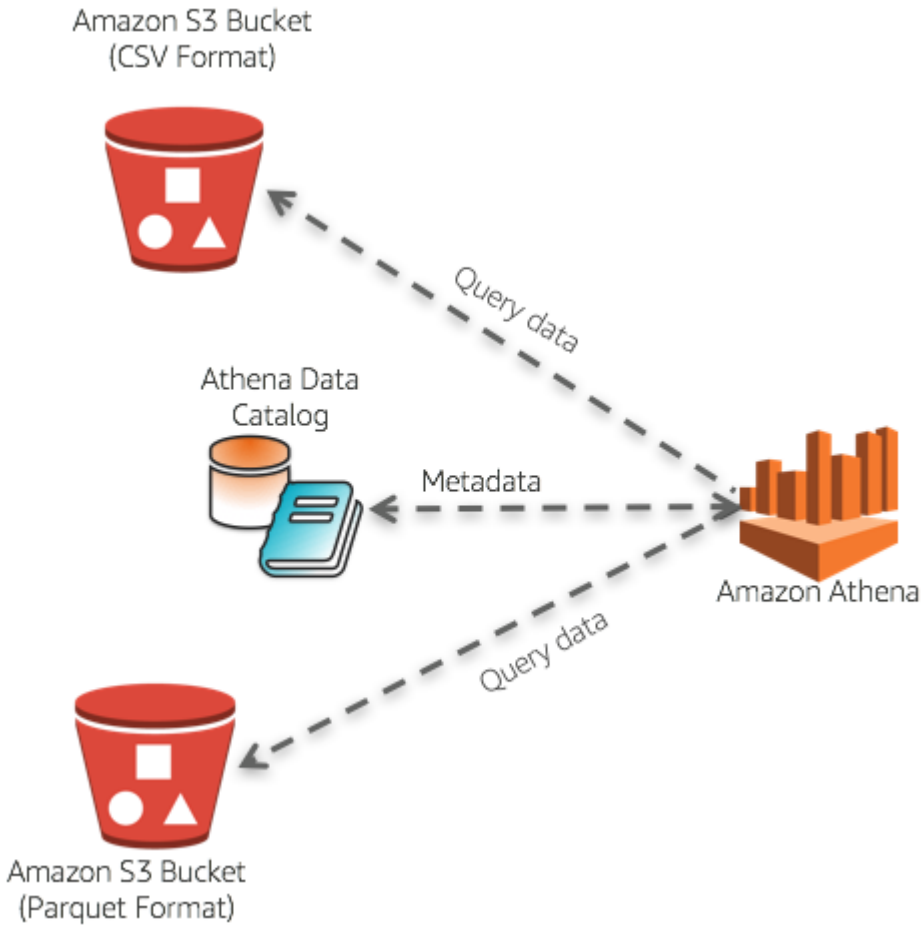
Lab 1: Serverless Analysis of data in Amazon S3 using Amazon Athena

Architectural Diagram



Creating Amazon Athena Database and Table

Amazon Athena uses Apache Hive to define tables and create databases. Databases are a logical grouping of tables. When you create a database and table in Athena, you are simply describing the schema and location of the table data in Amazon S3. In case of Hive, databases and tables don't store the data along with the schema definition unlike traditional relational database systems. The data is read from Amazon S3 only when you query the table. The other benefit of using Hive is that the metastore found in Hive can be used in many other big data applications such as Spark, Hadoop, and Presto. With Athena catalog, you can now have Hive-compatible metastore in the cloud without the need for provisioning a Hadoop cluster or RDS instance. For guidance on databases and tables creation refer Apache Hive documentation. The following steps provide guidance specifically for Amazon Athena.

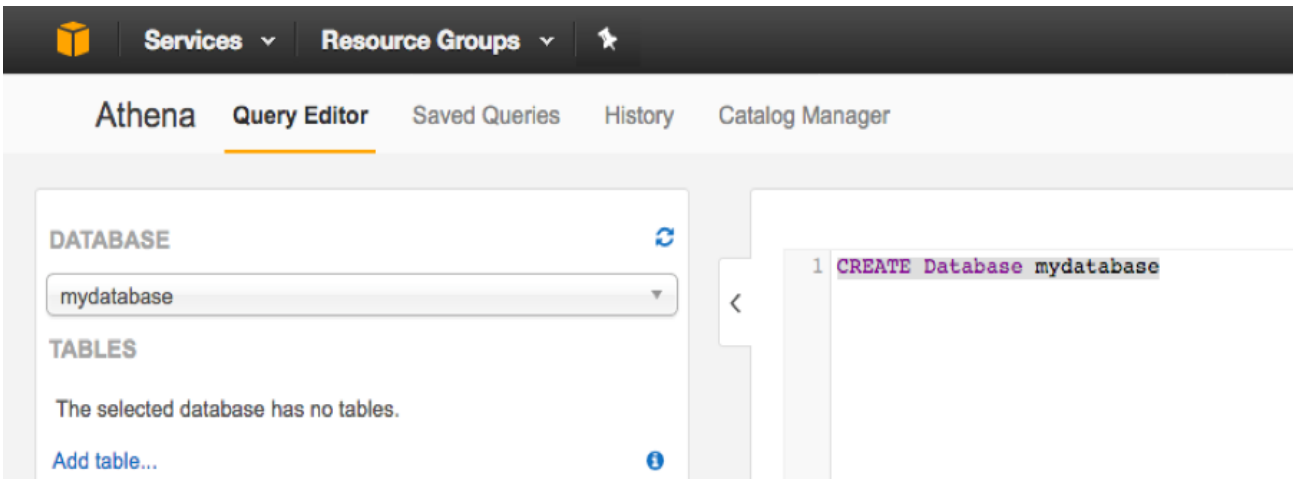
Create Database

1. Open the AWS Management Console for Athena.
2. If this is your first time visiting the AWS Management Console for Athena, you will get a Getting Started page. Choose **Get Started** to open the Query Editor. If this isn't your first time, the Athena **Query Editor** opens.

3. Make a note of the AWS region name, for example, for this lab you will need to choose the **US West (Oregon)** region.
4. In the Athena **Query Editor**, you will see a query pane with an example query. Now you can start entering your query in the query pane.
5. To create a database named *mydatabase*, copy the following statement, and then choose **Run Query**:

```
CREATE DATABASE mydatabase
```

6. Ensure *mydatabase* appears in the DATABASE list on the **Catalog** dashboard



Create a Table

Now that you have a database, you are ready to create a table that is based on the New York taxi sample data. You define columns that map to the data, specify how the data is delimited, and provide the location in Amazon S3 for the file.

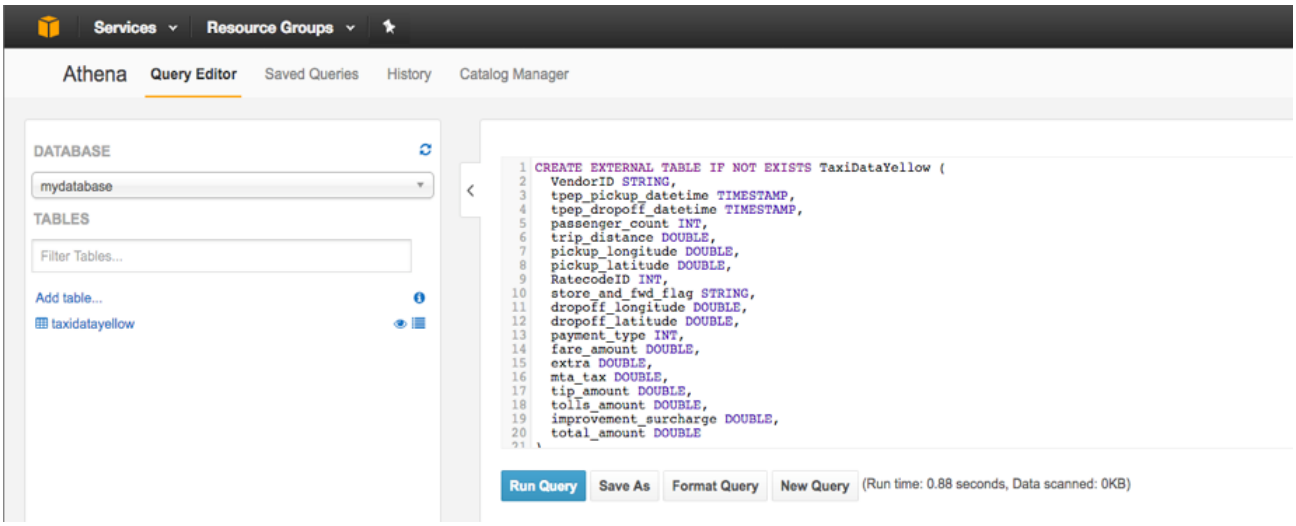
> **Note:** When creating the table, you need to consider the following:
>
> - You must have the appropriate permissions to work with data in the Amazon S3 location. For more information, refer Setting User and Amazon S3 Bucket Permissions.
> - The data can be in a different region from the primary region where you run Athena as long as the data is not encrypted in Amazon S3. Standard inter-region data transfer rates for Amazon S3 apply in addition to standard Athena charges.
> - If the data is encrypted in Amazon S3, it must be in the same region, and the user or principal who creates the table must have the appropriate permissions to decrypt the data. For more information, refer Configuring Encryption Options.
> - Athena does not support different storage classes within the bucket specified by the LOCATION clause, does not support the GLACIER

> storage class, and does not support Requester Pays buckets. For more information, see Storage Classes,Changing the Storage Class of an Object in Amazon S3, and Requester Pays Buckets in the Amazon Simple Storage Service Developer Guide.

1. Ensure that current AWS region is **US West (Oregon)** region
2. Ensure **mydatabase** is selected from the **DATABASE** list and then choose **New Query**.
3. In the query pane, copy the following statement to create TaxiDataYellow table, and then choose **Run Query**:

```
CREATE EXTERNAL TABLE IF NOT EXISTS TaxiDataYellow (
  VendorID STRING,
  tpep_pickup_datetime TIMESTAMP,
  tpep_dropoff_datetime TIMESTAMP,
  passenger_count INT,
  trip_distance DOUBLE,
  pickup_longitude DOUBLE,
  pickup_latitude DOUBLE,
  RatecodeID INT,
  store_and_fwd_flag STRING,
  dropoff_longitude DOUBLE,
  dropoff_latitude DOUBLE,
  payment_type INT,
  fare_amount DOUBLE,
  extra DOUBLE,
  mta_tax DOUBLE,
  tip_amount DOUBLE,
  tolls_amount DOUBLE,
  improvement_surcharge DOUBLE,
  total_amount DOUBLE
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 's3://us-west-2.serverless-analytics/NYC-Pub/yellow/'
```

> **Note:**
>
> - If you use CREATE TABLE without the EXTERNAL keyword, you will get an error as only tables with the EXTERNAL keyword can be created in Amazon Athena. We recommend that you always use the EXTERNAL keyword. When you drop a table, only the table metadata is removed and the data remains in Amazon S3.
> - You can also query data in regions other than the region where you are running Amazon Athena. Standard inter-region data transfer rates for Amazon S3 apply in addition to standard Amazon Athena charges.
> - Ensure the table you just created appears on the Catalog dashboard for the selected database.



Querying data from Amazon S3 using Amazon Athena

Now that you have created the table, you can run queries on the data set and see the results in AWS Management Console for Amazon Athena.

1. Choose **New Query**, copy the following statement into the query pane, and then choose **Run Query**.

```
SELECT * FROM TaxiDataYellow limit 10
```

Results for the above query look like the following:

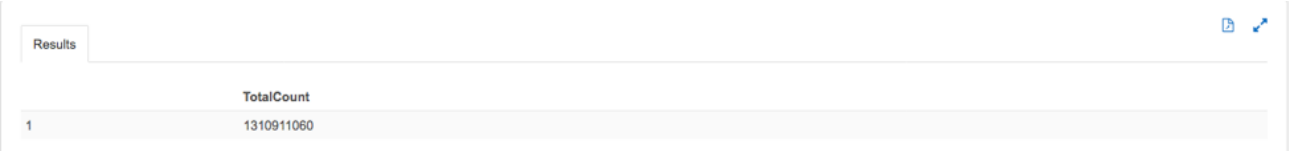
Results

	vendorid	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	ratecodeid	store_and_fwd_flag	dropoff_longitude	dropoff_latitude
1	CMT	2010-09-08 14:40:24.000	2010-09-08 14:46:56.000	1	1.3	-73.991867	40.748982	1	N	-73.995792	40.738708
2	CMT	2010-09-08 12:52:11.000	2010-09-08 13:04:51.000	1	1.8	-73.997253	40.725447	1	N	-73.989603	40.702025
3	CMT	2010-09-08 12:31:37.000	2010-09-08 12:37:27.000	1	1.2	-73.79027	40.64644	1	N	-73.943387	40.768977
4	CMT	2010-09-17 18:30:13.000	2010-09-17 18:32:33.000	1	0.6	-73.991052	40.749222	1	N	-73.96478	40.763778
5	CMT	2010-09-08 15:27:48.000	2010-09-08 15:35:50.000	1	1.9	-73.997845	40.720523	1	N	-73.971835	40.762908
6	CMT	2010-09-08 12:41:49.000	2010-09-08 12:56:52.000	1	2.0	-73.997692	40.724093	1	N	-74.011063	40.704315
7	CMT	2010-09-08 07:46:20.000	2010-09-08 08:05:52.000	4	2.8	-74.004775	40.739772	1	N	-74.011435	40.708125
8	CMT	2010-09-08 15:21:28.000	2010-09-08 15:24:41.000	1	0.7	-73.979777	40.761237	1	N	-73.953965	40.7669
9	CMT	2010-09-17 07:30:34.000	2010-09-17 07:37:58.000	1	0.9	-74.008032	40.739643	1	N	-73.985582	40.748675
10	CMT	2010-09-08 10:13:04.000	2010-09-08 10:33:16.000	1	3.6	-73.968153	40.768775	1	N	-73.988533	40.769248

2. Choose **New Query**, copy the following statement into the query pane, and then choose **Run Query** to get the total number of taxi rides for yellow cabs.

```
SELECT COUNT(1) as TotalCount FROM TaxiDataYellow
```

Results for the above query look like the following:



Results

	TotalCount
1	1310911060

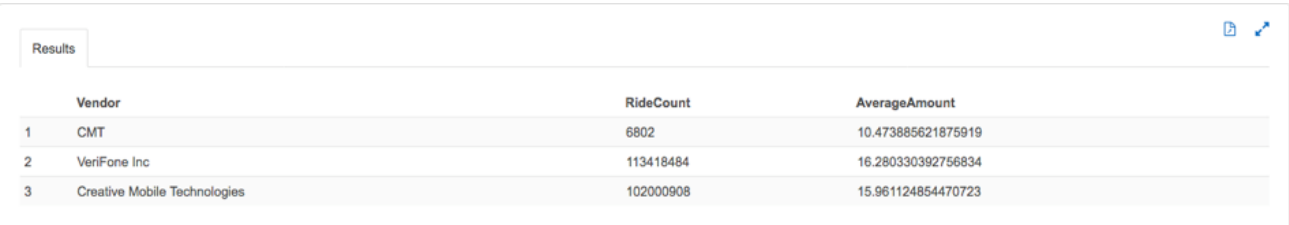
> **Note:** The current data format is CSV and this query is scanning **~207GB** of data and takes **~20.06** seconds to execute the query.

3. Make a note of query execution time for later comparison while querying the data set in Apache Parquet format.

4. Choose **New Query**, copy the following statement into the query pane, and then choose **Run Query** to query for the number of rides per vendor, along with the average fair amount for yellow taxi rides

```
SELECT
CASE vendorid
     WHEN '1' THEN 'Creative Mobile Technologies'
     WHEN '2' THEN 'VeriFone Inc'
     ELSE vendorid END AS Vendor,
COUNT(1) as RideCount,
avg(total_amount) as AverageAmount
FROM TaxiDataYellow
WHERE total_amount > 0
GROUP BY (1)
```

Results for the above query look like the following:



Results

	Vendor	RideCount	AverageAmount
1	CMT	6802	10.473885621875919
2	VeriFone Inc	113418484	16.280330392756834
3	Creative Mobile Technologies	102000908	15.961124854470723

Querying partitioned data using Amazon Athena

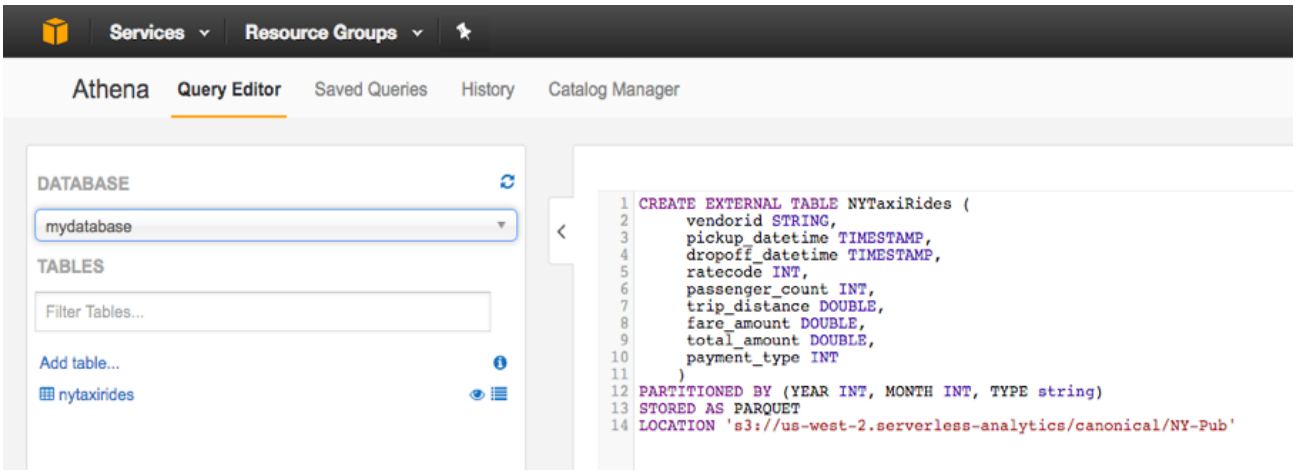
By partitioning your data, you can restrict the amount of data scanned by each query, thus improving performance and reducing cost. Athena leverages Hive for partitioning data. You can partition your data by any key. A common practice is to partition the data based on time, often leading to a multi-level partitioning scheme. For example, a customer who has data coming in every hour might decide to partition by year, month, date, and hour. Another customer, who has data coming from many different sources but loaded one time per day, may partition by a data source identifier and date.

Create a Table with Partitions

1. Ensure that current AWS region is **US West (Oregon)** region
2. Ensure **mydatabase** is selected from the DATABASE list and then choose **New Query**.
3. In the query pane, copy the following statement to create a the NYTaxiRides table, and then choose **Run Query**:

```
CREATE EXTERNAL TABLE NYTaxiRides (
  vendorid STRING,
  pickup_datetime TIMESTAMP,
  dropoff_datetime TIMESTAMP,
  ratecode INT,
  passenger_count INT,
  trip_distance DOUBLE,
  fare_amount DOUBLE,
  total_amount DOUBLE,
  payment_type INT
  )
PARTITIONED BY (YEAR INT, MONTH INT, TYPE string)
STORED AS PARQUET
LOCATION 's3://us-west-2.serverless-analytics/canonical/NY-Pub'
```

4.Ensure the table you just created appears on the Catalog dashboard for the selected database.



> **Note:** Running the following sample query on the NYTaxiRides table you just created will not return any result as no metadata about the partition is added to the Amazon Athena table catalog.
>
> ```
> SELECT * FROM NYTaxiRides limit 10
> ```

Adding partition metadata to Amazon Athena

Now that you have created the table you need to add the partition metadata to the Amazon Athena Catalog.

1. Choose **New Query**, copy the following statement into the query pane, and then choose **Run Query** to add partition metadata.

```
MSCK REPAIR TABLE NYTaxiRides
```

The returned result will contain information for the partitions that are added to NYTaxiRides for each taxi type (yellow, green, fhv) for every month for the year from 2009 to 2016

> **Note:** The MSCK REPAIR TABLE automatically adds partition data based on the New York taxi ride data to in the Amazon S3 bucket is because the data is already converted to Apache Parquet format partitioned by year, month and type, where type is the taxi type (yellow, green or fhv). If the data layout does not confirm with the requirements of MSCK REPAIR TABLE the alternate approach is to add each partition manually using ALTER TABLE ADD PARTITION. You can also automate adding partitions by using the JDBC driver.

Querying partitioned data set

Now that you have added the partition metadata to the Athena data catalog you can now run your query.

1. Choose **New Query**, copy the following statement into the query pane, and then choose **Run Query** to get the total number of taxi rides

```
SELECT count(1) as TotalCount from NYTaxiRides
```

Results for the above query look like the following:

Results

	TotatlCount
1	2870781820

> **Note:** This query executes much faster because the data set is partitioned and it in optimal format - Apache Parquet (an open source columnar).

> Following is a comparison of the execution time and amount of data scanned between the data formats:
>
> > **CSV Format:**
> >
> > ```
> > SELECT count(*) as count FROM TaxiDataYellow
> > ```
> >
> > Run time: **~20.06 seconds**, Data scanned: **~207.54GB**, Count: **1,310,911,060**
> >
> > ```
> > SELECT * FROM TaxiDataYellow limit 1000
> > ```
> >
> > Run time: **~3.13 seconds**, Data scanned: **~328.82MB**
>
> > **Parquet Format:**
> >
> > ```
> > SELECT count(*) as count FROM NYTaxiRides
> > ```
> >
> > Run time: **~5.76 seconds**, Data scanned: **0KB**, Count: **2,870,781,820**
> >
> > ```
> > SELECT * FROM NYTaxiRides limit 1000
> > ```
> >
> > Run time: **~1.13 seconds**, Data scanned: **5.2MB**

2. Choose **New Query**, copy the following statement into the query pane, and then choose **Run Query** to get the total number of taxi rides by year

```
SELECT YEAR, count(1) as TotalCount from NYTaxiRides GROUP BY YEAR
```

Results for the above query look like the following:

Results

	YEAR	TotalCount
1	2015	375327352
2	2010	338002306
3	2012	357088648
4	2011	353794416
5	2014	346065723
6	2016	411140768
7		168
8	2009	341792110
9	2013	347570329

3. Choose **New Query**, copy the following statement into the query pane, and then choose **Run Query** to get the top 12 months by total number of rides across all the years

```
SELECT YEAR, MONTH, COUNT(1) as TotalCount
FROM NYTaxiRides
GROUP BY (1), (2)
ORDER BY (3) DESC LIMIT 12
```

Results for the above query look like the following:

Results

	YEAR	MONTH	TotalCount
1	2016	5	36063253
2	2016	3	35722884
3	2016	4	35654963
4	2016	10	35614908
5	2016	12	35562724
6	2015	10	34934049
7	2016	6	34545922
8	2016	2	33772790
9	2016	11	33700663
10	2015	12	33354426
11	2016	9	33088841
12	2016	7	33014293

4. Choose **New Query**, copy the following statement into the query pane, and then choose **Run Query** to get the monthly ride counts per taxi time for the year 2016.

```
SELECT MONTH, TYPE, COUNT(1) as TotalCount
FROM NYTaxiRides
WHERE YEAR = 2016
GROUP BY (1), (2)
ORDER BY (1), (2)
```

Results for the above query look like the following:

Results

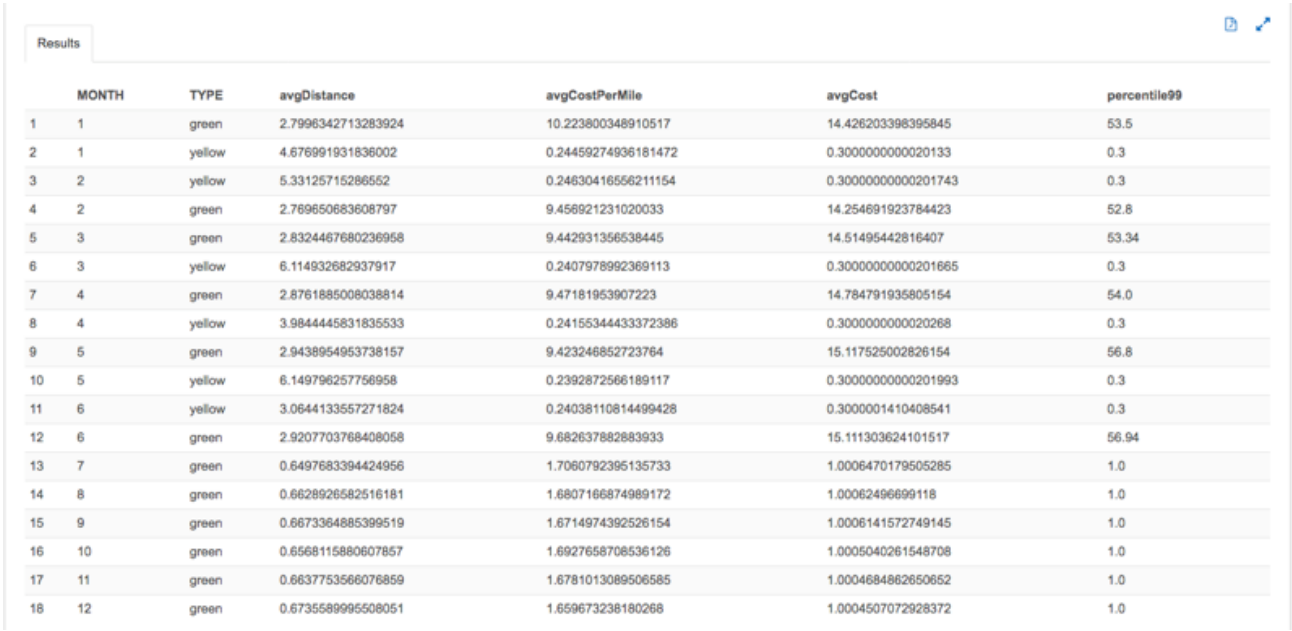
	MONTH	TYPE	TotalCount
1	1	fhv	8756726
2	1	green	1445285
3	1	yellow	21813716
4	2	fhv	9497970
5	2	green	1510722
6	2	yellow	22764098
7	3	fhv	9724587
8	3	green	1576393

> **Note:** Now the execution time is ~ 3 second, as the amount of data scanned by the query is restricted thus improving performance. This is because the data set is partitioned and it in optimal format – Apache Parquet, an open source columnar format.

5. Choose **New Query**, copy the following statement anywhere into the query pane, and then choose **Run Query**.

```
SELECT MONTH,
  TYPE,
  avg(trip_distance) as  avgDistance,
  avg(total_amount/trip_distance) as avgCostPerMile,
  avg(total_amount) as avgCost,
  approx_percentile(total_amount, .99) percentile99
FROM NYTaxiRides
WHERE YEAR = 2016 AND (TYPE = 'yellow' OR TYPE = 'green') AND trip_distan
GROUP BY MONTH, TYPE
ORDER BY MONTH
```

Results for the above query look like the following:



Results

	MONTH	TYPE	avgDistance	avgCostPerMile	avgCost	percentile99
1	1	green	2.7996342713283924	10.223800348910517	14.426203398395845	53.5
2	1	yellow	4.676991931836002	0.24459274936181472	0.3000000000020133	0.3
3	2	yellow	5.33125715286552	0.24630416556211154	0.30000000000201743	0.3
4	2	green	2.769650683608797	9.456921231020033	14.254691923784423	52.8
5	3	green	2.8324467680236958	9.442931356538445	14.51495442816407	53.34
6	3	yellow	6.114932682937917	0.2407978992369113	0.30000000000201665	0.3
7	4	green	2.8761885008038814	9.47181953907223	14.784791935805154	54.0
8	4	yellow	3.9844445831835533	0.24155344433372386	0.3000000000020268	0.3
9	5	green	2.9438954953738157	9.423246852723764	15.117525002826154	56.8
10	5	yellow	6.149796257756958	0.2392872566189117	0.30000000000201993	0.3
11	6	yellow	3.0644133557271824	0.24038110814499428	0.3000001410408541	0.3
12	6	green	2.9207703768408058	9.682637882883933	15.111303624101517	56.94
13	7	green	0.6497683394424956	1.7060792395135733	1.0006470179505285	1.0
14	8	green	0.6628926582516181	1.6807166874989172	1.00062496699118	1.0
15	9	green	0.6673364885399519	1.6714974392526154	1.0006141572749145	1.0
16	10	green	0.6568115880607857	1.6927658708536126	1.0005040261548708	1.0
17	11	green	0.6637753566076859	1.6781013089506585	1.0004684862650652	1.0
18	12	green	0.6735589995508051	1.659673238180268	1.0004507072928372	1.0

Creating Views with Amazon Athena

A view in Amazon Athena is a logical, not a physical table. The query that defines a view runs each time the view is referenced in a query. You can create a view from a SELECT query and then reference this view in future queries. For more information, see CREATE VIEW.

1. Ensure that current AWS region is **US West (Oregon)** region
2. Ensure **mydatabase** is selected from the DATABASE list.
3. Choose **New Query**, copy the following statement anywhere into the query pane, and then choose **Run Query**.

```
CREATE VIEW nytaxiridesmonthly AS
SELECT
    year,
    month,
    vendorid,
    avg(total_amount) as avg_Amt,
    sum (total_amount) as sum_Amt
FROM nytaxirides
where total_amount > 0
group by vendorid, year, month
```

You will see a new view called **nytaxiridesmonthly** created under **Views** under **Database** section in the left.

4. Choose **New Query**, copy the following statement anywhere into the query pane, and then choose **Run Query**.

```
SELECT * FROM nytaxiridesmonthly WHERE vendorid = '1'
```

Some of the view specific commands to try out are SHOW COLUMNS, SHOW CREATE VIEW, DESCRIBE VIEW, and DROP VIEW.

CTAS Query with Amazon Athena

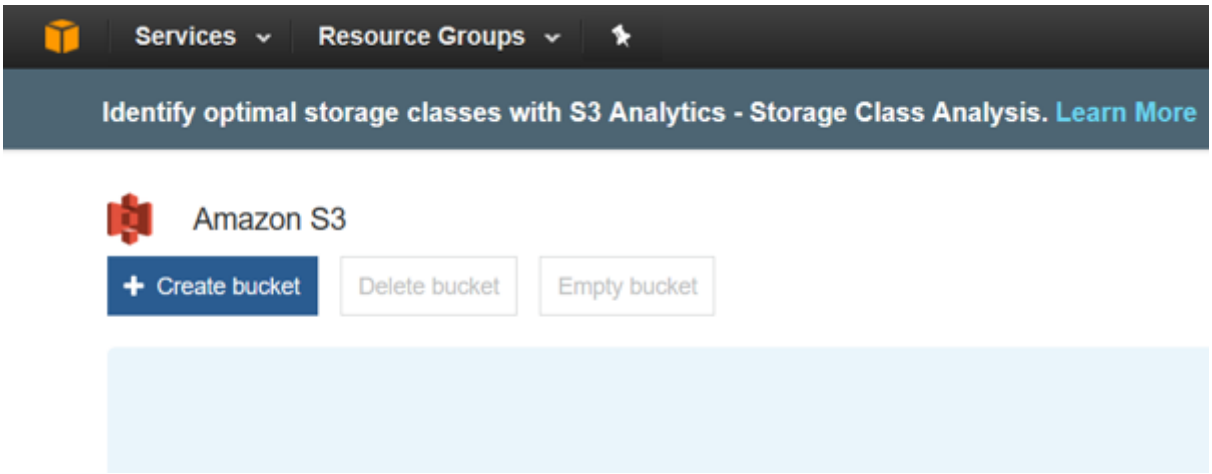
A CREATE TABLE AS SELECT (CTAS) query creates a new table in Athena from the results of a SELECT statement from another query. Athena stores data files created by the CTAS statement in a specified location in Amazon S3. For syntax, see CREATE TABLE AS.

Use CTAS queries to:

Create tables from query results in one step, without repeatedly querying raw data sets. This makes it easier to work with raw data sets. Transform query results into other storage formats, such as Parquet and ORC. This improves query performance and reduces query costs in Athena. For information, see Columnar Storage Formats. Create copies of existing tables that contain only the data you need.

Create an Amazon S3 Bucket

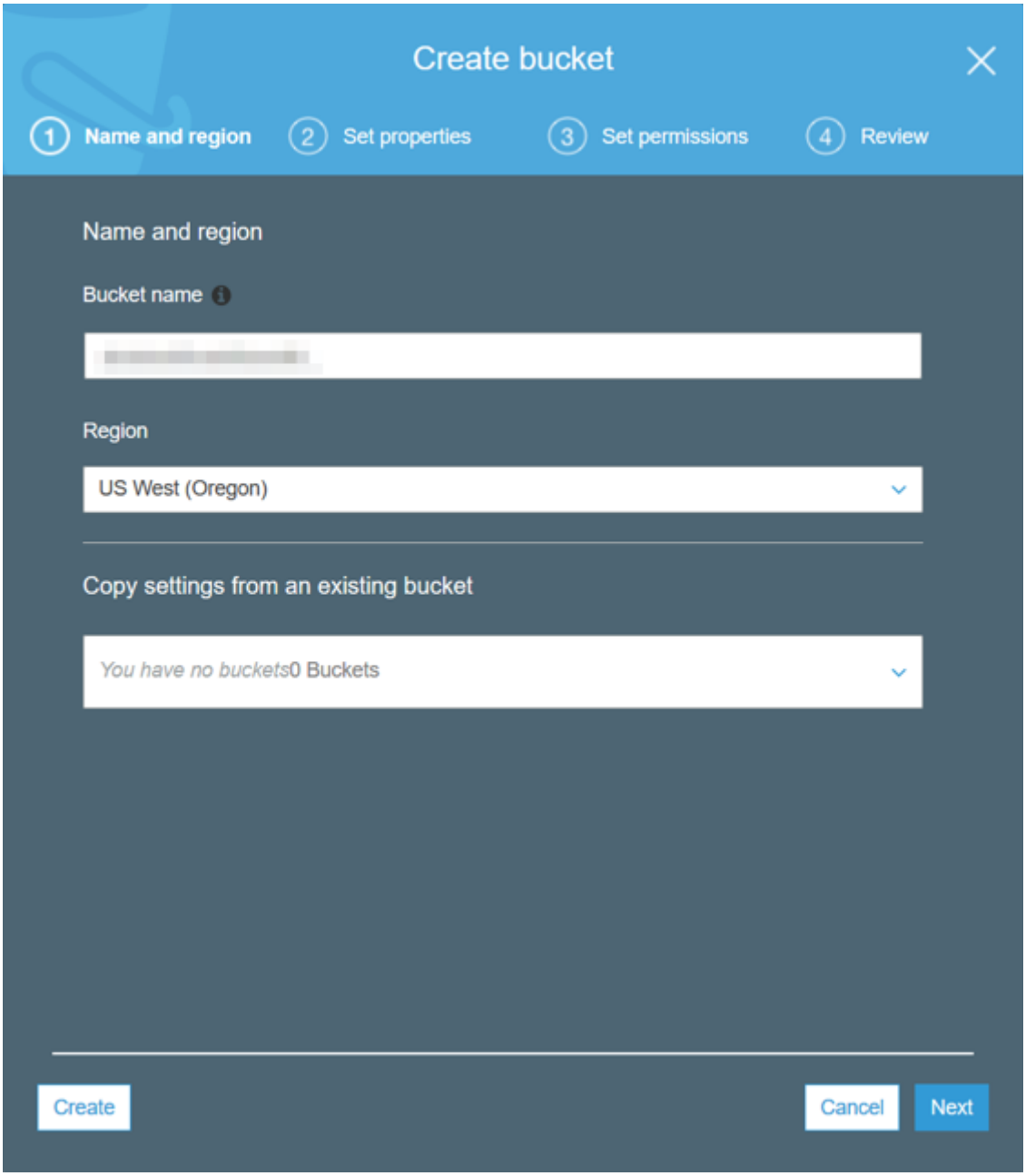
1. Open the AWS Management console for Amazon S3
2. On the S3 Dashboard, Click on **Create Bucket**.



3. In the **Create Bucket** pop-up page, input a unique **Bucket name**. So it's advised to choose a large bucket name, with many random characters and numbers (no spaces).

    1. Select the region as **Oregon**.
    2. Click **Next** to navigate to next tab.
    3. In the **Set properties** tab, leave all options as default.
    4. In the **Set permissions** tag, leave all options as default.

5. In the **Review** tab, click on **Create Bucket**



Repartitioning the dataset using CTAS Query

1. Ensure that current AWS region is **US West (Oregon)** region
2. Ensure **mydatabase** is selected from the DATABASE list.
3. Choose **New Query**, copy the following statement anywhere into the query pane, and then choose **Run Query**.

```
CREATE TABLE ctas_nytaxride_partitioned
WITH (
      format = 'PARQUET',
      external_location = 's3://<name-of-the-bucket-your-created>/ctas_nytaxri
      partitioned_by = ARRAY['month','type','vendorid']
      )
```

```
AS select
    ratecode, passenger_count, trip_distance, fare_amount, total_amount, montl
FROM nytaxirides where year = 2016 and (vendorid = '1' or vendorid = '2')
```

Go to the Amazon S3 bucket specified as the external location and inspect the format and key structure in which the new objects are written in.

Repartitioning and Bucketing the dataset using CTAS Query

4. Choose **New Query**, copy the following statement anywhere into the query pane, and then choose **Run Query**.

```
CREATE TABLE ctas_nytaxride_bucketed_partitioned
WITH (
     format = 'PARQUET',
     external_location = 's3://<name-of-the-bucket-your-created>/ctas_nytaxri
     partitioned_by = ARRAY['month', 'type'],
     bucketed_by = ARRAY['vendorid'],
     bucket_count = 3)
AS select
    ratecode, passenger_count, trip_distance, fare_amount, total_amount,vendo
FROM nytaxirides where year = 2016
```

> **Note:** This query will take approximately 6 minutes.

Go to the Amazon S3 bucket specified as the external location and inspect the format and key structure in which the new objects are written in.

Please refer to Partitioning Vs. Bucketing for more details.

License

This library is licensed under the Apache 2.0 License.