Lab 10. Tableau for Big Data

Nowadays, we have a lot of popular data platforms, for instance, Google BigQuery, Azure Data Warehouse, Hadoop, and Snowflake. In this lab, we will integrate Tableau Desktop with the most popular among them.

In this lab, we will cover the following topics:

- Connecting with Amazon Redshift
- Launching an Amazon Redshift cluster
- Connecting a Redshift cluster
- Loading sample data into the Redshift cluster
- Connecting Redshift with Tableau
- Creating a Tableau report
- Tuning Redshift for efficient Tableau performance
- Connecting to Amazon Redshift Spectrum
- Connecting to Snowflake
- Using SnowSQL CLI
- Connecting Tableau to Snowflake
- Connecting big data
- Accessing semi--structured data
- Connecting Amazon Elastic MapReduce with Apache Hive
- Creating sample data
- Connecting Tableau with Apache Hive

Technical requirements

To perform the recipes in this lab, you will need to have Tableau Desktop 2019.x installed. Moreover, you need internet access to register and download the trial versions of the products required. We will create an AWS account and launch AWS EMR, Redshift, and Spectrum. Finally, we will launch a Snowflake instance.

Introduction

Nowadays, almost all organizations are trying to become data-driven. Organizations collect data about sales, customer behavior, user experience, inventory, clickstream, marketing activity, and more. As a result, the data volume is huge and can't fit in a single computer. There are also other attributes of big data such as velocity, variety, and value. BI and data engineers should use different analytics platforms that are able to process big volumes of data, often unstructured or streaming in real time.

Another important trend in the industry is shifting to the cloud. There are multiple leaders such as AWS, Google, and Azure, which offer cloud infrastructure, high availability, and security of the data. The cloud gives us a lot of advantages and also gives us more time to focus on data processing and analysis.

Tableau supports more than 40 different data sources. There are plenty of data sources available for big data ecosystems, as follows:

- [**Cloud Data Platforms**]: Snowflake, Amazon Redshift, AWS Spectrum, Amazon Athena, Google BigQuery, and so on
- [**Hadoop**]: Cloudera, Hortonworks, Hive, Presto, and so on
- [**MPP databases**]: Teradata, Oracle Exadata, HP Vertica, Exasol, SAP HANA, and many more In this lab, we will learn how to connect the most popular big data platforms, such as Amazon Redshift, Snowflake, and Hadoop to Tableau. Moreover, we will look into the data lake concept and will connect our raw data to Tableau using Amazon Spectrum.

It is important to understand that the key element for working with massive datasets in Tableau is good data engineering execution, to make sure that all heavy lifting is performed by the big data systems. Tableau will just render the query results.

Connecting with Amazon Redshift

[**Amazon Web Services**] ([**AWS**]) completely changed the way that IT infrastructure is deployed. It is available on demand and is cost-effective. Amazon Redshift is one of the hundreds of AWS services and it is one of the most popular cloud data warehouses. It is fast, secure, and petabyte-scale. Redshift combines the following two important technologies:

- Columnar data store or column-oriented database
- [**Massively parallel processing**] ([**MPP**]) You can learn more about MPP and column databases on the Internet. You can find more information about Redshift at AWS's documentation: https://docs.aws.amazon.com/redshift/index.html{.ulink}

Getting ready

Before we start, we need to create an AWS account or use an existing one. AWS offers us a two month free trial, and then we can spin up the smallest Redshift cluster and do important network settings to open access from our local machine to Amazon Redshift. Finally, we will load some sample data into Redshift from an S3 bucket, and then we will query and connect Tableau Desktop.

In this section, we will go deep into data engineering design to show you the main principles of working with big data, so that we can use the strength of each tool. These patterns could be applicable to any other big data platform.

How to do it...

To launch an Amazon Redshift cluster, we should have an AWS account. Then, we need to set up security with AWS [**Identity and Access Management**] ([**IAM**]).

Creating an AWS account

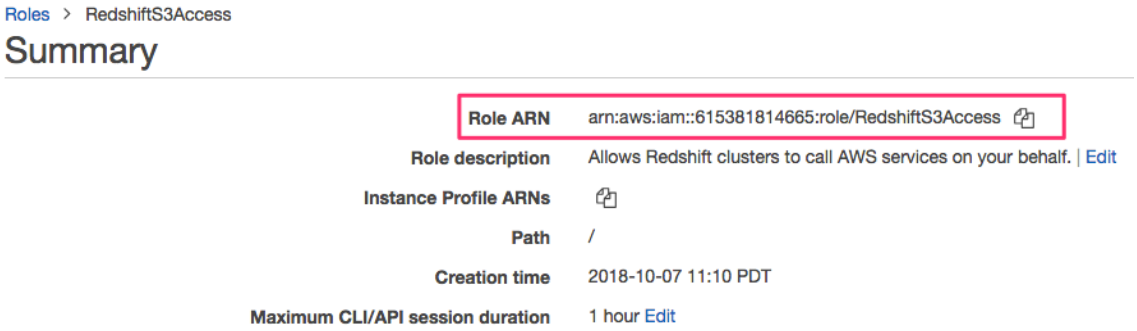
Go to https://aws.amazon.com/account/ and sign in to an existing account, or create a new account.

Creating an IAM role

We should create an [**IAM**] role to get access to data for another AWS resource, such as Amazon S3 buckets because our cluster needs some permissions to access the resources and the data. You can learn more about IAM and permissions to access other AWS resources here: https://docs.aws.amazon.com/redshift/latest/dg/copy-usage_notes-access-permissions.html

Follow these steps to create an IAM role:

1. Go to `IAM` and choose `Roles`.
2. Click `Create Role`.
3. Under `AWS services`, choose `Redshift`.
4. Under `Select your use case`, choose `Redshift Customizable`, and then click `Next`.
5. Attach `AmazonS3ReadOnlyAccess` and click `Next`.
6. Type the role name, `RedshiftS3Access`, and click `Create`.
7. Open the role that we just created and copy the [**Amazon Resource**][**Name**] ([**ARN**]): Let's view the results of the steps we carried out in the following screenshot:



We will need this later in this lab.

How it works...

AWS gives us an account for free where we can explore all the available services and features. In addition, AWS provides us with AWS Free Tier. You can learn about it here: https://aws.amazon.com/free/. In addition, we created an IAM role, to allow our Redshift cluster access the S3 bucket with data.

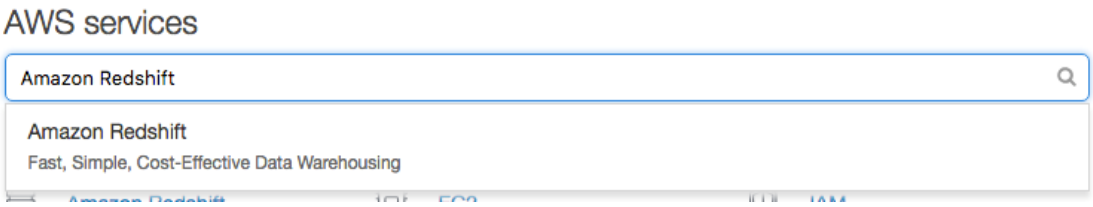
Launching an Amazon Redshift cluster

AWS allows us to create a powerful analytical data warehouse with just multiple clicks. In this recipe, we will create an Amazon Redshift cluster using AWS Free Tier. AWS provides us with two months of free usage of the basic cluster, which is still powerful and good for our purpose.

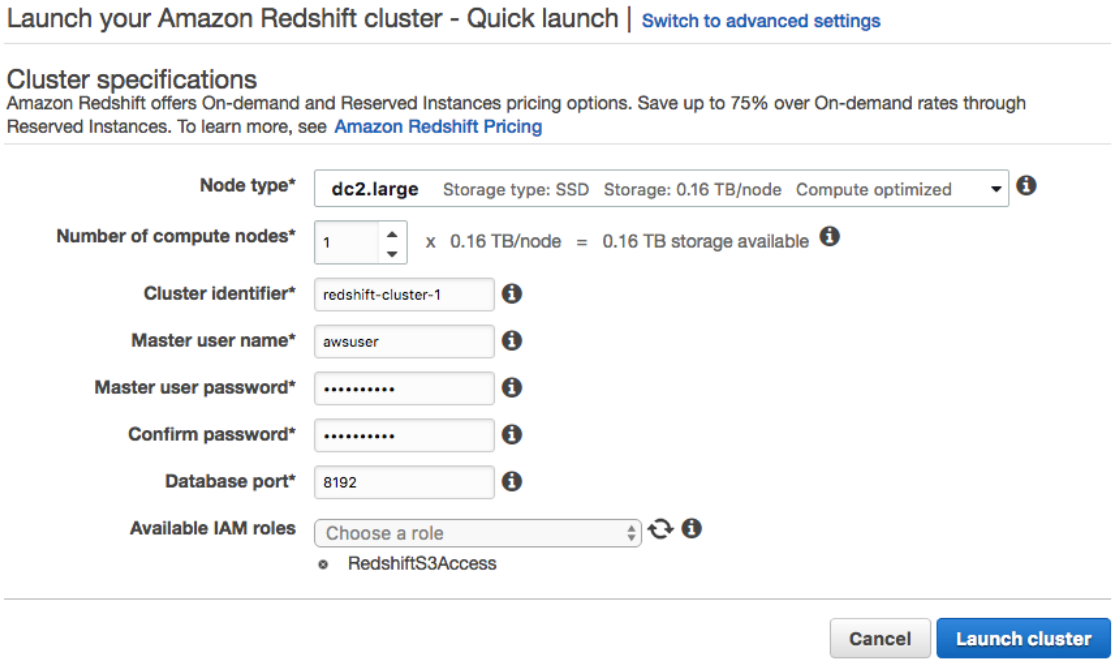
How to do it...

We should launch an Amazon Redshift cluster, as shown via the following steps:

1. Let's find `Amazon Redshift` among `AWS services`, as follows:



2. Click on `Quick Launch Cluster`.
3. Fill the options for the cluster, as shown in the following screenshot:



4. We should choose the smallest cluster if we want to use the trial version of AWS. We will choose our IAM roles and enter the password.
5. We should keep the password in a secure place, as we will need this for the Tableau connection. In this example, we are going to use a more powerful cluster `ds2.xlarge`, but we can work with the smallest available cluster.
6. Click on **`Launch cluster`**.

How it works...

As you might have noticed, the launching of Redshift is a very simple and straightforward approach. It is very important to decide what size cluster we want and give write permissions to AWS resources with IAM.

There's more...

You can learn more about the available sizes of clusters at https://aws.amazon.com/redshift/pricing/.

Connecting a Redshift cluster

The cluster should be created by now, and we should test that we can connect to the cluster from our local machine. The best way to do this is to download the SQL client and connect to it using the JDBC driver. It is the common pattern for databases or analytics tool connections.

The cluster should be created and we should test that we can connect to the cluster from our local machine. The best way to do this is to download an SQL client and connect using the JDBC driver. It is the common pattern for database or analytics tool connection.

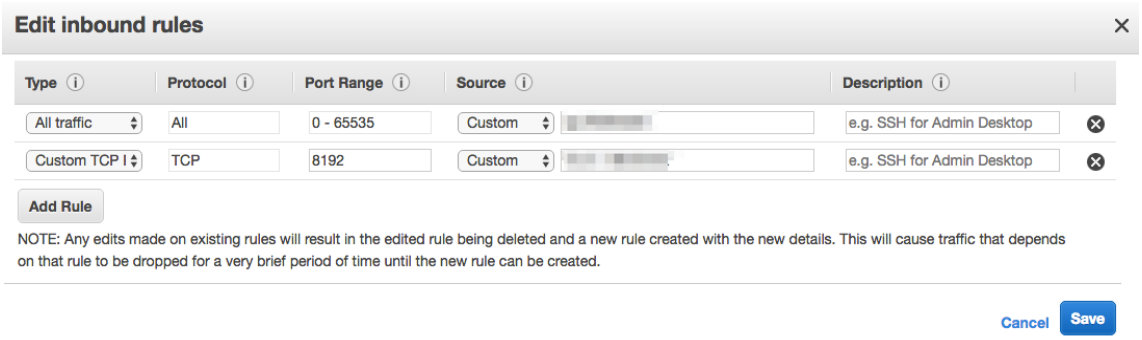
How to do it...

We will connect the Redshift cluster using the following steps:

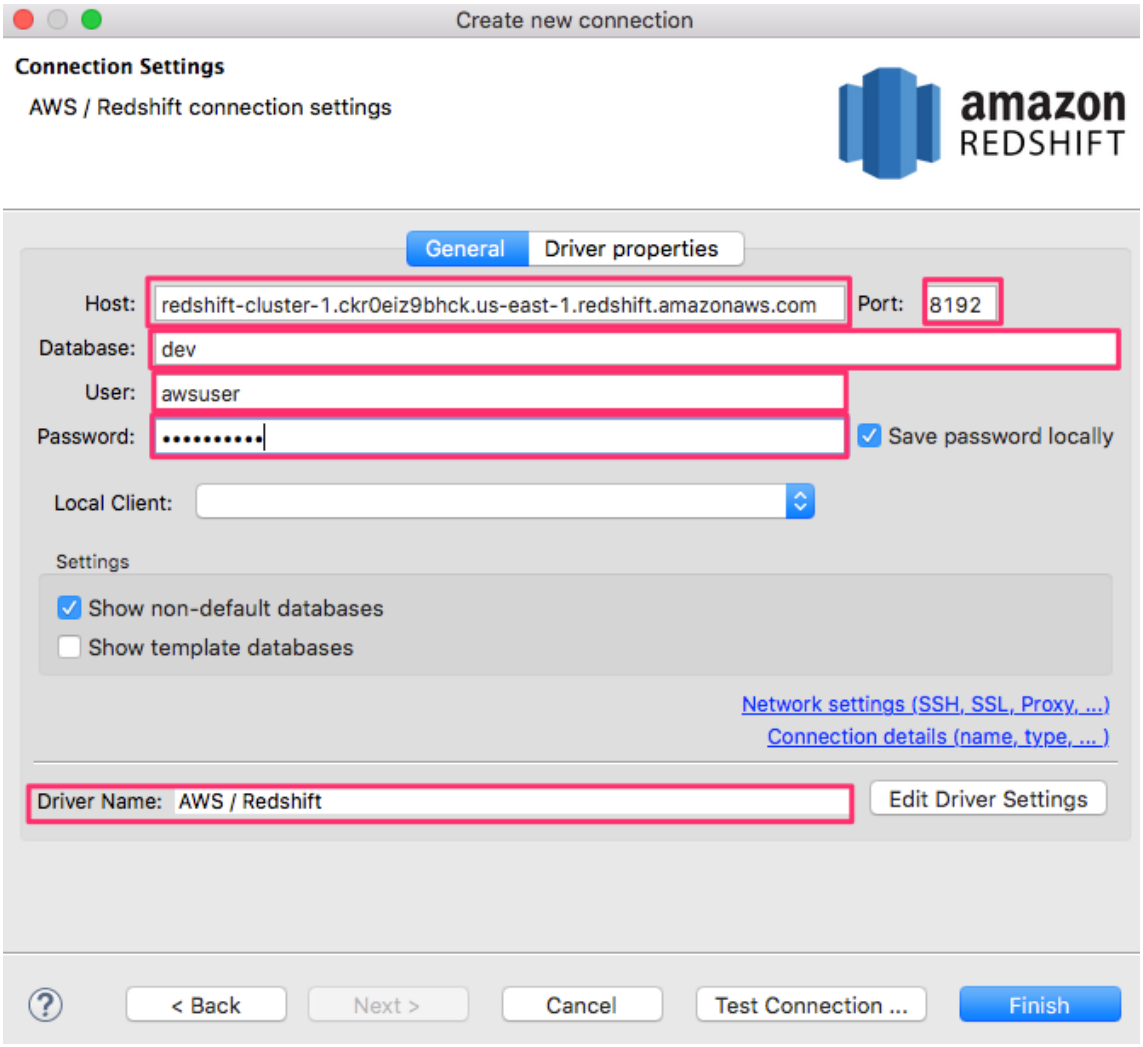
1. Let's download the Redshift SQL client. There are many available clients. Some of them are free, while others aren't. We will download the DBeaver SQL client. Go to https://dbeaver.io/download/ and download the

lastest available package for your OS from https://www.sql-workbench.eu/downloads.html. In my case, I will use macOS X (pkg installer + JRE).

2. We will now install DBeaver and launch it. It will offer us the option to create a new database connection. We should use `AWS Redshift`. The connection window will ask us about `Host`, `Database`, `Port`, `User`, and `Passwords`. We should copy them from the AWS console under our cluster properties.
3. By default, any resource in AWS is closed. We should adjust the security settings to allow access to the Redshift cluster from local machine. Among the `AWS Services`, find `EC2` and click on it.
4. On the left sidebar, click on `Security Groups`.
5. Edit your `VPC group`. In my case, I have only one default group. If you have more than one group, you can check for, the VPC group associated with your cluster.
6. Click `Edit inbound rules` and add a new rule. From the `Source`, you can choose `My IP`. Let's take a look at the results, which are shown in the following screenshot:



7. Now, we can go back to `DBeaver` and enter our credentials for a new Redshift database connection. We should copy these credentials from the AWS console under the Redshift properties, as shown in the following screenshot:



DBeaver will offer us the ability to download the Redshift JDBC driver. We will accept it and install the driver.

As a result, we have now connected to the Amazon Redshift cluster, and our next step is to get the sample data.

How it works...

To connect any data platform, we should obtain a driver and adjust the firewall. In our case, we've got an Amazon Redshift native JDBC driver. In addition, we had to use the SQL client to connect Redshift and query it. We used open source DBeaver, which served well for our purpose and works perfectly in real life.

There's more...

We might use a native PostgreSQL driver because Amazon Redshift was originally based on PostgreSQL. You can learn more about this at the following link:

https://docs.aws.amazon.com/redshift/latest/dg/c_redshift-postgres-jdbc.html

Moreover, you might consider the internal AWS SQL client---Query Editor.

You can also read about it at the following link:

https://aws.amazon.com/about-aws/whats-new/2018/10/amazon_redshift_announces_query_editor_to_run_queries_directly_from_the_aws_console/

In our case, it was important to demonstrate how Tableau Desktop can communicate with Redshift clusters remotely.

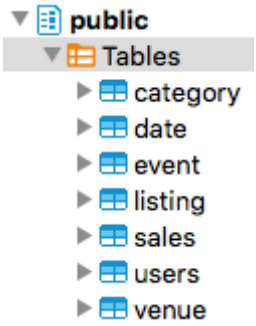
Loading sample data into the Redshift cluster

We should load sample data into Redshift to demonstrate how Tableau Desktop will connect to a huge dataset and query it.

How to do it...

To load data into the Redshift cluster, we should use Amazon S3 buckets, which consist of folders with files. We will use AWS samples and utilize the `COPY` command to load data into a cluster, as follows:

1. Copy and paste the SQL code from the `Create_Statement_Redshift.sql` file that is available for this lab.
2. Run these statements and the tables should be created, as seen in the following screenshot:



3. Then, we should load the data using the `copy` command. We will copy the commands from the `COPY` data to the `Redshift.txt` file and insert our ARN for each statement. Let's look at my example, in the following code block:

```
copy dwdate from's3://awssampledbuswest2/ssbgz/dwdate'
credentials 'aws_iam_role=arn:aws:iam::615381814665:role/RedshiftS3Access'
gzip compupdate off region 'us-west-2';
```

4. Run some sample queries to see a number of rows, as shown in the following screenshot:

```
select count(*) from customer; --3000000 rows
select count(*) from dwdate; --2556 rows
select count(*) from lineorder; --600037902 rows
select count(*) from part; --1400000 rows
select count(*) from supplier; --1000000 rows
```

The biggest table has 600 million rows, which is a huge amount of rows. The question is, is this big data or not? You can answer this question for yourself.

How it works...

Using the `COPY` command, we were able to load 600 million rows in seconds.

There's more...

The `COPY` command is a game changer for your data ingesting processes because it can instantly load big volumes of data into Redshift using a bulk methodology. You can learn about the `COPY` command here:

https://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html

Connecting Redshift with Tableau

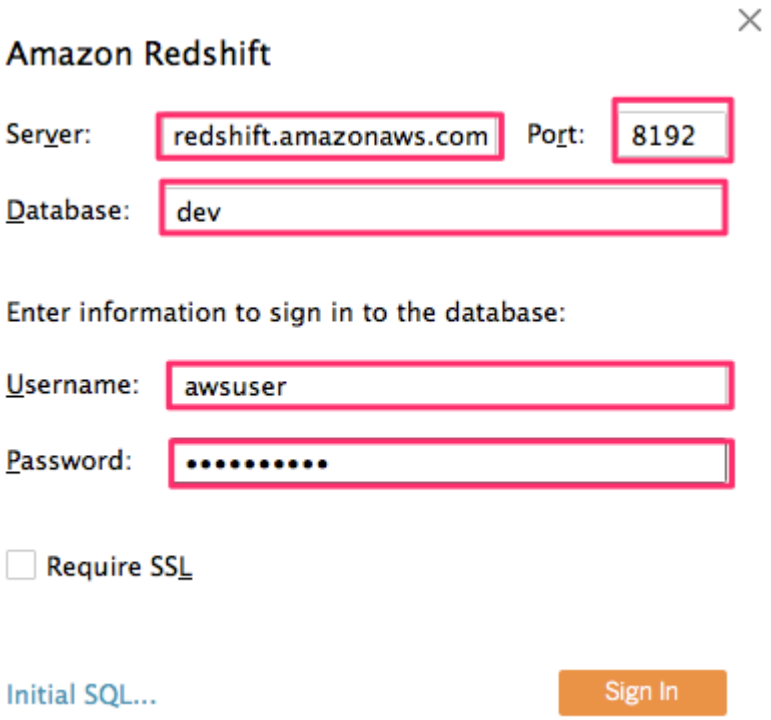
This is the main part of this lab. We are going to connect our huge sample dataset with Tableau Desktop. Moreover, we will tune Redshift to get the best possible performance from Tableau.

How to do it...

During this part of this lab, we will measure our performance based on the following three metrics:

- Load time
- Storage use
- Query performance Let's see how it is done by performing the following steps:

1. Open Tableau Desktop and click on **`Connect to Amazon Redshift`**. Fill in the credentials and click on **`Sign In`**:



Basically, we have now connected to Amazon Redshift. If you get an error, you should check your Firewall settings and make sure that you can connect with a SQL client to the cluster.

Now, we can choose the tables and create a Tableau data source. However, this isn't a true case. We need to make sure that we get the best possible performance. As a result, we need to learn more about our data usage, that is, SQL queries, patterns, tables, and joins.

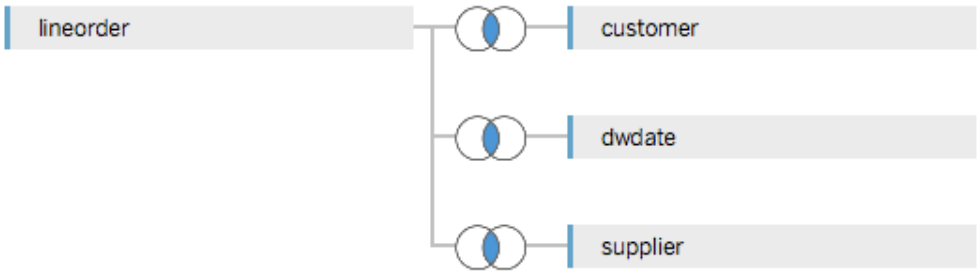
2. We should turn off Amazon Redshift caches to make accurate performance comparisons. In the following code block, we will run the following command:

```
set enable_result_cache_for_session to off;
```

3. Let's finish our Tableau Data Source. We should use the following tables:

- `customer`
- `lineorder`
- `supplier`
- `dwdate`

The following screenshot depicts the preceding listed tables:



There are joins too, as shown in the following list:

- `lineorder.lo_custkey = customer.c_custkey`
- `lineorder.lo_suppkey = supplier.s_suppkey`
- `lineorder.lo_orderdate = dwdate.d_datekey`

4. Finally, go to **`Sheet 1`**, and you will see that our data source has been created and is ready to use.

How it works...

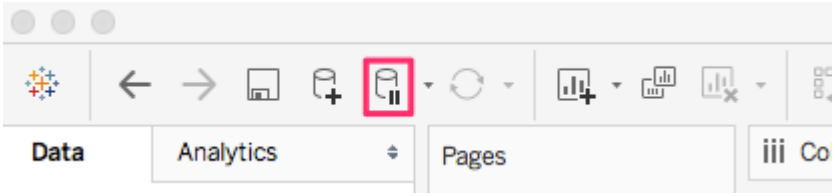
When Tableau Desktop establishes a live connection, it acts in the same way as the SQL client. The difference is that Tableau Desktop will generate an SQL query based on our Tableau Data Source and query Redshift every time we have updated the Tableau View by dragging and dropping the new object or changing filters.

Creating a Tableau report

Let's ask questions with Tableau. For example, we want to know the revenue in specific cities (`UNITED KI5` , `UNITED KI1`) in December 1997. Moreover, we want to order the result by revenue amount in descending order.

How to do it...

1. We are using a live connection, which means that each interaction with a report will generate an SQL query and we should wait. To avoid this, we will pause Tableau Auto Updates, as shown in the following screenshot:



2. Now, we can craft our report by dragging and dropping **`Dimensions`** and **`Measures`** to the canvas, as follows:

3. You should add the following filters:

- **`C City`**: `UNITED KI1` or `UNITED KI5`
- **`S City`**: `UNITED KI1` or `UNITED KI5`
- **`D YEAR`**: `Dec1997` Moreover, we should convert **`D Year`** into discrete and sort by
- *`Revenue`* *.

4. We should unpause auto updates and run our query. It takes ~18 seconds for my cluster. If we missed the time, we can log in to the **`AWS Console`** and navigate to the **`Redshift cluster`** | **`Queries Tab`** and see all executed queries, their time, and plan. You might see that, in the case of Tableau, we don't have actual queries; instead, we have something like **`fetch 10000 in SQL_CUR7`** . Tableau is using cursors and we can run the following query to see queries for currently active cursors:

```
SELECT
   usr.usename AS username
  , min(cur.starttime) AS start_time
  , DATEDIFF(second, min(cur.starttime), getdate()) AS run_time
  , min(cur.row_count) AS row_count
  , min(cur.fetched_rows) AS fetched_rows
  , listagg(util_text.text)
    WITHIN GROUP (ORDER BY sequence) AS query
FROM STV_ACTIVE_CURSORS cur
  JOIN stl_utilitytext util_text
    ON cur.pid = util_text.pid AND cur.xid = util_text.xid
  JOIN pg_user usr
    ON usr.usesysid = cur.userid
GROUP BY usr.usename, util_text.xid;
```

5. Let's run one more system query that will help us to identify the size of tables we are using in our Tableau data source, as follows:

```
select stv_tbl_perm.name as table, count(*) as mb from stv_blocklist, stv_tbl_perm
where stv_blocklist.tbl = stv_tbl_perm.id
and stv_blocklist.slice = stv_tbl_perm.slice and stv_tbl_perm.name in
('lineorder','part','dwdate','supplier')
group by stv_tbl_perm.name
order by 1 asc;
```

We can notice the size of the tables:

- `lineorder` = `34311mb`
- `part` = `92mb`
- `dwdate` = `80mb`
- `supplier` = `76mb` Finally, we can tune our cluster to improve the performance of the queries.

How it works...

We've built a new report and analyzed the SQL query that was generated by Tableau Desktop. In addition, we measured the performance and size of the tables.

There's more...

You can learn more about Redshift cursors at the following URL:

https://docs.aws.amazon.com/redshift/latest/dg/declare.html

Tuning Redshift for efficient Tableau performance

Every big data platform has special settings for tuning. For example, in our case, Redshift we will focus on the following three options:

- Sort keys
- Dist keys
- Compression You can learn more about Redshift tuning here:

https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-techniques-for-amazon-redshift/

How to do it..

We will adjust our tables to get the maximum from Redshift in terms of a place for computing, and, allow Tableau to efficiently render the results and visualize it as follows:

1. To choose the right sort key, we should evaluate our queries to find a date column that we are using for filters (the `WHERE` condition in SQL). For our huge fact table, it is the `lo_orderdate` column. For the remaining dimension tables, we will use their primary key as a sort key: `p_partkey`, `s_supkey`, `d_datekey`.
2. Then, we will choose candidates for the sort key. The following are the three types of distribution available in Redshift:

- The key distribution
- The all distribution
- The even distribution You can learn more about Redshift distribution at the following URL:

https://docs.aws.amazon.com/redshift/latest/dg/c_best-practices-best-dist-key.html

3. To find the best distribution style, we need to analyze the SQL query that is generated by Tableau and executed by Redshift. We should use the preceding query to extract the SQL from the cursor. Then, we should look to the execution plan by running our query with the word `EXPLAIN`, as follows:

```
explain
SELECT "customer"."c_city" AS "c_city", "dwdate"."d_year" AS "d_year",
"supplier"."s_city" AS "s_city",
SUM("lineorder"."lo_revenue") AS "sum_lo_revenue_ok"
FROM "public"."lineorder" "lineorder"
INNER JOIN "public"."customer" "customer" ON ("lineorder"."lo_custkey" =
"customer"."c_custkey")
INNER JOIN "public"."supplier" "supplier" ON ("lineorder"."lo_suppkey" =
"supplier"."s_suppkey")
INNER JOIN "public"."dwdate" "dwdate" ON ("lineorder"."lo_orderdate" =
"dwdate"."d_datekey")
WHERE (("customer"."c_city" IN ('UNITED KI1', 'UNITED KI5'))
AND (("supplier"."s_city" IN ('UNITED KI1', 'UNITED KI5'))
AND ("dwdate"."d_yearmonth" IN ('Dec1997'))))
GROUP BY 1, 2, 3
```

We will get the plan as follows:

	QUERY PLAN
1	XN HashAggregate (cost=8838907488.18..8838907488.19 rows=1 width=36)
2	-> XN Hash Join DS_BCAST_INNER (cost=60110.78..8838907483.69 rows=449 width=36)
3	Hash Cond: ("outer".lo_orderdate = "inner".d_datekey)
4	-> XN Hash Join DS_BCAST_INNER (cost=60078.75..8833946984.65 rows=37002 width=36)
5	Hash Cond: ("outer".lo_custkey = "inner".c_custkey)
6	-> XN Hash Join DS_BCAST_INNER (cost=15019.67..2216602842.55 rows=4697040 width=26)
7	Hash Cond: ("outer".lo_suppkey = "inner".s_suppkey)
8	-> XN Seq Scan on lineorder (cost=0.00..6000378.88 rows=600037888 width=16)
9	-> XN Hash (cost=15000.00..15000.00 rows=7868 width=18)
10	-> XN Seq Scan on supplier (cost=0.00..15000.00 rows=7868 width=18)
11	Filter: (((s_city)::text = 'UNITED KI1'::text) OR ((s_city)::text = 'UNITED KI5'::text))
12	-> XN Hash (cost=45000.00..45000.00 rows=23633 width=18)
13	-> XN Seq Scan on customer (cost=0.00..45000.00 rows=23633 width=18)
14	Filter: (((c_city)::text = 'UNITED KI1'::text) OR ((c_city)::text = 'UNITED KI5'::text))
15	-> XN Hash (cost=31.95..31.95 rows=31 width=8)
16	-> XN Seq Scan on dwdate (cost=0.00..31.95 rows=31 width=8)
17	Filter: ((d_yearmonth)::text = 'Dec1997'::text)

I've highlighted `BS_BCAST_INNER`. It means that the inner join was broadcasted across all slices. We should eliminate any broadcast and distribution steps. You can learn more about query patterns here:

https://docs.aws.amazon.com/redshift/latest/dg/t_evaluating_query_patterns.html

In our case, we should look at the join between the fact table with 600 mln and dimension tables. Based on fairly small rows in the dimension tables, we can distribute the dimension tables `SUPPLIER`, `PART`, and `DWDATE` across all nodes. For the `LINEORDER` table, we will use `lo_custkey` as a distribution key and for the `CUSTOMER` table, we will use the `c_custkey` as the distribution key.

4. Next, we should compress our data to make sure that we can reduce storage space, and also, reduce the size of the data that is read from storage. It decreases I/O and improves query performance. By default, all data is uncompressed. You can learn more about compression encodings here: https://docs.aws.amazon.com/redshift/latest/dg/c_Compression_encodings.html. We should use system tables in order to research the best compression encoding. Let's run the following query:

```
select col, max(blocknum)
from stv_blocklist b, stv_tbl_perm p
where (b.tbl=p.id) and name ='lineorder'
and col < 17
group by name, col
order by col;
```

It will show us the highest block number for each column in the `LINEORDER` table. Then, we can start to experiment with different encoding types in order to find the best. In addition, we should always analyze the table after changes in order to update table statistics. However, in our case, we can simply execute the `COPY` command with the auto compression parameter.

5. Let's apply the changes to our tables and run the same report. Copy the queries from the `Create_Statementv2_Redshift.sql` [** **]file and run them.
6. Then, we should reload data with autocompression. Run SQL from the `COPY` data to `Redshiftv2.txt` file. Don't forget to insert your ARN.
7. Let's refresh our Tableau workbook and see the improvements. Moreover, we can check the query plan and see the changes. In my case, it took 8 seconds.

How it works...

As you can see, the secret of working with big volumes of data is in good design patterns of the Data Platform. In the case of Redshift, we analyze the data structure, volume, and query patterns, and then apply the best practices of using sort keys, distribution styles, and compression encoding. It is obvious that we are depending on query patterns, and we can keep high performance for any queries. As a result, we should be very careful with the preceding options.

There's more...

It is important for cost-based optimization databases such as Redshift to keep table statistics up-to-date in case of any changes or updates by using the `ANALYZE` and `VACUUM` commands.

Connecting to Amazon Redshift Spectrum

In this section, we will upgrade our Redshift data warehouse by enabling Redshift Spectrum, which plays the role of a data lake and compliments our data warehouse. It gives us a powerful and serverless architecture and can handle a tremendous volume of data, which is true big data.

Amazon Spectrum extends Redshift data warehouse out to the Exabytes website. You can get all the benefits of open data formats and inexpensive storage, and we can easily scale out to thousands of nodes. Another benefit is the cost; we pay only for usage and storage using S3, which is fairly small in comparison with an analytical data warehouse.

Getting ready

We should update our IAM roles for Redshift by adding the following additional policies:

- **`AmazonS3FullAccess`**
- **`AmazonAthenaFullAccess`**
- **`AWSGlueConsoleFullAccess`** Spectrum will use the Glue or Athena data catalog. It is important to have the data in the region as our Redshift cluster. That's why we are going to use the `UNLOAD` command: so that we have the data in the same network with our Redshift. In real life, we can query data from other VPC in the same region.

How to do it...

1. Let's create an S3 bucket for our data. Go to **`S3`** and click on **`Create new bucket`**. Type the name `cookbook-spectrum`.
2. Now, we can unload data into this bucket and run this command, as shown in the following code block:

```
unload ('select * from lineorder')
to 's3://cookbook-spectrum/
iam_role 'arn:aws:iam::615381814665:role/RedshiftS3Access'
delimiter '\t'
```

3. Let's create the external schema, as follows:

```
create external schema datalake
from data catalog
database 'spectrumdb'
iam_role 'arn:aws:iam::615381814665:role/RedshiftS3Access'
create external database if not exists;
```

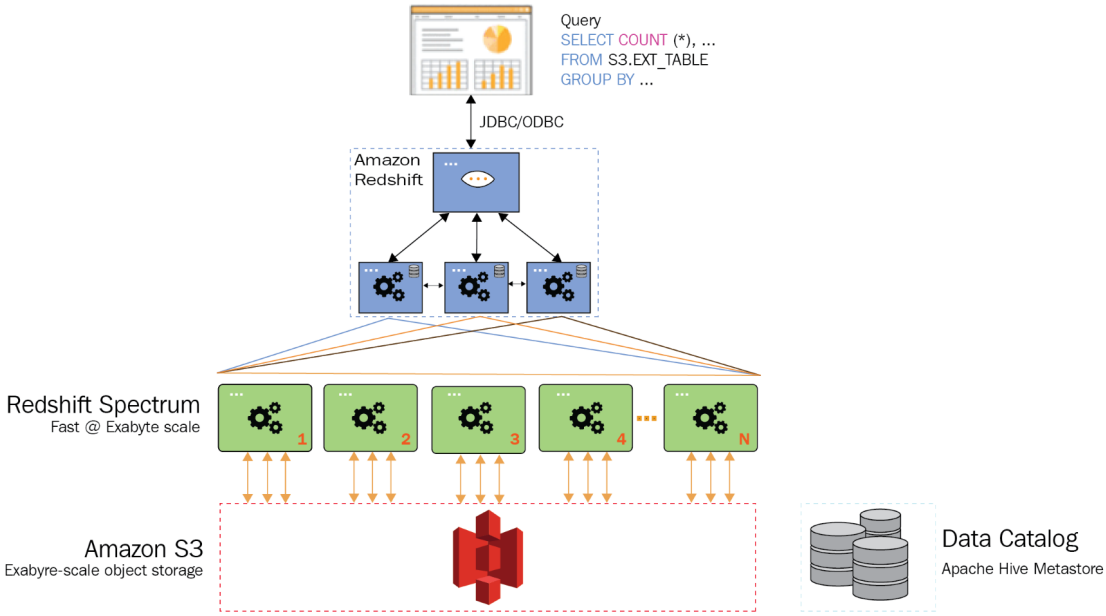
4. Let's create an external table in this schema, as follows:

```
create external table datalake.lineorder
(
  lo_orderkey INTEGER
  ,lo_linenumber INTEGER
  ,lo_custkey INTEGER
  ,lo_partkey INTEGER
  ,lo_suppkey INTEGER
  ,lo_orderdate INTEGER
  ,lo_orderpriority VARCHAR(15)
  ,lo_shippriority VARCHAR(1)
  ,lo_quantity INTEGER
  ,lo_extendedprice INTEGER
  ,lo_ordertotalprice INTEGER
  ,lo_discount INTEGER
  ,lo_revenue INTEGER
  ,lo_supplycost INTEGER
  ,lo_tax INTEGER
  ,lo_commitdate INTEGER
  ,lo_shipmode VARCHAR(10)
)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://cookbook-spectrum/
table properties ('numRows'='172000');
```

5. We can test this table by running queries against it, such as `SELECT * FROM datalake.lineorder`, or we can adjust our Tableau Data Source and use a Spectrum table instead of the initial one. However, it is better to tune the external table before use, otherwise Spectrum will scan the full table.

How it works...

The main benefit of Spectrum for the end user is that there are no changes; they can still use SQL, and query the same tables as well, for BI or ETL use. The following diagram demonstrates the life of a Spectrum query:



There's more...

By default, Spectrum will scan all your rows, which could become expensive. There is a good explanation about improving Spectrum performance at the following link:

https://docs.aws.amazon.com/redshift/latest/dg/c-spectrum-external-performance.html

We can learn more about the usage of Amazon Redshift Spectrum with this article: [*10 Best Practices for Amazon Redshift Spectrum*], which can be found at https://aws.amazon.com/blogs/big-data/10-best-practices-for-amazon-redshift-spectrum/.

Moreover, you can learn about Glue at https://docs.aws.amazon.com/glue/latest/dg/what-is-glue.html and Amazon Athena at https://docs.aws.amazon.com/athena/latest/ug/what-is.html.

Finally, you can drop your cluster if you no longer need it.

Connecting to Snowflake

Snowflake is the leading cloud data warehouse platform. Moreover, it is the first data warehouse that was created for the cloud. In this section, we will launch a Snowflake instance and connect to it with Tableau. Currently, Snowflake is available for AWS and Azure. We are going to use the AWS version of Snowflake.

Snowflake has a multi-cluster shared architecture, and it separates storage and computes. This allows it to easily scale up and down on the fly without any disruption.

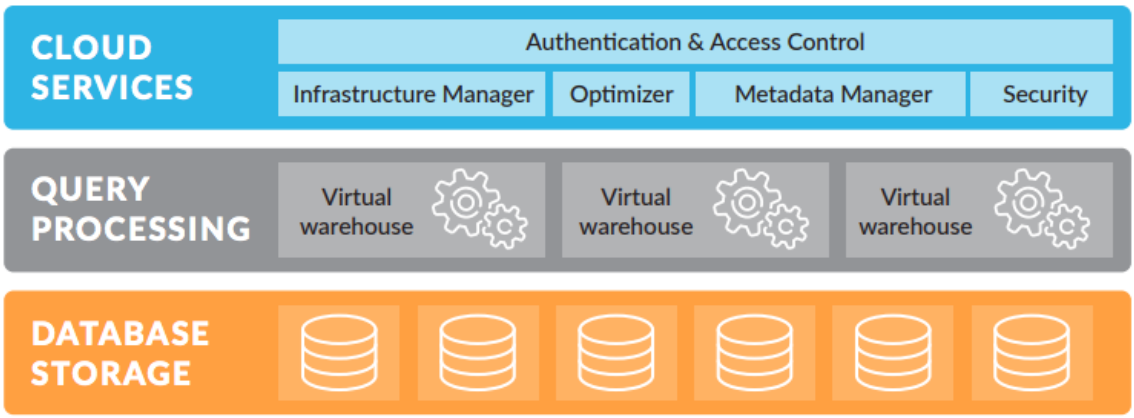
Snowflake is an SQL data warehouse that supports structured and semi-structured data, such as JSON, AVRO, or XML.

The Snowflake architecture consists of the following three layers:

- [**Cloud services**]: This is a collection of services such as authentication, and so on
- [**Query processing**]: Snowflake executes queries using a virtual warehouse, for example, an MPP cluster with multiple compute nodes using Amazon EC2 or Azure virtual machines

- [**Database storage**]: Snowflake stores optimized data in Amazon S3 or Azure Blob Storage

Let's take a look at the following diagram that depicts the preceding mentioned layers:



There is the same principle when we allow Snowflake to do the heavy lifting and then, Tableau renders the results. As a result, we don't need to use Tableau extracts.

With a live connection, we should avoid a custom SQL data source because often it is counterproductive, and could be run multiple times on the dashboard. Tableau itself can generate much more efficient queries.

Getting ready

We should start the free trial of Snowflake. We can choose either AWS or Azure. In my case, I will use AWS. After that, we will launch the cluster, and then, we can load sample data and connect the Tableau Desktop using a built-in Snowflake connector.

How to do it...

Let's begin with logging into Snowflake, and carrying out the following steps:

1. Go to the Snowflake, https://www.snowflake.com/, and click on `START FOR FREE`. Fill in the form and you'll get an email about Snowflake.
2. You will get an email with the link. Click the link and you will create the user and password. Then, you will see the worksheet of Snowflake. In my case, it was `https://<account name>.snowflakecomputing.com`.
3. After successful activation, you can go to `https://<account name>.snowflakecomputing.com`, where you can find a web-based GUI where you can create and manage all Snowflake objects. You can learn more here: https://docs.snowflake.net/manuals/user-guide/snowflake-manager.html.

How it works...

- Snowflake provides a free trial and gives you 400 credits. You can learn more about Snowflake Trial at https://www.snowflake.com/trial_faqs/

Using SnowSQL CLI

One of the common ways of interacting with the database is CLI. Let's do a quick exercise to load data into the Snowflake and then connect Tableau.

How to do it...

1. At the GUI, you can click on `Help` and choose `Download`.
2. Download the CLI client (`snowsql`) for your OS. In my case, I am using macOS.

3. Install the file.
4. Connect the Snowflake instance. In our case, we are using macOS. Open a command line and run the following command:

```
/Users/anoshind/Applications/SnowSQL.app/Contents/MacOS/snowsql -a <account name> -u
<user name>
```

In our case, we have `-a DZ27900 -u tableaucookbook`. It will ask for a password. After successful authentication, we connected to Snowflake.

5. Now, we can start to create Snowflake objects:

   a. Run this command to create a database:

```
create or replace database sf_tuts;
```

- 2. Run this command to create a target table:

```
create or replace table emp_basic (
first_name string, last_name string, email string, streetaddress string,  city string
,  start_date date );
```

- 3. Run this command to create our computing DW. Depending on the size, you can choose the power of your cluster:

```
create or replace warehouse sf_tuts_wh with
 warehouse_size='X-SMALL'   auto_suspend = 180   auto_resume = true
initially_suspended=true;
```

You can learn more about pricing options at:

https://docs.snowflake.net/manuals/user-guide/credits.html

4. Next, load the files into Snowflake. You should download files from the lab bundle (five `.xls` files) and put them into any directory, where `snowsql` can access it. In our case, we will copy them into `/tmp`.
5. Put file: `///<path>/employees0*.csv @sf_tuts.public.%emp_basic;`.
6. The Windows syntax will be different: put file `//C:\<path>\employees0*.csv @sf_tuts.public.%emp_basic;`.
7. This operation will upload and compress (`.gzip`) files into the stage for our table, `emp_basic`.
8. Now, we can copy data into the target table by executing the following command:

```
copy into emp_basic
  from @%emp_basic
  file_format = (type = csv field_optionally_enclosed_by='"')
  pattern = '.*employees0[1-5].csv.gz'
  on_error = 'skip_file';
```

We can see the output in the following screenshot:

```
tableaucookbook#SF_TUTS_WH@SF_TUTS.PUBLIC>copy into emp_basic
                                          from @%emp_basic
                                          file_format = (type = csv field_optionally_enclosed_by='"')
                                          pattern = '.*employees0[1-5].csv.gz'
                                          on_error = 'skip_file';
+--------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
| file               | status | rows_parsed | rows_loaded | error_limit | errors_seen | first_error | first_error_line | first_error_character | first_error_column_name |
|--------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------|
| employees03.csv.gz | LOADED |           5 |           5 |           1 |           0 | NULL        |             NULL |               NULL | NULL                     |
| employees05.csv.gz | LOADED |           5 |           5 |           1 |           0 | NULL        |             NULL |               NULL | NULL                     |
| employees02.csv.gz | LOADED |           5 |           5 |           1 |           0 | NULL        |             NULL |               NULL | NULL                     |
| employees01.csv.gz | LOADED |           5 |           5 |           1 |           0 | NULL        |             NULL |               NULL | NULL                     |
| employees04.csv.gz | LOADED |           5 |           5 |           1 |           0 | NULL        |             NULL |               NULL | NULL                     |
+--------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
5 Row(s) produced. Time Elapsed: 3.128s
```

As a result, we successfully loaded data from the local machine. Moreover, we can load huge datasets from S3 into Snowflake or other source systems.

How it works...

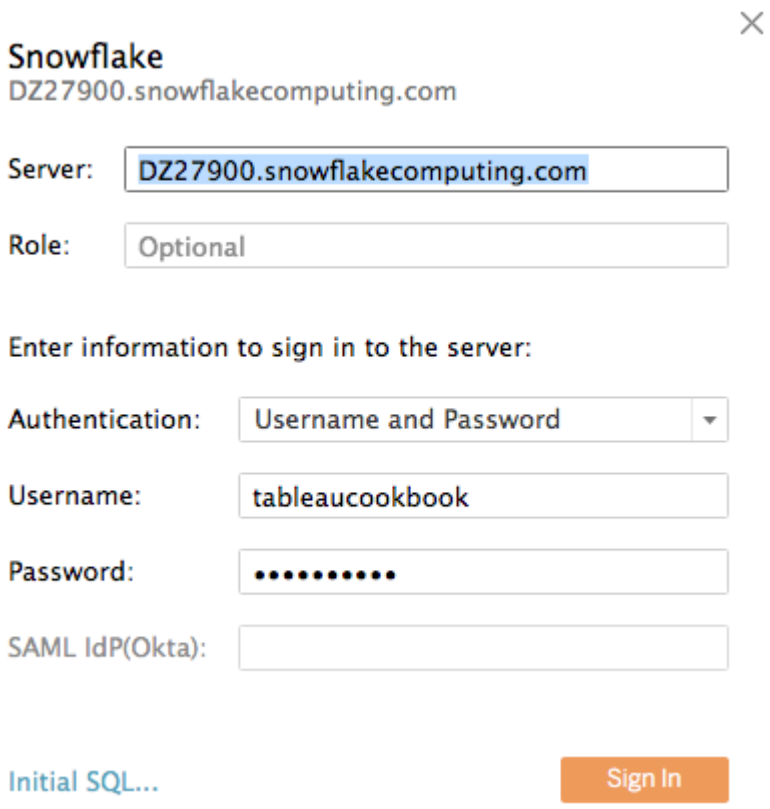
We used internal Snowflake storage in order to upload data into Snowflake and we used the Snowflake CLI tool to work with the Snowflake cluster remotely.

Connecting Tableau to Snowflake

We can connect Snowflake with Tableau in the same way as any other data warehouse. Let's do this!

How to do it...

1. First, we need to download and install the Snowflake ODBC Driver. We can download it from the same place that we downloaded `snowsql`.
2. After a successful installation, we create a new data source by filling in the connection details, as follows:



3. Then, we can choose the following:

- **Warehouse**: `SF_TUTS_WH`
- **Database**: `SF_TUTS`
- **Schema**: `PUBLIC`

As usual, we can drag and drop the table and see the data.

How it works...

It works in the same way as we connect Amazon Redshift or any other database by using the specific driver and our credentials. You can learn about other ways of connecting Snowflake here:

https://docs.snowflake.net/manuals/user-guide-connecting.html

Connecting big data

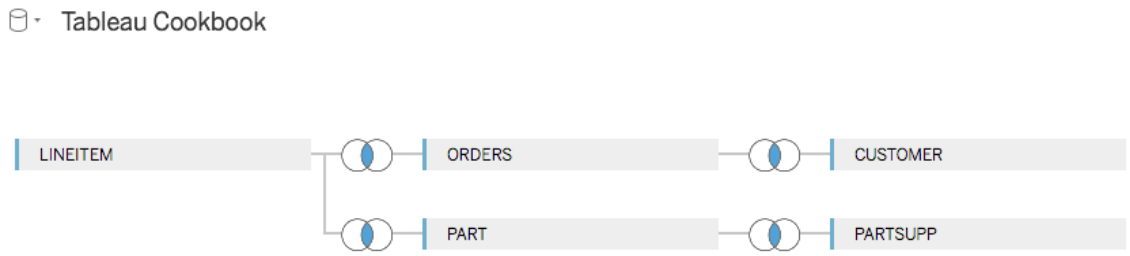
Now, we can try to query a bigger dataset and see how Tableau will work with big data. We shouldn't avoid Tableau Extracts, because our data size is huge. This is the great thing about the cloud. We can orchestrate with big data without loading it into our machine.

How to do it...

1. We can create a new data source and specify the following options:
   - **Warehouse**: `COMPUTE_WH`
   - **Database**: `SNOWFLAKE_SAMPLE_DATA`
   - **Schema**: `TPCH_SF1000`

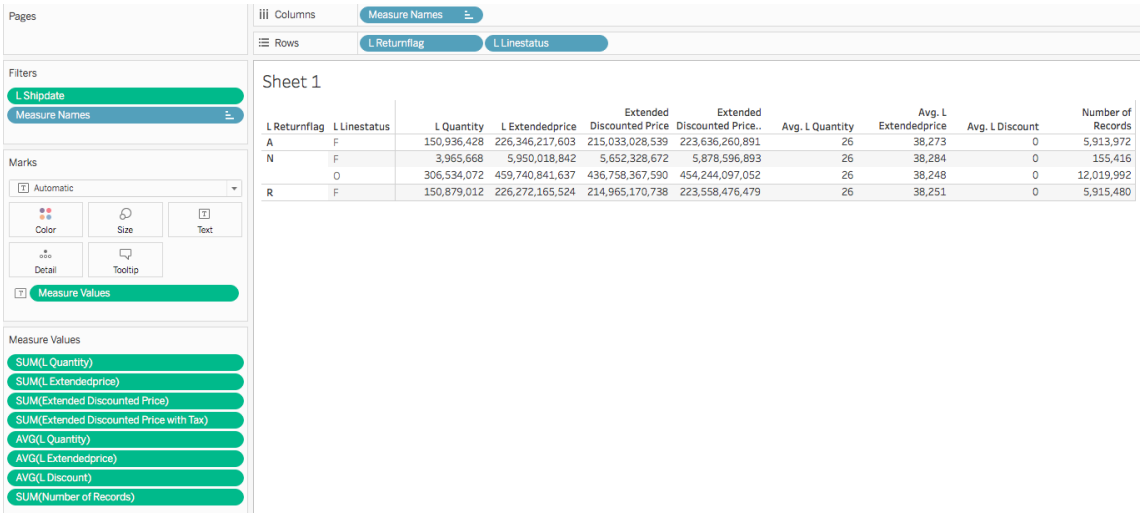
You can learn more about the TPC sample dataset at http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.5.0.pdf.

2. When we are choosing **Warehouse**, we are choosing to compute resources for our DW. In our case, `COMPUTE_WH` is a `X-Large`. We also have `SF_TUTS_WH` that is `X-Small`.
3. Build the data model, as shown in the following screenshot:



4. Let's answer the following business question: List the totals for the extended price, discounted extended price plus tax, average quantity, average discount, and average extended price, and group the result by **Return Flag** and **Line Status**. Using Tableau, we can drag and drop the following object. Do not forget to pause Auto Updates. In addition, we should create the following calculation fields:
   - **Extended Discounted Price**: `[L Extendedprice]*(1-[L Discount])`
   - **Extended Discounted Price with Tax**: `[L Extendedprice]*(1-[L Discount])*(1+[L Tax])`

Here is the screenshot showing the end result:



L Returnflag	L Linestatus	L Quantity	L Extendedprice	Extended Discounted Price	Extended Discounted Price..	Avg. L Quantity	Avg. L Extendedprice	Avg. L Discount	Number of Records
A	F	150,936,428	226,346,217,603	215,033,028,539	223,636,260,891	26	38,273	0	5,913,972
N	F	3,965,668	5,950,018,842	5,652,328,672	5,878,596,893	26	38,284	0	155,416
	O	306,534,072	459,740,841,637	436,758,367,590	454,244,097,052	26	38,248	0	12,019,992
R	F	150,879,012	226,272,165,524	214,965,170,738	223,558,476,479	26	38,251	0	5,915,480

5. We can check with Snowflake GUI about the actual plan by going to **`History`** and finding our **`Query`**. Because we are using a live connection, Tableau generates a query and uses only one table, `LINEITEM`, instead of the whole data model, which saves us time and money.

How it works...

We've created new Tableau data source based on Snowflake sample data. In addition, we had to specify a Virtual Warehouse (our horsepower) and source dataset.

There's more...

We can use Tableau Initial SQL to create temporary or aggregation tables. This can significantly improve performance in case if we are using custom SQL logic or want to use aggregate tables by materializing the result into a temp table.

Accessing semi--structured data

In the modern world, often we can meet unstructured data that is generating by machines, apps, sensors, and so on. There are the following two main attributes of semi-structured data that differ it from structure data:

- Semi-structured data can contain n-level hierarchies of nested information
- Structured data always needs a defined schema before loading it. Semi-structured data doesn't need this, so as a result we can create the schema on the fly Despite the fact that Tableau supports direct connection to the JSON format, we still have the same issue with big data, when we need more compute resource than Tableau allows us to use and also, we can collect data types such as Avro, ORC, Parquet, and XML.

Usually, we should parse unstructured data and write into the table. But not with Snowflake; it has a special data type `VARIANT` that allows us to store semi-structured data. Moreover, we can easily parse key-value pairs. You can run the following SQL and check how it looks:

```
select * from "SNOWFLAKE_SAMPLE_DATA"."WEATHER"."WEATHER_14_TOTAL" limit 1
```

Let's try to use the Snowflake sample database in order to see how it looks. Unfortunately, Tableau can't parse `VARIANT` data type, which is why we should create the SQL for Tableau based on a `VARIANT` column.

How to do it...

Let's create a Tableau workbook based on semi-structured data. At the Snowflake sample database we have a table `HOURLY_16_TOTAL` with 392 mln rows and 322 GB. This dataset keeps the last four days of hourly weather forecasts for 20,000+ cities.

1. Create a new Tableau data source with a Snowflake connection:
    - **`Warehouse`** : `COMPUTE_WH`
    - **`Database`** : `SNOWFLAKE_SAMPLE_DATA`
    - **`Schema`** : `WEATHER`
2. Drag and drop custom SQL element and use this query, that will parse the `VARIANT` data type, as follows:

```
select (V:main.temp_max - 273.15)::FLOAT * 1.8000 + 32.00 as temp_max_far,
(V:main.temp_min - 273.15)::FLOAT * 1.8000 + 32.00 as temp_min_far,
  cast(V:time as timestamp) time,
       V:city.coord.lat::FLOAT lat,
       V:city.coord.lon::FLOAT lon,
       v:city.name::VARCHAR,
       v:city.country::VARCHAR,
      v:city:id,
      V
```

From `"SNOWFLAKE_SAMPLE_DATA"."WEATHER"."HOURLY_16_TOTAL"` you can use another table that is much smaller, such as `DAILY_14_TOTAL` (37 mln rows). In addition, you can always specify `LIMIT` at the end of the query in order to limit you output.

3. Then we can build the dashboard in order to see the minimum temperature in Fahrenheit across the world, as follows:

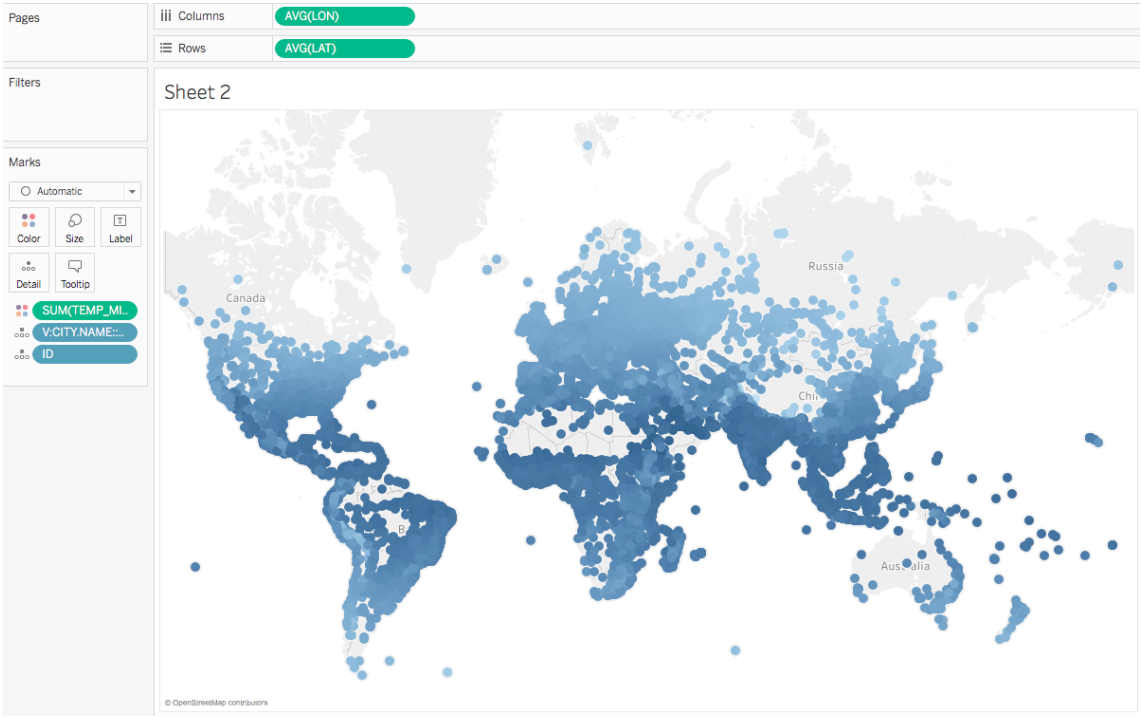


Tableau has special function pass-through functions that can send SQL expressions to the database, without being

interpreted by Tableau. In our case with weather data, we can create a new calculated field `WIND SPEED`: `RAWSQL_REAL("v:wind.speed::float",[V])` and it will add wind speed to our dataset.

How it works...

Tableau on top of Snowflake is fast because it is utilizing Snowflake's computing power and just renders the result in Tableau.

There's more...

According to the best practices, if our semi-structured data schema is static, we can parse it and load it into the Snowflake tables. We can use SQL queries from the Snowflake GUI or SnowSQL CLI, or we can use the ELT tool such as Matillion ETL. You can learn more about Snowflake and Matillion ETL here:

https://www.matillion.com/etl-for-snowflake/

Connecting Amazon Elastic MapReduce with Apache Hive

[**Amazon Elastic MapReduce**] ([**EMR**]) is a true big data platform. It provides a managed Hadoop framework that can process a huge volume of data across dynamically scalable Amazon EC2 instances. It allows us to run popular distributed frameworks with it, such as Presto, Hive, Spark, and others. This is a short video about Amazon EMR: https://youtu.be/S6Ja55n-o0M.

Organizations use Hadoop for big data use cases, such as clickstream analysis, log analysis, predictive analytics, data transformation, data lake, and many more. Often, business users need to work with raw data that is stored in Hadoop, in our case Amazon EMR. There are multiple ways to connect Hadoop with Tableau, such as Presto, Hive, and more.

Getting ready

We should launch the cluster, get sample data, and connect it via Tableau Desktop using Hive. Let's do this! But be aware, EMR cluster isn't free and you might spend $20-$30 in order go through scenarios.

How to do it..

Let's create our EMR cluster.

1. Go to the AWS account: https://aws.amazon.com/console/ and sign in.
2. Follow the instructions present here: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs.html in order to launch a cluster. You should perform the following three steps first:

   a. Set up prerequisites b. Launch the cluster c. Allow SSH access

We used the following names:

Bucket name	`tableau_cookbook`
Folder name	`tableau-cookbook-query-result`
Key pair name	`tableau-cookbook.pem`
Cluster name	`My Cluster for Tableau Cookbook`
Cluster parameters	`Core Hadoop: Hadoop 2.8.4 with Ganglia 3.7.2, Hive 2.3.3, Hue 4.2.0, Mahout 0.13.0, Pig 0.17.0, and Tez 0.8.4`

As a result, we should be able to successfully launch the EMR cluster. In addition, we can connect our cluster via SSH using the `tableau-cookbook.pem` key.

4. Let's connect the cluster by running the following command:

```
ssh -i ~/.ssh/tableua-cookbook.pem hadoop@ec2-18-215-157-216.compute-1.amazonaws.com
```

Note

I saved my key in the `.ssh` folder and the key should have 400 permissions. In the following screenshot we can see the following output:

```
|f45c89c859a9:.ssh anoshind$ ssh -i ~/.ssh/tableua-cookbook.pem hadoop@ec2-18-215-157-216.compute-1.amazonaws.com
Last login: Sat Oct 13 16:46:34 2018

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
15 package(s) needed for security, out of 28 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R::::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-17-64 ~]$
```

How it works...

Using AWS, we launched EMR cluster. In addition, we accessed it with SSH using key. You can learn more about EMR in AWS documantation: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs.html

Creating sample data

Next, we want to create a Hive external table on top of S3 logs and use EMR to compute the results. We can do this using the following three different methods:

- Using EMR CLI
- Using EMR console

- Using web GUI

How to do it..

It depends on your preferences. In my example, I will use EMR CLI. We should already be connected to the EMR cluster via SSH. Let's start to work with Hive:

1. In EMR CLI type `hive` and it will launch Hive.
2. Next, we can execute SQL commands. Let's create the table on top of the CloudFront logs that are stored in the S3 bucket. We will run this DDL, as follows:

```
hive>CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
  DateObject Date,
  Time STRING,
  Location STRING,
  Bytes INT,
  RequestIP STRING,
  Method STRING,
  Host STRING,
  Uri STRING,
  Status INT,
  Referrer STRING,
  OS String,
  Browser String,
  BrowserVersion String
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
  "input.regex" = "^(?!#)([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^
]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+[^\(]+[\(]([^\;]+).*\%20([^\/]+)
[\/](.*)$"
) LOCATION 's3://us-east-1.elasticmapreduce.samples/cloudfront/data';
```

Note

It is important that region location is in the same region as where we have the EMR cluster, that is in my case it is `us-east-1`.

3. We can test our new table, by running the following simple query:

```
Hive>SELECT os, COUNT(*) count FROM cloudfront_logs GROUP BY os;
```

Let's look at the results in the following screenshot:

```
hive> SELECT os, COUNT(*) count FROM cloudfront_logs GROUP BY os;
Query ID = hadoop_20181013171623_9c7dfe43-9086-47c6-8e36-4ab61d56630e
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1539374666378_0006)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 25.50 s
--------------------------------------------------------------------------------
OK
Android 855
Linux   813
MacOS   852
OSX     799
Windows 883
iOS     794
Time taken: 32.262 seconds, Fetched: 6 row(s)
```

You can see that the query works. Despite the fact that it has a fairly small amount of data, the query took 32 seconds. This is how Hadoop works, it takes a long time for initialization and other complementary steps. This is true for any big data system and should work for big data. You can also experiment with other SQL-friendly Hadoop tools, such as Impala, Presto, and so on. For us, the main feature of these guys is their connection to the Tableau via the ODBC driver.

4. Let's create one more table, using the [*Google Books N-grams*] dataset. It is an AWS public dataset available for everyone. You can read more about it here: https://registry.opendata.aws/google-ngrams/. Let's create the Hive table for 1-gram, as follows:

```
hive> CREATE EXTERNAL TABLE eng_1M_1gram(token STRING, year INT, frequency INT, pages
 INT, books INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY 't' STORED AS SEQUENCEFILE
  LOCATION 's3://datasets.elasticmapreduce/ngrams/books/20090715/eng-1M/1gram';
```

As a result, we will get one more table. This table has 261,823,186 rows.

How it works...

We used Apache Hive that was deployed on top of an EMR cluster. In addition, we queried two datasets using Hive SQL.

There's more...

You can learn more about Apache Hive here: https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hive.html

Connect Tableau with Apache Hive

The fastest way to connect Tableau to AWS EMR via Hive JDBC is to open an SSH tunnel to the master node. Let's connect.

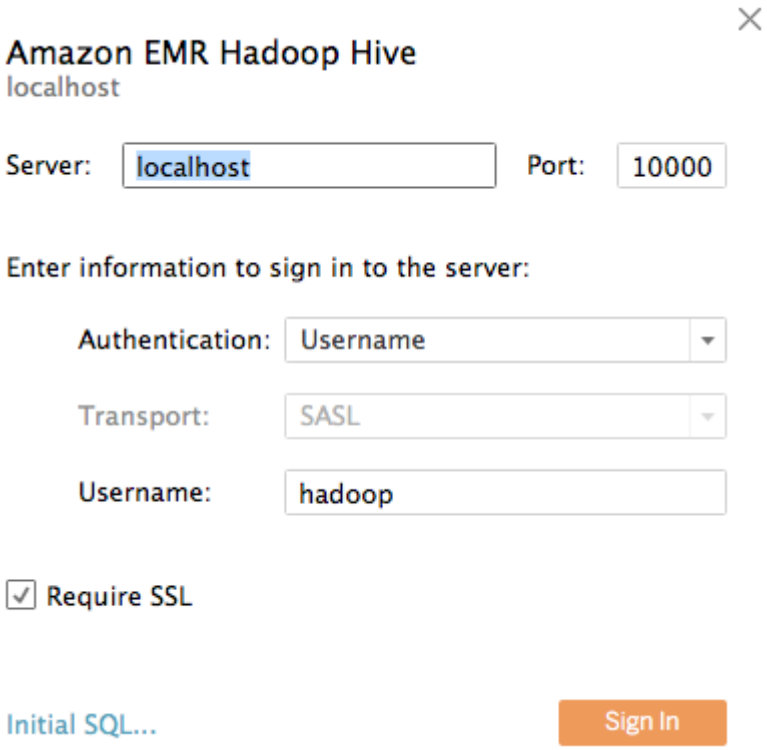
How to do it...

1. Open Terminal and run the following command:

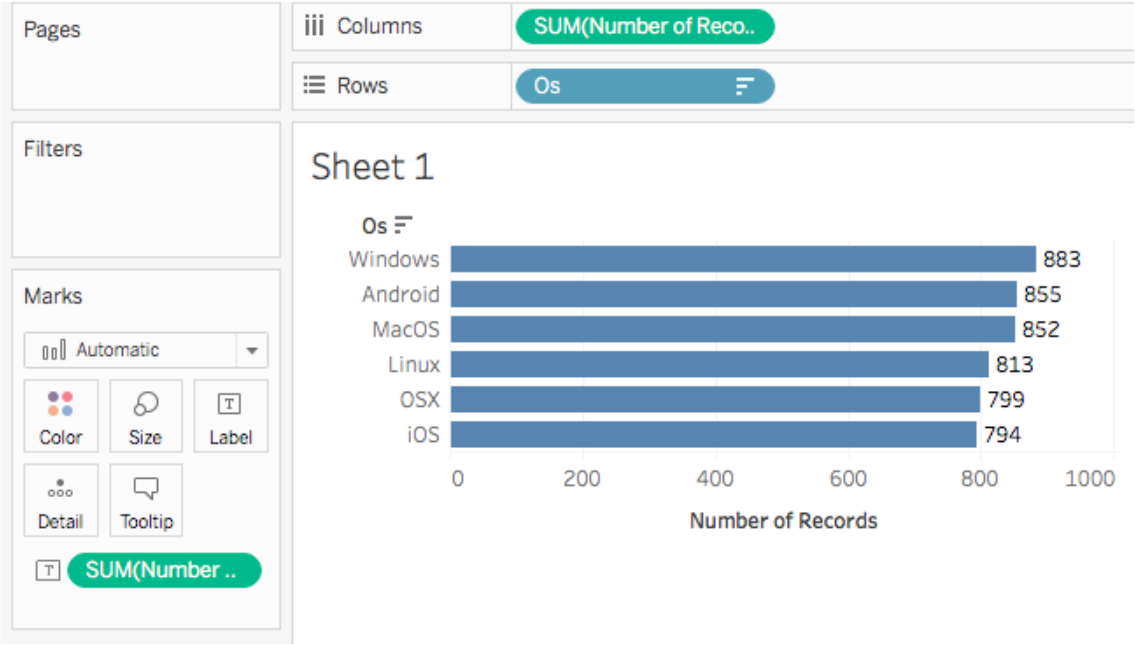
```
ssh -o ServerAliveInterval=10 -i ~/.ssh/tableau-cookbook.pem -N -L
10000:localhost:10000 hadoop@ec2-18-215-157-216.compute-1.amazonaws.com
```

But this could be a different command in your case.

2. Open Tableau Desktop and create a new connection using Amazon Hadoop EMR Hive.
3. Download and install ODBC drivers for your OS from:https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-bi-tools.html
4. Now, we can connect Hive, as follows:



5. Then choose `default` as a schema and `cloudfront_logs` as a table and go to the sheet. As you know, every drag and drop will initialize the SQL query and will trigger our EMR. In order to avoid this, we should pause Auto Updates and create our report, then run the final query, as follows:



As a result, we were able to connect to Tableau. You can connect to the bigger `eng_1M_1gram` table as well. There are lots of different techniques for accessing data from Hadoop, and often it depends on use case.

How it works...

In our scenario, we connected Tableau Desktop to EMR cluster via Apache Hive. As usual, we left all the heavy work to ERM cluster and used Apache Hive in order to get an SQL interface to the Hadoop cluster.

There's more...

You can also use Impala. It is also an open-source tool in the Hadoop ecosystem. Instead of using MapReduce, as Hive does, it is using MPP (similar to the analytical data warehouse), which gives us a better query performance but requires additional steps. Hive and Impala both use Hive Metastore, but Hive requires more time to process the query and it isn't very efficient for improvised analyses. Impala requires data to be stored on HDFS. Moreover, it depends on the memory resource of clusters. As a result, every tool has its own pros and cons. You can read more here:

https://vision.cloudera.com/impala-v-hive/