HEVO

A COMPLETE GUIDE ON REDSHIFT SPECTRUM

Table of Contents

Amazon Spectrum - Overview

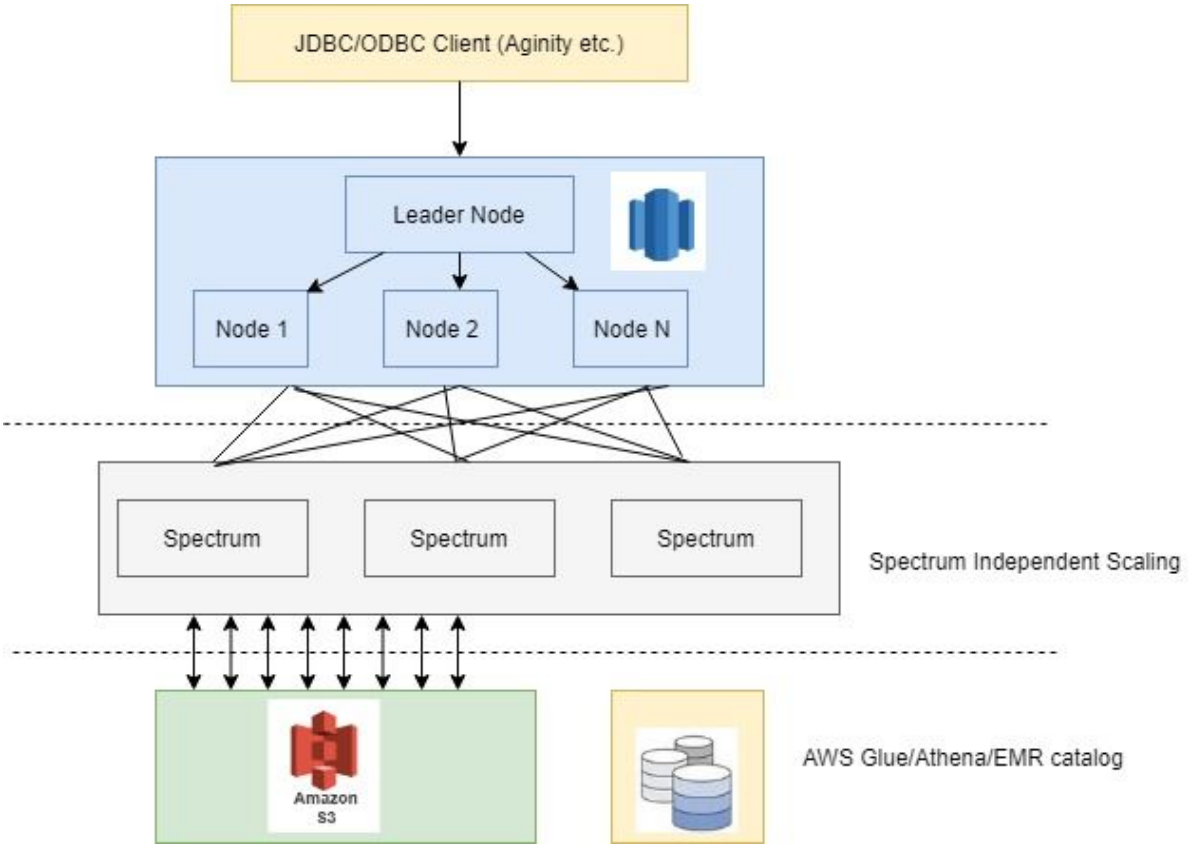
Amazon Redshift Spectrum is a thin layer which resides between Redshift cluster and S3. Redshift Spectrum data is independent of Redshift cluster but leverages data from S3. Redshift Spectrum allows processing of data without loading data into Redshift cluster (using COPY command and more) and take advantage of the power and flexibility of Amazon Redshift and S3 data. You will need to create a data source and issue your queries to the Redshift cluster. In the background, Redshift Spectrum scales thousands of instances based on query issued. The queries can refer any combination of data stored in Redshift cluster and S3 i.e. Redshift tables/views, columnar files, CSV files or S3 files of all major formats supported by Spectrum.

When you hit a query in Redshift cluster, it internally generates a query plan that minimizes the amount of S3 data that will be read, by taking advantage of Redshift data and S3 files. Redshift cluster generates final result based on the query plans generated by leader node.

In the image below, Spectrum layer is highlighted inside dotted lines between Redshift cluster and S3.

Architecture of the Data Flow in Spectrum:

Redshift client <-> Leader Nodes <-> Compute Nodes <-> **Spectrum Layer** <-> S3 <-> Data catalog



Redshift Spectrum Pricing

Spectrum pricing is based on the amount of data hit on S3 layer during query processing and is charged at the rate of $5 per terabyte. So, you do not pay anything when Redshift Spectrum query is not running. You can improve performance by compressing, partitioning, and by converting your data to a columnar format.

Redshift Spectrum Use-case

Amazon Spectrum can be leveraged when you are querying on large data sets of structured data. With Spectrum, you can store data in any format and make it available for processing when you need it. Amazon Spectrum automatically scales out to thousands of instances. It can even do processing of exabyte sized queries. Another use-case of Spectrum is to perform data visualization through Amazon Quicksight.

Creating Data Files for Queries in Amazon Redshift Spectrum

In a nutshell, Amazon Redshift Spectrum is directly reading data from S3. Here, you don't have to load data from S3 to Redshift using COPY command or by any other means. The beauty of Spectrum is that it supports various data format files which can be read from S3. Amazon Spectrum uses the same data file formats as used by other services such as Amazon Athena, Amazon EMR, Amazon Glue, etc. The supported file formats are mentioned below:

- AVRO → Avro is a compact binary format file containing data definition in JSON format.
- PARQUET → Parquet is an open source columnar format file for Hadoop.
- TEXTFILE → Textfile is a human-readable file containing a set of data definition per row/line.
- SEQUENCEFILE → Sequencefile is a data file containing data stored as key/value pair.
- RCFILE → RCfile is a record columnar file used in the Big data environment.
- RegexSerDe → It is a file using Regex serialization/deserialization methodology.
- ORC → ORC stands for Optimized Row Columnar file and is used by HIVE.
- CSV → CSV files are comma separated files
- JSON → JSON file is simple alternative format of xml containing content in name-value pair

Redshift Spectrum only supports timestamp format of type YYYY-MM-DD and HH:MM:SS:SSSSS.

Data types supported by Amazon Redshift Spectrum

Since Redshift Spectrum is an extended feature of Redshift, it supports same data formats as used in Redshift tables/views. However, there are some limitations in using DATE and TIMESTAMP data types. Please refer to the AWS documentation for complete details.

Data catalog for Redshift Spectrum

Data stored in Redshift Spectrum are in the form of tables called as External table. However, they are not a normal table stored in the cluster, unlike Redshift tables. The actual data is being stored in S3. You have to use standard Redshift SQL queries to examine those external tables. In Redshift, you need to create a schema in Redshift cluster; while in Redshift Spectrum, a schema is being referenced in the external database called data catalog. Data Catalog an index to the location and metrics of Spectrum data.

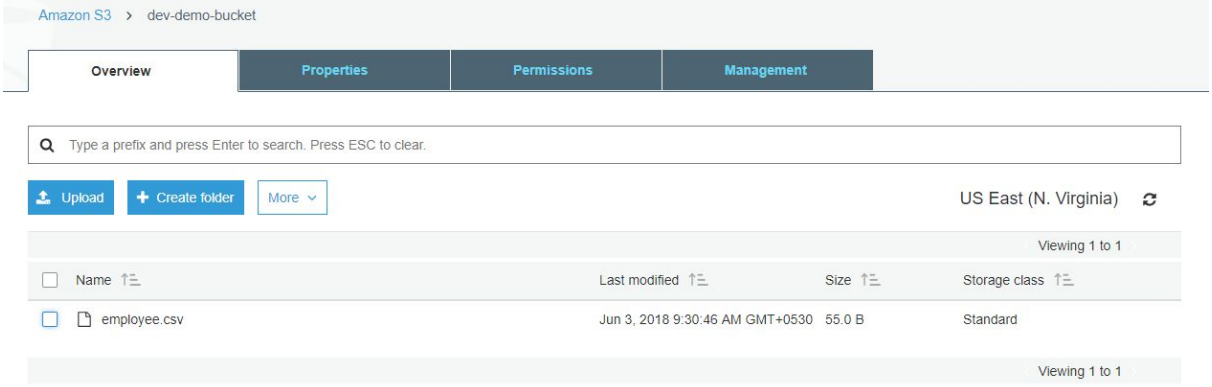
Before you load data in Redshift Spectrum tables, you need to create a data catalog which Spectrum needs to refer. Redshift spectrum refers data catalog from Amazon Athena/Glue/EMR. Redshift schema external tables can also be viewed in Amazon Athena/Glue/EMR and vice-versa (Database name and Spectrum schema name will be same).Here in this report, we will refer data catalog from Amazon Glue in our various examples. Typically in Amazon Glue, you run a crawler to take inventory of the data in your data stores but you can also generate the same manually.

Let us try to create Amazon Glue data catalog through crawler.

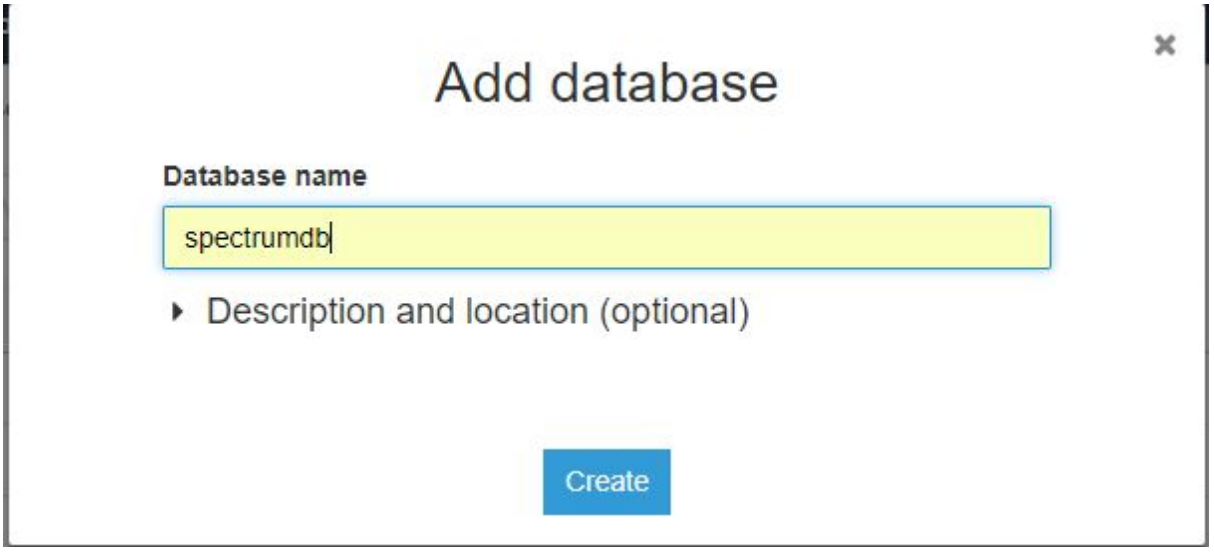
Step 1: Create a bucket dev-data-bucket and place a CSV file in the bucket. (employee.csv)

Content of employee.csv

1	a	10	100
3	c	10	300
5	e	10	500
2	b	20	200
4	d	20	400

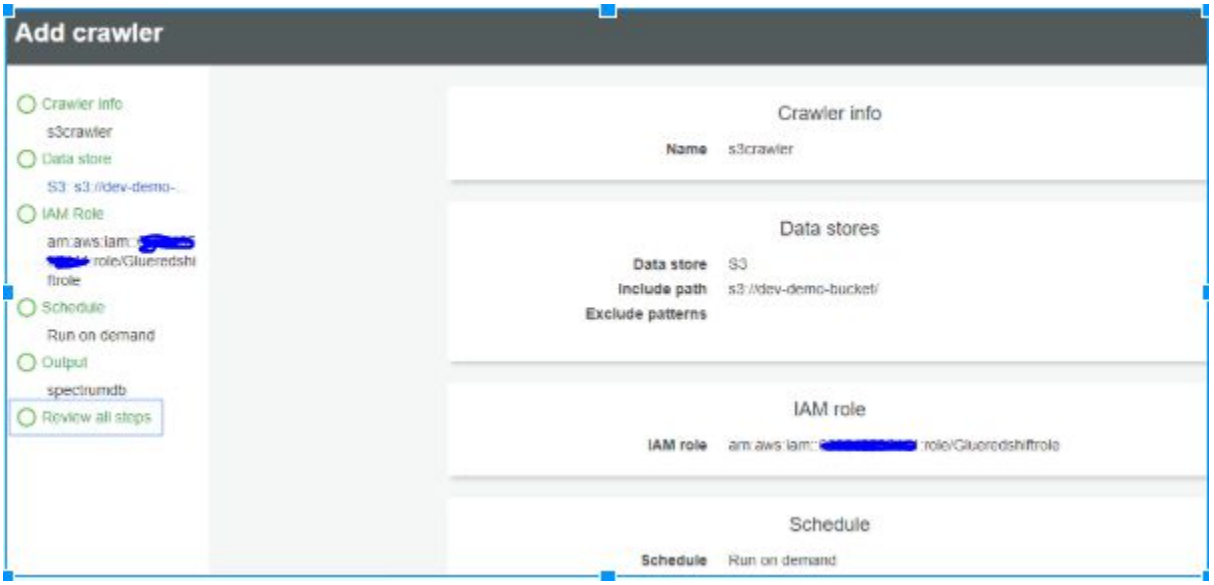


Step 2: Go to Amazon Glue and add a database **spectrumdb** (You can also create data catalog/database in Amazon Athena and Amazon EMR). Here in this example, we will refer Amazon Glue.

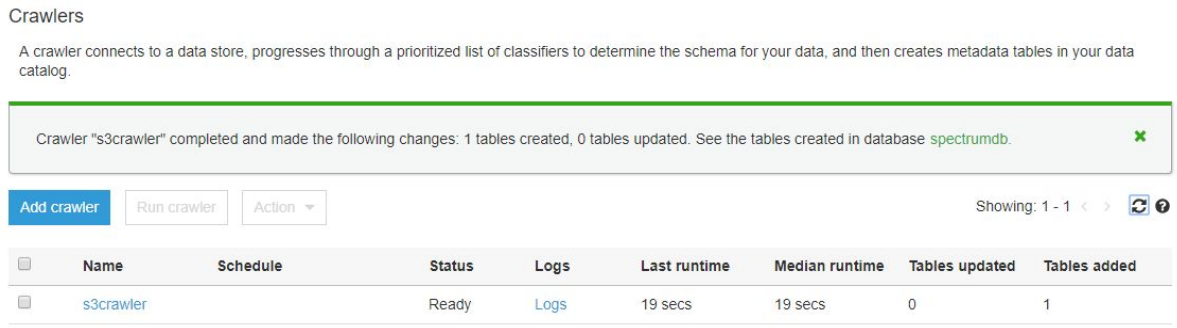


Step 3: Go to IAM console and assign Amazon Glue role to access Redshift services (Glueredshiftrole)

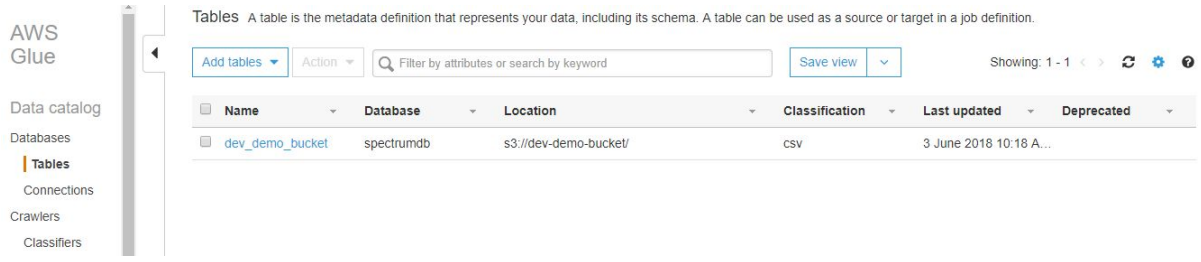
Generate data catalog using Amazon Glue crawler



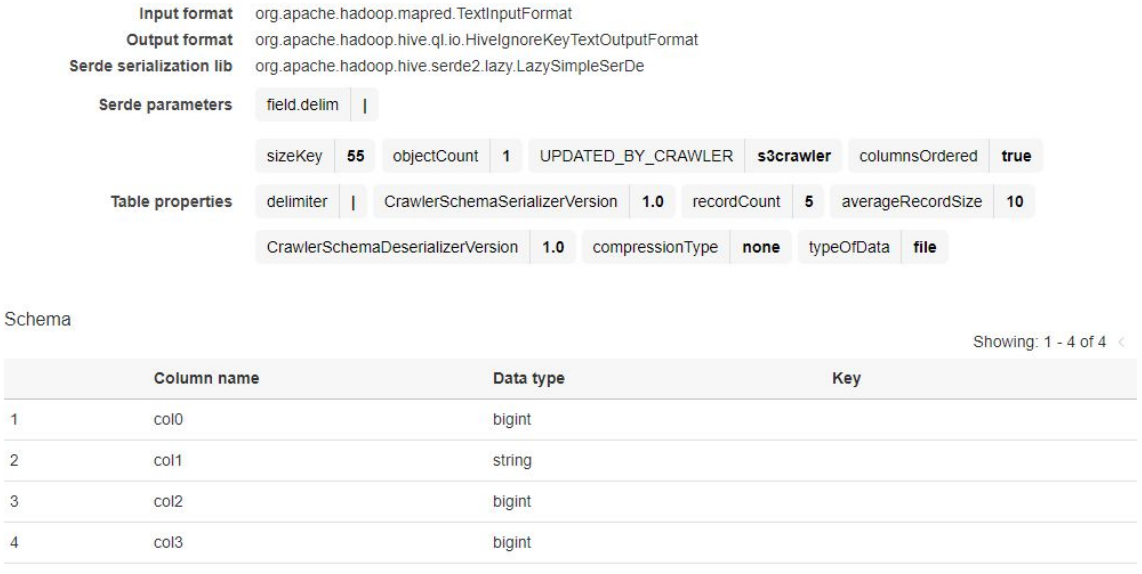
Step 4: Once the crawler is ready it will look as below



Let us see the table entry created through crawler (S3 file table)

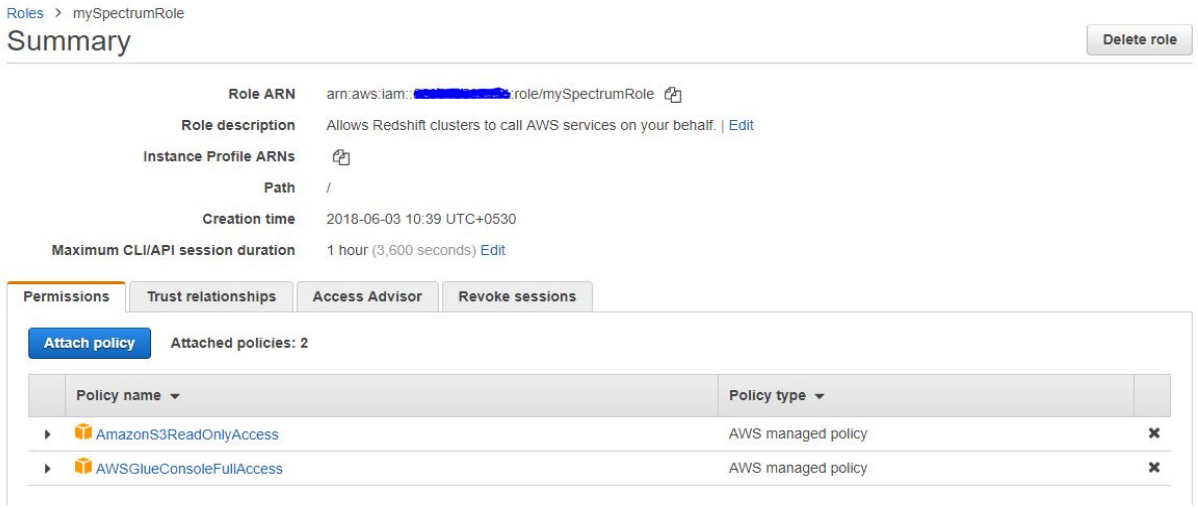


As you can see that crawler has created a table called dev_demo_bucket

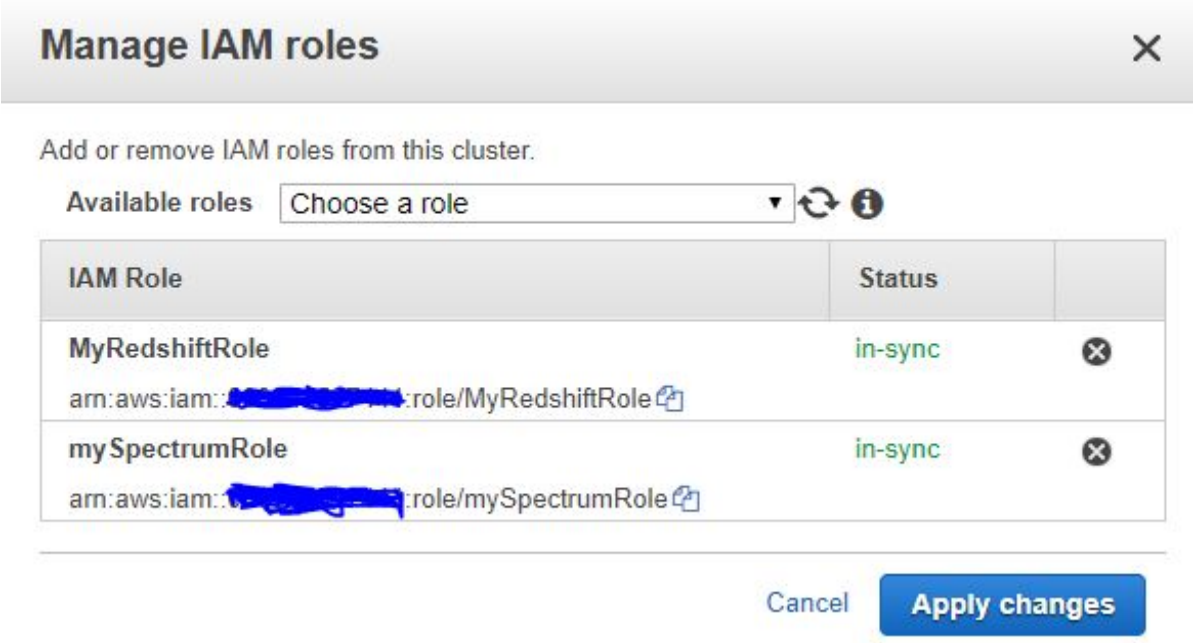


Creating External Schema

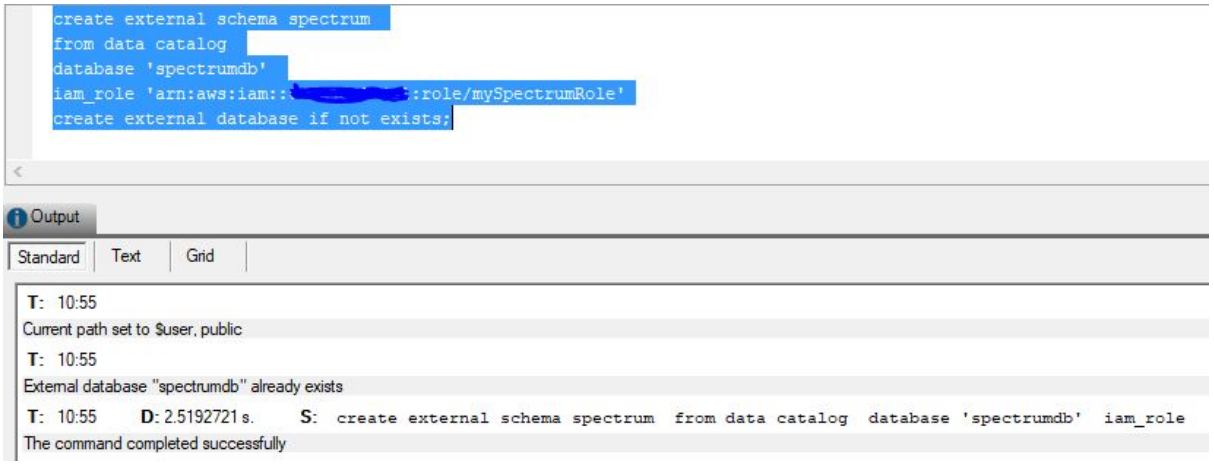
Since Redshift Spectrum uses data catalog from Amazon Glue/Amazon Athena/Amazon EMR, Redshift cluster requires permission to access S3 bucket files and full access on Glue/Athena/EMR. Let us create a new role mySpectrumrole and attach the policies AmazonS3ReadOnlyAccess and AWSGlueConsoleFullAccess to the role.



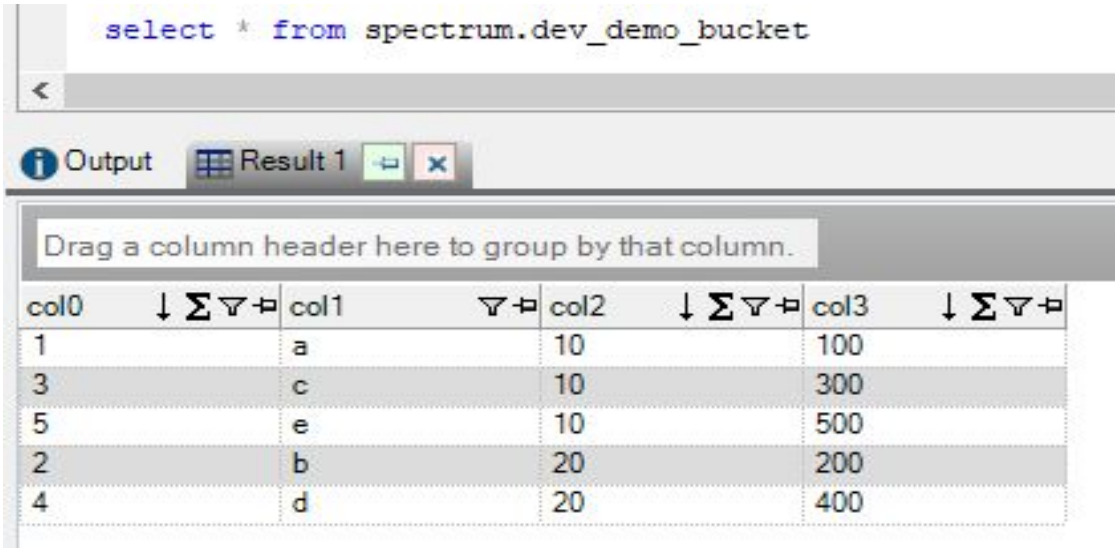
Next, associate “mySpectrumRole” role to the running cluster.



Now, we have attached all the necessary policies and our data catalog is ready. Let us create an external schema from data catalog in Redshift client. Remember your cluster and the Redshift Spectrum files must be in the same region. Now, login to Redshift client (Aginity here) and create external schema “spectrum” referring to data catalog schema **spectrumdb** created earlier in Amazon Glue as part of data cataloging.



The beauty of data cataloging is that you can access Amazon Glue/Athena/EMR tables or can even join to Redshift tables by accessing Redshift client. To illustrate here, we are using Aginity.



col0	col1	col2	col3
1	a	10	100
3	c	10	300
5	e	10	500
2	b	20	200
4	d	20	400

Let us try to join data catalog and Redshift tables (It should return data only for dept=10)

```
create table dept
(dept_no integer)

insert into dept values (10)

select * from spectrum.dev_demo_bucket a
inner join dept b
on a.col2=b.dept_no
```

Output | Result 1

Drag a column header here to group by that column.

col0	col1	col2	col3	dept_no
1	a	10	100	10
3	c	10	300	10
5	e	10	500	10

Creating External Tables for Amazon Redshift Spectrum

Since the schema does not reside in Redshift cluster rather it resides in the form of reference to other AWS Service (Glue/Athena/EMR), hence it is called external table.

Amazon Redshift Spectrum uses external tables to query data stored in S3. The syntax used to access Spectrum tables is same as used in Redshift tables. However, Spectrum tables are read-only and you cannot write into Spectrum tables.

To access Spectrum schema and tables you should be either:

1) Superuser
2) Access must have been granted on schema and tables to users for accessing schema and tables.

The following example creates a table named employee in the Amazon Redshift external schema named spectrum. The data contain pipe delimited text files.

```
Create external table spectrum.employee (
Emp_id integer,
Emp_name varchar(20),
Dept integer,
Salary integer)
row format delimited
fields terminated by '|'
stored as textfile
location 's3://dev-demo-bucket/';
```

The above command will automatically scan through all the files present inside s3://dev-demo-bucket/ and loads the data into spectrum.employee table.

There are few differences in Spectrum tables even though Spectrum tables look pretty similar to Redshift tables:-

1) In Spectrum table there is no concept of sort/distkeys.
2) Column compression (encoding) is not supported in external table.
3) Spectrum tables and S3 bucket should be in same region.
4) Spectrum tables are read-only.

5) You can't view details of Spectrum tables using the the same resources as used in Amazon Redshift i.e. PG_TABLE_DEF, STV_TBL_PERM. However, you can use different views, for example, SVV_EXTERNAL_TABLES, SVV_EXTERNAL_COLUMNS to refer details about external tables (This is covered in this report later.)

Partitioning the Redshift Spectrum External Tables

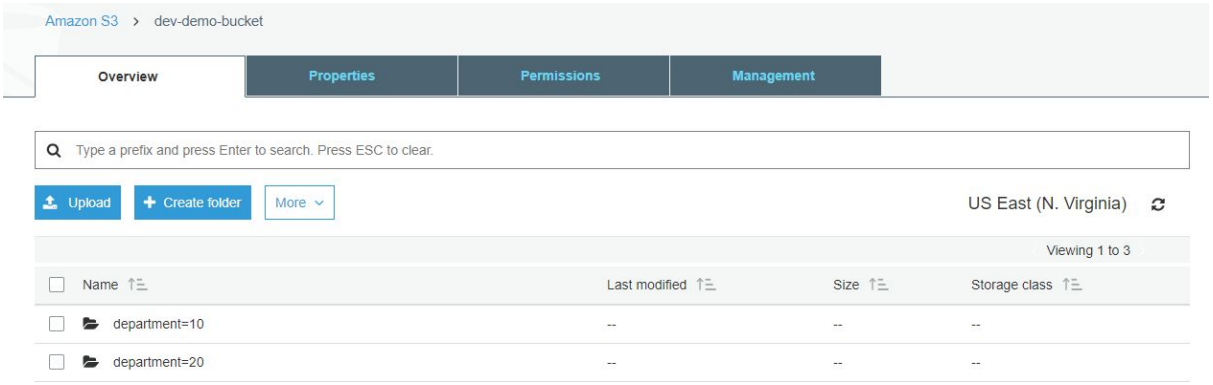
Redshift Spectrum supports handling of partitioned data. Partitioning of data means splitting the content of data using a partition key to segregate out the content stored in the external table. You can ensure faster access of data by applying filter condition in your query.

When you partition your data, you can restrict the amount of data Redshift Spectrum scans by filtering on the partition key. You can partition your data by any key. However, as a common practice, it is advisable to partition the data on date related columns.

Steps to partition the data

1. Store your data in folders in Amazon S3 according to your partition key.

Create a folder for each partition value and name the folder with the partition key and value.



Here in the above example, partition key is department and we have 2 folders for department 10 and 20 respectively.

Content of file inside “department=10” folder (10.csv) is mentioned below:

1	a	10	100
3	c	10	300
5	e	10	500

Content of file inside “department=20” folder (20.csv) is mentioned below:

2	b	20	200
4	d	20	400

2. Create an external table employee2 and specify the partition key in the PARTITIONED BY clause.

3. Add the partitions.

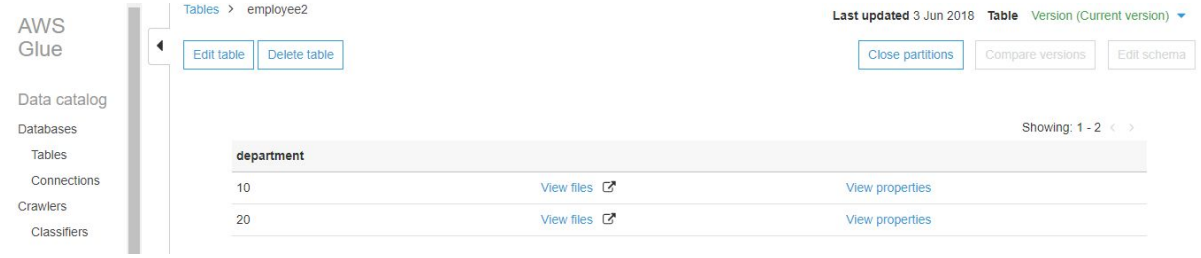
```
create external table spectrum.employee2 (
emp_id integer,
emp_name varchar(20),
dept integer,
salary integer)
partitioned by (department integer)
row format delimited
fields terminated by '|'
stored as textfile
location 's3://dev-demo-bucket/';

alter table spectrum.employee2
add partition(department=10)
location 's3://dev-demo-bucket/department=10/';
alter table spectrum.employee2
add partition(department=20)
location 's3://dev-demo-bucket/department=20/';

select * from spectrum.employee2
```

Output | Result 1

Drag a column header here to group by that column.

emp_id	emp_name	dept	salary	department
1	a	10	100	10
3	c	10	300	10
5	e	10	500	10
2	b	20	200	20
4	d	20	400	20

Same files can be viewed from Amazon Glue data catalog



Using a Manifest to Specify Data Files for Spectrum

Manifest files are files containing a list of entities or metadata for the set of files residing in S3. Using manifest file, you can place a list of files which Spectrum will be referring to and not the entire set of data. Manifest files in Spectrum would be a JSON file containing the URL of bucket name and object path. You can also specify various combination files residing in different S3 buckets. All the object files are not required to be in the same bucket.

Example of a sample manifest file:

```
employee.manifest
{
  "entries": [
    {"url":"s3://test1-bucket/April-data", "mandatory":true},
    {"url":"s3://test1-bucket/May-data", "mandatory":true},
    {"url":"s3://test2-bucket/April-data", "mandatory":true},
    {"url":"s3://test2-bucket/May-data", "mandatory":true}
  ]
}
```

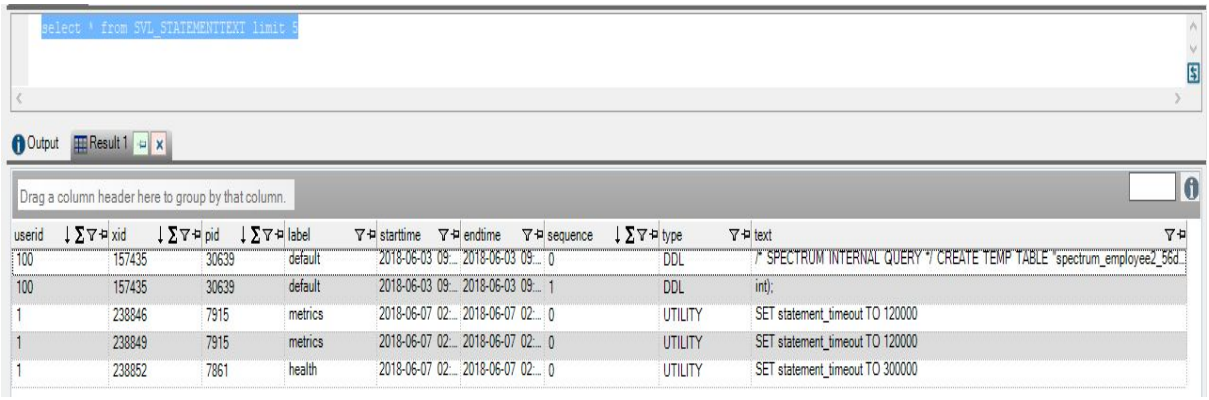
Example of an External table creation using Manifest file:

```
create external table spectrum.emp(
empid integer,
deptid integer,
salary integer)
row format delimited
fields terminated by '|'
stored as textfile
location 's3://testbucket/spectrum/employee.manifest'
```

In the above example, External table spectrum.emp will refer the data from s3://testbucket/spectrum/employee.manifest. It will have the content of data residing in above 4 files in the employee.manifest file. If you compare this example against the previous example of External table creation by specifying the S3 folder name then data was scanning through entire files residing in mentioned folder. On the other hand, while creating the external table by specifying the manifest file, Spectrum External table only refers explicit data files mentioned in the manifest file and not entire folder data files.

System views used in Redshift spectrum

SVL_STATEMENTTEXT view contains all the query details including DDL, Utility. For getting all details of query executed by Redshift/Spectrum, this is a great view to refer.



userid	xid	pid	label	starttime	endtime	sequence	type	text
100	157435	30639	default	2018-06-03 09:...	2018-06-03 09:...	0	DDL	/* SPECTRUM INTERNAL QUERY */ CREATE TEMP TABLE "spectrum_employee2_56d...
100	157435	30639	default	2018-06-03 09:...	2018-06-03 09:...	1	DDL	int);
1	238846	7915	metrics	2018-06-07 02:...	2018-06-07 02:...	0	UTILITY	SET statement_timeout TO 120000
1	238849	7915	metrics	2018-06-07 02:...	2018-06-07 02:...	0	UTILITY	SET statement_timeout TO 120000
1	238852	7861	health	2018-06-07 02:...	2018-06-07 02:...	0	UTILITY	SET statement_timeout TO 300000

The few important system views related to Spectrum are mentioned below:

- SVV_EXTERNAL_COLUMNS → This view contain details of columns for external tables.
- SVV_EXTERNAL_DATABASES → This view contain details for external databases.
- SVV_EXTERNAL_PARTITIONS → This view contain details for partitions in external tables.
- SVV_EXTERNAL_SCHEMAS → This view contain details about external schemas.
- SVV_EXTERNAL_TABLES → This view contain details for external tables.
- SVL_QLOG → This view contains the log of all the queries run against the database.
- SVL_S3LOG → This view contain details of Spectrum queries at segment and node slice level (This view is useful in troubleshooting issue while creating external tables from S3).
- SVL_S3PARTITION → This view contain details of Spectrum partitions at segment and node slice level.

- SVL_S3QUERY → This view contain details about Amazon Redshift Spectrum queries at the segment and node slice level. (This view is useful in getting details about queries run against Spectrum)
- SVL_S3QUERY_SUMMARY→ This view contains the summary of all Spectrum queries (S3 queries) that have been run on the system. SVL_S3QUERY_SUMMARY aggregates detail from SVL_S3QUERY at the segment level.

Redshift Spectrum Query Best Practices

1) Amazon Redshift Spectrum pushes many compute intensive tasks into another layer called Spectrum layer and hence queries use much less of cluster's processing capacity. If you are working in typical data warehousing environment, it is advisable to keep your fact tables/high volume data into Spectrum layer (S3) and keep dimension tables/small tables data in Redshift cluster.

2) With Spectrum, you can keep a large volume of data in S3 and to reduce cost you can terminate on-demand Redshift cluster once your job is finished.

3) Use Apache Parquet files for better performance and lower cost. Apache Parquet is a columnar storage format files and contains compressed data. This improves the performance and lower cost since Amazon Redshift Spectrum charges you by the amount of data that is scanned from S3 per query.

4) Partition files on frequently used columns. As a general practice, it is advisable to apply partition on columns frequently used for filtering. Make sure data is not skewed on partitioning columns. File size distribution should be as uniform as possible.

5) Last but not the least while working with large data set environment it is always better to check the query performance by hitting SVL_S3QUERY_SUMMARY view.

Following are some things to look for in SVL_S3QUERY_SUMMARY:

- Count of files processed by the Redshift Spectrum query.
- The number of rows of data scanned from Amazon S3 (In bytes).
- The number of rows returned from the Amazon S3 to the cluster (In bytes).
- The maximum and average duration of Redshift Spectrum requests. The queries should not be long-running, if it is long running this could be a case of bottleneck.

Example:-

```
select query, segment, elapsed, s3_scanned_rows as scanned_rows,
s3_scanned_bytes as scanend_bytes, s3query_returned_rows as returned_rows,
files, s3query_returned_bytes as returned_bytes
from svl_s3query_summary
where query = pg_last_query_id();
```

Query	Segment	Elapsed	Scanned_rows	Scanned_bytes	Returned_rows	Files	Returned_bytes
123	2	123456	200000	2000000	9000	1	200000

If you are looking for an ETL solution, consider Hevo!

If you want to load any data easily into Spectrum without any hassle, you can try out Hevo. Hevo automates the flow of data from various sources to Amazon Spectrum in real time and at zero data loss. In addition to migrating data, you can also build aggregates and joins on to create materialized views that enable faster query processing.

About Author:

Ankur Shrivastava is a AWS Solution Designer with hands-on experience on Data Warehousing, ETL, and Data Analytics. He is an AWS Certified Solution Architect Associate. In his free time, he enjoys all outdoor sports and practices.