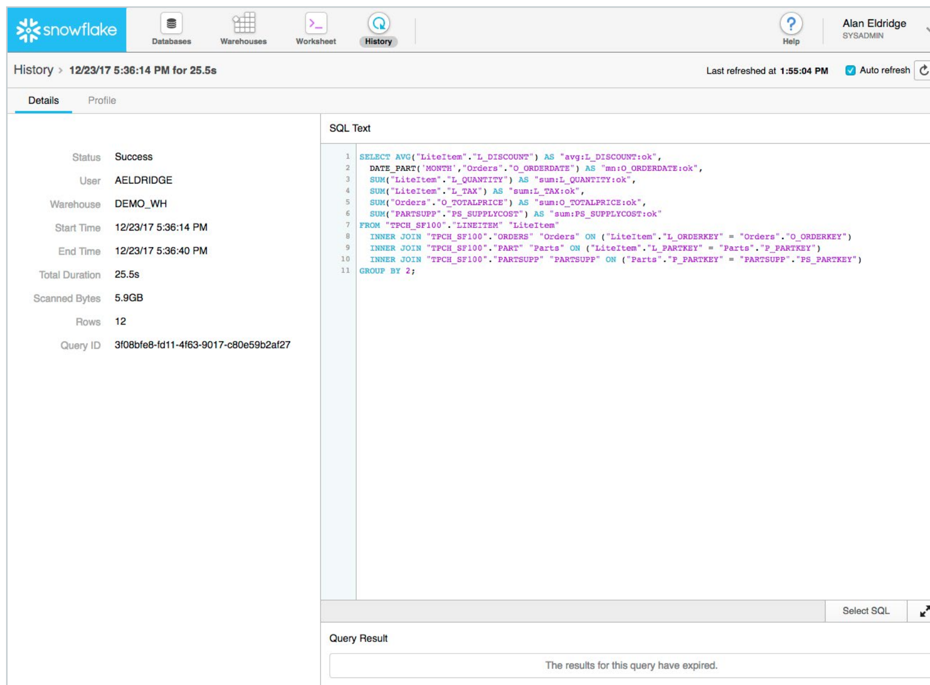


Figure 54

This is also where you can access the query profile.

SNOWFLAKE QUERY PROFILE

Query Profile is a powerful tool for understanding the mechanics of queries. Implemented as a page in the Snowflake web interface, it provides a graphical tree view and detailed execution statistics for the major components of the processing plan for a query. It is designed to help troubleshoot issues in SQL query expressions that commonly cause performance bottlenecks.

To access Query Profile:

1. Go to the **Worksheet** or **History** page in the Snowflake web interface.
2. Click on **Query ID** for a completed query.
3. In the query details, click the **Profile** tab.

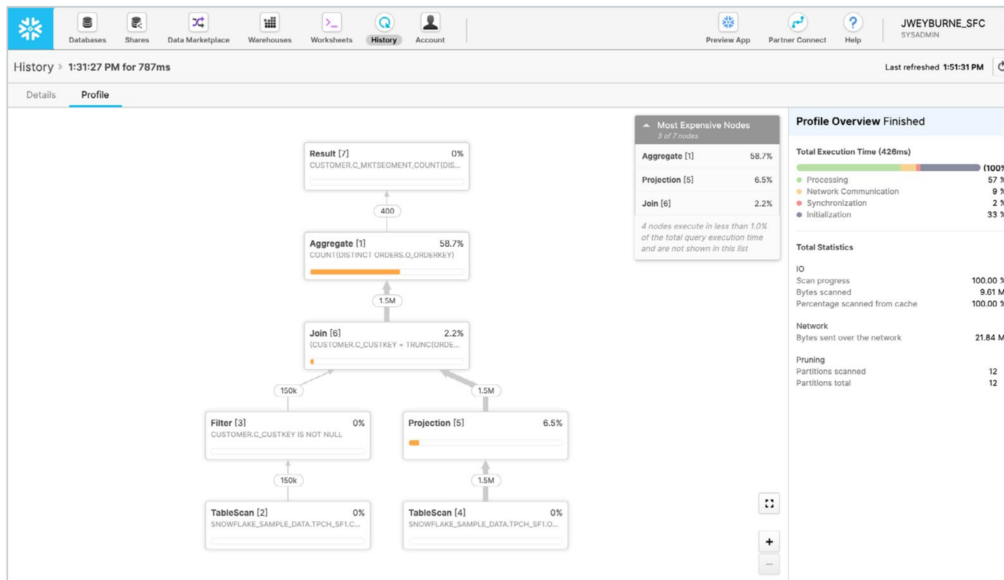


Figure 55

To help you analyze query performance, the Details panel provides two classes of profiling information, execution time and statistics.

EXECUTION TIME

Execution time provides information about where the time was spent during the processing of a query. It is broken down into the following categories:

- **Processing:** time spent on data processing by the CPU
- **Local Disk IO:** time when the processing was blocked by local disk access
- **Remote Disk IO:** time when the processing was blocked by remote disk access
- **Network Communication:** time when the processing was waiting for the network data transfer
- **Synchronization:** time spent on various synchronization activities between participating processes
- **Initialization:** time spent setting up the query processing

Statistics

A major source of information provided in the Details panel consists of various statistics that can help you identify common problems that occur during queries.

"Exploding" joins

Two common mistakes users make is joining tables without providing a join condition (resulting in a "Cartesian Product") and getting the join condition incorrect (resulting in records from one table matching multiple records from another table). For such queries, the join operator produces significantly (often by orders of magnitude) more tuples than it consumes.

You can observe this by looking at the number of records produced by a join operator or noticing that the join operation consumes a lot of time.

The following example shows input in the hundreds of records but output in the hundreds of thousands:

```
select tt1.c1, tt1.c2
from tt1
join tt2 on tt1.c1 = tt2.c1
    and tt1.c2 = tt2.c2;
```

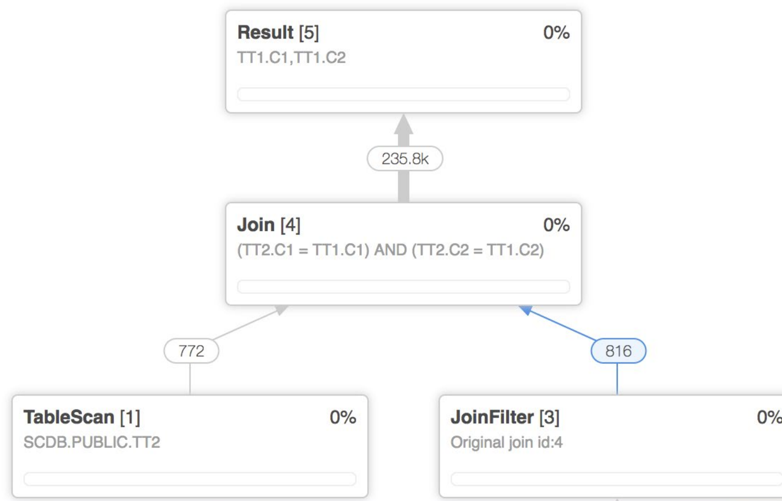


Figure 56

The recommended action for this problem is to review the join conditions defined in the Tableau Connect dialog box and correct any omissions.

Queries too large to fit in memory

For some operations (for example, duplicate elimination for a huge), the amount of memory available might not be sufficient to hold intermediate results. As a result, the query processing engine will start *spilling* the data to a local disk. If the local disk space is insufficient, the spilled data is then saved to remote disks.

This spilling can have a profound effect on query performance, especially if data is spilled to a remote disk. To alleviate this, you should use a larger warehouse (effectively increasing the available memory and local disk space for the operation) or process data in smaller batches.

Inefficient pruning

Snowflake collects rich statistics on data. Query filters dictate which parts of a table it reads. However, effective filtering requires the data storage order to be correlated with the query filter attributes.

To determine the efficiency of pruning, compare the number of partitions scanned to the total number of partitions in the TableScan operators. If the former is a small fraction of the latter, pruning is efficient. If not, the pruning did not have an effect.

Of course, pruning can help only for queries that filter out a significant amount of data. If the pruning statistics do not show data reduction, but a Filter operator above TableScan filters out several records, this might signal that a different data organization might be needed.

Clustering

If the query profile shows inefficient pruning related, and it's believed that this is related to the ordering of data, one option to consider is clustering. Snowflake supports designating one or more table columns/expressions as a clustering key for the table.

Using a clustering key to co-locate similar rows enables several benefits for very large tables, including:

- Improved scan efficiency in queries by skipping data that does not match filtering predicates.
- Better column compression than in tables with no clustering. This is especially true when other columns are strongly correlated with the columns that comprise the clustering key.
- After a key has been defined on a table, no additional administration is required, unless you choose to drop or modify the key. All future maintenance on the rows in the table (to ensure optimal clustering) is performed automatically by Snowflake (see the guide on Automatic Clustering).

Although clustering can substantially improve the performance and reduce the cost of some queries, the compute resources used to perform clustering consume credits. As such, you should cluster only when queries will benefit substantially from the clustering.

Linking performance data between Tableau and Snowflake

One of the challenges users face when examining performance is connecting queries run in Snowflake with the workbooks and user sessions in Tableau that generated them. One useful way to do this is to set the QUERY_TAG session variable using an initial SQL statement.

Example

The following statement uses an initial SQL block to set the QUERY_TAG session variable:

```
ALTER SESSION SET QUERY_TAG = [WorkbookName][TableauApp][TableauServerUser];
```

The resulting value is then available from the Query History table function and can be used to attribute queries to specific workbooks and users:

Figure 57

Example

With the previous example, you still need access to Query History in Snowflake to get more details. However, using the same initial SQL for every data source you create, you can insert the session ID to an audit table, then use the audit table to create a Tableau dashboard. The dashboard gives an overview of SQL Statements that were executed in Snowflake and many other key metrics such as execution time, bytes scanned, partitions scanned, compilation time, credits used, warehouse size, and other key metrics.

Follow these steps:

1. Create a Tableau audit table in Snowflake.

```
CREATE OR REPLACE TABLE TABLEAU_AUDIT
(QID integer, exectime timestamp,
 APP string,tabserver string,workbookname string,  version string);
```

2. In the initial SQL of the data source, use the following SQL statement to insert the unique session identifier and the Tableau parameters listed.

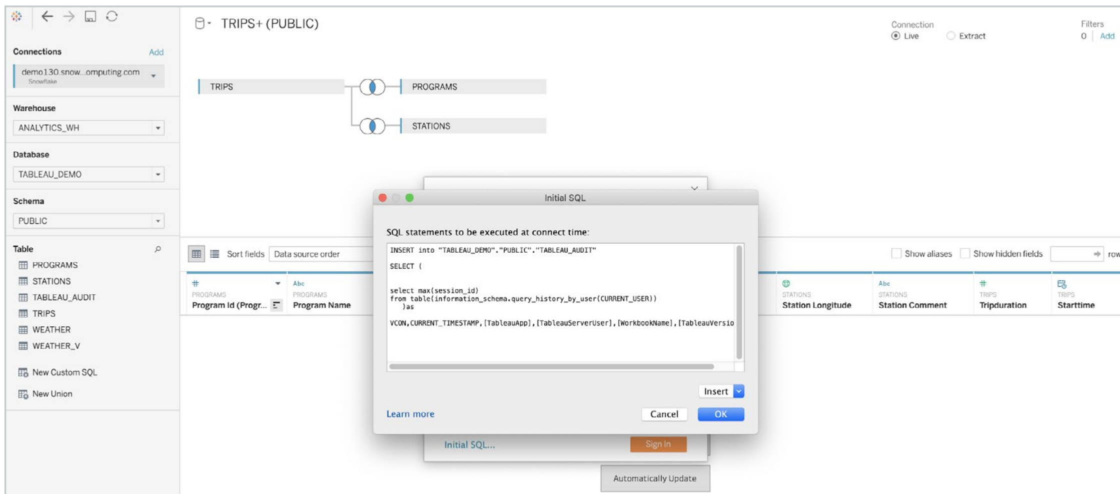


Figure 58

Initial SQL used in the above screenshot:

INSERT into "TABLEAU_DEMO"."PUBLIC"."TABLEAU_AUDIT"

```
SELECT (
select max(session_id)
from table(information_schema.query_history_by_user(CURRENT_USER))
  )as
VCON,CURRENT_TIMESTAMP,[TableauApp],[TableauServerUser],[WorkbookName],[TableauVersion] ;
```

3. Create the Tableau dashboard by combining the audit table with SNOWFLAKE.ACCOUNT_USAGE. QUERY_HISTORY.

```
 Select * from SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY SF
INNER JOIN TABLEAU_DEMO.PUBLIC.TABLEAU_AUDIT T on F.SESSION_ID = T.QID
where SF.QUERY_TEXT like 'SELECT%'
```

The following sample showcases query text, partitions scanned, and what the warehouse was used for. This is just a sample. You can filter results by workbook names, usernames, warehouse size, and many other metrics.

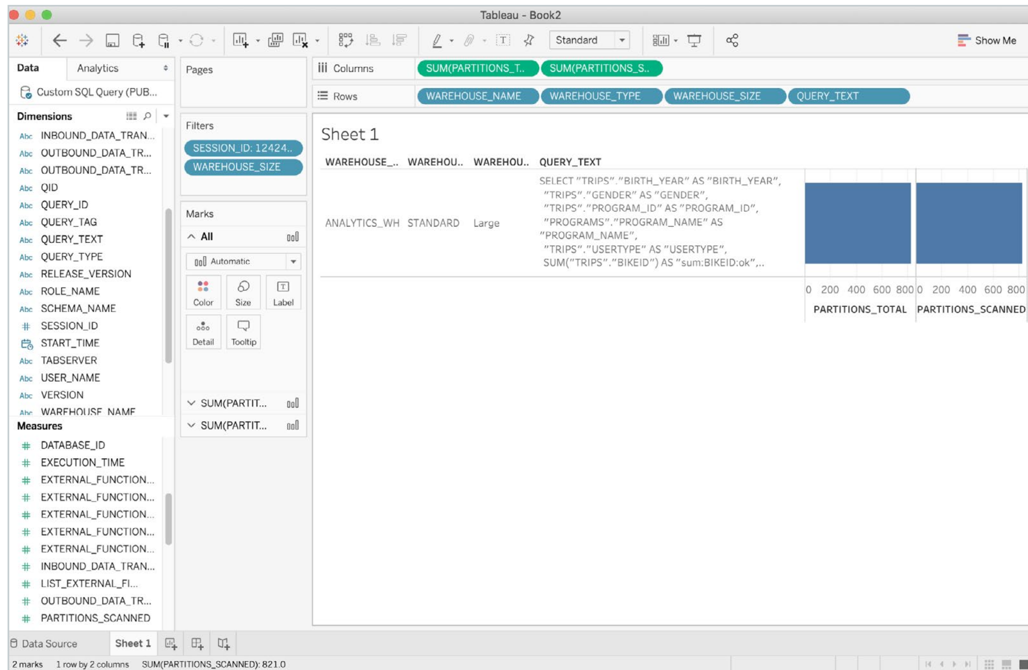


Figure 59

Using an audit table will retain the history as long as you need.

CONCLUSION

Tableau and Snowflake's platform combine to enable you to analyze massive amounts of data. Using best practices for connecting, caching, and performance monitoring enables faster queries, and taking advantage of features like Snowflake Time Travel and Secure Data Sharing empower you to become even more data-driven.

For more information about any of the features described in this white paper, consult the Tableau support site or Snowflake documentation.

ABOUT SNOWFLAKE

Snowflake shatters barriers that prevent organizations from unleashing the true value from their data. Thousands of customers around the world mobilize their data in ways previously unimaginable with Snowflake's cloud data platform—a solution for data warehousing, data lakes, data engineering, data science, data application development, and data exchange. Snowflake provides the near-unlimited scale, concurrency, and performance our customers in a variety of industries want, while delivering a single data experience that spans multiple clouds and geographies. Our cloud data platform is also the engine that drives the Data Cloud—the global ecosystem where thousands of organizations have seamless and governed access to explore, share, and unlock the potential of data. Learn how you can mobilize your data at **snowflake.com**

© 2021 Snowflake. All rights reserved.