11
Ad Hoc Queries with Amazon Athena

In *Chapter 8, Identifying and Enabling Varied Data Consumers*, we explored a variety of data consumers. Now in this chapter, we will start examining the AWS services that some of these different data consumers may want to use, starting with those that need to use SQL to run ad hoc queries on data in the **data lake**.

SQL syntax is widely used for querying data in a variety of databases, and there is a large number of people that know SQL, making it a skill that is fairly easy to find. As a result, there is significant demand from various data consumers for the ability to query data that is in the data lake using SQL, without having to first move the data into a dedicated traditional database.

Amazon Athena is a serverless, fully managed service that lets you use SQL and Spark to directly query data in the data lake, as well as query various other database sources. It requires no setup, and there are options to either pay for the service based only on the amount of data that is scanned by your SQL queries, or to reserve a specific amount of capacity and pay for that capacity reservation (referred to as provisioned capacity).

In this chapter we will examine Athena features and functionality, such as how Athena can be used to query data directly in the data lake, how you can use Athena to query data from other data sources with Query Federation, and how Athena provides functionality for governance and cost management, with workgroups. We also provide some recommended best practices to help you optimize your Athena SQL queries for both cost and performance.

This chapter covers the following topics:

- An introduction to Amazon Athena
- Tips and tricks to optimize Amazon Athena SQL queries
- Exploring advanced Athena functionality
- Managing groups of users with Amazon Athena workgroups
- Hands-on – creating an Amazon Athena workgroup and configuring Athena settings

- Hands-on – switching workgroups and running queries

Technical requirements

In the hands-on sections of this chapter, you will perform administrative tasks related to Amazon Athena (such as creating a new Athena workgroup) and run Athena queries. As mentioned at the start of this book, we strongly recommend that, for the exercises in this book, you use a sandbox account where you have full administrative permissions.

For this chapter, at a minimum, you will need permissions to manage Athena workgroups, permissions to run Athena queries, access to the **AWS Glue Data Catalog** for databases and tables to be queried, and read access to the relevant underlying S3 storage.

A user that has the **AmazonAthenaFullAccess** and **AmazonS3ReadOnlyAccess** policies attached should have sufficient permissions for the exercises in this chapter. However, note that a user with these roles will have access to all S3 objects in the account, all Glue resources, all Athena permissions, as well as various other permissions, so this should only be granted to users in a sandbox account. Such broad privileges should be avoided for users in production accounts.

You can find the code files of this chapter in the GitHub repository using the following link: `https://github.com/PacktPublishing/Data-Engineering-with-AWS-2nd-edition/tree/main/Chapter11`.

An introduction to Amazon Athena

Amazon Athena was originally launched as a service that simply provided a way to run **SQL** queries against data in an S3-based data lake. However, over the years, AWS had added a lot of additional functionality to Athena, enabling features like running queries against other databases (not just S3-based data), and supporting the use of Spark based Notebooks for querying data (in addition to SQL queries).

Structured Query Language (**SQL**) was invented at **IBM** in the 1970s but has remained an extremely popular language for querying data throughout the decades. Every day, millions of people across the world use SQL directly to explore data in a variety of databases, and many more use applications (whether business applications, mobile applications, or others) that, under the covers, use SQL to query a database.

Facebook, the social media network, has very large datasets and complex data analysis requirements and found that existing tools in the Hadoop ecosystem were not able to meet their needs. As a result, Facebook created an internal solution for being able to run SQL queries on their very large datasets, using standard SQL semantics, and in 2013, Facebook released this as an open-source solution called **Presto**.

In late 2016, AWS announced the launch of Amazon Athena, a new service that would enable customers to directly query structured and semi-structured data that exists in Amazon S3. In the launch announcement, Amazon indicated that Athena was a managed version of Presto, with standard SQL support. This provided the power of the **Presto SQL analytics engine** as a serverless service to AWS customers.

SQL is broadly broken into two parts:

- **Data Definition Language** (**DDL**), which is used to create and modify database objects.
- **Data Manipulation Language** (**DML**), which is used to query and manipulate data.

In 2021, AWS upgraded the Amazon Athena engine to v2, which included support for features such as federated queries, new geospatial functions, and schema evolution support for Parquet files. This was followed by the launch of Athena engine v3 in October 2022, which included reliability improvements, performance improvements, and updates incorporated from the Trino open-source project (`https://trino.io/`). While both engine versions are available for use at the time of writing, Athena engine v2 will be deprecated at some point, and therefore if starting a new project it is always recommended to configure the project to use the latest Athena engine version.

Amazon Athena requires a Hive-compatible data catalog that provides the metadata for the data being queried. The catalog provides a logical view (databases that contain tables, which consist of rows, along with columns of a specific data type), and this maps to physical files stored in Amazon S3. Athena originally had its own data catalog, but today, it requires the use of the AWS Glue Data Catalog as its Hive compatible data store.

Amazon Athena makes it easy to quickly start querying data in an Amazon S3-based data lake, but there are some important things to keep in mind to optimize

SQL queries with Amazon Athena, as we will discuss in the next section.

Tips and tricks to optimize Amazon Athena queries

When **raw data** is ingested into the data lake, we can immediately create a table for that data in the AWS Glue Data Catalog (either using a **Glue crawler** or by running DDL statements with Athena to define the table). Once the table has been created, we can start exploring the table by using Amazon Athena to run SQL queries against the data.

However, raw data is often ingested in plaintext formats such as CSV or JSON. And while we can query the data in this format for ad hoc data exploration, if we need to run complex queries against large datasets, these raw formats are not efficient to query. There are also ways that we can optimize the SQL queries that we write to make the best use of the underlying Athena query engine, which we will review in this chapter.

By default, Amazon Athena's cost is based on the amount of compressed data that is scanned to resolve your SQL query, so anything that can be done to reduce the amount of data scanned improves query performance and reduces query cost. And if using Athena Provisioned Capacity, while you don't pay based on the amount of data scanned, optimized file formats and queries use provisioned resources more efficiently, and queries are also more performant.

In this section, we will review several ways that we can optimize our analytics for increased performance and better cost efficiency.

Common file format and layout optimizations

The most impactful and easiest transformations that a data engineer can apply to raw files are those that transform the raw files into an optimized file format, and that structure the layout of files in an optimized way.

Transforming raw source files to optimized file formats

As we discussed in *Chapter 7, Transforming Data to Optimize for Analytics*, file formats such as **Apache Parquet** are designed for analytics and perform much better than raw data formats such as CSV or JSON. So, transforming your raw source

files into a format such as Parquet is one of the most important things a data engineer can do to improve the performance of Athena queries. Review the *Optimizing the file format* section of *Chapter 7, Transforming Data to Optimize for Analytics*, for a more comprehensive look at the benefits of Apache Parquet files.

In *Chapter 7* we also reviewed how the AWS Glue service can be used to transform your files into optimized formats. However, Amazon Athena can also transform files using a concept called **Create Table As Select (CTAS)**. With this approach, you run a CTAS statement using Athena, and this instructs Athena to create a new table based on a SQL select statement against a different table.

In the following example, `customers_csv` is the table that was created on the data we imported from a database to our data lake, and the data is in CSV format. If we want to create a Parquet version of this table so that we can query it efficiently, we could run the following SQL statement using Athena:

```
CREATE TABLE customers_parquet
WITH (
      format = 'Parquet',
      parquet_compression = 'SNAPPY')
AS SELECT *
FROM customers_csv;
```

The preceding statement will create a new table called `customers_parquet`. The underlying files for this table will be in Parquet format and compressed using the Snappy compression algorithm. The contents of the new table will be the same as the `customers_csv` table since our query specified `SELECT *`, meaning select all data.

If you are bringing in specific datasets regularly (such as every night), then in most scenarios, it would make sense to configure and schedule an AWS Glue job to perform the conversion to Parquet format. But if you're doing ad hoc exploratory work on various datasets, or a one-time data load from a system, then you may want to consider using Amazon Athena to perform the transformation. Note that there are some limitations in using Amazon Athena to perform these types of transforms, so refer to the *Considerations and Limitations for CTAS Queries* (https://docs.aws.amazon.com/athena/latest/ug/considerations-ctas.html) page in the Amazon Athena documentation for more details.

Partitioning the dataset

This is also a concept that we covered in more detail in *Chapter 7, Transforming Data to Optimize for Analytics*, but we will discuss it again now briefly as, after using an optimized file format such as Parquet, this is the next most impactful thing you can do to increase the performance of your analytic queries. Review the *Optimizing with Data Partitioning* section of *Chapter 7, Transforming Data to Optimize for Analytics*, for more details on partitioning.

A common data partitioning strategy is to partition files by columns related to date. For example, in our `sales` table, we could have a `YEAR` column, a `MONTH` column, and a `DAY` column that reflect the year, month, and day of a specific sales transaction, respectively. When the data is written to S3, all sales data related to a specific day will be written out to the same S3 prefix path.

Our partitioned dataset may look as follows:

```
/datalake/transform_zone/sales/YEAR=2023/MONTH=9/DAY=29/sales1.parquet
/datalake/transform_zone/sales/YEAR=2023/MONTH=9/DAY=30/sales1.parquet
/datalake/transform_zone/sales/YEAR=2023/MONTH=10/DAY=1/sales1.parquet
/datalake/transform_zone/sales/YEAR=2023/MONTH=10/DAY=2/sales1.parquet
```

NOTE

The preceding partition structure is a simple example because generally, with large datasets, you would expect to have multiple Parquet files in each partition, and not just a single file.

Partitioning provides a significant performance benefit when you filter the results of your query based on one or more partitioned columns using the `WHERE` clause. For example, if a data analyst needs to query the total sales for the last day of September 2023, they could run the following query:

```
select sum(SALE_AMOUNT) from SALES where YEAR = '2023' and MONTH = '9' and DAY = '30'
```

Based on our partitioning strategy, the preceding query would only need to read the file (or files) in the single S3 prefix of `/datalake/transform_zone/sales/YEAR=2023/MONTH=9/DAY=30`.

Even if we want to query the data for a full month or year, we still significantly reduce the number of files that need to be scanned, compared to having to scan all

the files for all the years if we did not partition our data.

As covered in *Chapter 7, Transforming Data to Optimize for Analytics*, you can specify one or more columns to partition by when writing out data using **Apache Spark**. Alternatively, you can use Amazon Athena CTAS statements to create a partitioned dataset. However, note that a single CTAS statement in Athena can only create a maximum of 100 partitions.

Other file-based optimizations

Using an optimized file format (such as Apache Parquet) and partitioning your data are generally the two strategies that will have the biggest positive impact on analytic performance. However, several other strategies can fine-tune performance, which we will cover here briefly.

Optimizing file size: It is important to avoid having a large number of small files if you want to optimize your analytic queries. For each file in S3, the analytic engine (in this case, Amazon Athena) needs to do the following:

- Open the file.
- Read the Parquet metadata to determine whether the query needs to scan the contents of the file.
- Scan the contents of the file if the file contains data needed for the query.
- Close the file.

There can be significant **Input/Output (I/O)** overhead in listing out very large numbers of files and then processing each file. **Airbnb** has an interesting blog post on *Medium* (`https://medium.com/airbnb-engineering/on-spark-hive-and-small-files-an-in-depth-look-at-spark-partitioning-strategies-a9a364f908`) that explains an issue they had where one of their data pipeline jobs ended up creating millions of files, and how this caused significant outages for them.

While there are no hard rules regarding file size, it is generally recommended that to optimize for analytics you should aim for file sizes of at least 128 MB for objects in your S3-based data lake. When using file formats such as Parquet that are splitable (meaning the query engine can split the file and process different parts of the file in parallel), the maximum file size is not as much of an issue. But if processing files that are not splitable, such as Gzipped CSV files, it is important to have multiple files that the query engine can read in parallel, rather than a single large file.

Bucketing: Bucketing is a concept that is related to partitioning. However, with bucketing, you group rows of data together in a specified number of buckets, based on the hash value of a column (or columns) that you specify. Athena engine v2 is not compatible with the bucketing implementation that's used in Spark, but Athena engine v3 supports both Hive bucketing and the Spark bucketing algorithm.

For example, if you have data on all airline flights that you often query based on the airport code of where the flight departed from, you could bucket your data on the origin airport code. That way, all records for flights that originated from a specific airport may be in a single file in each partition, and Athena would only need to scan that file when a query was based on the origin airport and limited to a single partition.

Refer to the Amazon Athena documentation on *Partitioning and bucketing in Athena* for more information (`https://docs.aws.amazon.com/athena/latest/ug/ctas-partitioning-and-bucketing.html`).

Partition Projection: In scenarios where you have a very large number of partitions, there can be a significant overhead for Athena to read all the information about partitions from the Glue catalog. To improve performance, you can configure partition projection, where you provide a configuration pattern to reflect your partitions. Athena can then use this configuration information to determine possible partition values, without needing to read the partition information from the catalog.

For example, if you have a column called `YEARMONTH` that you partition on, and you have data going back to 2013, you could configure the partition projection range as `201301,NOW` and the partition projection format as `yyyyMM`. Athena would then be able to determine all possible valid partitions for that period without needing to read the partition information from the Glue catalog. For more information on partition projection, see the AWS documentation titled *Partition Projection with Amazon Athena* (`https://docs.aws.amazon.com/athena/latest/ug/partition-projection.html`).

In addition to the file and layout optimizations, there are also ways to write SQL queries so that the queries are optimized for the Athena analytic engine. We will cover some of these optimizations in the next section.

Writing optimized SQL queries

The way that SQL queries are written can also have a significant impact on the performance of the query.

In this section, we will review some best practices that will help provide optimal performance of queries. It's recommended that, as a data engineer, you share these best practices with data analysts and others using Athena to run queries.

That said, there are other ways, beyond the three best practices we will outline here, to go deep into query optimization. For example, you (or your end user data analysts) can use the `EXPLAIN` statement as part of an Athena query to view the logical execution plan of a specific SQL statement. You can then make modifications to your SQL statement and review the `EXPLAIN` query plan to understand how that changes the underlying execution plan. For more information, see the AWS Athena documentation titled *Using EXPLAIN and EXPLAIN ANALYSE in Athena*: `https://docs.aws.amazon.com/athena/latest/ug/athena-explain-statement.html`.

Let's look now at three important ways that you can optimize your queries.

Selecting only the specific columns that you need

When exploring data, it is common to run queries that select all columns by specifying the start of the query as `select *`. However, remember that the Parquet format that we recommend for analytics is a columnar-based file format, meaning that data stored on disk is grouped by columns rather than rows. When you specify a specific column to query, the analytic engine (such as Athena) can read the data for that column only.

If you have a table with a lot of columns (and it is not uncommon for tables to have over a hundred columns), specifying just the columns that are important to your query can significantly increase the performance of your query.

Take a scenario where you have a table with 150 columns, but your specific query only needs data from 15 of the columns. If using a columnar data format such as Parquet, then Athena would only need to scan approximately 10% of the dataset, compared to a query that uses a `select *` to query all columns.

Using approximate aggregate functions

The Presto and Trino database engines (and therefore Athena) supports a wide variety of functions and operators that can be used in queries. These include functions that can be used in calculations against large datasets in a data lake. They are used for tasks such as the following:

- Working out the sum of all sales for this month compared to last month
- Calculating the average number of orders per store
- Determining the total number of unique users that accessed our e-commerce store yesterday
- Other advanced statistical calculations

For some calculations, you may need to get a fully accurate calculation, such as when determining sales figures for formal financial reporting. At other times, you may just need an approximate calculation, such as for getting an estimate on how many unique visitors came to our website yesterday.

For those scenarios, where you can tolerate some deviation in the result, Presto and Trino provide approximate aggregate functions, and these offer significant performance improvements compared to the equivalent fully accurate version of the function.

For example, if we needed to calculate the approximate number of unique users that browsed our e-commerce store in the past 7 days, and we could tolerate a standard deviation of 2.3% in the result, we could use the `approx._distinct` function, as follows:

```
SELECT
     approx_distinct(userid)
FROM
     estore_log
WHERE
     visit_time > to_iso8601(current_timestamp - interval '7' day)
```

For more information on supported functions in Athena, including approximate functions, refer to the Athena documentation titled *Functions in Amazon Athena*: https://docs.aws.amazon.com/athena/latest/ug/functions.html.

Athena also includes some options that you can enable to further improve query performance, such as the ability to reuse query results, as we discuss next.

Reusing Athena query results

One of the options introduced with Athena engine v3, is the ability to specify that you want to reuse query results from previous queries, for a specific time period. When running a query you can opt-in to reusing previous query results, and can specify a maximum age for reusing results (with a default of 60 minutes, but the ability to reuse results for up to 7 days).

When you enable this option for a query, Athena will look for a previous execution of the exact same query within the specified time period, and within the current workgroup. If Athena finds the results of the same query, it does not rerun the query, but rather points to the previous result location or fetches the data from it. This can lead to significant performance improvements and reduced cost.

Note that Athena does not check for changes in the source data, so it is possible that using this feature will result in stale data being returned. Query reuse is based purely on finding an exact match for the query being run, within the same workgroup, and confirming that the result was generated within the time period specified (or 60 minutes if no maximum age is specified).

We don't have space to cover all query optimization techniques in this chapter. However, the AWS documentation provides a deeper dive into these optimizations, so for more information, refer to the AWS Athena documentation titled *Performance Tuning in Athena* at `https://docs.aws.amazon.com/athena/latest/ug/performance-tuning.html`.

Now that you have a good idea of the core functionality provided in Athena, and understand how to optimize your queries, let's move on to how to get even more out of Athena. In the next section we look at advanced functionality, such as how Athena can be used to query other data sources beyond the data files stored in Amazon S3, and how you can also query your data using Spark code through Athena.

Exploring advanced Athena functionality

As we've discussed several times in this book, Athena lets you query data that has been loaded into the data lake using standard SQL semantics. But since the launch of Athena, AWS has added additional functionality to enhance Athena to make it an even more powerful query engine.

One of those major enhancements, which became available in 2021 with Athena query engine v2, was the ability to run federated queries, which we will look at next.

Querying external data sources using Athena Federated Query

Query federation, also sometimes referred to as data virtualization, is the process of querying multiple external data sources, in different database engines or other systems, through a single SQL query statement.

Data lakes are designed to collect data from multiple systems in an organization and bring it into centralized storage, where the data can be combined in ways that unlock value for the business. However, it is not practical to bring every single dataset that an organization has created into the centralized storage of the data lake. For some datasets, the organization either does not need to keep all historical data for a dataset, or the data is currently in a system that already stores historical data. In these scenarios, it may make more sense to query the source dataset directly and combine data from the source with data in the data lake on the fly.

If a dataset needs to be queried by multiple teams, is queried often, and queries need to return very large amounts of data, then it is generally best to load that dataset directly into the data lake. Also, if you need to repeatedly query a system that is already under a relatively heavy load, you can reduce that load by loading data from the system to the data lake in off-peak hours, rather than running federated queries during peak times.

But if you're performing ad hoc queries, or if the data only needs to be queried by a small number of teams with a relatively low frequency of querying, then using the *Athena Federated Query* functionality to access the data makes sense. Several people have run performance testing with Athena Federated Query and have proven the ability to query many tens of thousands of records per second.

Let's look at an example of how this functionality could be used. Athena Federated Query could enable a data analyst to run a single SQL query that combines data from the following datasets:

- Master customer data in Amazon S3
- Current order information in Amazon Aurora
- Shipment tracking data in Amazon DynamoDB

- Product catalog data in Amazon Redshift

An illustration of how this query would work with Athena federation, which uses AWS Lambda based connectors, is shown in the following figure:

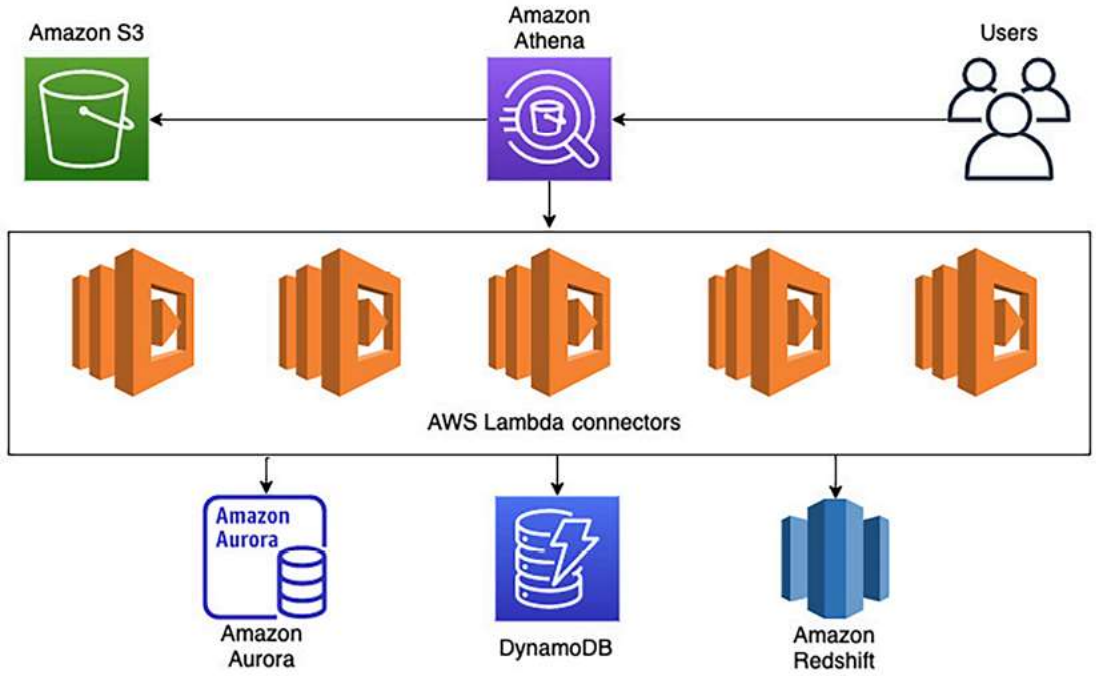

Figure 11.1: Amazon Athena query federation

Another use case would be if you do a nightly load of data from an external system into your data lake, but a few of your queries need to be able to reference some real-time data. For example, if supplier order information was loaded into the data lake each night, but you had a query that needed to calculate the total number of orders for a specific supplier for the year up to the present time, your query could do the following:

- Read supplier order information from the S3 data lake for all orders from the beginning of the year up until yesterday.
- Read any orders from today from your SAP HANA system.

There are many other potential use cases where data processing can be simplified by having the ability to use SQL to read and manipulate data from multiple systems. Rather than having to read independently from multiple systems, and then programmatically process the data, data processing is simplified by processing the data with a single SQL query.

In April 2023, Athena added support for creating views on external data sources. **Views** are a common SQL construct that are used to mask complex queries on data, by effectively creating a virtual table, that when queried, constructs the results by running the underlying query specified in the view definition.

For example, you can create a view based on a select statement that queries only certain columns in an underlying table. This effectively creates a virtual table, and when a user queries this view (or virtual table) with a `select *` statement, Athena runs the underlying query that only selected specific columns, and only returns those columns. This can be used for use cases such as limiting access to sensitive data (a table may have a column that contains a phone number, but you create the view to select all other columns except the phone number, and when the view is queried the results will not include the phone number column). You could also create a view that queries a number of systems using federated queries, but enables users to query the view without needing to know anything about the external databases.

Athena includes a number of pre-built connectors for querying various data sources, but also enables you to create custom connectors, as we discuss next.

Pre-built connectors and custom connectors

Athena Federated Query uses code running in **AWS Lambda** to connect to and query data, as well as metadata, from external systems. When a query runs that uses a connected data source, Athena invokes the relevant Lambda function/s to read metadata, identifies parts of the tables that need to be read, and launches multiple Lambda functions to read the data in parallel.

AWS has connectors that enable federated queries against over 30 popular data sources, including the following:

- A **JDBC connector** for connecting to sources such as MySQL, Postgres, and Redshift.
- A **DynamoDB connector** for reading from the Amazon-managed NoSQL database.
- A **Redis connector** for reading data from Redis instances.
- A **CloudWatch logs and CloudWatch metrics connector**, enabling you to query your application log files and metrics using SQL.

- An **AWS CMDB connector** that integrates with several AWS services to enable SQL queries against your AWS resources. Integrated services include EC2, RDS, EMR, and S3.
- A **Google BigQuery** connector that enables queries across clouds for data in Google BigQuery tables.
- An **Apache Kafka** connector that enables querying real-time data in Apache Kafka (and Amazon MSK) topics. With this connector you can join data in a Kafka topic with data in other topics, or with data in your Amazon S3 based data lake.

The full list of connectors can be found in the *Using Amazon Athena Federated Query* section of the documentation at `https://docs.aws.amazon.com/athena/latest/ug/connect-to-a-data-source.html`.

In addition to the connectors made available by AWS, anyone can create custom connectors to connect to external systems. If you can make a network connection from AWS Lambda to the target system, whether on-premises or in the cloud, you could potentially create an Athena Federated Query connector for that system.

Third-party companies are also able to create connectors for Athena Federated Query. For example, a company called **Trianz** has created connectors for **GreenPlum** and **GoogleSheets**.

To learn more about building custom connectors, see the Athena documentation titled *Developing a data source connector using the Athena Query Federation SDK*, at `https://docs.aws.amazon.com/athena/latest/ug/connect-data-source-federation-sdk.html`.

Another relatively new feature in Athena (announced at re:Invent 2022) is the ability to use Athena to run Spark based processing of data via Notebooks. We look into this feature in the next section.

Using Apache Spark in Amazon Athena

In November 2022, Amazon announced new functionality for Athena that enables users to interactively explore their data lake data using the power of **Apache Spark**. This functionality includes a simplified notebook experience (compatible with Jupyter notebooks) where users can write and run code in blocks, and immediately see the results of the code execution.

To use this functionality, you need to create a new Athena workgroup (which we cover in more detail later in this chapter) and configure the workgroup engine to be Spark. You can then create a new Notebook associated with the Spark workgroup, and specify options such as the idle timeout, the driver node size, executor size, as well as the maximum concurrent executors. You can then immediately start using Spark code to interactively query your data lake data.

Using Athena notebooks you can also plot your data, creating visualizations based on your data. For example, you can load data into a Spark data frame based on a SQL query, covert the data frame into a Python Pandas data frame, and then use various libraries to plot the data. For an example walk-through of how to do this, refer to the AWS blog *Explore your data lake using Amazon Athena for Apache Spark* at `https://aws.amazon.com/blogs/big-data/explore-your-data-lake-using-amazon-athena-for-apache-spark/`.

Amazon Athena for Apache Spark is useful for data engineers that are familiar with coding with Spark and Python, and want to interactively explore their data using code. Spark compute resources are deployed rapidly and Athena auto-scales compute resources based on the requirements of the queries that are run (up to the maximum concurrency that you specify when creating the Notebook).

Amazon Athena also includes support for a number of open table formats, which we look at in the next section.

Working with open table formats in Amazon Athena

Amazon Athena includes support for working with a number of popular **open table formats**, including **Apache Iceberg**, **Apache Hudi**, and the **Linux Foundation Delta Lake** table format. These table formats, which we cover in more detail in *Chapter 14, Building Transactional Data Lakes*, enable data lakes to provide data integrity. **ACID transactions** provided by these formats (referring to the properties of **atomicity**, **consistency**, **isolation** and **durability**), enable data lakes to perform updates to data and provide a consistent view of data, in a way that is similar to what was previously only available in databases and data warehouses.

However, the Athena support for each table format differs. At the time of writing, Apache Iceberg has the most support, enabling you to both query Iceberg data and perform updates and deletes to data in Iceberg. For Delta Lake format tables, Athena supports read-only queries (statements such as `UPDATE`, `INSERT` and

`DELETE` are not currently supported). For the Hudi table format, Athena can read those tables, but does not support writing to those tables.

For the latest support information on these open table formats, see the Athena documentation titled *Using Athena ACID transactions* at `https://docs.aws.amazon.com/athena/latest/ug/acid-transactions.html`.

Let's now look at how you can provision dedicated Athena capacity, instead of using the default on-demand capacity.

Provisioning capacity for queries

By default, Athena usage is priced by the amount of data that a query scans. However, Athena also includes the option to provision a specific amount of capacity to use for queries. When using provisioned capacity, you are charged based on the compute resources you provision, and there is not a per query charge based on data scanned.

When you provision capacity for Athena, you get dedicated processing capacity for your queries. To use this functionality, you specify the number of **Data Processing Units (DPUs)** that you need for your queries, and you assign one or more Workgroups to use that capacity (we discuss Workgroups in more detail in the next section of this chapter).

DPUs are an indication of the compute resources assigned for your provisioned capacity reservation. One DPU provides for 4 CPUs, and 16 GB of memory. The more DPUs assigned, the more concurrent queries you can run. Amazon provides the following guidance for the number of DPUs needed for a specific number of concurrent queries:

- For 10 concurrent queries, Amazon recommends 40 DPUs
- For 20 concurrent queries, Amazon recommends 96 DPUs
- For 30 or more concurrent queries, Amazon recommends 240 DPUs

The above provides general guidance on number of DPUs needed, but the actual number needed is based on your requirements and analysis patterns. If you do not have enough capacity to run all your queries at peak times, Athena will queue you query and run it when capacity becomes available.

At the time of writing, Athena provisioned capacity is billed at $0.30 per DPU hour billed. While billing is per minute, there is a minimum billing period of 8 hours

for any capacity that you provision. There is also a requirement to provision a minimum of 24 DPUs in a reservation, and you can increase DPU capacity in 4 DPU increments at any time. Capacity can be held for as long as needed, and can also be cancelled at any time (however you are always billed for a minimum of 8 hours for each capacity reservation).

If you have a few data analysts or data engineers that use Athena on an ad hoc basis to better understand the data in your data lake, then using Athena on-demand makes sense. However, if you have a large number of data consumers that are doing constant queries of your data, or you use Athena for performing data transformation tasks on a regular basis, then Athena provisioned capacity may help you to reduce costs.

When you create an Athena Provisioned Capacity reservation, you need to assign one or more workgroups to that reservation. In the next section we do a deeper dive into Athena workgroups.

Managing groups of users with Amazon Athena workgroups

Athena workgroups are a powerful mechanism for separating different groups and types of user queries (such as SQL based queries, and Spark Notebook queries), for applying cost controls, assigning provisioned capacity, and for implementing strong governance on Athena usage.

All queries within Athena are run in a specific workgroup, and you can apply a number of configuration settings to each workgroup to control costs and governance for the users in that workgroup. The following list shows the configuration items that can be controlled at the workgroup level:

- The analytic engine that users in this workgroup use (either Athena SQL or Apache Spark)
- Whether the Athena engine version used for this workgroup is selected and upgraded automatically, or whether you manually specify the engine version to use
- The S3 location where the results of queries are written to, and whether these files are encrypted or not
- Various general settings, such as the option to publish query metrics to Amazon CloudWatch, and an option to prevent users overriding the workgroup settings

- Limits for each query (the amount of data a single query is allowed to scan)
- Query limits for SQL engine workgroups (how much data can the workgroup as a whole scan within a specified time period)

These options enable administrators to manage governance concerns (where results are written to, and encryption settings), as well as cost controls (how much data can be scanned for SQL queries, at either the user or workgroup level).

Athena workgroups enable administrators to separate groups of users and automated ETL processes into separate workgroups, each with their own settings. By using IAM policy settings, users (or processes) can be restricted to only be able to run their queries within specified workgroups.

Let's start by looking at the cost controls enabled by workgroups.

Managing Athena costs with Athena workgroups

By default, Athena SQL query costs are based on the amount of data that is scanned by a query, and in the first section of this chapter, we looked at some of the ways that data can be optimized so that queries would scan less data, and therefore reduce costs.

However, some of those optimizations are based on writing efficient SQL queries, and it's not unusual for organizations to be concerned that users are going to accidentally run SQL queries that are not optimized and end up scanning massive amounts of data. As such, organizations want a way to control the amount of data that's scanned by different users or teams when they are not using provisioned capacity.

By using Athena workgroups, an administrator can either assign the workgroup to use provisioned capacity (in which case there is no per query cost), or they can place cost controls on the workgroup.

Per query data usage control

You can configure the maximum amount of data that can be scanned by a single SQL query using per query data usage controls. If a user runs a query and Athena ends up trying to scan more data than allowed by the control, the query is cancelled. However, note that the AWS account is still billed for the amount of data that was scanned up until the query was cancelled.

As a practical example, you may have a group of users that are relatively inexperienced with SQL and want to have a sandbox environment where they can run ad hoc queries safely. In this scenario, you could create an Athena workgroup called *sandbox* and configure these users to have access to the sandbox workgroup. You could configure the workgroup to have a per-query limit of 100 GB, for example, which would ensure that no individual query would cost more than $0.50.

Per query data limits are useful for scenarios where you want to have hard control over the amount of data that's scanned by each query. However, this control is restrictive in that it automatically cancels any query that exceeds the specified amount of data that's scanned. An alternative option for controlling costs is to configure workgroup data usage controls.

Athena workgroup data usage controls

With workgroup data usage controls, you have the flexibility to configure the maximum amount of data that's scanned by the entire workgroup, within a specified time period. When the limit for the workgroup is reached, queries are not automatically cancelled, but an **Amazon Simple Notification Service** (**SNS**) message is triggered, or a Lambda function is run to take some programmatic action.

For each workgroup, you can configure multiple alerts. For example, you can configure an initial workgroup data usage control for a maximum data scan of 3 TB per day, and have that trigger an SNS message to an SNS topic that sends an email to an administrator when the limit is reached.

You can also configure a second alert, that when data scanned reaches 5 TB for the workgroup in a day, another SNS message is sent to a topic, where the SNS topic triggers a Lambda function that programmatically updates the workgroup to have a `STATE` of `DISABLED` (preventing any additional queries from being run within that workgroup).

You also have the option of adding tags to a Workgroup. If that tag is configured as a cost allocation tag in the *Billing and Cost Management* console, the costs associated with running queries in that Workgroup appear in your *Cost and Usage Reports* with that cost allocation tag. This helps you understand and monitor Athena costs by Workgroup. To learn more about monitoring costs with *cost allocation tags*, see the following AWS documentation: https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/cost-alloc-tags.html.

Having reviewed how to place limits on costs related to Athena SQL queries, let's now look at how to implement additional workgroup settings for implementing strong governance controls.

Implementing governance controls with Athena workgroups

As discussed previously, Athena workgroups enable the separation of query execution between different users, teams, or systems. Every Athena query runs within a workgroup, and you can use IAM policies to limit which workgroups a specific user, or process, has access to.

Most organizations have concerns around governance and security, and workgroups include functionality that can address some of these concerns. Common concerns include the following:

- Athena saves the results of all queries, as well as associated metadata, on S3. These results could contain confidential information, so organizations want to ensure this data is protected.
- Multiple teams in an organization may use Athena in the same AWS account, and organizations want items such as query history to be stored separately for each team.

In the Athena console, users can save queries that they frequently run, and a list of historical queries that they have run are also available. However, these lists only show queries for the workgroup where the query is run, so splitting up teams or projects into different workgroups ensures that query history and saved queries are visible only to the specific team associated with the workgroup. These could be separate workgroups for each team, separate workgroups for different applications, or separate workgroups for different types of users.

By default, each user can control several settings, but workgroups enable an administrator to override the users' settings, forcing them to use the workgroup settings.

The following are some of the configuration items that an administrator can enforce for members of a workgroup:

- **Query Result Location**: This is the S3 path where the results of Athena queries will be written. Users can set a query result location, but if this is set for the

workgroup and **Override client-side settings** is set on the workgroup, then this location will be used for all the queries that are run in this workgroup. This enables an organization to control where query result files are stored in S3, and the organization can set strict access control options on this location to prevent unauthorized users from gaining access to query results.
For example, each team can be assigned a different workgroup, and their IAM access policy can be configured to only allow read access to their query results.

- **Encrypt Query Results**: This option can be used to enforce that query results are encrypted, helping organizations keep in line with their corporate security requirements.
- **Metrics**: You can choose to send metrics to CloudWatch logs, which will reflect items such as the number of successful queries, the query runtime, and the amount of data that's been scanned for all the queries that are run within this workgroup.
- **Override client-side settings**: If this item is not enabled, then users can configure their user settings for things such as query result location, and whether query results are encrypted. Therefore, it is important to enable this setting to ensure that query results are protected and corporate governance standards are met.
- **Requester pays S3 buckets**: When creating a bucket in Amazon S3, one of the options that's available is to configure the bucket so that the user that queries the bucket pays for the API access costs. By default, Athena will not allow queries against buckets that have been configured for requester pays, but you can allow this by enabling this item.
- **Tags**: You can provide as many `key:value` tags as needed to help with items such as cost allocation, or for controlling access to a workgroup. For example, you may have two Workgroups that have different settings for the query output location, based on different projects or use cases. You could provide a tag with the name of the team and then, through IAM policies, provide team members access to all Workgroups that are tagged with their team name.

For examples of how to manage access to workgroups using tags or workgroup names, see the Amazon Athena documentation titled *Tag-Based IAM Access Control Policies* (`https://docs.aws.amazon.com/athena/latest/ug/taqgs-access-control.html`).

So far in this chapter we have learned more about the features and functionality available in Athena, such as how you can use connectors to query data in a variety of systems (such as Google BigQuery, or a PostgreSQL database) using Athena,

and how Workgroups can be used to manage cost and enforce strong governance. We also reviewed some best practices for optimizing your Athena SQL queries from both a cost and performance perspective. Now, let's get hands-on with some of these concepts by creating and configuring a new Athena workgroup, and then running some SQL queries on our data lake data.

Hands-on – creating an Amazon Athena workgroup and configuring Athena settings

In this section, we're going to create and configure a new Athena workgroup, and set a per query data limit:

1. Log into **AWS Management Console** and access the Athena service using this link: `https://console.aws.amazon.com/athena`.
2. Expand the left-hand menu and click on **Workgroups** to access the workgroup management page.

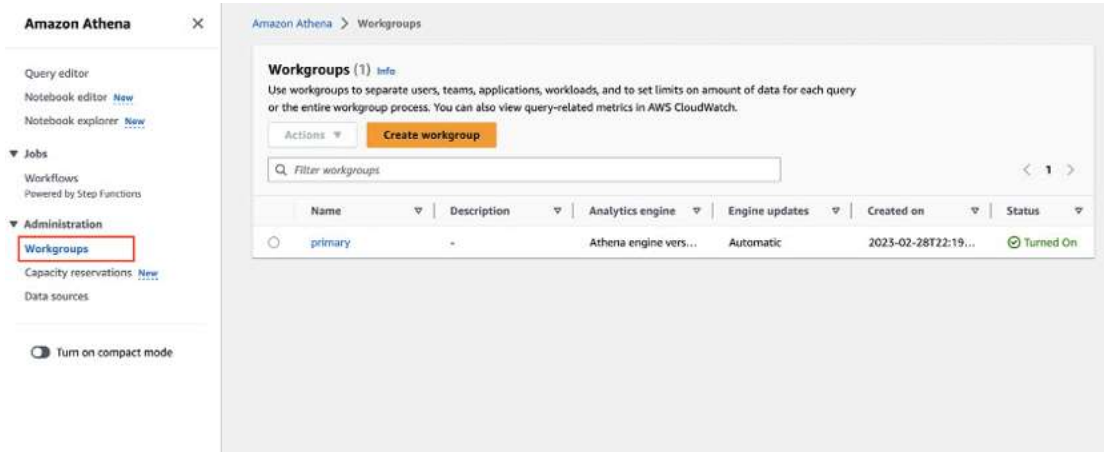

Figure 11.2: Athena Console showing Workgroups

3. On the **workgroup management** page, click on **Create workgroup** and enter the following values for our new workgroup. For the items not listed here, leave the defaults as-is:
    1. **Workgroup name**: Provide a descriptive name for the workgroup, such as `datalake-user-sandbox`.
    2. **Description**: Optionally, provide a description for this workgroup, such as `Sandbox workgroup for new datalake-users`.
    3. **Query result location** (in the **Query result configuration** section): In the hands-on exercises in *Chapter 4*, *Data Cataloging, Security, and Governance*, we created a bucket to store our Athena query results in (named `aws-`

`athena-query-results-dataengbook-<initials>`). Click the **Browse S3 button** next to **Query result location**, and select the previously created query result bucket selector, and then click **Choose**. To make the location of our query results unique for this workgroup, add the workgroup name to the end of the path. For example, the full path should be something like `s3://aws-athena-query-results-dataengbook-gse23/datalake-user-sandbox/` . Make sure that you include the trailing slash at the end of the path.

4. **Encrypt Query Results**: Tick this box to ensure that our query results are encrypted. When you select this, you will see several options for controlling the type of encryption. For our purposes, select **SSE_S3** for **Encryption type** (this specifies that we want to use S3 Server-Side Encryption rather than our own unique KMS encryption key).
5. **Override client-side settings** (under the **Settings** section): If we want to prevent our users from changing items such as the query result's location or encryption settings, we need to ensure that we select this option.

4. In the **Per query data usage control** section, we can specify a **Data limit** to limit the scan size for individual queries run in this workgroup. If a query scans more than this amount of data, the query will be canceled. Set the **Data limit** size to `10 GB` .

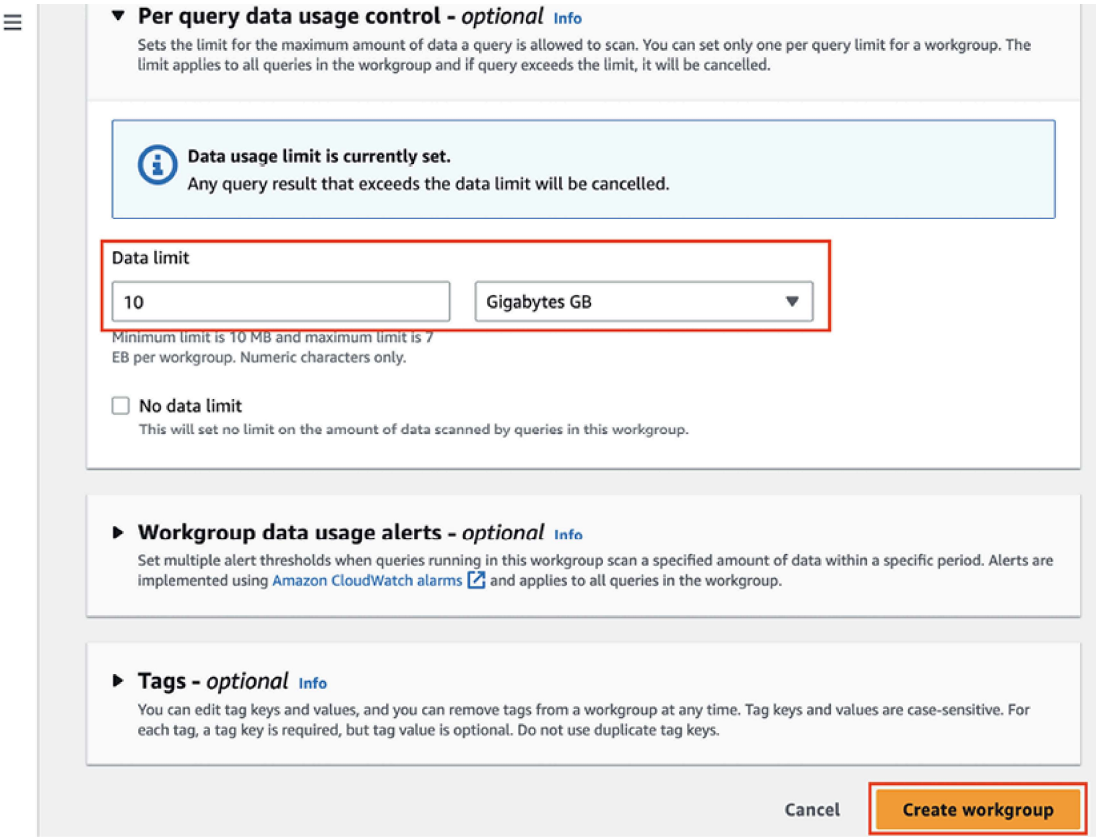

Figure 11.3: Athena workgroup – Per query data usage control

5. Optionally add any **Tags** you want to specify, and then click on **Create workgroup**.

We can also set a **workgroup data usage control** to manage the total amount of data that is scanned by all users of the workgroup over a specific period. We are not going to cover this now, but if you would like to explore setting this up, refer to the AWS documentation titled *Setting Data Usage Controls Limits*: https://docs.aws.amazon.com/athena/latest/ug/workgroups-setting-control-limits-cloudwatch.html.

Let's now get hands-on with how to switch into our new workgroup, and how to run some queries using Athena.

Hands-on – switching workgroups and running queries

By default, all users operate in the **primary** workgroup, but users can switch between any workgroup that they have access to. You can control workgroup access via IAM policies, as detailed in the AWS documentation titled *IAM Policies for Accessing Workgroups* : https://docs.aws.amazon.com/athena/latest/ug/workgroups-iam-policy.html

In the previous section, we created and configured a new workgroup, so we can now run some SQL queries and explore Athena's functionality further:

1. In the left-hand menu, click on **Query editor**. Once in the Query editor, use the **Workgroup** drop-down list selector to change to your newly created sandbox workgroup.

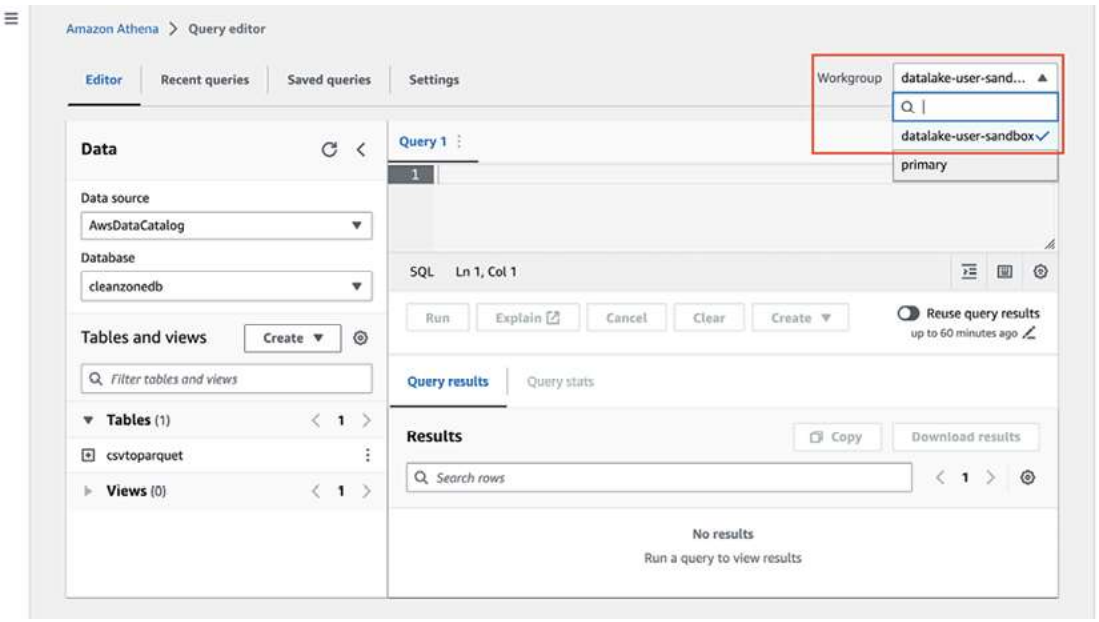

Figure 11.4: Switching Workgroups in the Athena Console

2. A pop-up dialog may appear for you to acknowledge that all the queries that are run in this workgroup will use the settings we configured previously. This is because we chose to **Overwrite client-side settings** when creating the workgroup. Click **Acknowledge** to confirm this.
3. In the Query editor, let's run our first query to determine which category of films is most popular with our streaming viewers. We're going to query the `streaming_films` table, which was the denormalized table we created in *Chapter 7, Transforming Data to Optimize for Analytics*. On the left-hand side of the Athena query editor, select the `curatedzonedb` from the **Database** drop-down, and then run the following query in the query editor:

```
SELECT category_name,
        count(category_name) streams
FROM streaming_films
GROUP BY category_name
ORDER BY streams DESC
```

This query performs the following tasks:

1. It selects the category name and a count of the total number of entries of that category in the table, and then it renames the count of queries column to create a new column heading of `streams`.
2. It selects this data from the `streaming_films` table. Since we selected the `curatedzonedb` from the dropdown on the left-hand side, Athena automati-

cally assumes that the table we are querying is in that selected database, so we don't need to specifically reference `curatedzonedb` in our query, although we could.

3. Then, it groups the results by `category_name`, meaning that one record will be returned per category.
4. Finally, it sorts the results by the `streaming` column, in descending order, so that the first result is the category with the highest number of streams. In the following screenshot, we can see that `Sports` was the most popular category from our streaming catalog:

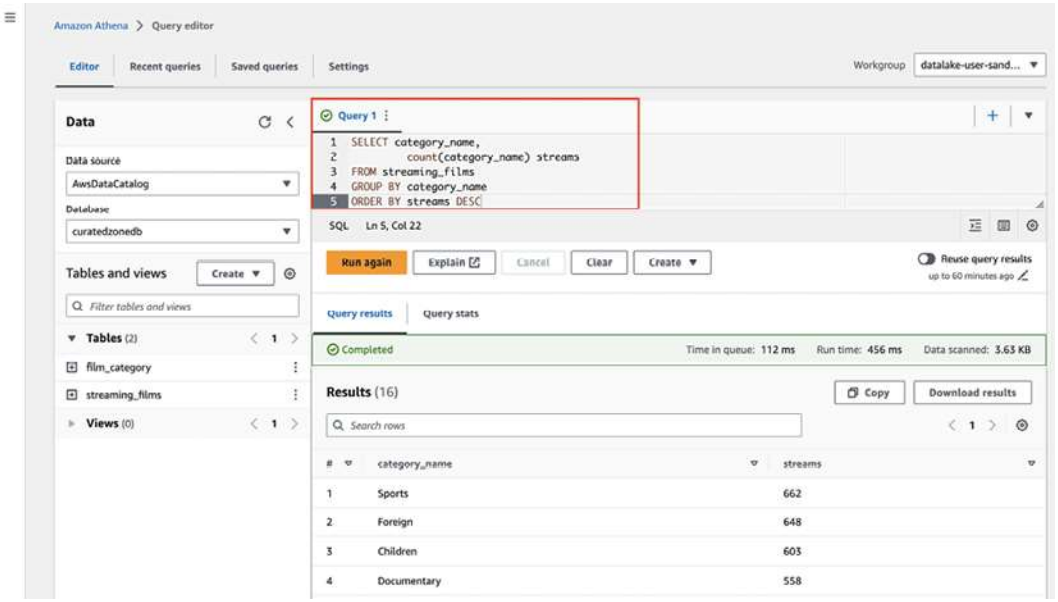

Figure 11.5: Athena query for the top streaming categories

> Note that the data in the `streaming_films` table was randomly generated by the Kinesis Data Generator utility in *Chapter 6, Ingesting Streaming and Batch Data*, so your results regarding the top category may be different.

4. Notice how under the query results on the right, there is an option for **Reuse query results**, which will re-use the previous result for an identical query run in the last 60 minutes, without rerunning the query (as we covered earlier in this chapter). Optionally run the query again a few times with this option not enabled and note the **run time** for each query. Then enable the **Reuse query results** option, and run the query a few more times, and compare the run-times. The query we have is a very simple query, but for complex queries, or

for cases where you have a BI or dashboarding tool using Athena, this option can make a significant performance difference.

5. If we have a query that we think we may want to run regularly (such as seeing the top category each day), we can **save** the query so that we don't need to re-type it each time we want to run it. To do so, click the three dots at the top of the query window, and click on **Save as**, as shown in the following screenshot.

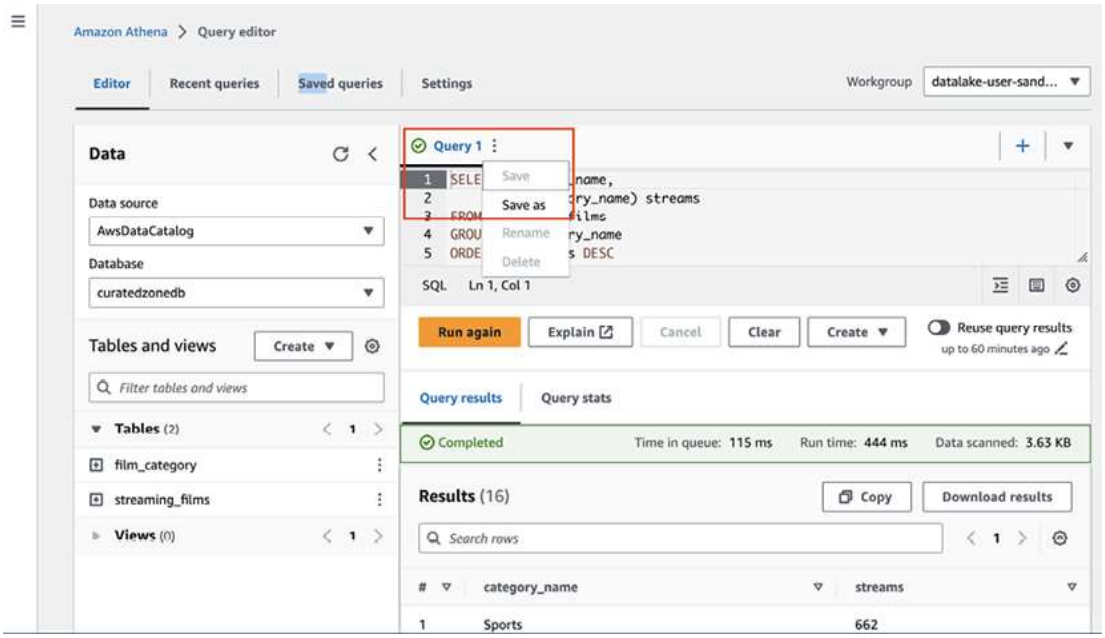

Figure 11.6: Save an Athena Query

6. Provide a name for the query (such as `Overall-Top-Streaming-Categories`) and a description (such as `Returns a list of all categories, sorted by highest number of views`). Then, click **Save query**.
7. Now, let's modify our query slightly to find out which `State` streamed the most movies out of our streaming catalog. Click on the plus (+) sign at the top right of the query window to open a new query window and enter the following query, and then click **Run**:

```
SELECT state,
        count(state) count
FROM streaming_films
GROUP BY state
ORDER BY count desc
```

Running this query using the data I generated indicates that **Alaska** was the most popular state for streaming movies from our catalog. Again, though, your results

may be different due to the random data we generated using the **Kinesis Data Generator**:

1. Click on the three dots and then **Save as** and provide a name (such as **Overall-Top-Streaming-States**) and a description for this query, and then click on **Save query**.
2. Click on the down arrow on the top right-hand side of the query editor and click **Close all tabs**. Then, via the top sAthena menu, click on **Saved queries**. Here, we can see the list of queries that we have previously saved, and we can easily select a query from the list if we want to run that query again. Note that saved queries are saved as part of the workgroup, so any of our team members that have access to this workgroup will also be able to access any queries that we have saved. If you click on one of the saved queries, it will open the query in a **New query** tab.
3. At the top of the Athena menu, click on **Recent queries**. Here, we can see a list of all the recent queries that have been run in this workgroup:

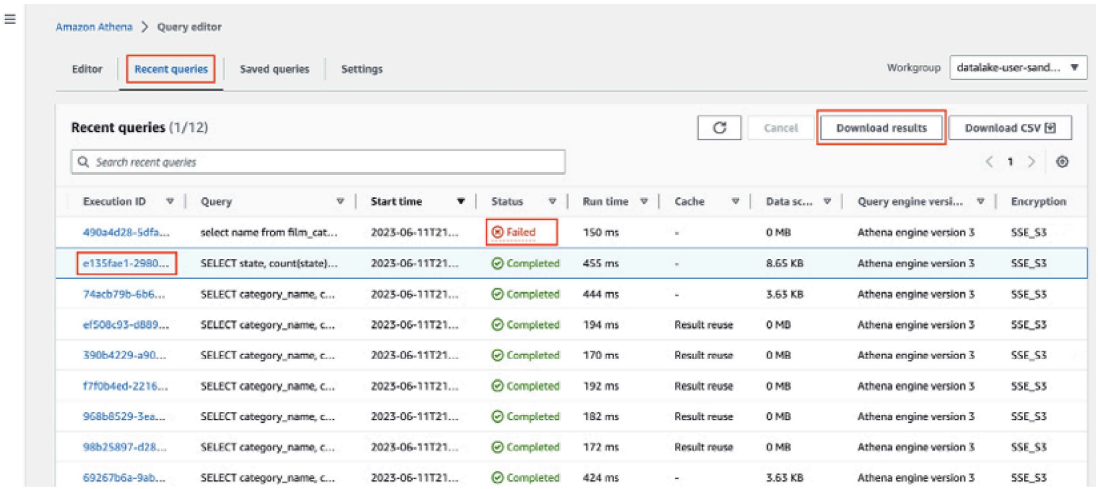

Figure 11.7: Amazon Athena recent queries tab

From here, we can take several actions, as follows:

1. If we want to rerun a query, we can click on the **Execution ID** of the query and it will open the query in a new query window.
2. To download the results that the query generated as a CSV file, click the selector for a query and then click on **Download Results** . Remember that query results are also always stored on S3 in the location set for **Query Result Location**. Note that the **Download CSV** button can be used to download the list of queries, along with the query details, in a CSV file.

3. To see the details of why a query failed, click on **Failed** under the Status column. A pop-up box will provide details of the error message that caused the failure.
4. Other information that we can see in the **Recent queries** tab is the Athena **engine version** that was used, whether the query result was returned from the query **cache** (which happens when we use the **Reuse query** option), as well as the **Run time** for the query.

Note that the **Recent queries** tab keeps a record of all the queries that have been run in the past 45 days.

In these hands-on exercises, you configured an Athena workgroup and made use of that workgroup to run several queries against data that you previously loaded and transformed in the data lake. You also learned how to save queries and view query history.

Summary

In this chapter, we learned more about the Amazon Athena service, an AWS-managed service that builds on the Apache Presto and Trino solutions to enable you to run SQL or Spark based queries against your data. We also looked at how to optimize our data and SQL queries to increase query performance and reduce costs.

Then, we explored advanced Athena functionality, including how Athena can be used as a SQL query engine not only for data in an Amazon S3 data lake, but also for external data sources such as other database systems, data warehouses, and even CloudWatch logs, using Athena Query Federation.

We wrapped up the theory part of this chapter by looking at Athena workgroups, which let us manage governance and costs, and they can be used to enforce specific settings for different teams or projects, and can also be used to limit the amount of data that is scanned by queries. In the last section of this chapter, we got hands-on with Athena, first creating a new workgroup, and then using that workgroup to run a number of SQL queries.

In the next chapter, we will explore another Amazon tool for data consumers as we look at how we can create rich visualizations and dashboards using **Amazon QuickSight**.

Learn more on Discord

To join the Discord community for this book – where you can share feedback, ask questions to the author, and learn about new releases – follow the QR code below:

https://discord.gg/9s5mHNyECd