Amazon Athena User Guide

Release 1.0

Amazon Web Services

Jan 19, 2017

1 Contents **1**

1.1 What is Amazon Athena? . . . 1
1.2 Setting Up Amazon Athena . . . 3
1.3 Getting Started . . . 6
1.4 Athena Catalog Management . . . 10
1.5 Creating Databases and Tables . . . 13
1.6 Accessing Amazon Athena with JDBC . . . 14
1.7 Supported Formats and SerDes . . . 23
1.8 Partitioning Data . . . 35
1.9 Converting to Columnar Formats . . . 39
1.10 SQL and HiveQL Reference . . . 43
1.11 Known Limitations . . . 63
1.12 Service Limits . . . 64
1.13 Document History . . . 65
1.14 About Amazon Web Services . . . 65

CHAPTER 1

Contents

1.1 What is Amazon Athena?

Amazon Athena is an interactive query service that makes it easy to analyze data directly in Amazon Simple Storage Service using standard SQL. With a few actions in the AWS Management Console, customers can point Athena at their data stored in Amazon S3 and begin using standard SQL to run ad-hoc queries and get results in seconds.

Athena is serverless, so there is no infrastructure to set up or manage, and customers pay only for the queries they run. Athena scales automatically—executing queries in parallel—so results are fast, even with large datasets and complex queries.

1.1.1 When should I use Athena?

Amazon Athena helps you analyze data stored in Amazon S3. You can use Athena to run ad-hoc queries using ANSI SQL, without the need to aggregate or load the data into Athena. You use Athena to process unstructured, semi-structured, and structured data sets. Examples include: CSV, JSON, or columnar data formats such as Apache Parquet and Apache ORC. Athena integrates with Amazon QuickSight for easy visualization. You can also use Athena to generate reports or to explore data with business intelligence tools or SQL clients, connected via a JDBC driver.

1.1.2 Accessing Athena

There are currently two ways to access Athena: using the AWS Management Console or through a JDBC connection. To get started with the console, see *Getting Started*. To learn how to use the JDBC, see *Accessing Amazon Athena with JDBC*.

1.1.3 Creating Tables

Before you can create tables, it is important to first know what is meant by the terms “database” and “table.”

What are tables?

Tables are a definition of how your data are stored. Tables are essentially metadata that describes your data in a way similar to a relation, although it is important to emphasize that tables and databases in Athena do not represent a true relational database.

What are databases?

In Athena, databases simply are a logical grouping of tables. Synonymous terms include catalog and namespace.

Athena uses an internal data catalog to store information and schemas about the databases and tables that you create for your data stored in Amazon S3. You can modify the catalog using data definition language (DDL) statements or via the AWS Management Console. Any schemas you define are automatically saved unless you explicitly delete them. Athena applies schemas on-read, which means that your table definitions are applied to your data in Amazon S3 when queries are being executed. There is no data loading or transformation required. You can delete table definitions and schema without impacting the underlying data stored on Amazon S3.

Amazon Athena uses Presto, a distributed SQL engine, to execute your queries. You define data using tables created using the Hive DDL in the Athena Query Editor in the console. There are also examples with sample data within Athena to show you how to do this. Athena also has a wizard to get you started with creating a table based on data stored in Amazon S3.

For more information, see *Creating Databases and Tables*.

1.1.4 Querying Data

You query data using the Athena Query Editor window that you used to create your table. Athena enables you to write DDL statements or SQL queries directly from the Query Editor. Results are automatically stored in Amazon S3. You can change the base prefix of where the results should be shared by choosing a setting. You also have the option to download results in CSV format. Athena supports ANSI SQL standard queries.

For more information, see *Getting Started*.

1.1.5 How to Get Started with Athena

- See the *Getting Started* tutorial for an in-depth walkthrough of how to create a table and write queries in the Athena Query Editor.
- Run the Athena on-boarding tutorial in the console. You can do this by logging into the AWS Management Console for Athena.

1.2 Setting Up Amazon Athena

If you've already signed up for Amazon Web Services (AWS), you can start using Amazon Athena immediately. If you haven't signed up for AWS, or if you need assistance querying data using Athena, first complete the tasks below:

1.2.1 Sign Up for AWS

When you sign up for AWS, your account is automatically signed up for all services in AWS, including Athena. You are charged only for the services that you use. When you use Athena, you use Amazon S3 to store your data. Athena has no AWS Free Tier pricing.

If you have an AWS account already, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

To create an AWS account

1. Open http://aws.amazon.com/, and then choose **Create an AWS Account**.
2. Follow the online instructions. Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

Note your AWS account number, because you need it for the next task.

1.2.2 Create an IAM User

An AWS Identity and Access Management (IAM) user is an account that you create to access services. It is a different user than your main AWS account. As a security best practice, we recommend that you use the IAM user's credentials to access AWS services. Create an IAM user, and then add the user to an IAM group with administrative permissions or and grant this user administrative permissions. You can then access AWS using a special URL and the credentials for the IAM user.

If you signed up for AWS but have not created an IAM user for yourself, you can create one using the IAM console. If you aren't familiar with using the console, see Working with the AWS Management Console.

To create a group for administrators

1. Sign in to the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Groups** then **Create New Group**.
3. For **Group Name**, type a name for your group, such as `Administrators`, and **Next Step**.
4. In the list of policies, select the check box next to the **AdministratorAccess** policy. You can use the **Filter** menu and the **Search** field to filter the list of policies.
5. Choose **Next Step**, then **Create Group**. Your new group is listed under **Group Name**.

To create an IAM user for yourself, add the user to the administrators group, and create a password for the user

1. In the navigation pane, choose **Users**, and then **Create New Users**.
2. For **1**, type a user name.
3. Clear the check box next to **Generate an access key for each user** and then **Create**.
4. In the list of users, select the name (not the check box) of the user you just created. You can use the **Search** field to search for the user name.
5. Choose **Groups** then **Add User to Groups**.
6. Select the check box next to the administrators and choose **Add to Groups**.
7. Choose the **Security Credentials** tab. Under **Sign-In Credentials**, choose **Manage Password**.
8. Choose **Assign a custom password**. Then type a password in the **Password** and **Confirm Password** fields. When you are finished, choose **Apply**.
9. To sign in as this new IAM user, sign out of the AWS console, then use the following URL, where `your_aws_account_id` is your AWS account number without the hyphens (for example, if your AWS account number is 1234-5678-9012, your AWS account ID is 123456789012):

   ```
   https://*your_account_alias*.signin.aws.amazon.com/console/
   ```

   It is also possible the sign-in link will use your account name instead of number. To verify the sign-in link for IAM users for your account, open the IAM console and check under **IAM users sign-in link** on the dashboard.

1.2.3 Create an IAM policy for using Athena Service

Attach the Athena managed policy to the IAM account you are using to access Athena. This policy allows the service to query Amazon S3 as well as write the results of your queries to a separate bucket. You also need to attach a custom policy for Amazon S3 buckets if you are not the account owner of the bucket.

Managed policies

This is the current managed policy, AWSQuicksightAthenaAccess, available in IAM for integrating with Amazon QuickSight:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "athena:CancelQueryExecution",
        "athena:GetCatalogs",
        "athena:GetExecutionEngine",
        "athena:GetExecutionEngines",
        "athena:GetNamespace",
```

```
        "athena:GetNamespaces",
        "athena:GetQueryExecution",
        "athena:GetQueryExecutions",
        "athena:GetQueryResults",
        "athena:GetTable",
        "athena:GetTables",
        "athena:RunQuery"
      ],
      "Resource": [ "*" ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:ListBucketMultipartUploads",
        "s3:ListMultipartUploadParts",
        "s3:AbortMultipartUpload",
        "s3:CreateBucket",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::aws-athena-query-results-*"
      ]
    }
  ]
}
```

This is the current managed policy, AmazonAthenaFullAccess, available in IAM for full access to Athena. Again, policies must be attached to users for buckets that they do not own but wish to access in Athena:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "athena:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:ListBucketMultipartUploads",
        "s3:ListMultipartUploadParts",
        "s3:AbortMultipartUpload",
```

```
        "s3:CreateBucket",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::aws-athena-query-results-*"
      ]
    }
  ]
}
```

Athena Policy Actions

The following are the descriptions for the Athena actions found in the managed policy. These allow the service to perform those specific actions on behalf of the attached user:

athena:CancelQueryExecution Cancels a query that is currently running.

athena:GetCatalogs Retrieves all available catalogs. A catalog is a collection of databases (e.g., "namespace" or "schema" can be synonymously used in place of database).

athena:GetExecutionEngine(s) Retrieves the execution engines on which Athena runs the query.

athena:GetNamespace(s) Retrieves namespace information for a given catalog. The terms "namespace","schema", and "database" are often used interchangeably to represent a collection of tables. In all cases outside of this condition key and its description, a namespace is referred to as a database.

athena:GetQueryExecution(s) Retrieves the status of one or more queries that are being executed.

athena:GetQueryResults Retrieves results of the query when it reaches a SUCCEEDED state.

athena:GetTable(s) Retrieves information about one or more tables that exist within a database.

athena:RunQuery Submits a query to Athena for execution. There is a limit of five (5) concurrent queries per account.

For information about Amazon S3 actions, see the topic called Actions for Amazon S3.

1.3 Getting Started

This tutorial walks you through using Amazon Athena to query data. You'll create a table based on sample data stored in Amazon Simple Storage Service, query the table, and check the results of the query.

The tutorial is using live resources, so you are charged for the queries that you run. You aren't charged for the sample data sets that you use, but if you upload your own data files to Amazon S3, charges do apply.

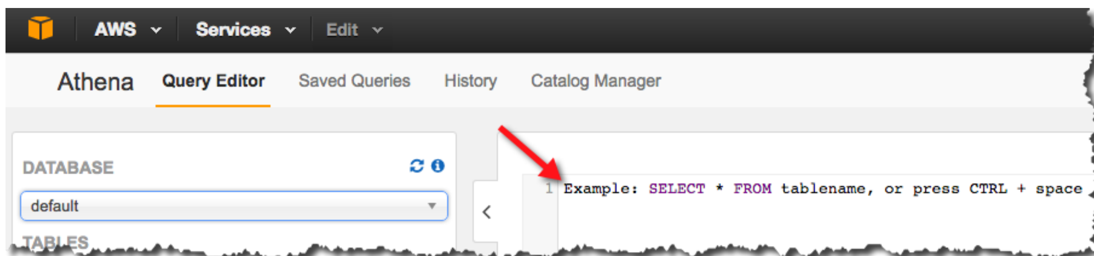
1.3.1 Prerequisites

If you have not already done so, sign up for an account in *Setting Up Amazon Athena*.

1.3.2 Step 1: Create a Database

You first need to create a database in Athena.

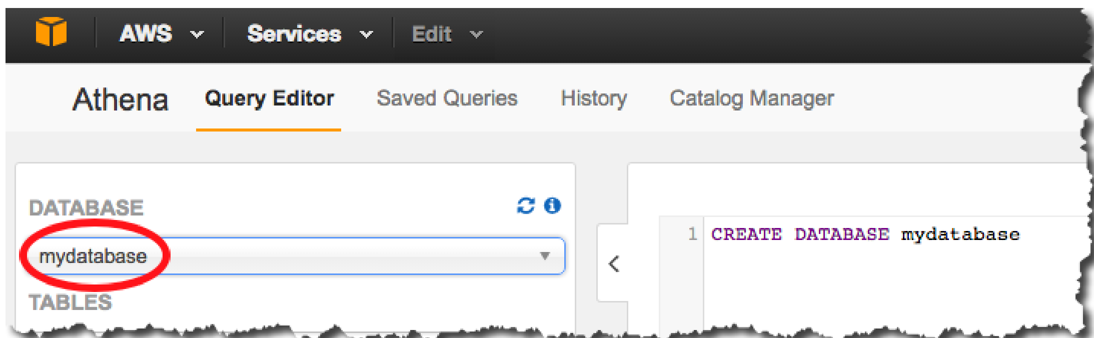
1. Open the AWS Management Console for Athena.
2. If this is your first time visiting the AWS Management Console for Athena, you'll go to a Getting Started page. Choose **Get Started** to open the Query Editor. If it isn't your first time, the Athena Query Editor opens.
3. In the Athena Query Editor, you see a query pane with an example query. Start typing your query anywhere in the query pane.



4. To create a database named `mydatabase`, enter the following CREATE DATABASE statement, and then choose *Run Query*:

```
CREATE DATABASE mydatabase
```

5. Confirm that the catalog display refreshes and `mydatabase` appears in the *DATABASE* list in the *Catalog* dashboard on the left side.



1.3.3 Step 2: Create a Table

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

1. Make sure that `mydatabase` is selected for *DATABASE* and then choose *New Query*.
2. In the query pane, enter the following CREATE TABLE statement, and then choose *Run Query*:

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
  `Date` 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://athena-examples/cloudfront/plaintext/';
```

The table is created and appears in the *Catalog* dashboard for your database.

1.3.4 Step 3: Query Data

Now that you have a table with data, you can run queries on the data and see the results in Athena.

1. Choose *New Query*, enter the following statement anywhere in the query pane, and then choose *Run Query*:

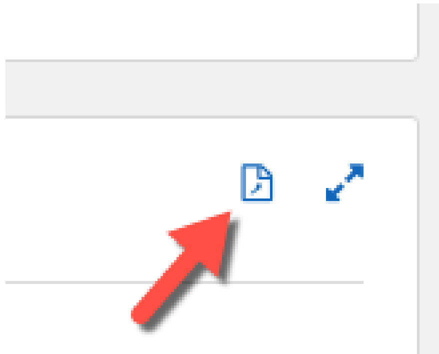
```
SELECT os, COUNT(*) count FROM cloudfront_logs WHERE date BETWEEN
↪date '2014-07-05' AND date '2014-08-05' GROUP BY os;
```

   Results are returned that look like the following:

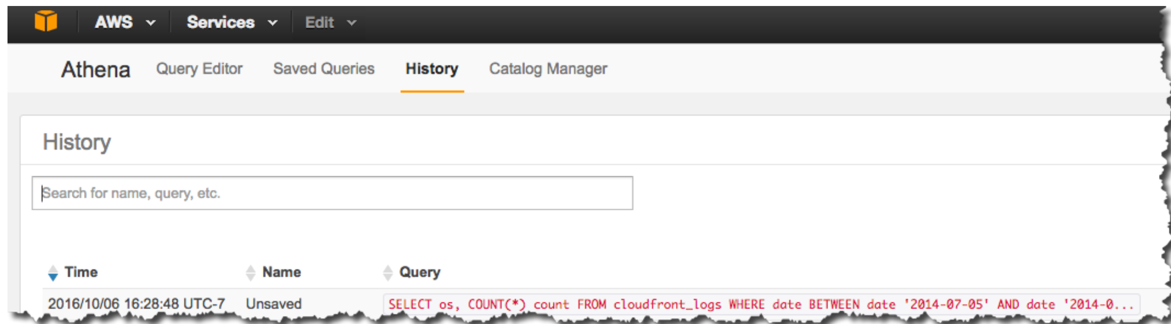
Results

	os	count
1	iOS	794
2	MacOS	852
3	OSX	799
4	Windows	883
5	Linux	813
6	Android	855

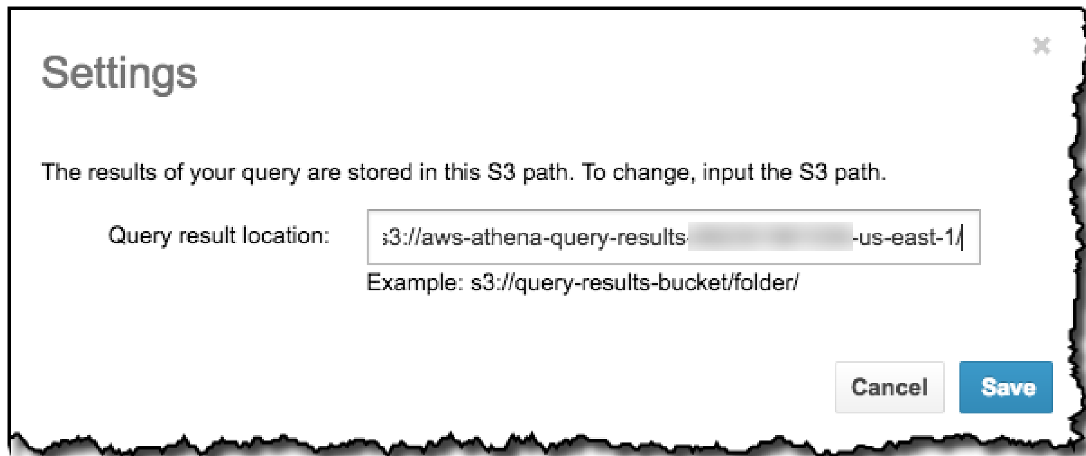
2. Optionally, you can save the results of a query to CSV by choosing the file icon on the *Results* pane.



3. You can also view the results of previous queries or queries that may take some time to complete. Choose *History* then either search for your query or choose *View* or *Download* to view or download the results of previous completed queries. This also displays the status of queries that are currently running.



Query results are also stored in Amazon S3 in a bucket called aws-athena-query-results-*ACCOUNTID-REGION*. You can change the default location in the console by choosing *Settings* in the upper right pane.

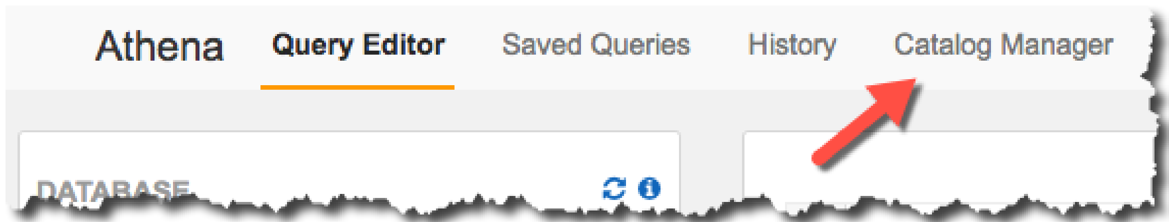


1.4 Athena Catalog Management

Amazon Athena uses an internal data catalog to store information and schemas about the databases and tables that you create for your data stored in Amazon S3. You can modify the catalog using DDL statements or via the AWS Management Console. Any schemas that you define are automatically saved unless you explicitly delete them. Athena applies schemas on-read, which means that your table definitions are applied to your data in Amazon S3 when queries are being executed. There is no data loading or transformation required. You can delete table definitions and schema without impacting the underlying data stored on Amazon S3.

1.4.1 Browse the catalog

1. Open the AWS Management Console for Athena.
2. If you have not used Athena, you see a Getting Started page. Choose *Get Started*.
3. Choose *Catalog Manager*.



4. Select a database, for example, **default**.
5. In the database, select a table. The table display shows the schema for the table on the *Columns* tab.

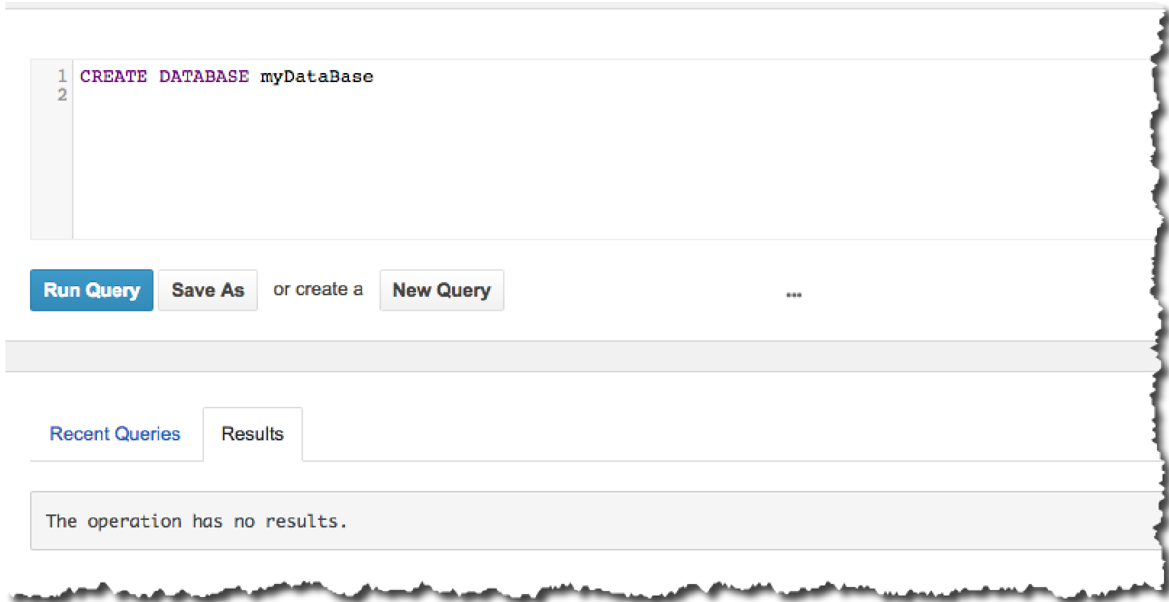
Other table information, including table data location, can be found on the *Properties* tab.

1.4.2 To create a table using the wizard

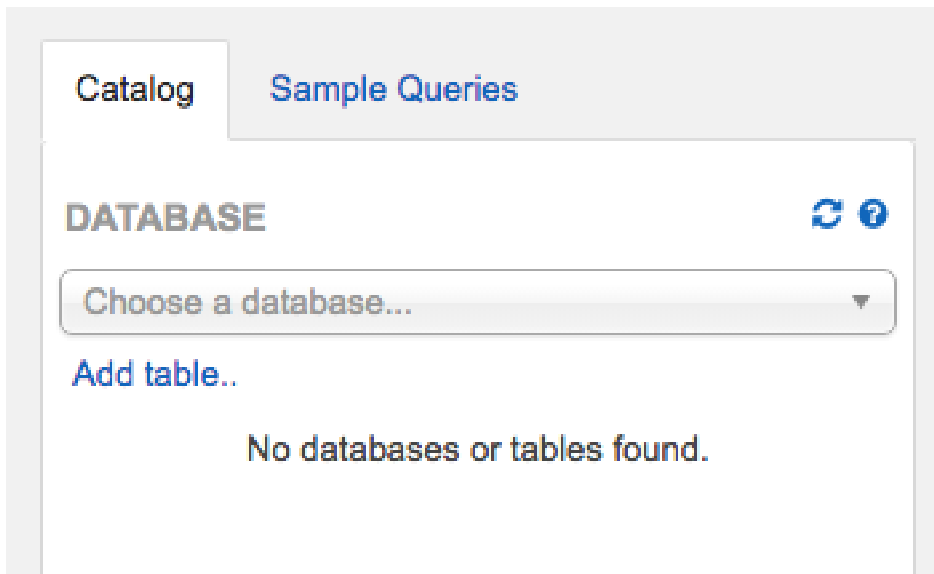
1. Open the AWS Management Console for Athena.
2. Under the database display in the Query Editor, choose *Add table*, which displays a wizard.
3. Follow the steps for creating your table.

1.4.3 To create a database using Hive DDL

1. Open the Athena console.
2. Choose *Query Editor*.
3. Enter `CREATE DATABASE myDataBase` and choose *Run Query*.



4. Select your database from the menu. It is likely to be an empty database.



1.4.4 To create a table using Hive DDL

The Athena Query Editor displays the current database. If you create a table without further qualification, the database where the table is created is the one chosen in the *Databases* section on the *Catalog* tab.

1. In the database you created in *To create a database using Hive DDL*, create a table by entering the following statement and choosing *Run Query*:

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
    `Date` 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://athena-examples/cloudfront/plaintext/';
```

2. If the table was successfully created, you can then run queries against your data. For more information, see *Getting Started*.

1.5 Creating Databases and Tables

1.5.1 Hive Data Definition Language (DDL)

Amazon Athena uses Apache Hive data definition statements to define tables. You can run DDL statements using the Athena console, via a JDBC driver, or using the Athena create table wizard. When you create a new table schema in Athena, the schema is stored in a data catalog and used when executing queries, but it does not modify your data in Amazon S3.

Athena uses an approach known as schema-on-read, which allows you to project your schema on to your data at the time you execute a query. This eliminates the need for any data loading or transformation.

Athena uses Hive to define tables and create databases, which are essentially a logical namespace of tables. This is a slightly different meaning for these terms than for traditional relational database systems, because the data is not stored with the database and in the schema defined by a table. When you create a database and table in Athena, you are simply describing the schema and where the table data are located in Amazon S3 for read-time querying.

However, when you query that data, you use standard SQL. You can find guidance for how to create databases and tables using Apache Hive documentation, but the following provides guidance specifically for Athena.

Hive supports multiple data formats through the use of serializer-deserializer (SerDes) libraries and complex schemas can be defined using regular expressions. A list of supported SerDes can be found in *Supported Formats and SerDes*.

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. The Athena catalog enables you to have this same Hive-compatible metastore in the cloud without needing to provision a cluster or RDS instance to host the metastore.

Functions Supported

The functions supported in Athena queries are those found within Presto. For more information, see Functions and Operators in the Presto documentation.

CREATE TABLE AS type statements

Athena does not support, for example, CREATE TABLE AS SELECT, which creates a table from the result of a SELECT query statement.

Transactional data transformations are not supported

Athena does not currently support transaction-based operations on table data.

Operations that change table states are ACID

When you create, update, or delete tables, those operations are guaranteed ACID-compliant. For example, if multiple users or clients attempt to create or alter an existing table at the same time, only one will be successful.

All tables are EXTERNAL

If you use CREATE TABLE without the EXTERNAL keyword, you will get an error; only tables with the EXTERNAL keyword can be created. We recommend that you always use the EXTERNAL keyword. When you drop a table in Athena, only the table metadata is removed; the data remains in Amazon S3.

1.6 Accessing Amazon Athena with JDBC

Access Amazon Athena using a Java Database Connectivity (JDBC) driver available on Amazon Simple Storage Service.

1.6.1 Athena JDBC Driver

Using this driver allows you to connect to popular third-party applications such as SQL Workbench. You can also use this driver to run queries programmatically against Athena.

For example, you cannot currently run more than one query against Athena in the AWS Management Console. However, with the JDBC driver, you can submit a script that runs more than one query. By default, you can run five (5) queries concurrently from an account. You can request a service limit increase to raise this limit.

1.6.2 Downloading the Driver

JDBC 4.1-compatible driver:
https://s3.amazonaws.com/athena-downloads/drivers/AthenaJDBC41-1.0.0.jar.

Use the AWS CLI with the following command:

```
aws s3 cp s3://athena-downloads/drivers/AthenaJDBC41-1.0.0.jar [local_
↪directory]
```

1.6.3 JDBC URL Format

The format of the JDBC connection string for Athena is the following:

> *jdbc:awsathena://athena.REGION.amazonaws.com:443*

Current REGION values are us-east-1 and us-west-2.

1.6.4 Driver Class Name

To use the driver in custom applications, you need to set up your Java class path to the location of the JAR file that you downloaded from s3://athena-downloads/drivers/ in the previous section. This makes the classes within the JAR available for use. The main JDBC driver class is `com.amazonaws.athena.jdbc.AthenaDriver`.

1.6.5 Credentials

Credentials are required to gain access to AWS services and resources, such as Athena and the Amazon S3 buckets to access.

For providing credentials in Java code you should use a class which implements the AWSCredentialsProvider interface. You would then set the JDBC property, `aws_credentials_provider_class`, equal to the class name and make sure that it is included in your classpath. To include constructor parameters, you set the JDBC property, `aws_credentials_provider_arguments`. For more information, see *Using a Credentials Provider*.

Another method to supply credentials—used in BI tools like SQL Workbench, for example—would be to supply the credentials used for the JDBC as AWS access key and AWS secret key for the JDBC properties for user and password, respectively.

1.6.6 JDBC Driver Options

You can configure the following options for the JDBC driver.

Table 1.1: JDBC Options

Property Name	Description	Default Value	Is Required
s3_staging_dir	The Amazon S3 location to which your query output is written. The JDBC driver then asks Athena to read the results and provide rows of data back to the user.	N/A	Yes
aws_credentials_provider_class	The credentials provider class name, which implements the AWSCredentialsProvider interface.	N/A	No
aws_credentials_provider_arguments	Arguments for the credentials provider constructor as comma-separated values.	N/A	No
max_error_retries	The maximum number of retries that the JDBC client attempts to make a request to Athena.	10	No
connection_timeout	The maximum amount of time, in milliseconds, to make a successful connection to Athena before an attempt is terminated.	10,000	No
socket_timeout	The maximum amount of time, in milliseconds, to wait for a socket in order to send data to Athena.	10,000	No
retry_base_delay	Minimum delay amount, in milliseconds, between retrying attempts to connect Athena.	100	No
retry_max_backoff_time	Maximum delay amount, in milliseconds, between retrying attempts to connect Athena.	1000	No
log_path	Local path of the Athena JDBC driver logs. If no log path is provided, then no log files are created.	N/A	No
log_level	Log level of the Athena JDBC driver logs. Valid values: INFO, DEBUG, WARN, ERROR, ALL, OFF, FATAL, TRACE.	N/A	No

1.6.7 Use Athena with SQL Workbench

Follow these instructions as a general guideline for how to access Athena with a JDBC driver.

Prerequisites

This tutorial assumes that:

- You have downloaded and installed SQL Workbench for your operating system.
- You have set up Athena according to *Setting Up Amazon Athena*.
- The AWS JAVA SDK is included in your classpath, specifically the `aws-java-sdk-core` module, which includes the authorization packages (com.amazonaws.auth.*) referenced in the example.

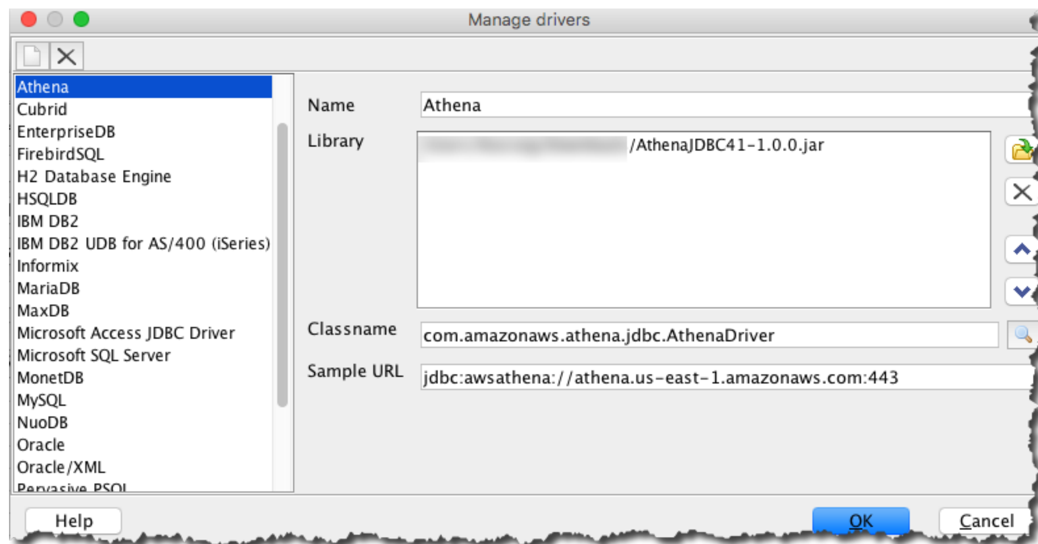
Configuring SQL Workbench

1. Download the Athena driver and place it in the SQL Workbench directory.

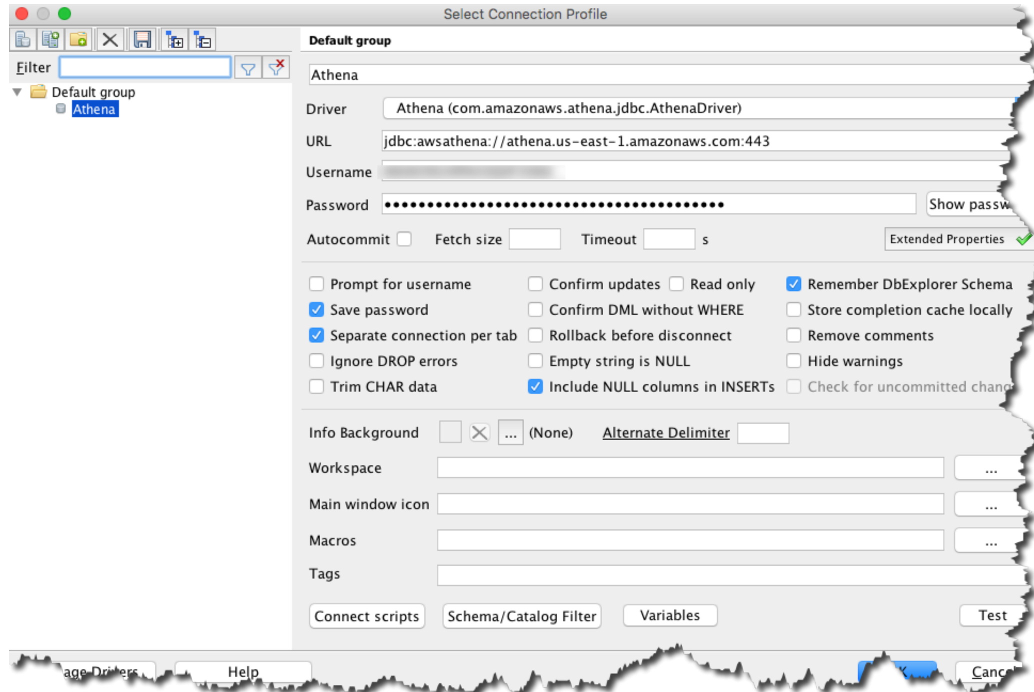
2. Open SQL Workbench.
3. Configure an Athena driver by clicking *File*, *Manage Drivers...*.
4. For *Name*, type something like "Athena JDBC Driver".
5. For *Library*, type the path for the location to which you downloaded your driver. For example, on a Linux machine, this might look like: */usr/local/SqlWorkBench-121/AthenaJDBC41-1.0.0.jar*.
6. For *Classname*, enter the full class name: *com.amazonaws.athena.jdbc.AthenaDriver*.
7. For *Sample URL*, enter the URL, replacing REGION with your desired region. Currently, the supported regions are us-east-1 and us-west-2.

   jdbc:awsathena://athena.REGION.amazonaws.com:443

8. Click *OK*.



9. Set up a connection by clicking *File*, *Connect window*.
10. Create a new connection profile and call it "Athena".
11. Under *Driver*, select the Athena driver (com.amazonaws.athena.jdbc.AthenaDriver).
12. For *URL*, enter the connection string. For example, in us-east-1, this would be *jdbc:awsathena://athena.us-east-1.amazonaws.com:443/*.
13. For *Username* and *Password*, enter your AWS access key and secret key, respectively.
14. Under *Extended Properties*, enter a desired value for *s3_staging_dir* that is in the same region where you are working, and then click *OK*. This setting is necessary to place the query results in Amazon S3 so you can download them locally using the JDBC driver. For more information about other options, see *JDBC Driver Options*.
15. You can leave other options at their default condition and click *OK*.



Querying Data

In the *Statement* window, you can enter a series of queries on your data. You can also use a CREATE statement to add new tables. The JDBC uses the *default* database but you can also create databases and use them. In this case, you should use the database identifier as a namespace prefix to your table name when writing your queries, to distinguish between tables in the default and custom databases.

Although you can enter a series of queries in the *Statement* window, keep in mind that you can only run five (5) simultaneous queries per account.

1. Create a table in the default database using the example in the Getting Started chapter. Here's the CREATE TABLE statement:

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
   `Date` 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://athena-examples/cloudfront/plaintext/';
```

Click *Execute*.

2. Run a simple query such as *SELECT DISTINCT os FROM cloudfront_logs*, and view the result.

Exploring Data

There is also a more slightly interactive way to view your Athena data using the *Database Explorer* tab. On this tab, you can select tables and view information such as the CREATE TABLE statement or a window into the data.

1. On the *Database Explorer* tab, select the *default* schema (database).

2. Select the *cloudfront_logs* table. This loads the *Columns* tab, which shows the table schema.

Columns | SQL source | Data | Indexes | References | Referenced by

COLUMN_NAME Table: "AwsDataCatalog"."default".cloudfront_log

COLUMN_NAME	DATA_TYPE	PK	NULLABLE	DEFAULT	AUTOINCREMENT	COMPUTED	REMARKS	JDBC Type	POSITION
date	date	NO	NO		NO	NO		DATE	1
time	string	NO	NO		NO	NO		LONGNVARCHAR	2
location	string	NO	NO		NO	NO		LONGNVARCHAR	3
bytes	int	NO	NO		NO	NO		INTEGER	4
requestip	string	NO	NO		NO	NO		LONGNVARCHAR	5
method	string	NO	NO		NO	NO		LONGNVARCHAR	6
host	string	NO	NO		NO	NO		LONGNVARCHAR	7
uri	string	NO	NO		NO	NO		LONGNVARCHAR	8
status	int	NO	NO		NO	NO		INTEGER	9
referrer	string	NO	NO		NO	NO		LONGNVARCHAR	10
os	string	NO	NO		NO	NO		LONGNVARCHAR	11
browser	string	NO	NO		NO	NO		LONGNVARCHAR	12
	strin	NO	NO		N			LONGNVARCHAR	13

3. Open other tabs. Two other tabs are of immediate interest:

   - *SQL Source* shows something similar to the output of *EXPLAIN CREATE TABLE cloudfront_logs*

Columns | SQL source | Dat

Edit in

```
1  DROP EXTERNAL_TABLE "AwsDataCatalog"."default".cloudfront_logs;
2
3  CREATE EXTERNAL_TABLE "AwsDataCatalog"."default".cloudfront_logs
4  (
5      date            date      NOT NULL,
6      time            string    NOT NULL,
7      location        string    NOT NULL,
8      bytes           int       NOT NULL,
9      requestip       string    NOT NULL,
10     method          string    NOT NULL,
11     host            string    NOT NULL,
12     uri             string    NOT NULL,
13     status          int       NOT NULL,
14     referrer        string    NOT NULL,
15     os              string    NOT NULL,
16     browser         string    NOT NULL,
17     browserversion  string    NOT NULL
```

- *Data* shows rows returned from your table. It may take time to load the data.

Columns | SQL source | Data | Indexes | References | Referenced by | Triggers

Table: "AwsDataCatalog"."default".cloudfront_logs Rows: 4996 Autoload

date	time	location	bytes	requestip	method	host	uri	status	referrer	os	browser	br
2014-07-05	15:00:00	LHR3	4260	10.0.0.15	GET	eabcd12345678.cloudfront.net	/test-image-1.jpeg	200	-	Linux	Opera	3.0
2014-07-05	15:00:00	MIA3	10	10.0.0.15	GET	eabcd12345678.cloudfront.net	/test-image-1.jpeg	304	-	OSX	Firefox	3.0
2014-07-05	15:00:00	MIA3	4252	10.0.0.15	GET	eabcd12345678.cloudfront.net	/test-image-3.jpeg	200	-	Linux	Lynx	3.0
2014-07-05	15:00:00	FRA2	4257	10.0.0.8	GET	eabcd12345678.cloudfront.net	/test-image-2.jpeg	200	-	iOS	Firefox	3.0
2014-07-05	15:00:03	HKG1	4261	10.0.0.15	GET	eabcd12345678.cloudfront.net	/test-image-2.jpeg	200	-	Linux	Opera	3.
2014-07-05	15:00:03	HKG1	4252	10.0.0.15	GET	eabcd12345678.cloudfront.net	/test-image-1.jpeg	200	-	Windows	IE	3.0
2014-07-05	15:00:04	MIA3	4257	10.0.0.12	GET	eabcd12345678.cloudfront.net	/test-image-3.jpeg	200	-	MacOS	Opera	3.0
2014-07-05	15:00:04	LAX1	4261	10.0.0.15	GET	eabcd12345678.cloudfront.net	/test-image-1.jpeg	200	-	Android	Firefox	3.0
2014-07-05	15:00:04	SFO4	4252	10.0.0.15	GET	eabcd12345678.cloudfront.net	/test-image-2.jpeg	200	-	OSX	Safari	3.
2014-07-05	15:00:04	LAX1	1	10.0.0.15	GET	eabcd12345678.cloudfront.net	/test-image-1.jpeg	304	-	Linux	Opera	
2014-07-05	15:00:05	MIA3	4252	10.0.0.15	GET	eabcd12345678.cloudfront.net	/test-image-2.jpeg	200	-	MacOS	Opera	
2014-07-05	15:00:05	MIA3	4260	10.0.0.15	GET	eabcd12345678.cloudfront.net	/test-image-1.jpeg	200	-	Android	Firefox	3.
2014-07-05	15:00:05	SFO4	10	10.0.0.15	GET	eabcd12345678.cloudfront.net	/test-image-1.jpeg	304	-	MacOS	Firefox	3.
2014-07-05	15:00:06	DUB2	6	10.0.0.3	GET	eabcd12345678.cloudfront.net	/test-image-3.jpeg	304	-	iOS	Safari	

1.6.8 Using the JDBC Driver with the JDK

The following code examples demonstrate how to use the JDBC driver in an application.

Creating a Driver

```
Properties info = new Properties();
info.put("user", "AWSAccessKey");
info.put("password", "AWSSecretAccessKey");
info.put("s3_staging_dir", "s3://S3 Bucket Location/");
```

```
Class.forName("com.amazonaws.athena.jdbc.AthenaDriver");

Connection connection = DriverManager.getConnection("jdbc:awsathena://athena.
↪us-east-1.amazonaws.com:443/", info);
```

Using a Credentials Provider

The following examples demonstrate different ways to use a credentials provider that implements the AWSCredentialsProvider interface with the JDBC.

Credentials provider with a single argument

```
Properties myProps = new Properties();
myProps.put("aws_credentials_provider_class","com.amazonaws.auth.
↪PropertiesFileCredentialsProvider");
myProps.put("aws_credentials_provider_arguments","/Users/myUser/.
↪athenaCredentials");
```

In this case, the filename called `/Users/myUser/.athenaCredentials` should contain the following:

```
accessKey = ACCESSKEY
secretKey = SECRETKEY
```

You replace the right hand part of the assignments with your account's AWS access and secret keys.

Credentials provider with a multiple arguments

This example shows a credentials provider that uses an access and secret key as well as a session token, for example. The signature of the class looks like the following:

```
public CustomSessionCredentialsProvider(String accessId, String secretKey,␣
↪String token)
{
  //...
}
```

You would then set the properties as follows:

```
Properties myProps = new Properties();
myProps.put("aws_credentials_provider_class","com.amazonaws.athena.jdbc.
↪CustomSessionCredentialsProvider");
String providerArgs = "My_Access_Key," + "My_Secret_Key," + "My_Token";
myProps.put("aws_credentials_provider_arguments",providerArgs);
```

Using InstanceProfileCredentialsProvider

If you use the InstanceProfileCredentialsProvider, you don't need to supply any credential provider arguments because they are provided using the EC2 instance profile for the instance on which you are running your application. You would still set the `aws_credentials_provider_class` property to this class name, however.

Executing a SELECT Query

```
Statement statement = connection.createStatement();
ResultSet queryResults = statement.executeQuery("SELECT * FROM cloudfront_logs
↪");
```

Executing CREATE/ALTER Statements

```
Statement statement = connection.createStatement();
ResultSet queryResults = statement.executeQuery("CREATE EXTERNAL TABLE␣
↪tableName ( Col1 String ) LOCATION 's3://bucket/tableLocation");
```

Full Example Listing Tables

```
import java.sql.*;
import java.util.Properties;
import com.amazonaws.athena.jdbc.AthenaDriver;
import com.amazonaws.auth.PropertiesFileCredentialsProvider;


public class AthenaJDBCDemo {

  static final String athenaUrl = "jdbc:awsathena://athena.us-east-1.
↪amazonaws.com:443";

  public static void main(String[] args) {

      Connection conn = null;
      Statement statement = null;

      try {
          Class.forName("com.amazonaws.athena.jdbc.AthenaDriver");
          Properties info = new Properties();
          info.put("s3_staging_dir", "s3://my-athena-result-bucket/test/");
          info.put("log_path", "/Users/myUser/.athena/athenajdbc.log");
          info.put("aws_credentials_provider_class","com.amazonaws.auth.
↪PropertiesFileCredentialsProvider");
          info.put("aws_credentials_provider_arguments","/Users/myUser/.
↪athenaCredentials");
          String databaseName = "default";

          System.out.println("Connecting to Athena...");
          conn = DriverManager.getConnection(athenaUrl, info);

          System.out.println("Listing tables...");
          String sql = "show tables in "+ databaseName;
          statement = conn.createStatement();
          ResultSet rs = statement.executeQuery(sql);
```

```
            while (rs.next()) {
                //Retrieve table column.
                String name = rs.getString("tab_name");

                //Display values.
                System.out.println("Name: " + name);
            }
            rs.close();
            conn.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            try {
                if (statement != null)
                    statement.close();
            } catch (Exception ex) {

            }
            try {
                if (conn != null)
                    conn.close();
            } catch (Exception ex) {

                ex.printStackTrace();
            }
        }
        System.out.printf("Finished connectivity test.");
    }
}
```

1.7 Supported Formats and SerDes

The data formats presently supported are:

1.7.1 CSV

SerDe Name

CSV SerDe

OpenCSVSerde is not supported.

Library Name

org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

Examples

DDL:

```
DROP TABLE flight_delays_csv;

CREATE EXTERNAL TABLE flight_delays_csv (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
    carrier STRING,
    tailnum STRING,
    flightnum STRING,
    originairportid INT,
    originairportseqid INT,
    origincitymarketid INT,
    origin STRING,
    origincityname STRING,
    originstate STRING,
    originstatefips STRING,
    originstatename STRING,
    originwac INT,
    destairportid INT,
    destairportseqid INT,
    destcitymarketid INT,
    dest STRING,
    destcityname STRING,
    deststate STRING,
    deststatefips STRING,
    deststatename STRING,
    destwac INT,
    crsdeptime STRING,
    deptime STRING,
    depdelay INT,
    depdelayminutes INT,
    depdel15 INT,
    departuredelaygroups INT,
    deptimeblk STRING,
    taxiout INT,
    wheelsoff STRING,
    wheelson STRING,
    taxiin INT,
    crsarrtime INT,
    arrtime STRING,
    arrdelay INT,
    arrdelayminutes INT,
    arrdel15 INT,
    arrivaldelaygroups INT,
```

```
    arrtimeblk STRING,
    cancelled INT,
    cancellationcode STRING,
    diverted INT,
    crselapsedtime INT,
    actualelapsedtime INT,
    airtime INT,
    flights INT,
    distance INT,
    distancegroup INT,
    carrierdelay INT,
    weatherdelay INT,
    nasdelay INT,
    securitydelay INT,
    lateaircraftdelay INT,
    firstdeptime STRING,
    totaladdgtime INT,
    longestaddgtime INT,
    divairportlandings INT,
    divreacheddest INT,
    divactualelapsedtime INT,
    divarrdelay INT,
    divdistance INT,
    div1airport STRING,
    div1airportid INT,
    div1airportseqid INT,
    div1wheelson STRING,
    div1totalgtime INT,
    div1longestgtime INT,
    div1wheelsoff STRING,
    div1tailnum STRING,
    div2airport STRING,
    div2airportid INT,
    div2airportseqid INT,
    div2wheelson STRING,
    div2totalgtime INT,
    div2longestgtime INT,
    div2wheelsoff STRING,
    div2tailnum STRING,
    div3airport STRING,
    div3airportid INT,
    div3airportseqid INT,
    div3wheelson STRING,
    div3totalgtime INT,
    div3longestgtime INT,
    div3wheelsoff STRING,
    div3tailnum STRING,
    div4airport STRING,
    div4airportid INT,
    div4airportseqid INT,
    div4wheelson STRING,
    div4totalgtime INT,
    div4longestgtime INT,
```

```
    div4wheelsoff STRING,
    div4tailnum STRING,
    div5airport STRING,
    div5airportid INT,
    div5airportseqid INT,
    div5wheelson STRING,
    div5totalgtime INT,
    div5longestgtime INT,
    div5wheelsoff STRING,
    div5tailnum STRING
)
    PARTITIONED BY (year STRING)
    ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ','
      ESCAPED BY '\\'
      LINES TERMINATED BY '\n'
    LOCATION 's3://athena-examples/flight/csv/';

MSCK REPAIR TABLE flight_delays_csv;
```

Query: Top 10 routes delayed by more than 1 hour

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_csv
WHERE depdelayminutes > 60
GROUP BY origin, dest
ORDER BY 3 DESC
LIMIT 10;
```

Note: The flight table data comes from Flights provided by US Department of Transportation, Bureau of Transportation Statistics. Desaturated from original.

1.7.2 TSV

SerDe Name

TSV SerDe

Library Name

org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

1.7.3 Parquet Serde

SerDe Name

Parquet SerDe

Library Name

org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe

Examples

DDL:

```
DROP TABLE flight_delays_pq;
CREATE EXTERNAL TABLE flight_delays_pq (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
    carrier STRING,
    tailnum STRING,
    flightnum STRING,
    originairportid INT,
    originairportseqid INT,
    origincitymarketid INT,
    origin STRING,
    origincityname STRING,
    originstate STRING,
    originstatefips STRING,
    originstatename STRING,
    originwac INT,
    destairportid INT,
    destairportseqid INT,
    destcitymarketid INT,
    dest STRING,
    destcityname STRING,
    deststate STRING,
    deststatefips STRING,
    deststatename STRING,
    destwac INT,
    crsdeptime STRING,
    deptime STRING,
    depdelay INT,
    depdelayminutes INT,
    depdel15 INT,
    departuredelaygroups INT,
```

```
    deptimeblk STRING,
    taxiout INT,
    wheelsoff STRING,
    wheelson STRING,
    taxiin INT,
    crsarrtime INT,
    arrtime STRING,
    arrdelay INT,
    arrdelayminutes INT,
    arrdel15 INT,
    arrivaldelaygroups INT,
    arrtimeblk STRING,
    cancelled INT,
    cancellationcode STRING,
    diverted INT,
    crselapsedtime INT,
    actualelapsedtime INT,
    airtime INT,
    flights INT,
    distance INT,
    distancegroup INT,
    carrierdelay INT,
    weatherdelay INT,
    nasdelay INT,
    securitydelay INT,
    lateaircraftdelay INT,
    firstdeptime STRING,
    totaladdgtime INT,
    longestaddgtime INT,
    divairportlandings INT,
    divreacheddest INT,
    divactualelapsedtime INT,
    divarrdelay INT,
    divdistance INT,
    div1airport STRING,
    div1airportid INT,
    div1airportseqid INT,
    div1wheelson STRING,
    div1totalgtime INT,
    div1longestgtime INT,
    div1wheelsoff STRING,
    div1tailnum STRING,
    div2airport STRING,
    div2airportid INT,
    div2airportseqid INT,
    div2wheelson STRING,
    div2totalgtime INT,
    div2longestgtime INT,
    div2wheelsoff STRING,
    div2tailnum STRING,
    div3airport STRING,
    div3airportid INT,
    div3airportseqid INT,
```

```
    div3wheelson STRING,
    div3totalgtime INT,
    div3longestgtime INT,
    div3wheelsoff STRING,
    div3tailnum STRING,
    div4airport STRING,
    div4airportid INT,
    div4airportseqid INT,
    div4wheelson STRING,
    div4totalgtime INT,
    div4longestgtime INT,
    div4wheelsoff STRING,
    div4tailnum STRING,
    div5airport STRING,
    div5airportid INT,
    div5airportseqid INT,
    div5wheelson STRING,
    div5totalgtime INT,
    div5longestgtime INT,
    div5wheelsoff STRING,
    div5tailnum STRING
)
PARTITIONED BY (year STRING)
STORED AS PARQUET
LOCATION 's3://athena-examples/flight/parquet/'
tblproperties ("parquet.compress"="SNAPPY");

MSCK REPAIR TABLE flight_delays_pq;
```

Query:

Top 10 routes delayed by more than 1 hour

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_pq
WHERE depdelayminutes > 60
GROUP BY origin, dest
ORDER BY 3 DESC
LIMIT 10;
```

Note: The flight table data comes from Flights provided by US Department of Transportation, Bureau of Transportation Statistics. Desaturated from original.

1.7.4 ORC

SerDe Name

OrcSerDe

Library Name

org.apache.hadoop.hive.ql.io.orc.OrcSerde

Examples

```
DROP TABLE flight_delays_orc;
CREATE EXTERNAL TABLE flight_delays_orc (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
    carrier STRING,
    tailnum STRING,
    flightnum STRING,
    originairportid INT,
    originairportseqid INT,
    origincitymarketid INT,
    origin STRING,
    origincityname STRING,
    originstate STRING,
    originstatefips STRING,
    originstatename STRING,
    originwac INT,
    destairportid INT,
    destairportseqid INT,
    destcitymarketid INT,
    dest STRING,
    destcityname STRING,
    deststate STRING,
    deststatefips STRING,
    deststatename STRING,
    destwac INT,
    crsdeptime STRING,
    deptime STRING,
    depdelay INT,
    depdelayminutes INT,
    depdel15 INT,
    departuredelaygroups INT,
    deptimeblk STRING,
    taxiout INT,
    wheelsoff STRING,
    wheelson STRING,
    taxiin INT,
    crsarrtime INT,
    arrtime STRING,
    arrdelay INT,
    arrdelayminutes INT,
```

```
    arrdel15 INT,
    arrivaldelaygroups INT,
    arrtimeblk STRING,
    cancelled INT,
    cancellationcode STRING,
    diverted INT,
    crselapsedtime INT,
    actualelapsedtime INT,
    airtime INT,
    flights INT,
    distance INT,
    distancegroup INT,
    carrierdelay INT,
    weatherdelay INT,
    nasdelay INT,
    securitydelay INT,
    lateaircraftdelay INT,
    firstdeptime STRING,
    totaladdgtime INT,
    longestaddgtime INT,
    divairportlandings INT,
    divreacheddest INT,
    divactualelapsedtime INT,
    divarrdelay INT,
    divdistance INT,
    div1airport STRING,
    div1airportid INT,
    div1airportseqid INT,
    div1wheelson STRING,
    div1totalgtime INT,
    div1longestgtime INT,
    div1wheelsoff STRING,
    div1tailnum STRING,
    div2airport STRING,
    div2airportid INT,
    div2airportseqid INT,
    div2wheelson STRING,
    div2totalgtime INT,
    div2longestgtime INT,
    div2wheelsoff STRING,
    div2tailnum STRING,
    div3airport STRING,
    div3airportid INT,
    div3airportseqid INT,
    div3wheelson STRING,
    div3totalgtime INT,
    div3longestgtime INT,
    div3wheelsoff STRING,
    div3tailnum STRING,
    div4airport STRING,
    div4airportid INT,
    div4airportseqid INT,
    div4wheelson STRING,
```

```
    div4totalgtime INT,
    div4longestgtime INT,
    div4wheelsoff STRING,
    div4tailnum STRING,
    div5airport STRING,
    div5airportid INT,
    div5airportseqid INT,
    div5wheelson STRING,
    div5totalgtime INT,
    div5longestgtime INT,
    div5wheelsoff STRING,
    div5tailnum STRING
)
PARTITIONED BY (year String)
STORED AS ORC
LOCATION 's3://athena-examples/flight/orc/'
tblproperties ("parquet.compress"="ZLIB");

MSCK REPAIR TABLE flight_delays_orc
```

Query:

Top 10 routes delayed by more than 1 hour

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_pq
WHERE depdelayminutes > 60
GROUP BY origin, dest
ORDER BY 3 DESC
LIMIT 10
```

1.7.5 JSON

There are two SerDes for JSON: the native Hive/HCatalog JsonSerde and the OpenX SerDe

SerDe Names

Hive-JsonSerDe

Openx-JsonSerDe

Library Name

Use one of the following:

org.apache.hive.hcatalog.data.JsonSerDe

org.openx.data.jsonserde.JsonSerDe

Examples

The following DDL statement uses the Hive JsonSerde:

```
CREATE EXTERNAL TABLE impressions (
    requestBeginTime string,
    adId string,
    impressionId string,
    referrer string,
    userAgent string,
    userCookie string,
    ip string,
    number string,
    processId string,
    browserCookie string,
    requestEndTime string,
    timers struct<modelLookup:string, requestTime:string>,
    threadId string, hostname string,
    sessionId string
)   PARTITIONED BY (dt string)
ROW FORMAT  serde 'org.apache.hive.hcatalog.data.JsonSerDe'
with serdeproperties ( 'paths'='requestBeginTime, adId, impressionId,
↪ referrer, userAgent, userCookie, ip' )
LOCATION 's3://REGION.elasticmapreduce/samples/hive-ads/tables/
↪impressions';
```

The following DDL statement replaces the Hive Serde with the OpenX Serde:

```
CREATE EXTERNAL TABLE impressions (
    requestBeginTime string,
    adId string,
    impressionId string,
    referrer string,
    userAgent string,
    userCookie string,
    ip string,
    number string,
    processId string,
    browserCookie string,
    requestEndTime string,
    timers struct<modelLookup:string, requestTime:string>,
    threadId string, hostname string,
    sessionId string
)   PARTITIONED BY (dt string)
ROW FORMAT  serde 'org.openx.data.jsonserde.JsonSerDe'
with serdeproperties ( 'paths'='requestBeginTime, adId, impressionId,
↪ referrer, userAgent, userCookie, ip' )
LOCATION 's3://REGION.elasticmapreduce/samples/hive-ads/tables/
↪impressions';
```

1.7.6 Apache Web Server logs (RegexSerDe)

SerDe Name

RegexSerDe (Apache Web Server logs)

Library Name

org.apache.hadoop.hive.serde2.RegexSerDe

Examples

Here's an example for creating a table from AWS CloudFront logs using the RegExSerDe from the Getting Started tutorial:

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
  `Date` 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://athena-examples/cloudfront/plaintext/';
```

1.7.7 Custom Delimiters

The SerDe library, InputFormat, and OutputFormat for Custom Delimiters are listed below.

SerDe Name

Custom Delimiters

Library Name

org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

In order to use these formats you must specify a serializer-deserializer class (SerDe) so that Athena knows the format of the table. A SerDe is a custom library that tells the Athena-managed Catalog how to handle your data.

1.7.8 Compression Formats

Currently, Snappy, Zlib, and GZIP are the supported compression formats. LZO is not supported.

1.8 Partitioning Data

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.

To create a table with partitions, you must define it during the CREATE TABLE statement. Use PARTITIONED BY to define the keys you want to partition data by. There are two scenarios discussed below:

1. Data is already partitioned, stored on Amazon S3, and you need to access the data on Athena.
2. Data is not partitioned.

1.8.1 Scenario 1: Data already partitioned and stored on S3 in hive format

Storing Partitioned Data

Partitions are stored in separate folders in Amazon S3. For example, here is the partial listing for sample ad impressions:

```
% aws s3 ls s3://elasticmapreduce/samples/hive-ads/tables/impressions/

    PRE dt=2009-04-12-13-00/
    PRE dt=2009-04-12-13-05/
    PRE dt=2009-04-12-13-10/
    PRE dt=2009-04-12-13-15/
    PRE dt=2009-04-12-13-20/
    PRE dt=2009-04-12-14-00/
    PRE dt=2009-04-12-14-05/
    PRE dt=2009-04-12-14-10/
    PRE dt=2009-04-12-14-15/
    PRE dt=2009-04-12-14-20/
```

```
    PRE dt=2009-04-12-15-00/
    PRE dt=2009-04-12-15-05/
```

Here, logs are stored with the column name (dt) set equal to date, hour, and minute increments. When you give a DDL with the location of the parent folder, the schema, and the name of the partitioned column, Athena can query data in those subfolders.

Creating a Table

To make a table out of this data, create a partition along 'dt' as in the following Athena DDL statement:

```
CREATE EXTERNAL TABLE impressions (
    requestBeginTime string,
    adId string,
    impressionId string,
    referrer string,
    userAgent string,
    userCookie string,
    ip string,
    number string,
    processId string,
    browserCookie string,
    requestEndTime string,
    timers struct<modelLookup:string, requestTime:string>,
    threadId string,
    hostname string,
    sessionId string)
PARTITIONED BY (dt string)
ROW FORMAT  serde 'org.apache.hive.hcatalog.data.JsonSerDe'
    with serdeproperties ( 'paths'='requestBeginTime, adId, impressionId,␣
↪referrer, userAgent, userCookie, ip' )
LOCATION 's3://elasticmapreduce/samples/hive-ads/tables/impressions/' ;
```

This table uses Hive's native JSON serializer-deserializer to read JSON data stored in Amazon S3. For more information about the formats supported, see *Supported Formats and SerDes*.

After you execute this statement in Athena, choose **New Query** then execute:

```
MSCK REPAIR TABLE impressions
```

Athena loads the data in the partitions.

Query the Data

Now, query the data from the impressions table using the partition column. Here's an example:

```
SELECT dt,impressionid FROM impressions WHERE dt<'2009-04-12-14-00' and dt>=
↪'2009-04-12-13-00' ORDER BY dt DESC LIMIT 100
```

This query should show you data similar to the following:

```
2009-04-12-13-20    ap3HcVKAWfXtgIPu6WpuUfAfL0DQEc
2009-04-12-13-20    17uchtodoS9kdeQP1x0XThKl5IuRsV
2009-04-12-13-20    JOUf1SCtRwviGw8sVcghqE5h0nkgtp
2009-04-12-13-20    NQ2XP0J0dvVbCXJ0pb4XvqJ5A4QxxH
2009-04-12-13-20    fFAItiBMsgqro9kRdIwbeX60SROaxr
2009-04-12-13-20    V4og4R9W6G3QjHHwF7gI1cSqig5D1G
2009-04-12-13-20    hPEPtBwk45msmwWTxPVVo1kVu4v11b
2009-04-12-13-20    v0SkfxegheD90gp31UCr6FplnKpx6i
2009-04-12-13-20    1iD9odVgOIi4QWkwHMcOhmwTkWDKfj
2009-04-12-13-20    b31tJiIA25CK8eDHQrHnbcknfSndUk
```

1.8.2 Scenario 2: Data is not partitioned

A layout like the following does not, however, work for automatically adding partition data with MSCK REPAIR TABLE:

```
aws s3 ls s3://athena-examples/elb/plaintext/ --recursive

2016-11-23 17:54:46   11789573 elb/plaintext/2015/01/01/part-r-00000-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:46    8776899 elb/plaintext/2015/01/01/part-r-00001-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:46    9309800 elb/plaintext/2015/01/01/part-r-00002-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47    9412570 elb/plaintext/2015/01/01/part-r-00003-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47   10725938 elb/plaintext/2015/01/01/part-r-00004-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:46    9439710 elb/plaintext/2015/01/01/part-r-00005-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47          0 elb/plaintext/2015/01/01_$folder$
2016-11-23 17:54:47    9012723 elb/plaintext/2015/01/02/part-r-00006-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47    7571816 elb/plaintext/2015/01/02/part-r-00007-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47    9673393 elb/plaintext/2015/01/02/part-r-00008-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48   11979218 elb/plaintext/2015/01/02/part-r-00009-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48    9546833 elb/plaintext/2015/01/02/part-r-00010-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48   10960865 elb/plaintext/2015/01/02/part-r-00011-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48          0 elb/plaintext/2015/01/02_$folder$
2016-11-23 17:54:48   11360522 elb/plaintext/2015/01/03/part-r-00012-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48   11211291 elb/plaintext/2015/01/03/part-r-00013-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48    8633768 elb/plaintext/2015/01/03/part-r-00014-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49   11891626 elb/plaintext/2015/01/03/part-r-00015-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
```

```
2016-11-23 17:54:49    9173813 elb/plaintext/2015/01/03/part-r-00016-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49   11899582 elb/plaintext/2015/01/03/part-r-00017-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49          0 elb/plaintext/2015/01/03_$folder$
2016-11-23 17:54:50    8612843 elb/plaintext/2015/01/04/part-r-00018-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50   10731284 elb/plaintext/2015/01/04/part-r-00019-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50    9984735 elb/plaintext/2015/01/04/part-r-00020-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50    9290089 elb/plaintext/2015/01/04/part-r-00021-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50    7896339 elb/plaintext/2015/01/04/part-r-00022-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    8321364 elb/plaintext/2015/01/04/part-r-00023-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51          0 elb/plaintext/2015/01/04_$folder$
2016-11-23 17:54:51    7641062 elb/plaintext/2015/01/05/part-r-00024-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51   10253377 elb/plaintext/2015/01/05/part-r-00025-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    8502765 elb/plaintext/2015/01/05/part-r-00026-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51   11518464 elb/plaintext/2015/01/05/part-r-00027-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    7945189 elb/plaintext/2015/01/05/part-r-00028-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    7864475 elb/plaintext/2015/01/05/part-r-00029-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51          0 elb/plaintext/2015/01/05_$folder$
2016-11-23 17:54:51   11342140 elb/plaintext/2015/01/06/part-r-00030-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    8063755 elb/plaintext/2015/01/06/part-r-00031-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    9387508 elb/plaintext/2015/01/06/part-r-00032-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    9732343 elb/plaintext/2015/01/06/part-r-00033-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52   11510326 elb/plaintext/2015/01/06/part-r-00034-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    9148117 elb/plaintext/2015/01/06/part-r-00035-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52          0 elb/plaintext/2015/01/06_$folder$
2016-11-23 17:54:52    8402024 elb/plaintext/2015/01/07/part-r-00036-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    8282860 elb/plaintext/2015/01/07/part-r-00037-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52   11575283 elb/plaintext/2015/01/07/part-r-00038-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53    8149059 elb/plaintext/2015/01/07/part-r-00039-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53   10037269 elb/plaintext/2015/01/07/part-r-00040-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
```

```
2016-11-23 17:54:53   10019678 elb/plaintext/2015/01/07/part-r-00041-ce65fca5-
↪d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53          0 elb/plaintext/2015/01/07_$folder$
2016-11-23 17:54:53          0 elb/plaintext/2015/01_$folder$
2016-11-23 17:54:53          0 elb/plaintext/2015_$folder$
```

In this case, you would have to use ALTER TABLE ADD PARTITION to add each partition manually.

For example to load the data in s3://athena-examples/elb/plaintext/2015/01/01/, you can run the following:

```
ALTER TABLE elb_logs_raw_native_part ADD PARTITION (year='2015',month='01',
↪day='01') location 's3://athena-examples/elb/plaintext/2015/01/01/'
```

You can also automate adding partitions by using the *JDBC driver*.

1.9 Converting to Columnar Formats

Your Amazon Athena query performance improves if you convert your data into open source columnar formats such as Apache Parquet or ORC.

You can do this to existing Amazon S3 data sources by creating a cluster in Amazon EMR and converting it using Hive. The following example using the AWS CLI shows you how to do this with a script and data stored in Amazon S3.

1.9.1 Overview

The process for converting to columnar formats using an EMR cluster is as follows:

- Create an EMR cluster with Hive installed.
- In the step section of the cluster create statement, you can specify a script stored in Amazon S3, which points to your input data and creates output data in the columnar format in an Amazon S3 location. In this example, the cluster auto-terminates.

  For more information, here's an example script beginning with the CREATE TABLE snippet:

```
ADD JAR /usr/lib/hive-hcatalog/share/hcatalog/hive-hcatalog-core-1.0.0-
↪amzn-5.jar;
CREATE EXTERNAL TABLE impressions (
  requestBeginTime string,
  adId string,
  impressionId string,
  referrer string,
  userAgent string,
  userCookie string,
  ip string,
  number string,
  processId string,
  browserCookie string,
  requestEndTime string,
```

```
  timers struct<modelLookup:string, requestTime:string>,
  threadId string,
  hostname string,
  sessionId string)
PARTITIONED BY (dt string)
ROW FORMAT  serde 'org.apache.hive.hcatalog.data.JsonSerDe'
with serdeproperties ( 'paths'='requestBeginTime, adId, impressionId,␣
↪referrer, userAgent, userCookie, ip' )
LOCATION 's3://${REGION}.elasticmapreduce/samples/hive-ads/tables/
↪impressions' ;
```

Note: Replace `REGION` in the `LOCATION` clause with the region where you are running queries. For example, if your console is in us-east-1, `REGION` will be `s3://us-east-1.elasticmapreduce/samples/hive-ads/tables/`.

This creates the table in Hive on the cluster which uses samples located in the Amazon EMR samples bucket. On Amazon EMR release 4.7.0, you need to include the ADD JAR line to find the appropriate JsonSerDe. The prettified sample data looks like the following:


```
{
    "number": "977680",
    "referrer": "fastcompany.com",
    "processId": "1823",
    "adId": "TRktxshQXAHWo261jAHubijAoNlAqA",
    "browserCookie": "mvlrdwrmef",
    "userCookie": "emFlrLGrm5fA2xLFT5npwbPuG7kf6X",
    "requestEndTime": "1239714001000",
    "impressionId": "1I5G20RmOuG2rt7fFGFgsaWk9Xpkfb",
    "userAgent": "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0;␣
↪SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506;␣
↪InfoPa",
    "timers": {
        "modelLookup": "0.3292",
        "requestTime": "0.6398"
    },
    "threadId": "99",
    "ip": "67.189.155.225",
    "modelId": "bxxiuxduad",
    "hostname": "ec2-0-51-75-39.amazon.com",
    "sessionId": "J9NOccA3dDMFlixCuSOtl9QBbjs6aS",
    "requestBeginTime": "1239714000000"
}
```


In Hive, you need to load the data from the partitions, so the script runs the following:

```
MSCK REPAIR TABLE impressions;
```

The script then creates a table that stores your data in a Parquet-formatted file on Amazon S3:

```
CREATE EXTERNAL TABLE  parquet_hive (
    requestBeginTime string,
    adId string,
    impressionId string,
    referrer string,
    userAgent string,
    userCookie string,
    ip string
)   STORED AS PARQUET
LOCATION 's3://myBucket/myParquet/';
```

The data are inserted from the *impressions* table into *parquet_hive*:

```
INSERT OVERWRITE TABLE parquet_hive
SELECT
requestbegintime,
adid,
impressionid,
referrer,
useragent,
usercookie,
ip FROM impressions WHERE dt='2009-04-14-04-05';
```

The script stores the above impressions table columns from the date, 2009-04-14-04-05, into s3://myBucket/myParket/ in a Parquet-formatted file. The full script is located on Amazon S3 at:

s3://athena-examples/conversion/write-parquet-to-s3.q

- After your EMR cluster is terminated, you then create your table in Athena, which uses the data in the format produced by the cluster.

1.9.2 Prerequisites

- You need to be able to create EMR clusters. For more information about Amazon EMR, see the Amazon EMR Management Guide.
- Follow the instructions found in *Setting Up Amazon Athena*.

1.9.3 Example: Converting data to Parquet using an EMR cluster

1. Use the AWS CLI to create a cluster. If you need to install the AWS CLI, see Installing the AWS Command Line Interface in the AWS CLI User Guide.
2. You need roles to use Amazon EMR, so if you haven't used Amazon EMR before, create the default roles using the following command:

```
aws emr create-default-roles
```

For more information, see Create and Use IAM Roles for Amazon EMR in the Amazon EMR Management Guide.

3. Create an Amazon EMR cluster using the emr-4.7.0 release to convert the data using the following AWS CLI emr create-cluster command:

```
export REGION=us-east-1
export SAMPLEURI=s3://${REGION}.elasticmapreduce/samples/hive-ads/
↪tables/impressions/
export S3BUCKET=myBucketName

aws emr create-cluster --applications Name=Hadoop Name=Hive␣
↪Name=HCatalog \
--ec2-attributes KeyName=myKey,InstanceProfile=EMR_EC2_DefaultRole,
↪SubnetId=subnet-mySubnetId \
--service-role EMR_DefaultRole --release-label emr-4.7.0 --instance-
↪type \
m4.large --instance-count 1 --steps Type=HIVE,Name="Convert to␣
↪Parquet",\
ActionOnFailure=CONTINUE,ActionOnFailure=TERMINATE_CLUSTER,Args=[-f,\
s3://athena-examples/conversion/write-parquet-to-s3.q,-hiveconf,
↪INPUT=${SAMPLEURI},-hiveconf,OUTPUT=s3://{$S3BUCKET}/myParquet,-
↪hiveconf,REGION=${REGION}] \
--region ${REGION} --auto-terminate
```

   A successful request gives you a cluster ID. You can monitor the progress of your cluster using the AWS Management Console or using the cluster ID with the *list-steps* subcommand in the AWS CLI:

```
aws emr list-steps --cluster-id myClusterID
```

   Look for the script step status. If it is COMPLETED, then the conversion is done and you are ready to query the data.

4. Now query the data in Athena. First, you need to create the same table that you created on the EMR cluster.

   You can use the same statement as above. Log into Athena and enter the statement in the *Query Editor* window:

```
CREATE EXTERNAL TABLE  parquet_hive (
    requestBeginTime string,
    adId string,
    impressionId string,
    referrer string,
    userAgent string,
    userCookie string,
    ip string
)   STORED AS PARQUET
LOCATION 's3://myBucket/myParquet/';
```

   Choose *Run Query*.

5. Run the following query to show that you can query this data:

```
SELECT * FROM parquet_hive LIMIT 10;
```

Alternatively, you can select the view (eye) icon by the parquet_hive table in *Catalog*:

The results should show output similar to this:

Athena Results

	requestbegintime	adid	impressionid	referrer	useragent
1	1239682352000	sn07U0dSU2BUek2lkJ1EKGXmhxDwhs	5EM6xQDRXRPRwvMx4wPCWIE03930q6	cartoonnetwork.com	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.1) Gecko/20090715 F
2	1239682686000	XaiowOqorg6rcCpUrgPr0iHtO91r27	TAr4P6gEnLsweSViaABw6BmEL4lnF1	cartoonnetwork.com	Echoping/6.0.2
3	1239682753000	c2sNCqusvnv7RPqCMpr0h7jFVVruDw	ON6doUquwLE4a1pnVLhLjilHmJBuHk	cartoonnetwork.com	Echoping/6.0.2
4	1239682506000	4Xkt3ErCHRw1sN1rXrnMHg9rdJTlpo	87GLC447C88J7sqfCudcCgHgtMTg5A	cartoonnetwork.com	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_6; en-us) AppleWebKit/525.27
5	1239682573000	nAAuKDKRp26pWULS1wbBbbVEvrMHjS	cqodkEKNQ91QpDvHJ6esitkaTtvEia	cartoonnetwork.com	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_6; en-us) AppleWebKit/525.27
6	1239682387000	Muvf2gHNwxS5RpNnxTQgPEHfmrqQAJ	SEgg69XEtRmgWNIHRkdP0pLhvpVELx	cartoonnetwork.com	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4
7	1239682595000	couJJD6RLuqOQ6Hpxg3jjVXUXRXof4	5f0frAsugNDl65euRaxHM18qCuXRR7	cartoonnetwork.com	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4
8	1239682537000	lO9o9TUFqSTS0hKaetlXX8xgaN7fVF	Hu62Kiuu9ejeSiWkFrJPDtrjqKQGGM	cartoonnetwork.com	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
9	1239682667000	QPORxxngM5oDxkvnmBNEgAtF1w0War	tpETvVWt6fP5STPgFt7FckLhCCIns1	cartoonnetwork.com	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET C
10	1239682347000	2RWcfpDa1nXleuUXwKjhaWnoqDrbSm	MhTBNA5Qpl3djlU6JWRGlq8whjFvqP	corriere.it	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.507

1.10 SQL and HiveQL Reference

Amazon Athena is based on the Hive metastore and Presto .

Athena syntax consists of a combination of standard ANSI SQL for queries (select) and relational operations (join) and HiveQL DDL statements for altering metadata (create, alter).

1.10.1 SQL Queries

SELECT

Description

Retrieves rows from zero or more tables.

Synopsis

```
[ WITH with_query [, ...] ]
SELECT [ ALL | DISTINCT ] select_expression [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition ]
[ UNION [ ALL | DISTINCT ] union_query ]
```

```
[ ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...] ]
[ LIMIT [ count | ALL ] ]
```

Parameters

[WITH with_query [, ...]] Precedes the SELECT list in a query and defines one or more subqueries for use within the SELECT query. Each subquery defines a temporary table, similar to a view definition, which can be referenced in the FROM clause. These tables are used only during the execution of the query. You can use WITH to flatten nested queries or to simplify subqueries. Using WITH to create recursive queries is not supported. WITH works with multiple subqueries and the relations within the WITH clause can chain.

with_query syntax is:

```
subquery_table_name [ ( column_name [,...] ) ] AS
(subquery)
```

Where:

- subquery_table_name is a unique name for a temporary table that defines the results of the WITH clause subquery. Each subquery must have a table name that can be referenced in the FROM clause.
- column_name [,...] is an optional list of output column names. The number of column names must be equal to or less than the number of columns defined by subquery.
- subquery is any query statement

[ALL | DISTINCT] select_expr select_expr determines the rows to be selected. Use DISTINCT to return only distinct values when a column contains duplicate values. ALL is the default. Using ALL is treated the same as if it were omitted; all rows for all columns are selected and duplicates are kept.

FROM from_item [, ...] Indicates the input to the query, where from_item can be a view, a join construct, or a subquery as described below.

The from_item can be either:

- table_name [[AS] alias [(column_alias [,...])]]

  Where table_name is the name of the target table from which to select rows, alias is the name to give the output of the SELECT statement, and column_alias defines the columns for the alias specified.

–OR–

- join_type from_item [ON join_condition | USING (join_column [,...])]

  Where join_type is one of:

  - [INNER] JOIN
  - LEFT [OUTER] JOIN

- RIGHT [OUTER] JOIN
- FULL [OUTER] JOIN
- CROSS JOIN
- ON join_condition | USING (join_column [,...]) Where using join_condition allows you to specify column names for join keys in multiple tables, and using join_column requires join_column to exist in both tables.

[WHERE condition] Filters results according to the condition you specify.

[GROUP BY [ALL | DISTINCT] grouping_expressions [, ...]] Divides the output of the SELECT statement into rows with matching values. ALL and DISTINCT determine whether duplicate grouping sets each produce distinct output rows. If omitted, ALL is assumed. The grouping_expressions element can be any function (such as SUM, AVG, COUNT, etc.) performed on input columns or be an ordinal number that selects an output column by position, starting at one. GROUP BY expressions can group output by input column names that don't appear in the output of the SELECT statment. All output expressions must be either aggregate functions or columns present in the GROUP BY clause. ''grouping_expressions'' allow you to perform complex grouping operations. You can use a single query to perform analysis that requires aggregating multiple column sets. These complex grouping operations don't support expressions comprising input columns. Only column names or ordinals are allowed. You can often use UNION ALL to achieve the same results as these GROUP BY operations, but queries that use GROUP BY have the advantage of reading the data once, whereas UNION ALL reads the underlying data three times and may produce inconsistent results when the data source is subject to change. GROUP BY CUBE generates all possible grouping sets for a given set of columns. GROUP BY ROLLUP generates all possible subtotals for a given set of columns.

[HAVING condition] Used with aggregate functions and the GROUP BY clause. Controls which groups are selected, eliminating groups that don't satisfy condition. This filtering occurs after groups and aggregates are computed.

[UNION [ALL | DISTINCT] union_query]] Combines the results of more than one SELECT statement into a single query. ALL or DISTINCT control which rows are included in the final result set. ALL causes all rows to be included, even if the rows are identical, while DISTINCT causes only unique rows to be included in the combined result set. DISTINCT is the default. Multiple UNION clauses are processed left to right unless you use parentheses to explictly define the order of processing.

[ORDER BY expression [ASC | DESC] [NULLS FIRST | NULLS LAST] [, ...]] Sorts a result set by one or more output expression. When the clause contains multiple expressions, the result set is sorted according to the first expression. Then the second expression is applied to rows that have matching values from the first expression, and so on. Each expression may specify output columns from SELECT or an ordinal number for an output column by position, starting at one. ORDER BY is evaluted as the last step after any GROUP BY or HAVING clause. ASC and DESC determine whether results are sorted in ascending or descending order. The default null ordering is NULLS LAST, regardless of ascending or descending sort order.

LIMIT [count | ALL] Restricts the number of rows in the result set to count. LIMIT ALL is the same as omitting the LIMIT clause. If the query has no ORDER BY clause, the results are arbitrary.

TABLESAMPLE BERNOULLI | SYSTEM (percentage) Optional operator to select rows from a table based on a sampling method. `BERNOULLI` selects each row to be in the table sample with a probability of `percentage`. All physical blocks of the table are scanned, and certain rows are skipped based on a comparison between the sample `percentage` and a random value calculated at runtime. With `SYSTEM`, the table is divided into logical segments of data, and the table is sampled at this granulariy. Either all rows from a particular segment are selected, or the segment is skipped based on a comparison between the sample `percentage` and a random value calculated at runtime. `SYTSTEM` sampling is dependent on the connector. This method does not guarantee independent sampling probabilities.

[UNNEST (array_or_map) [WITH ORDINALITY]] Expands an array or map into a relation. Arrays are expanded into a single column. Maps are expanded into two columns (*key*, *value*). You can use `UNNEST` with multiple arguments, which are expanded into multiple columns with as many rows as the highest cardinality argument. Other columns are padded with nulls. The `WITH ORDINALITY` clause adds an ordinality column to the end. `UNNEST` is usually used with a `JOIN` and can reference columns from relations on the left side of the `JOIN`.

Examples

```
SELECT * from table;
```

```
SELECT os, COUNT(*) count FROM cloudfront_logs WHERE date BETWEEN date '2014-
↪07-05' AND date '2014-08-05' GROUP BY os;
```

1.10.2 DDL Statements

ALTER DATABASE SET DBPROPERTIES

Description

Creates one or more properties for a database. The use of `DATABASE` and `SCHEMA` are interchangeable; they mean the same thing.

Synopsis

```
ALTER (DATABASE|SCHEMA) database_name
  SET DBPROPERTIES ('property_name'='property_value' [, ...] )
```

Parameters

SET DBPROPERTIES ('property_name'='property_value' [, ...] Specifies a property or properties for the database named `property_name` and establishes the value for each of the properties

respectively as `property_value`. If `property_name` already exists, the old value is overwritten with `property_value`.

Examples

```
ALTER DATABASE jd_datasets
  SET DBPROPERTIES ('creator'='John Doe', 'department'='applied mathematics');

ALTER SCHEMA jd_datasets
  SET DBPROPERTIES ('creator'='Jane Doe');
```

ALTER TABLE ADD PARTITION

Description

Creates one or more partition columns for the table. Each partition consists of one or more distinct column name/value combinations. A separate data directory is created for each specified combination, which can improve query performance in some circumstances. Partitioned columns don't exist within the table data itself, so if you use a column name that has the same name as a column in the table itself, you get an error.

Synopsis

```
ALTER TABLE table_name ADD [IF NOT EXISTS]
  PARTITION
  (partition_col1_name = partition_col1_value
  [,partition_col2_name = partition_col2_value]
  [,...])
  [LOCATION 'location1']
  [PARTITION
  (partition_colA_name = partition_colA_value
  [,partition_colB_name = partition_colB_value
  [,...])]
  [LOCATION 'location2']
  [,...]
```

Parameters

[IF NOT EXISTS] Causes the error to be suppressed if a partition with the same definition already exists.

PARTITION (partition_col_name = partition_col_value [,...]) Creates a partition with the column name/value combinations that you specify. Enclose `partition_col_value` in string characters only if the data type of the column is a string.

[LOCATION 'location'] Specifies the directory in which to store the paritions defined by the preceding statement.

Examples

```
ALTER TABLE orders ADD
  PARTITION (dt = '2014-05-14', country = 'IN');

ALTER TABLE orders ADD
  PARTITION (dt = '2014-05-14', country = 'IN')
  PARTITION (dt = '2014-05-15', country = 'IN');

ALTER TABLE orders ADD
  PARTITION (dt = '2014-05-14', country = 'IN') LOCATION 's3://mystorage/path/
→to/INDIA_14_May_2014';
  PARTITION (dt = '2014-05-15', country = 'IN') LOCATION 's3://mystorage/path/
→to/INDIA_15_May_2014';
```

ALTER TABLE DROP PARTITION

Synopsis

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION (partition_spec) [,␣
→PARTITION (partition_spec)]
```

Description

Drops one or more specified partitions for the named table.

Parameters

[IF EXISTS] Causes the error message to be suppressed if the partition specified does not exist.

PARTITION (partition_spec) Each `partition_spec` specifies a column name/value combination in the form `partition_col_name = partition_col_value [,...]`.

Examples

```
ALTER TABLE orders DROP PARTITION (dt = '2014-05-14', country = 'IN');
ALTER TABLE orders DROP PARTITION (dt = '2014-05-14', country = 'IN'),␣
→PARTITION (dt = '2014-05-15', country = 'IN');
```

ALTER TABLE RENAME PARTITION

Description

Renames a partition column, `partition_spec`, for the table named `table_name`, to `new_partition_spec`.

Synopsis

```
ALTER TABLE table_name PARTITION (partition_spec) RENAME TO PARTITION (new_
↪partition_spec)
```

Parameters

PARTITION (partition_spec) Each `partition_spec` specifies a column name/value combination in the form `partition_col_name = partition_col_value [,...]`.

Examples

```
ALTER TABLE orders PARTITION (dt = '2014-05-14', country = 'IN') RENAME TO␣
↪PARTITION (dt = '2014-05-15', country = 'IN');
```

ALTER TABLE SET LOCATION

Synopsis

```
ALTER TABLE table_name [ PARTITION (partition_spec) ] SET LOCATION 'new␣
↪location'
```

Description

Changes the location for the table named `table_name`, and optionally a partition with `partition_spec`.

Parameters

PARTITION (partition_spec) Specifies the partition with parameters `partition_spec` whose location you want to change. The `partition_spec` specifies a column name/value combination in the form `partition_col_name = partition_col_value`.

SET LOCATION 'new location' Specifies the new location, which must be an Amazon S3 location.

Examples

```
ALTER TABLE customers PARTITION (zip='98040', state='WA') SET LOCATION 's3://
↪mystorage/custdata';
```

ALTER TABLE SET TBLPROPERTIES

Description

Adds custom metadata properties to a table sets their assigned values.

Synopsis

```
ALTER TABLE table_name SET TBLPROPERTIES ('property_name' = 'property_value'␣
↪[ , ... ])
```

Parameters

SET TBLPROPERTIES ('property_name' = 'property_value' [, ...]) Specifies the metadata properties to add as `'property_name'` and the value for each as `'property value`''.` `If ``property_name` already exists, its value is reset to `property_value`.

Note: Managed tables are not supported, so setting 'EXTERNAL'='FALSE' has no effect.

Examples

```
ALTER TABLE orders SET TBLPROPERTIES ('notes'="Please don't drop this table.
↪");
```

CREATE DATABASE

Description

Creates a database. The use of `DATABASE` and `SCHEMA` is interchangeable. They mean the same thing.

Synopsis

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
  [COMMENT `database_comment`]
  [LOCATION 'S3_loc']
  [WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]
```

Parameters

[IF NOT EXISTS] Causes the error to be suppressed if a database named `database_name` already exists.

[COMMENT database_comment] Establishes the metadata value for the built-in metadata property named `comment` and the value you provide for `database_comment`.

[LOCATION S3_loc] Specifies the location where database files and metastore will exist as `S3_loc`. The location must be an Amazon S3 location.

[WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]] Allows you to specify custom metadata properties for the database definition.

Examples

```
CREATE DATABASE clickstreams;

CREATE DATABASE IF NOT EXISTS clickstreams
  COMMENT 'Site Foo clickstream data aggregates';
  LOCATION 's3://myS3location/clickstreams'
  WITH DBPROPERTIES ('creator'='Jane D.', 'Dept.'='Marketing analytics');
```

CREATE TABLE

Description

Creates a table with the name the parameters you specify.

Synopsis

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS]
 [db_name.]table_name [(col_name data_type [COMMENT col_comment] [, ...] )]
 [COMMENT table_comment]
 [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
 [CLUSTERED BY (col_name, col_name, ...) [SORTED BY (col_name [ASC|DESC], ...
↪)] INTO num_buckets BUCKETS]
 [SKEWED BY ( col_name1 [, col_name2 , ... ] ) ON ( ( "col_name1_valueX" [,
↪"col_name2_valueA" , ... ] ) [, ("col_name1_valueY" [, "col_name2_valueB") ,
↪ ... ] ) ] [STORED AS DIRECTORIES]]
 [ROW FORMAT row_format]
```

```
[STORED AS file_format] [WITH SERDEPROPERTIES (...)] ]
[LOCATION 's3_loc']
[TBLPROPERTIES (property_name=property_value, ...)]
```

Parameters

[EXTERNAL] Specifies that the table is based on an underlying data file that exists in Amazon S3, in the `LOCATION` you specify. When you create an external table, the data referenced must comply with the default format or the format you specify with the `ROW FORMAT`, `STORED AS`, and `WITH SERDEPROPERTIES` clauses.

[IF NOT EXISTS] Causes the error message to be suppressed if a table named `table_name` already exists.

[db_name.]table_name Specifies the `table_name` that identifies the table you create. The optional `db_name` parameter specifies the database where the table exists. If omitted, the current database is assumed.

[(col_name data_type [COMMENT col_comment] [, ...])] Specifies each column to be created, identified by `col_name`, along with the column's data type. The `data_type` can be any of:

- **primitive_type**
  - TINYINT
  - SMALLINT
  - INT
  - BIGINT
  - BOOLEAN
  - FLOAT
  - DOUBLE
  - STRING
  - BINARY
  - TIMESTAMP
  - DECIMAL [(precision,scale)]
  - DATE
  - VARCHAR
  - CHAR
- **array_type**
  - ARRAY < data_type >
- **map_type**

 – MAP < primitive_type, data_type >

- **struct_type**

 – STRUCT < col_name : data_type [COMMENT col_comment] [, ...] >

- **union_type**

 – UNIONTYPE < data_type, data_type [, ...] >

[COMMENT table_comment] Creates the `comment` table property and populates it with the `table_comment` you specify.

[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)] Creates a partitioned table with one or more partition columns that have the `col_name`, `data_type` and `col_comment` specified. A table can have one or more paritions, which consist of a distinct column name and value combination. A separate data directory is created for each specified combination, which can improve query performance in some circumstances. Partitioned columns don't exist within the table data itself, so if you use a `col_name` that has the same name as a column in the table itself, you'll get an error.

[CLUSTERED BY (col_name, col_name2 [, ...]) [SORTED BY (sort_col_name [ASC|DESC], ...)] INTO num_bucket
Creates clusters and corresponding buckets, which allows for more efficient sampling in some query cases. Records in `col_name` will be hashed into buckets by `num_buckets`. This ensures that records in `col_name` will be stored in the same bucket. The optional `SORTED BY` clause specifies that the cluster and buckets should be created based on values in the columns you list as `sort_col_name` in either ascending (`ASC`) or descending (`DESC`) order. The column or columns in this clause are part of the table definition as opposed to partitioned columns.

SKEWED BY (col_name1 [, col_name2 , ...]) ON (("col_name1_valueX" [, "col_name2_valueA" , ...]) [, ("col_na
Specifies skewing parameters for the table. Skewing can help accelerate queries when a few values occur often in a particular column–for example, if a column "country" were predominantly filled with the values "India" and "US", the table and column would be a good candidate for skewing on the "country" column. You can use skewing with or without partitioning. The column or columns specified by `col_name1`, `col_name2`, and so on are the columns on which to base the skew, followed by the values that should be skewed from each column respectively. Values must be in quoted strings. Each column can have one or more corresponding values, which must be specified in the same order as the column names. In other words, `column_name1` corresponds to `col_name1_valueX`, `col_name1_valueY`, and so on. Similarly, `column_name2` corresponds to `col_name2_valueA`, `col_name2_valueB` and so on. For each column and value specified, records are split into separate files so that queries can skip or include records based on the input values. The optional `STORED AS DIRECTORIES` clause specifies that list bucketing should be used on skewed tables. List bucketing can't be used with normal bucketing operations (for example, `CLUSTERED BY`), external tables, or tables created with `LOAD DATA`.

[ROW FORMAT row_format] Specifies the row format of the table and its underlying source data if applicable. For `row_format` you can specify one or more delimiters with the `DELIMITED` clause or, alternatively, use the `SERDE` clause as described below. If `ROW FORMAT` is omitted or `ROW FORMAT DELIMITED` is specified, a native SerDe is used.

- [DELIMITED FIELDS TERMINATED BY char [ESCAPED BY char]]
- [DELIMITED COLLECTION ITEMS TERMINATED BY char]

- [MAP KEYS TERMINATED BY char]
- [LINES TERMINATED BY char]
- [NULL DEFINED AS char] – (Note: Available in Hive 0.13 and later)

–OR–

- SERDE 'serde_name' [WITH SERDEPROPERTIES ("property_name" = "property_value", "property_name" = "property_value" [, ...])]

  The `serde_name` indicates the SerDe to use. The `WITH SERDEPROPERTIES` clause allows you to provide one or more custom properties allowed by the SerDe.

[STORED AS file_format] Specifies the file format for table data. If omitted, `TEXTFILE` is the default. Options for `file_format` are:

- SEQUENCEFILE
- TEXTFILE
- RCFILE
- ORC
- PARQUET
- AVRO
- INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname

[LOCATION 'S3_loc'] Specifies the location of the table with a pointer to S3. For example, `` `s3://mystorage` ``.

[TBLPROPERTIES (property_name=property_value, ...)] Allows you to specify custom metadata key/value pairs for the table definition in addition to predefined table properties, such as `"comment"`.

Examples

```
CREATE EXTERNAL TABLE IF NOT EXISTS mydatabase.cloudfront_logs (
  Date 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://athena-examples/cloudfront/plaintext/';
```

DESCRIBE TABLE

Description

Shows the list of columns, including partition columns, for the named column. This allows you to examine the attributes of a complex column.

Synopsis

```
DESCRIBE [EXTENDED | FORMATTED] [db_name.]table_name [PARTITION partition_
→spec] [col_name ( [.field_name] | [.'$elem$'] | [.'$key$'] | [.'$value$'] )]
```

Parameters

[EXTENDED | FORMATTED] Determines the format of the output. If you specify `EXTENDED`, all metadata for the table is output in Thrift serialized form. This is useful primarily for debugging and not for general use. Use `FORMATTED` or omit the clause to show the metadata in tabular format.

[PARTITION partition_spec] Lists the metadata for the partition with `partition_spec` if included.

[col_name ([.field_name] | [.'$elem$'] | [.'key'] | [.'$value$'])*] Specifies the column and attributes to examine. You can specify `.field_name` for an element of a struct, `'$elem$'` for array element, `'$key$'` for a map key, and `'$value$'` for map value. You can specify this recursively to further explore the complex column.

Examples

```
DESCRIBE orders;
```

DROP DATABASE

Description

Removes the named database from the system. If the database contains tables, you must either drop the tables before executing `DROP DATABASE` or use the `CASCADE` clause. The use of `DATABASE` and `SCHEMA` are interchangeable. They mean the same thing.

Synopsis

```
DROP {DATABASE|SCHEMA} [IF EXISTS] database_name [RESTRICT|CASCADE]
```

Parameters

[IF EXISTS] Causes the error to be suppressed if `database_name` doesn't exist.

[RESTRICT|CASCADE] Determines how tables within `database_name` are regarded during the `DROP` operation. If you specify `RESTRICT`, the database will not be dropped if it contains tables. This is the default behavior. Specifying `CASCADE` causes the database and all its tables to be dropped.

Examples

```
DROP DATABASE clickstreams;

DROP SCHEMA IF EXISTS clickstreams CASCADE;
```

DROP TABLE

Description

Removes the metadata table definition for the table named `table_name` and deletes the underlying data unless the table was created as an external table. When you drop an external table, the underlying data remains intact.

> **Caution:** Any tables created using the CREATE TABLE statement without the EXTERNAL clause are known as *managed* tables. Dropping managed tables deletes underlying data files in Amazon S3 as well as the metadata table definition. Regardless of whether the table is external or managed, the metadata table definition is lost when the table is dropped.

Synopsis

```
DROP TABLE [IF EXISTS] table_name [PURGE]
```

Parameters

[IF EXISTS] Causes the error to be suppressed if `table_name` doesn't exist.

[PURGE] Applies to managed tables. Ignored for external tables. Specifies that data should be removed permanently rather than being moved to the `.Trash/Current` directory.

Examples

```
DROP TABLE fulfilled_orders;

DROP TABLE IF EXISTS fulfilled_orders PURGE;
```

MSCK REPAIR TABLE

Description

Recovers partitions and data associated with partitions. Use this statement when you add partitions to the catalog. It is possible it will take some time to add all partitions. If this operation times out, it will be in an incomplete state where only a few partitions are added to the catalog. You should run the statement on the same table until all partitions are added. For more information, see *Partitioning Data*.

Synopsis

```
MSCK REPAIR TABLE table_name
```

Parameters

Examples

```
MSCK REPAIR TABLE table_name
```

SHOW COLUMNS

Description

Lists the columns in the table schema.

Synopsis

```
SHOW COLUMNS IN 'table_name'
```

Examples

```
SHOW COLUMNS IN clicks;
```

SHOW CREATE TABLE

Description

Returns a SQL statement that can create a table or view.

Synopsis

```
SHOW CREATE TABLE [db_name.]table_name
```

Parameters

TABLE [db_name.]table_name Analyzes an existing table named `table_name` to generate the query that created it. Optionally, you can use `db_name` to specify the database. If omitted, the context defaults to the current database.

Examples

```
SHOW CREATE TABLE orderclickstoday;

SHOW CREATE TABLE salesdata.orderclickstoday;
```

SHOW DATABASES

Description

Lists all databases defined in the metastore. You can use `DATABASES` or `SCHEMAS`. They mean the same thing.

Synopsis

```
SHOW {DATABASES | SCHEMAS} [LIKE 'regular_expression']
```

Parameters

[LIKE 'regular_expression'] Filters the list of databases to those that match the `regular_expression` you specify. Wildcards can only be `*`, which indicates any character, or `|`, which indicates a choice between characters.

Examples

```
SHOW SCHEMAS;

SHOW DATABASES LIKE '*analytics';
```

SHOW PARTITIONS

Description

Lists all the partitions in the table.

Synopsis

```
SHOW PARTITIONS 'table_name'
```

Parameters

Examples

```
SHOW PARTITIONS clicks;
```

SHOW TABLES

Description

Lists all the base tables and views in a database.

Synopsis

```
SHOW TABLES [IN database_name] ['regular_expression']
```

Parameters

[IN database_name] Specifies the `database_name` from which tables will be listed. If omitted, the database from the current context is assumed.

['regular_expression'] Filters the list of tables to those that match the `regular_expression` you specify. Wildcards can only be `*`, which indicates any character, or `|`, which indicates a choice between characters.

Examples

```
SHOW TABLES;

SHOW TABLES IN marketing_analytics 'orders*';
```

SHOW TBLPROPERTIES

Description

Lists table properties for the named table.

Synopsis

```
SHOW TBLPROPERTIES table_name [('property_name')]
```

Parameters

[('property_name')] If included, only the value of the property named `property_name` is listed.

Examples

```
SHOW TBLPROPERTIES orders;

SHOW TBLPROPERTIES orders('comment');
```

VALUES

Synopsis

```
VALUES row [, ...]
```

Description

Creates a table, which can be anonymous, or which you can name with the AS clause to specify column names, a table name, or both.

Parameters

row This can be a single expression or `( column_expression [,...] )`.

Examples

```
VALUES 263, 264, 265;
```

1.10.3 Unsupported DDL

The following native Hive DDLs are not supported by Athena:

- ALTER INDEX
- ALTER TABLE `table_name` ARCHIVE PARTITION
- ALTER TABLE `table_name` CLUSTERED BY
- ALTER TABLE `table_name` EXCHANGE PARTITION
- ALTER TABLE `table_name` NOT CLUSTERED
- ALTER TABLE `table_name` NOT SKEWED
- ALTER TABLE `table_name` NOT SORTED
- ALTER TABLE `table_name` NOT STORED AS DIRECTORIES
- ALTER TABLE `table_name` partitionSpec ADD COLUMNS
- ALTER TABLE `table_name` partitionSpec CHANGE COLUMNS
- ALTER TABLE `table_name` partitionSpec COMPACT
- ALTER TABLE `table_name` partitionSpec CONCATENATE
- ALTER TABLE `table_name` partitionSpec REPLACE COLUMNS
- ALTER TABLE `table_name` partitionSpec SET FILEFORMAT
- ALTER TABLE `table_name` RENAME TO
- ALTER TABLE `table_name` SET SKEWED LOCATION

- ALTER TABLE `table_name` SKEWED BY
- ALTER TABLE `table_name` TOUCH
- ALTER TABLE `table_name` UNARCHIVE PARTITION
- COMMIT
- CREATE INDEX
- CREATE ROLE
- CREATE TABLE `table_name` LIKE `existing_table_name`
- CREATE TEMPORARY MACRO
- CREATE VIEW
- DELETE FROM
- DESCRIBE DATABASE
- DFS
- DROP INDEX
- DROP ROLE
- DROP TEMPORARY MACRO
- EXPORT TABLE
- GRANT ROLE
- IMPORT TABLE
- INSERT INTO
- LOCK DATABASE
- LOCK TABLE
- REVOKE ROLE
- ROLLBACK
- SHOW COMPACTIONS
- SHOW CURRENT ROLES
- SHOW GRANT
- SHOW INDEXES
- SHOW LOCKS
- SHOW PRINCIPALS
- SHOW ROLE GRANT
- SHOW ROLES
- SHOW TRANSACTIONS

- START TRANSACTION
- UNLOCK DATABASE
- UNLOCK TABLE

1.10.4 Functions

The functions supported in Athena queries are those found within Presto. For more information, see Functions and Operators in the Presto documentation.

1.11 Known Limitations

The following are known limitations in Amazon Athena.

1.11.1 Miscellaneous

- User-defined functions (UDF or UDAFs) are not supported.
- Stored procedures are not supported.
- Currently, Athena does not support any transactions found in Hive or Presto. For a full list of keywords not supported, see *Unsupported DDL*.
- LZO is not supported. We suggest using Snappy instead.
- You can use Athena to query underlying Amazon S3 bucket data that's in a different region from the region where you initiate the query (using either the Athena console or a JDBC connection string). These queries are supported only for Amazon S3 bucket data in the following regions:
  - ap-northeast-1
  - ap-southeast-1
  - ap-southeast-2
  - ca-central-1
  - cn-north-1
  - eu-central-1
  - eu-west-1
  - sa-east-1
  - us-east-1
  - us-east-2
  - us-west-1
  - us-west-2

1.11.2 Tips and Tricks

The following tips and tricks might help you avoid surprises when working with Athena.

Table names that begin with an underscore

Use backticks if table names begin with an underscore. For example:

```
CREATE TABLE myUnderScoreTable (
`_id` string,
`_index`string,
...
```

For the LOCATION clause, using a trailing slash

In the LOCATION clause, use a trailing slash for your folder or bucket, NOT filenames or glob characters. For example:

Use:

```
s3://path_to_bucket/
```

Don't Use:

```
s3://path_to_bucket
s3://path_to_bucket/*
s3://path_to_bucket/mySpecialFile.dat
```

Athena table names are case-insensitive

If you are interacting with Apache Spark, then your table column names must be lowercase. Athena is case-insensitive but Spark requires lowercase table names.

Athena table names only allow underscore character

Athena table names cannot contain special characters, other than underscore (_).

1.12 Service Limits

Note: You can request a limit increase by contacting AWS Support.

- Currently, you can only submit one query at a time and you can only have 5 (five) concurrent queries at one time per account.

- Query timeout: 30 minutes
- Number of databases: 100
- Table: 100 per database
- Number of partitions: 20k per table
- You may encounter a limit for Amazon S3 buckets per account, which is 100. Athena also needs a separate bucket to log results.

1.13 Document History

The following describes the important changes to the documentation since the last release of Athena:

- **Latest documentation update: Jan 19, 2017** Removed topic for ALTER `table_name` RENAME TO. It is not currently supported. Added it to the list of unsupported statements.

December 9, 2016 Added information to known limitations about region support for queries that originate in different regions than underlying Amazon S3 bucket data being queried.

December 6, 2016 Corrected errors discovered after initial launch.

November 29, 2016 Documentation first created.

1.14 About Amazon Web Services

Amazon Web Services (AWS) is a collection of digital infrastructure services that developers can leverage when developing their applications. The services include computing, storage, database, and application synchronization (messaging and queuing). AWS uses a pay-as-you-go service model: you are charged only for the services that you—or your applications—use. For new AWS users, a free usage tier is available. On this tier, services are free below a certain level of usage. For more information about AWS costs and the Free Tier, see Use the AWS Free Tier. To obtain an AWS account, visit the AWS home page and click **Create a Free Account**.