4
Data Governance, Security, and Cataloging

Data governance and security are some of the most important topics to cover in a book that is all about data. Having the most efficient data pipelines, the fastest data transformations, and the best data consumption tools is not worth much if the data is not kept secure and governed correctly. Data must also be stored and accessed in a way that complies with local laws, and the data needs to be cataloged so that it is discoverable and useful to the organization.

Sadly, it is not uncommon to read about data breaches and poor data handling by organizations, and the consequences of this can include reputational damage to the organization, as well as potentially massive penalties imposed by the government. And once an organization causes damage to their customers (such as exposing them to potential identity theft through a data breach), it is difficult for the organization to regain that trust.

It is also not uncommon for organizations to find that they have massive quantities of data, but that they are not able to maximize the value of that data since it is siloed (contained across many different systems that are not connected) or of poor quality, or users just have no trust in the data they have access to.

In this chapter, we will do a deeper dive into best practices for handling data responsibly, and for making sure that the value of data can be maximized for an organization. We will cover the following topics:

- The many different aspects of data governance
- Data security, access, and privacy
- Data quality, data profiling, and data lineage
- Business and technical data catalogs
- AWS services that help with data governance
- Hands-on – configuring Lake Formation permissions

Technical requirements

To complete the hands-on exercises included in this chapter, you will need an AWS account where you have access to a user with administrator privileges (as covered in *Chapter 1, An Introduction to Data Engineering*).

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/Chapter04`

The many different aspects of data governance

Data governance is a wide-ranging topic, covering many different aspects. If you do a Google search for the definition of data governance, you are likely to see many different definitions. At its core though, data governance is the various things an organization needs to do to ensure the secure, compliant, and effective use of data, from the time the data is created through to when it is archived or deleted.

This includes processes that make the data **discoverable**, **understandable**, and **usable**, while ensuring that the data is of **high quality** and is **protected and secured**. This covers all data within an organization; however, for the purposes of this book, we will only be focusing on data governance for analytic data.

Most organizations consist of many different business units or teams, and each of these generates their own data, and also has their own specific requirements for accessing data from other parts of the organization. But if a team has no way to centrally publish their datasets, or to discover datasets published by other parts of the organization, this significantly impacts the value of the data. Data is most valuable to an organization when all data generated can be used across all parts of the organization (in a well-governed manner).

But just being able to discover data generated across the organization is not helpful if the data is not of good quality, cannot be easily understood, or cannot be trusted. Also, if it is not easy to get access to discovered data in a secure way, without significant delays and complicated processes, then the value of that data is impacted.

But governance also extends to ensuring that the data that is generated is secured and handled in a responsible way.

Data security dictates how an organization should protect data to ensure that it is stored securely (such as in an encrypted state), and that access by unauthorized entities is prevented. For example, all the things an organization does to prevent falling victim to a ransomware attack, or having their data stolen and sold on the dark web, fall under data security.

The **responsible handling of data** involves making sure that only people who need access to specific datasets have that access (such as ensuring that data is not just generally open to all users of a system without considering whether they need access to that data to perform their jobs). Responsible handling also means ensuring that an organization only uses and processes data on individuals in approved ways, and that organizations provide data disclosures as required by law.

As the security of data needs to be the top priority for an organization, let's start with a deeper dive into the topics of security, privacy, and

data handling.

Data security, access, and privacy

Not providing adequate protection and security for an organization's data, or not complying with relevant governance laws, can end up being a very expensive mistake for an organization.

According to an article on CSO Online titled *The biggest data breach fines, penalties, and settlements so far* (https://www.csoonline.com/article/3410278/the-biggest-data-breach-fines-penalties-and-settlements-so-far.html), penalties and expenses related to data breaches have cost companies over $4.4 billion (and counting).

For example, Equifax, the credit agency firm, had a data breach in 2017 that exposed the personal and financial information of nearly 150 million people. As a result, they agreed to pay at least $575 million in a settlement with several United States government agencies, and U.S. states.

But beyond financial penalties, a data breach can also do incalculable damage to an organization's reputation and brand. Once you lose the trust of your customers, it can be very difficult to earn that trust back.

Beyond data breaches where personal data is stolen from an organization's system, failure to comply with local regulations relating to how data is handled can also be costly. There are an increasing number of laws that define under what conditions a company may collect, store, and process personal information. Not complying with these laws can result in significant penalties for an organization, even in the absence of a data breach.

For example, Google was hit with a fine of more than $50 million for failing to adequately comply with aspects of a European regulation known as the **General Data Protection Regulation** (**GDPR**). Google

appealed the decision, but in 2020, the decision was upheld by the courts, leaving the penalty on Google in place.

Common data regulatory requirements

No matter where you operate in the world, there are very likely several regulations concerning data privacy and protection that you need to be aware of, and plan for, as a data engineer. A small selection of these include the following:

- The **General Data Protection Regulation (GDPR)** in the European Union
- The **California Consumer Privacy Act (CCPA)** and the **California Privacy Rights Act (CPRA)** in California, USA
- The **Personal Data Protection (PDP) Bill** in India
- The **Protection of Personal Information Act (POPIA)** in South Africa

These laws can be complex and cover many different areas, which is far beyond the scope of this book. However, generally, they involve individuals having the right to know what data a company holds about them; ensuring adequate protection of personal information that the organization holds; enforcing strict controls around data being processed; and in some cases, the right of an individual to request their data being deleted from an organization's system.

In the case of GDPR, an organization is subject to the regulations if they hold data on any resident of the European Union, even if the organization does not have a legal presence in the EU.

In addition to these broad data protection and privacy regulations, many regulations apply additional requirements to specific industries or functions. Let's take a look at some examples:

- The **Health Insurance Portability and Accountability Act** (**HIPAA**), which applies to organizations that store an individual's healthcare and medical data
- The **Payment Card Industry Data Security Standard** (**PCI DSS**), which applies to organizations that store and process credit card data

Understanding what these regulations require and how best to comply with them is often complex and time-consuming. In this chapter, we will look at general principles that can be applied to protect data used in analytic pipelines; however, this chapter is not intended as a guide on how to comply with any specific regulation.

GDPR specifies that in certain cases, an organization must appoint a **Data Protection Officer** (**DPO**). The DPO is responsible for training staff involved in data processing and conducting regular audits, among other responsibilities.

If your organization has a DPO, ensure you set up a time to meet with the DPO to fully understand the regulations that may apply to your organization and how this may affect analytic data. Alternatively, work with your **Chief Information Security Officer** (**CISO**) to ensure your organization seeks legal advice on which data regulations may apply.

If you must participate in a compliance audit for an analytic workload running in AWS, review the *AWS Artifact* service (`https://aws.amazon.com/artifact/`), a self-service portal for on-demand access to AWS's compliance reports.

Core data protection concepts

There are several concepts and terminology related to protecting data that are important for a data engineer to understand. In this section, we will briefly define some of these.

Personally identifiable information (PII)

Personally identifiable information (**PII**) is a term commonly used in North America to reference any information that can be used to identify an individual. This can refer to either the information on its own being able to identify an individual or where the information can be combined with other linkable information to identify an individual. It includes information such as full name, social security number, IP address, and photos or videos.

PII also covers data that provides information about a specific aspect of an individual (such as a medical condition, location, or political affiliation).

Personal data

Personal data is a term that is defined in GDPR and is considered to be similar to, but broader than, the definition of PII. Specifically, GDPR defines personal data as follows:

> *Any information relating to an identified or identifiable natural person ("data subject"); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person.*

GDPR, Article 4, Definitions (https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679#d1e1374-1-1)

Encryption

Encryption is a mathematical technique of encoding data using a key in such a way that the data becomes unrecognizable and unusable. An authorized user who has the key used to encrypt the data can use the key to decrypt the data and return it to its original plaintext form.

Encrypted data may be able to be decrypted by a hacker without the key through the use of advanced computational resources, skills, and time. However, a well-designed and secure encryption algorithm increases the difficulty of decrypting the data without the key, increasing the security of the encrypted data.

There are two important types of encryption and both should be used for all data and systems:

1. **Encryption in transit**: This is the process of encrypting data as it moves between systems. For example, a system that migrates data from a database to a data lake should ensure that the data is encrypted before being transmitted, that the source and target endpoints are authenticated, and the data can then be decrypted at the target for processing. This helps ensure that if someone can intercept the data stream during transmission, the data is encrypted and therefore unable to be read and used by the person who intercepted the data. A common way to achieve this is to use the **Transport Layer Security** (**TLS**) protocol for all communications between systems.
2. **Encryption at rest**: This is the encryption of data that is written to a storage medium, such as a disk. After each phase of data processing, all the data that is persisted to disk should be encrypted.

Encryption (in transit and at rest) is a key tool for improving the security of your data, but other important tools should also be considered, as covered in the subsequent sections.

Anonymized data

Anonymized data is data that has been altered in such a way that personal data is irreversibly de-identified, rendering it impossible for any PII data to be identified. For example, this could involve replacing PII data with randomly generated data in such a way that the randomization cannot be reversed to recreate the original data.

Another way anonymization can be applied is to remove most of the PII data so that only a few attributes that may be considered PII remain, but with enough PII data removed to make it difficult to identify an individual. However, this contains risk, as it is often still possible to identify an individual even with only minimal data. A well-known study (https://dataprivacylab.org/projects/identifiability/paper1.pdf) found that with just the ZIP code, gender, and date of birth information, 87% of the population in the United States can be uniquely identified.

Pseudonymized data/tokenization

Pseudonymized data is data that has been altered in such a way that personal data is de-identified. While this is similar to the concept of anonymized data, the big difference is that with pseudonymized data, the original PII data can still be accessed (if a user has a legitimate need to access the PII data and has the necessary system authorization).

Pseudonymized data is defined by GDPR as data that cannot be attributed to a specific data subject without the use of separately kept "*additional information*."

There are multiple techniques for creating pseudonymized data. For example, you can replace a full name with a randomly generated token, a different name (so that it looks real but is not), a hash repre-

senting the name, and more. However, whichever technique is used, it must be possible to still access the original data.

One of the most popular ways to do this is to have a tokenization system generate a unique, random token that replaces the PII data.

For example, when a raw dataset is ingested into the data lake, the first step may be to pass the data through the **tokenization** system. This system will replace all PII data in the dataset with an anonymous token and record each **real_data token substitution** in a secure database. Once the data has been transformed, if a data consumer requires access and is authorized to access the PII data, they can pass the dataset to the tokenization system to be detokenized (that is, have the tokens replaced with the original, real values).

The benefit of a tokenization system is that the generated token is random and does not contain any reference to the original value, and there is no way to determine the original value just from the token. If there is a data breach that can steal a dataset with tokenized data, there is no way to perform reverse engineering on the token to find the original value.

However, the tokenization system itself contains all the PII data, along with the associated tokens. If an entity can access the tokenized data and is also able to comprise the tokenization system, they will have access to all PII data.

Therefore, it is important that the tokenization system is completely separate from the analytic systems containing the tokenized data, and that the tokenization system is protected properly.

On the other hand, **hashing** is generally considered the least secure method of de-identifying PII data, especially when it comes to data types with a limited set of values, such as social security numbers and names.

Hashing uses several popular hashing algorithms to create a hash of an original value. An original value, such as the name "*John Smith*," will always return the same hash value for a specific algorithm.

However, all possible social security numbers and most names have been passed through popular hashing algorithms and lookup tables have been created, known as **rainbow tables**. Using these rainbow tables, anyone can take a hashed name or social security number and quickly identify the original value.

For example, if you use the SHA-256 hashing algorithm, the original value of "*John Smith*" will always return "*ef61a579c907bbed674c0dbcbcf7f7af8f851538eef7b8e58c5bee0b8cfdac4a.*"

If you used the SHA-256 hashing algorithm to de-identify your PII data, it would be very easy for a malicious actor to determine that the preceding value referenced "*John Smith*" (just try Googling the preceding hash and see how quickly the name John Smith is revealed). While there are approaches to improving the security of a hash (such as salting the hash by adding a fixed string to the start of the value), it is still generally not recommended to use hashing for any data that has a well-known, limited set of values, or values that could be guessed.

Authentication

Authentication is the process of validating that a claimed identity is that identity. A simple example is when you log in to a **Google Mail** (**Gmail**) account. You provide your identity (your Gmail email address) and then validate that it is you by providing something only you should know (your password), and possibly also a second factor of authentication (by entering the code that is texted to your cell phone, for example).

Authentication does not specify what you can access but does attempt to validate that you are who you say you are. Of course, authentication

systems are not foolproof. Your password may have been compromised on another website, and if you had the same password for your Gmail account, someone could use that to impersonate you. If you have **multi-factor authentication** (**MFA**) enabled, you receive a code on your phone or a physical MFA device that you need to enter when logging in, and that helps to further secure and validate your identity.

Federated identity is a concept related to authentication and means that responsibility for authenticating a user is done by another system. For example, when logging in to the AWS Management Console, your administrator could set up a federated identity so that you use your Active Directory credentials to log in via your organization's access portal, and the organization's Active Directory server authenticates you. Once authenticated, the Active Directory server confirms to the AWS Management Console that you have been successfully authenticated as a specific user. This means you do not need a separate username and password to log in to the AWS system, but you can use your existing Active Directory credentials to be authenticated to an identity in AWS.

Authorization

Authorization is the process of authorizing access to a resource based on a validated identity. For example, when you log in to your Google account (where you are authenticated by your password, and perhaps a second factor such as a code that is texted to your phone), you may be authorized to access that identity's email, and perhaps also the Google Calendar and Google Search history for that identity.

For a data analytics system, once you validate your identity with authentication, you need to be authorized to access specific datasets. A data lake administrator can, for example, authorize you to access data that is in the Conformed Zone of the data lake, but not grant you access to data in the Raw Zone.

Putting these concepts together

Getting data protection right, and ensuring that you comply with local compliance regulations, does not happen by itself. It is important that you plan for and thoughtfully execute the process of protecting your data and ensuring data compliance. This will involve using some of the concepts introduced previously, such as the following:

- Making sure PII data is replaced with a token as the first processing step after ingestion (and ensuring that the tokenization system is secure).
- Encrypting all data at rest with a well-known and reliable encryption algorithm and ensuring that all connections use secure encrypted tunnels (such as by using the TLS protocol for all communications between systems).
- Implementing federated identities where user authorization for analytic systems is performed via a central corporate identity provider, such as Active Directory. This ensures that, for example, when a user leaves the company and their Active Directory account is terminated, their access to analytic systems in AWS is terminated as a result.
- Implementing least privilege access, where users are authorized for the minimum level of permissions that they need to perform their job.

This is also not something that a data engineer should do in isolation. You should work with your organization's security and governance teams to ensure you understand any legal requirements for how to process and secure your analytical data. You should also regularly review, or audit, the security policies in place for your analytic systems and data.

Ensuring the security of your data, protecting PII, and making sure that data is used appropriately need to be a top priority for every organization. However, if the data that is being generated is not of high

quality, and not trusted and understood by people across your organization, then the value of that data is limited. In the next section, we look at important ways to manage the quality of your data, and ways to build trust in your data across the organization.

Data quality, data profiling, and data lineage

In this section, we look at three different, but related, concepts: data quality, data profiling, and data lineage. Each of these aspects of data governance are important tools for ensuring that data that is shared within your organization is of high quality, and that teams across your organization can have confidence when accessing and using the data.

Data quality

Having high-quality data is essential for ensuring that an organization is equipped to make the best data-driven decisions, and to be effective in all activities that are data-driven (such as marketing campaigns).

There are many different aspects to measuring **data quality**, and data quality is important in all phases of the data lifecycle. If data in the source production database is not captured correctly, then when that data is copied over to analytical systems, the analytical systems will have incorrect or missing data. For example, if the source system does not enforce that the date of birth is captured when creating a new customer record, then an analytical system using that data cannot rely on using a date of birth field, as it may be null for some records. Invalid records can also be created when users do manual data entry, as they may make a typo or capture a number incorrectly.

However, data quality issues can be introduced at any part of the data pipeline. As data is bought in from multiple different systems, datasets are joined, and calculations are made, data can be corrupted. Also, at times, ETL jobs in a pipeline may fail or be interrupted at some point,

and the resulting data that is generated may be incomplete. This can lead to further issues down the line with subsequent jobs.

Data quality can be measured in many different ways, depending on the perspective of the person reviewing the data. Some of the common ways that data quality is measured include:

- **Accuracy** – does the data reflect the real facts?
- **Completeness** – does the data have all the required fields completed, such as date of birth or email?
- **Consistency** – does data from different data sources match and do the formats, such as the date format, match?
- **Timeliness** – is the data up to date?

For a data engineer, though, the primary concern is whether the content of the data is as expected. For example, if a column in a dataset is expected to be an integer, but some rows have strings in that field, then that could potentially cause the ETL pipeline to fail. Or, if a data processing job depends on each row having a valid email address, but some rows are missing an email address (or have an invalid email address), this again could cause a job to fail. Another example is where a column is meant to have a percentage between 0 and 100, and some rows have values that are outside of that range.

As a result, it is common for data engineers to create a data quality check job as part of the data pipeline. In these data quality jobs, the data engineer will specify a number of rules that are used to evaluate the quality of data, and a data quality report can be generated that provides details on the results. For example, a data quality rule could be created that checks to make sure that the email address field has a valid value for at least 90% of rows in the dataset. Or, the data quality check could ensure that every phone number value in a dataset consists of exactly 10 characters. The data engineer can then decide on what action to take for rows that do not pass the data quality rule check.

There are various commercial and open-source tools that can be used to help create these jobs to evaluate data quality, and the AWS Glue service also includes functionality for evaluating data against a set of rules. Later in this chapter, we will do a deeper dive into AWS Glue Data Quality functionality.

Data profiling

Data profiling is the process of analyzing a dataset, and then reporting on various aspects of the data content. This is useful to give both data engineers and other potential users of the data better insight into the underlying data. Data profiling can also be very useful to quickly identify potential data quality issues in a dataset, as we explain below.

A data profiling job analyzes all the data in a dataset, and then provides metrics for each column in the dataset. For example, if one of the columns in the dataset is a string, the data profiling job may report on the following attributes of that column:

Missing values. This reports on how many rows have a null, or no data, for this column. If we are planning to use a dataset for an email-based marketing campaign, but the data profiling job indicates that 54% of rows have a null for the email address, then we can very quickly determine that this dataset would not be good for our planned use case.

Distinct values. This reports on how many distinct values are contained in this column. For example, if the column was to report on which US state a customer was in, we would expect there to be no more than 50 distinct values, since there are 50 states in the USA. If a higher number was reported for this column, then we would have to assume that data was not captured consistently, such as some rows having **NY** recorded for state, while other rows have captured the full name of **New York**.

Unique values. This reports on how many rows have a unique value for this column (i.e., it is the only row that contains a specific value for this column).

Minimum/maximum string length. This reports on the number of characters for the shortest string in the column, as well as the longest string in the column.

A dataset that includes number-based columns (such as integers or doubles) may report on similar values, but instead of reporting on minimum/maximum string length, for numbers, the report may include the minimum/maximum value. For a column that records a percentage that is expected to be between 0 and 100, if the minimum reported was a negative number, or the maximum recorded was over 100, then this would indicate likely data quality issues.

In this way, data profiling is closely related to data quality, as it helps to identify potential data quality issues very quickly. But data quality requirements may be different for different users of a dataset.

For example, a team that uses the master customer data table for reporting on the number of users in each country or state may not care whether every customer record has an email address and phone number, but does require the name of the *Country* or *State* that is part of the user's address to be captured consistently. On the other hand, a marketing team that wants to send email-based promotions to all users may not care whether the customers' *Country* or *State* has been captured consistently, but does require that they have a valid *email address* for every customer.

In this way, data profiling does not make any assessment about the quality of data, but rather just reports on the profile, or shape, of the data. The end user who wants to use a specific dataset for a specific purpose can then review the data profile information to get a quick assessment as to whether the dataset seems to be appropriate for their

use or not. If the data profile does not highlight any obvious problems with the data based on the specific use case, the data engineer can request access to the dataset and then establish a data quality job to further analyze the data.

For example, even if the data profile report indicates that 100% of rows have an email address captured, the data engineer can create a data quality rule to validate whether every email address appears to be valid by ensuring each email address contains an "at" sign and a string with a dot separator in the domain portion of the email address. In this way, if one of the data records has an email value of "anna@gmail," this would not be a null value, so the data profiling report may still indicate that 100% of records have an email address specified. But it is the data quality rule that would evaluate the specific value and be able to determine that the domain portion has not been correctly captured.

Data lineage

Another concept in data management that helps to build trust in a dataset is the understanding of how a dataset has been created, which can be captured with data lineage. **Data lineage** allows a user to view information about the original sources of a dataset, as well as the transformations that have been applied to those sources.

For example, let's say you work for a company that has a streaming movie business and you need to identify the top 10 movies for each country that you operate in. To do this, you create a table called `country_top_movies` by joining your `customer`, `movies`, and `streaming_stats` tables. In your ETL job, you group by country and limit the results to the top 10 movies for each country.

When you look at the data lineage for this job, you will see the three source tables, as well as details about the transformations that were applied to these tables after the join.

When a data analyst is searching for a dataset, being able to view the data lineage that shows how that table was created helps to build trust that the table is appropriate for their use case. They gain insight into what the source tables were and the types of transformations that were applied, and using that information can decide whether the resulting dataset meets their requirements.

The results of data quality checks, information on the data profile, and a visualization of data lineage can all be recorded in a data catalog, enabling users to search for datasets and dig deeper into information about that dataset. Let's look at the role of data catalogs in more detail.

Business and technical data catalogs

You have probably heard about swamps, even if you have never actually been to one. Generally, swamps are known to be wet areas that smell pretty bad, and where some trees and other vegetation may grow, but the area is generally not fit to be used for most purposes (unless, of course, you're an ogre similar to Shrek, and you make your home in the swamp!).

In contrast to a swamp, when most people think about a lake, they picture beautiful scenery with clean water, a beautiful sunset, and perhaps a few ducks gently floating on the water. Most people would hate to find themselves in a swamp if they thought they were going to visit a beautiful lake.

In the world of data lakes, as a data engineer, you want to provide an experience that is much like the pure and peaceful lake described previously, and you want to avoid your users finding that the lake looks more like a swamp. However, if you're not careful, your data lake can become a data swamp, where there are lots of different pieces of data around, but no one is sure what data is there. Then, when they do happen to find some data, they don't know where the data came from or whether it can be trusted. Ultimately, a data swamp can be a dumping ground for data that is not of much use to anyone. Or, more danger-

ously, it can allow people to use datasets that may not be the right dataset for the purpose and the data quality may be low or out of date. Making business decisions on the wrong dataset can lead to the wrong decision being made, which can negatively impact a company financially, or reputationally.

To avoid creating a data swamp, you need a well-organized data catalog.

Implementing a data catalog to avoid creating a data swamp

With some careful upfront planning and the right tools and policies, it is possible to avoid a data swamp and instead offer your users a well-structured, easy-to-navigate data lake.

Avoiding a data swamp is easy in theory – you just need two important things:

- A central data catalog that can be used to keep a searchable record of all the datasets in the data lake
- Policies that ensure useful metadata is added to all the entries in the data catalog, and policies for ensuring that only high-quality data is allowed to be added to the data catalog

While that may sound pretty straightforward, the implementation details matter and things are not always as simple in real life. You need to have a data platform that has rules in place to only allow data to be added to the central catalog that meets your specific requirements (such as certain metadata that must be associated with the dataset, and ensuring that the dataset meets certain data quality rules).

If you allow anyone to publish any dataset in your central catalog, you are very likely to end up with a data swamp. There may be many datasets published in the catalog, but users will not be able to identify

which are trustworthy and useful datasets, or which are updated regularly, and will not have the context to help them understand where the dataset came from (its lineage) and what business purpose it serves.

When we talk about a central catalog, we are generally referring to a business data catalog, and the business catalog is usually associated with one or more technical catalogs. Let's take a closer look at the different types of catalogs.

Business data catalogs

The business data catalog (often referred to just as the data catalog) is a central repository that stores metadata about the different datasets in your environment. The purpose of the catalog is to enable users across your organization to discover available datasets and learn more about those datasets through the associated metadata.

For example, you may have a dataset called *Top Movies by Country* that lists the top 10 most streamed movies over the past 8 hours for each country your movie steaming company operates in. This dataset is created by an ETL job that runs every 8 hours, joining data from multiple different database tables. The ETL job and dataset are created by the **Playback Operations** team, as they need this data so that their streaming video player can display the current most popular movies based on the country a user is in. The table that is created has the following fields:

- Country
- Movie_Rank
- Movie_ID
- Title
- Genre
- Runtime
- Release_Year

When the Playback Operations team added the *Top Movies by Country* dataset to the business data catalog, they included the following metadata:

- **Description of the dataset**: Top 10 movies over last 8 hours grouped by country
- **Tags**: top10; ranking; countries; movies
- **Owning team**: Playback Operations
- **Data Owner**: Anchal Priya
- **Update Frequency**: 8 hours

As part of the process of creating the new dataset entry in the business data catalog, the dataset entry is linked with the technical catalog, meaning that the details about each of the fields in the dataset are also included in the business data catalog.

The data catalog also captures certain metadata automatically, such as the date/time when the dataset was last updated, a data quality score that can be automatically imported from the data quality tool, and the popularity of the dataset based on how many other users have requested access to the dataset. It is also possible to have the lineage and profile of the dataset stored in the data catalog, giving other users a better understanding of how this data was generated.

Recently, the marketing team started a new project that depends on having data about which genre of movies are most popular in each country (such as Action in Brazil, and Comedy in Argentina). They initially thought that they would need to create their own new ETL job to join many different datasets to get this information, but wisely decided that they should first search the corporate data catalog to see if there is an existing dataset that may have this information.

One of the data analysts on the marketing team logs in to the data catalog, and searches for *top movie genres by country*. In the search results, the *Top Movies by Country* dataset is listed, and the data analyst

is able to explore more information about the dataset, view the list of fields, and even see feedback that other users of the dataset have left. In this way, the data catalog is much like the catalog of products on eCommerce sites like Amazon.com. Users can search for products, explore more information about the products, and look at reviews left by others who provide feedback on the products.

The data analyst realizes that this dataset has the data they need in order to be able to easily identify the top movie genre by country. Within the data catalog, they are able to click a button to request access to the data, and provide a short note indicating why they need access. This automatically gets routed to the dataset owner (Anchal Priya, as recorded in the catalog) who can then approve or deny the request. If approved, an automatic process grants the data analyst on the marketing team access to the dataset.

Popular data catalog solutions outside of AWS include the **Collibra Data Catalog**, the **Informatica Enterprise Data Catalog**, **Atlan**, and **Amundsen** (an open-source data catalog). At the AWS re:Invent conference in December 2022, AWS announced **Amazon DataZone**, a new service that includes business data-catalog-type functionality.

Amazon DataZone is a service from AWS that provides built-in data governance features, helping to unlock the power of data across an organization. This includes functionality for a business data catalog, as we cover in more detail in *Chapter 15*, *Implementing a Data Mesh Strategy*.

As shown in this section, the purpose of the business data catalog is to enable users to discover and browse datasets that are available within the organization. Let's now take a look at technical catalogs, which are primarily used by analytic applications to work with data in a data lake.

Technical data catalogs

Technical catalogs are those that map data files in a data lake (such as files stored in Amazon S3 storage) to a logical representation of those files in the form of databases and tables. The Hive metastore is a well-known technical catalog that stores metadata for Hive tables (such as the table schema, location, and partition information). These are primarily technical attributes of the table, and the AWS Glue Data Catalog is an example of a Hive-compatible Metastore (meaning analytic services designed to work with a Hive metastore catalog can use the Glue catalog).

Services such as Amazon Athena, AWS Glue ETL, and Amazon EMR enable you to run queries on data in an Amazon S3 data lake, just by referencing a database and table name and without needing to know the actual location in S3. When you run a query on a specific database/table, the analytic service references the technical catalog to determine where the underlying files that make up this table are located in Amazon S3. In addition, the analytic service can also access information on the schema of the table (column names and column types), as well as the format of those files (such as CSV or Parquet format), so that it uses the correct reader to access the data.

A business data catalog (discussed in the previous section) can often integrate with the technical catalog. For example, with Amazon DataZone it is possible to import databases that have been registered in the Glue technical data catalog. Once imported, a user can add additional business metadata that users of the DataZone catalog can use to discover datasets. When users search the DataZone catalog, they can search and view the business metadata that was manually added, but can also access technical metadata imported from Glue, such as the schema (a list of columns and data types). In a similar way, third-party tools such as the Collibra data catalog can also integrate with the Glue catalog in order to import technical attributes.

In the next section, we do a deeper dive into AWS services that relate to data governance, and this includes a closer look at the technical

data catalog provided by AWS.

AWS services that help with data governance

As already discussed in this chapter, data governance is a wide-ranging subject covering many different aspects of working with data. And within AWS, there are a number of different tools and services that assist with data governance, which we will explore in this section.

The AWS Glue/Lake Formation technical data catalog

The AWS Glue Data Catalog is a technical catalog that provides a logical mapping (databases, tables, and columns) to data that is stored in files in Amazon S3. However, in addition to cataloging data files in Amazon S3, the AWS Glue Data Catalog can also store schema and metadata information about tables in other databases, such as Amazon Redshift, Amazon RDS, Amazon DynamoDB, and more.

Within AWS, there are actually two services for interacting with the data catalog. So far, we have only discussed the **AWS Glue Data Catalog**, but the **AWS Lake Formation** service also provides an interface for the same catalog.

It is important to understand that there is only a single data catalog, but both Glue and Lake Formation provide an interface to the catalog. For example, if we edit the schema of the `csvtoparquent` table that we created in the hands-on exercise in *Chapter 3*, and use the Glue console to change the data type of the `favorite_num` column from `bigint` to `int`, we will see that reflected in the Lake Formation console as well.

In the following screenshot, we see the `csvtoparquent` table, showing the schema, in the AWS Glue console.

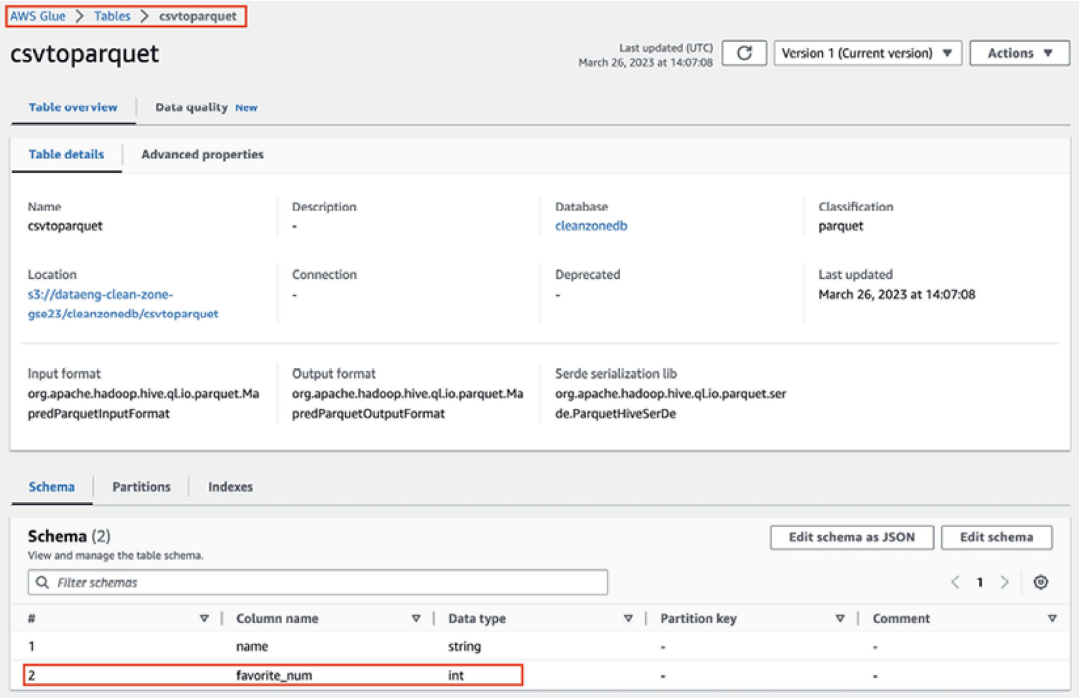

Figure 4.1: AWS Glue console showing table schema

If we look at the same table in the **Lake Formation** console, we can see similar information, including that the current version of this table is Version 1 (this was created when we updated Version 0 by changing the data type for the `favorite_num` column).

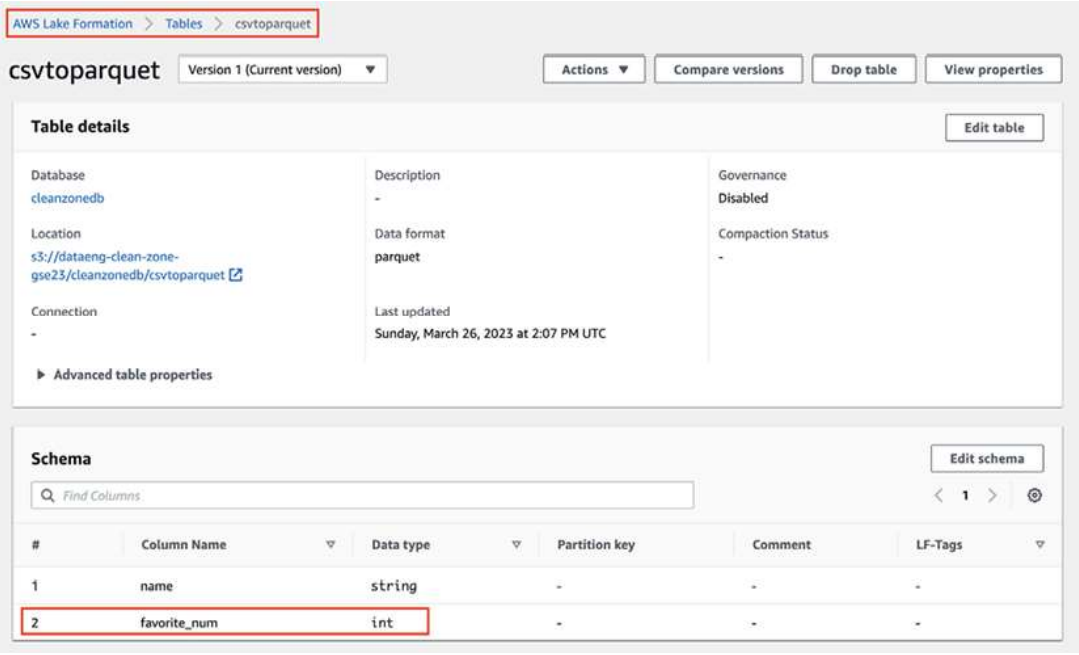

Figure 4.2: AWS Lake Formation console showing table schema

The Lake Formation and Glue consoles are similar in both design and functionality, and so whether you use the Glue or Lake Formation console comes down to personal preference.

The data catalog can be referenced by various analytical tools to work with data in the data lake. For example, Amazon Athena can reference the data catalog to enable users to run queries against databases and tables in the catalog. Athena uses the catalog to get the following information, which is required to query data in the data lake:

- The Amazon S3 location where the underlying data files are stored
- Metadata that indicates the file format type for the underlying files (such as CSV or Parquet)
- Details of the serialization library, which should be used to serialize the underlying data
- Metadata that provides information about the data type for each column in the dataset
- Information about any partitions that are used for the dataset

A data engineer must help put automation in place to ensure that all the datasets that are added to a data lake are cataloged and that the appropriate metadata is added.

In *Chapter 3*, *The AWS Data Engineers Toolkit*, we discussed AWS Glue Crawlers, a process that can be run to examine a data source, infer the schema of the data source, and then automatically populate the technical data catalog with information on the dataset.

A data engineer should consider building workflows that make use of Glue Crawlers to run after new data is ingested, to have the new data automatically added to the data catalog. Or, when a new data engineering job is being bought into production, a check can be put in

place to make sure that the Glue API is used to update the data catalog with details of the new data.

Note that adding data to the technical catalog is a separate process from adding data to a business data catalog. While having data in the Glue/Lake Formation catalog enables many different AWS services to work with the data (such as Amazon Athena, Amazon EMR, etc.), the Glue/Lake Formation interfaces were not designed as business catalogs. As such, they do not provide the best interface for searching for different datasets, or for capturing business-related metadata (such as data owner, confidentiality level, data quality score, etc.). Therefore, to make data discoverable, you need a separate process to ensure that datasets are added to a business catalog.

AWS Glue DataBrew for profiling datasets

As discussed earlier in this chapter, data profiling examines a dataset in order to report on the profile, or shape, of the data. This includes information on each of the columns, such as the number of rows, distinct and unique values, min and max values for number data types, and min and max string length for string data types. It can also report on null or missing values within a column.

In order to generate profile information on a dataset, you can configure and run a **Glue DataBrew** profile job. When the job finishes examining a dataset, you can view information about the data profile within the Glue DataBrew console, and you can generate a JSON file that contains profile information for the dataset.

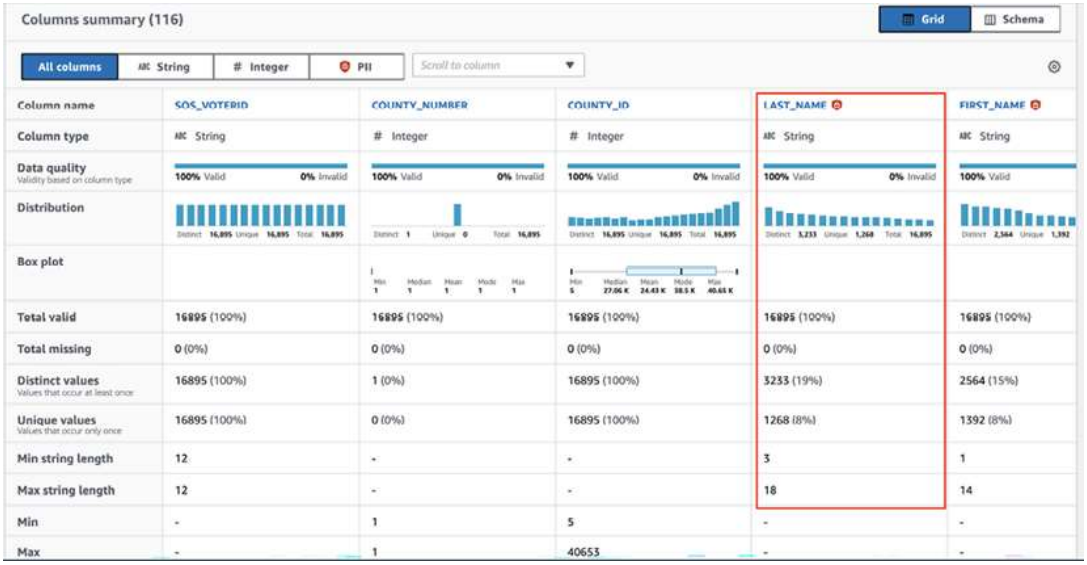

Figure 4.3: AWS Glue DataBrew profiling job results

In the above screenshot, we can see a sample data profile as shown in the AWS Glue DataBrew console. Note that this is just a subset of the data profile information that DataBrew generated, but in this screenshot, we can see, for example, that there is a `LAST_NAME` column with 16,895 rows. We can further see that there are 3,233 distinct values (last names that occur at least once), as well as 1,268 unique values (last names that occur only once in this dataset). We also see that the shortest last name is 3 characters long, and the longest last name has 18 characters.

AWS Glue Data Quality

The AWS Glue service includes functionality for monitoring and measuring the quality of your datasets. With **AWS Glue Data Quality**, you can either evaluate data that is in S3 and has been cataloged in the Glue Data Catalog, or you can evaluate data as part of a data processing job using **AWS Glue Studio**.

Glue Data Quality is based on the open-source **Deequ data quality framework** and uses the **Data Quality Definition Language** (**DQDL**) in order to define data quality rules. The following are some of the ex-

pressions that you can use in DQDL rules to test the quality of a dataset:

- **ColumnExists** – checks whether a column exists. For example, `ColumnExists "date_of_birth"`.
- **ColumnLength** – checks the length of each row in the column. For example, if your dataset has a postal code/ZIP code column and you know that the ZIP code should always be 5 characters in length, you can write a rule such as `ColumnLength "Postal_Code" = 5`.
- **Completeness** – checks the percentage of non-null rows for the specific column. For example, if you wanted to make sure that at least 95% of the rows in the dataset had a value for email, you could write a rule such as `Completeness "email" > 0.95`.
- **RowCount** – checks that the dataset contains a specified number of rows. This can be within a range, for example, we can check if the dataset has between 10,000 and 15,000 rows with the following rule: `RowCount between 10000 and 15000`.

There are many other rule types that can be used to evaluate data quality, either for data already cataloged in the data catalog or for data being processed with a Glue Studio job. Both Glue Studio and the Glue Data Catalog provide a visual tool for building a set of data quality rules.

In addition, the **AWS Glue DataBrew** service also includes a visual editor for implementing data quality rules to evaluate the quality of your data.

AWS Key Management Service (KMS) for data encryption

AWS KMS simplifies the process of creating and managing security keys for encrypting and decrypting data in AWS. The AWS KMS service is a core service in the AWS ecosystem, enabling users to easily manage data encryption across several AWS services.

There are a large number of AWS services that can work with AWS KMS to enable data encryption, including the following AWS analytical services:

- **Amazon AppFlow**
- **Amazon Athena**
- **Amazon EMR**
- **Amazon Kinesis Data Streams/Kinesis Firehose/Kinesis Video Streams**
- **Amazon Managed Streaming for Kafka (MSK)**
- **Amazon Managed Workflows for Apache Airflow (MWAA)**
- **Amazon Redshift**
- **Amazon S3**
- **AWS Data Migration Service (DMS)**
- **AWS Glue/Glue DataBrew**
- **AWS Lambda**

The full list of compatible services can be found at https://aws.amazon.com/kms/features/#AWS_Service_Integration.

Permissions can be granted to users to make use of the keys for encrypting and decrypting data, and all use of AWS KMS keys is logged in the AWS CloudTrail service. This enables an organization to easily audit the use of keys to encrypt and decrypt data.

For example, with Amazon S3, you can enable Amazon S3 Bucket Keys, which configures an S3 bucket key to encrypt all new objects in the bucket with an AWS KMS key. This is significantly less expensive than using **Server Side Encryption – KMS (SSE-KMS)** to encrypt each object in a bucket with a unique key.

To learn more about configuring Amazon S3 Bucket Keys, see https://docs.aws.amazon.com/AmazonS3/latest/userguide/bucket-key.html.

It is important that you carefully protect your KMS keys and that you put safeguards in place to prevent a KMS key from being accidentally (or maliciously) deleted. If a KMS key is deleted, any data that has been encrypted with that key is effectively lost and cannot be decrypted.

Because of this, you must schedule the deletion of your KMS keys and specify a waiting period of between 7 and 30 days before the key is deleted. During this waiting period, the key cannot be used, and you can configure a CloudWatch alarm to notify you if anyone attempts to use the key.

If you use AWS Organizations to manage multiple AWS accounts as part of an organization, you can create a **Service Control Policy** (**SCP**) to prevent any user (even an administrative user) from deleting KMS keys in child accounts.

Amazon Macie for detecting PII data in Amazon S3 objects

Amazon Macie is a managed service that uses machine learning, along with pattern matching, to discover and protect sensitive data. Amazon Macie identifies sensitive data, such as PII data, in an Amazon S3 bucket and provides alerts to warn administrators about the presence of such sensitive data. Macie can also be configured to launch an automated response to the discovery of sensitive data, such as a step function that runs to automatically remediate the potential security risk.

Macie can identify items such as names, addresses, and credit card numbers that exist in files on S3. These items are generally considered to be PII data, and as discussed previously, these should ideally be tokenized before data processing. Macie can also be configured to recognize custom sensitive data types to alert the user to sensitive data that may be unique to a specific use case.

The AWS Glue Studio Detect PII transform for detecting PII data in datasets

While Amazon Macie detects PII data directly in S3 files, an alternate option is to use the **AWS Glue Studio Detect PII transform** to detect PII data during a data processing job. When creating a job in AWS Glue Studio, you can add the **Detect Sensitive Data** transform and configure it based on your requirements. This includes selecting whether to examine every cell in the dataset for PII data or whether to just sample a limited number of rows for each column to detect PII data. You can also select what to do when PII data is detected, including options for redacting the text (replacing it with a preset string), applying a SHA-256 hash to the value, or just reporting on the PII data that is detected.

Amazon GuardDuty for detecting threats in an AWS account

While **Amazon GuardDuty** is not directly related to analytics on AWS, it is a powerful service that helps protect an AWS account. GuardDuty is an intelligent threat detection service that uses machine learning to monitor your AWS account and provide proactive alerts about malicious activity and unauthorized behavior.

GuardDuty analyzes several AWS-generated logs, including the following:

- CloudTrail S3 data events (a record of all actions taken on S3 objects)
- CloudTrail management events (a record of all usage of AWS APIs within an account)
- VPC flow logs (a record of all network traffic within an AWS VPC)
- DNS logs (a record of all DNS requests within your account)

By continually analyzing these logs to identify unusual access patterns or data access, Amazon GuardDuty can proactively alert you to potential issues, and also helps you automate your response to threats.

AWS Identity and Access Management (IAM) service

AWS IAM is a service that provides both authentication and authorization for the AWS Console, **command-line interface** (**CLI**), and **application programming interface** (**API**) calls.

AWS IAM also supports the federation of identities, meaning that you can configure IAM to use another identity provider for authentication, such as Active Directory or Okta.

Note that this section is not intended as a comprehensive guide to Identity and Access Management on AWS, but it does provide information on foundational concepts that are important for anyone working within the AWS cloud to understand. For a deeper understanding of the AWS IAM service, refer to the *AWS Identity and Access Management user guide* (`https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html`).

Several IAM identities are important to understand:

- **AWS account root user**: When you create an AWS account, you provide an email address to be associated with that account, and that email address becomes the root user of the account. You can log in to the AWS Management Console using the root user, and this user has full access to all the resources in the account. However, it is strongly recommended that you do not use this identity to log in and perform everyday tasks, but rather create an IAM user for everyday use.

- **IAM user**: This is an identity that you create and can be used to log in to the AWS Console, run CLI commands, or make API calls. An IAM user can have a login name and password that's used for Console access, and can have up to two associated access keys that can be used to authenticate this identity when using the AWS CLI or API. While you can associate IAM policies directly with an IAM user, the recommended method to provide access to AWS resources is to make the user part of a user group that has relevant IAM policies attached.
- **IAM user groups**: An IAM user group is used to provide permissions that can be associated with multiple IAM users. You provide permissions (via IAM policies) to an IAM user group, and all the members of that group then inherit those permissions.
- **IAM roles**: An IAM role can be confusing at first as it is similar to an IAM user. However, an IAM role does not have a username or password and you cannot directly log in or identify as an IAM role. However, an IAM user can assume the identity of an IAM role, taking on the permissions assigned to that role. An IAM role is also used in identity federation, where a user is authenticated by an external system, and that user identity is then associated with an IAM role. Finally, an IAM role can also be used to provide permissions to AWS resources (for example, to provide permissions to an AWS Lambda function so that the Lambda function can access specific AWS resources).

To grant authorization to access AWS resources, you can attach an **IAM policy** to an IAM user, IAM user group, or IAM role. These policies grant, or deny, access to specific AWS resources, and can also make use of conditional statements to further control access.

These identity-based policies are JSON documents that specify the details of access to an AWS resource. These policies can either be configured within the AWS Management Console, or the JSON documents can be created by hand.

There are three types of identity-based policies that can be utilized:

1. **AWS-managed policies**: These are policies that are created and managed by AWS and provide permissions for common use cases. For example, the `AdministratorAccess` managed policy provides full access to every service and resource in AWS, while the `DatabaseAdministrator` policy provides permissions for setting up, configuring, and maintaining databases in AWS.
2. **Customer-managed policies**: These are policies that you create and manage to provide more precise control over your AWS resources. For example, you can create a policy and attach it to specific IAM users/groups/roles that provide access to a list of specific S3 buckets and limit that access to only be valid during specific hours of the day or for specific IP addresses.
3. **Inline policies**: These are policies that are written directly for a specific user, group, or role. These policies are tied directly to the user, group, or role, and therefore apply to one specific entity only.

The following policy is an example of a customer-managed policy that grants read access to a specific S3 bucket:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3::: de-landing-zone"
        },
  {
           "Effect": "Allow",
           "Action": [
               "s3:GetObject"
],
           "Resource": ["arn:aws:s3::: de-landing-zone/*"]
```

```
        }
        ]
    }
```

The policy takes the form of a JSON document. In this instance, the policy does the following:

1. `Allow` access (you can also create policies that `Deny` access).
2. Allows access for `Action` of `s3:GetObject` and `s3:ListBucket`, meaning authorization is given to run the Amazon S3 `GetBucket` and `ListBucket` actions (via the Console, CLI, or API).
3. For `ListBucket`, the resource is set as the `de-landing-zone` bucket. For `GetObject`, the resource is set as `de-landing-zone/*`. This results in the principal being granted access to list the `de-landing-zone` bucket, and read access to all the objects inside the `de-landing-zone` bucket.

You could further limit this policy to only be allowed if the user was connecting from a specific IP address, at a certain time of day, or various other limitations. For example, to limit this permission to users from a specific IP address, you could add the following to the policy:

```
"Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "12.13.15.16/32",
                        "45.44.43.42/32"
                    ]
                }
            }
```

Once you have created a customer-managed policy, you can attach the policy to specific IAM user groups, IAM roles, or IAM users.

Traditional data lakes on AWS used IAM policies to control access to data in an Amazon S3-based data lake. For example, a policy would be created to grant access to different zones of the data lake storage in S3, and then that policy would be attached to different IAM users, user groups, or roles.

However, when creating a large data lake that may contain multiple buckets or S3 prefixes that relate to specific business units, it can be challenging to manage S3 permissions through these JSON policies. Each time a new data lake location is created, the data engineer would need to make sure that the JSON policy document were updated to configure permissions for the new location.

To make managing large S3-based data lakes easier, AWS introduced a service called **AWS Lake Formation**, which enables permissions for the data lake to be controlled by the data lake administrator from within the AWS Management Console (or via the AWS CLI or AWS API).

Using AWS Lake Formation to manage data lake access

AWS Lake Formation is a service that simplifies setting up and managing a data lake. A big part of the Lake Formation service is the ability to manage access (authorization) to data lake databases and tables without having to manage fine-grained access through JSON-based policy documents in the IAM service.

Lake Formation enables a data lake administrator to grant fine-grained permissions on data lake databases, tables, and columns using the familiar database concepts of grant and revoke for permissions management. A data lake administrator, for example, can grant SELECT permissions (effectively READ permission) for a specific data lake table to a specific IAM user or role.

Lake Formation permissions management is another layer of permissions that is useful for managing fine-grained access to data lake resources, but it works with IAM permissions and does not replace IAM permissions. A recommended way to do this is to apply broad permissions to a user or user group in an IAM policy, but then apply fine-grained permissions with Lake Formation.

Permissions management before Lake Formation

Before the release of the Lake Formation service, all data lake permissions were managed at the Amazon S3 level using IAM policy documents written in JSON. These policies would control access to resources such as the following:

- The data catalog objects in the Glue Data Catalog (such as permissions to access Glue databases and tables)
- The underlying physical storage in Amazon S3 (such as the Parquet or CSV files in an Amazon S3 bucket)
- Access to analytical services (such as Amazon Athena or AWS Glue)

For example, the IAM policy would provide several Glue permissions, including the ability to read catalog objects (such as Glue tables and table partitions) and the ability to search tables. However, the resources section of the policy would restrict these permissions to the specific databases and tables that the user should have access to.

The policy would also have a section that provided permissions to the underlying S3 data. For each table that a user needed to access in the Glue Data Catalog, they would need both Glue Data Catalog permissions for the catalog objects, as well as Amazon S3 permissions for the underlying files.

The last part of the IAM policy would also require the user to have access to relevant analytical tools, such as permissions to access the

Amazon Athena service.

Permissions management using AWS Lake Formation

With **AWS Lake Formation**, permissions management is changed so that broad access can be provided to Glue catalog objects in the IAM policy, and fine-grained access is controlled via AWS Lake Formation permissions.

With Lake Formation, data lake users do not need to be granted direct permissions on underlying S3 objects as the Lake Formation service can provide temporary credentials to compatible analytic services to access the S3 data.

It is important to note that Lake Formation permissions access only works with compatible analytic services, which, at the time of writing, include the following AWS services:

- Amazon Athena
- Amazon QuickSight
- Amazon EMR
- Amazon Redshift Spectrum
- AWS Glue
- Amazon SageMaker Studio

If using these compatible services, AWS Lake Formation is a simpler way to manage permissions for your data lake. The data lake user still needs an associated IAM policy that grants them access to the AWS Glue service, the Lake Formation service, and any required analytic engines (such as Amazon Athena). However, at the IAM level, the user can be granted access to all AWS Glue objects. Then, the Lake Formation permissions layer can be used to control which specific Glue catalog objects (databases and tables) can be accessed by the user.

As the Lake Formation service passes temporary credentials to compatible analytic services to read data from Amazon S3, data lake users no longer need any direct Amazon S3 permissions to be provided in their IAM policies.

Hands-on – configuring Lake Formation permissions

In this hands-on section, we will use the AWS Management Console to configure **Lake Formation permissions**.

However, before we implement Lake Formation permissions, we're going to create a new data lake user and configure their permissions using just IAM permissions. We'll then go through the process of updating a Glue database and table to use Lake Formation permissions, and then grant Lake Formation permissions to our data lake user.

Configuring the Glue Crawler

While not covered in this chapter, we will provide a hands-on section with details on how to configure the Glue Crawler in *Chapter 6, Ingesting Batch and Streaming Data*.

Creating a new user with IAM permissions

To start, let's create a new IAM user that will become our data lake user. We will initially use IAM to grant our data lake user the following permissions:

- Permission to access a specific database and table in the Glue Data Catalog
- Permission to access specific S3 locations that contain the underlying files associated with our Glue table objects

- Permission to use the Amazon Athena service to run SQL queries against the data lake

First, let's create a new IAM policy that grants the required permissions for using Athena and Glue, but limits those permissions to only the `CleanZoneDB` in the Glue catalog. To do this, we're going to copy the Amazon-managed policy for Athena Full Access, but we will modify the policy to limit access to just a specific Glue database, and we will add S3 permissions to the policy. Let's get started:

1. Log in to the AWS Management Console and access the IAM service using this link: https://console.aws.amazon.com/iam/home.
2. On the left-hand side, click on **Policies**, and then for **Filter Policies**, type in `Athena`.
3. From the filtered list of policies, expand the `AmazonAthenaFullAccess` policy.
4. Click on the **Copy** button to copy the policy to your computer clipboard.

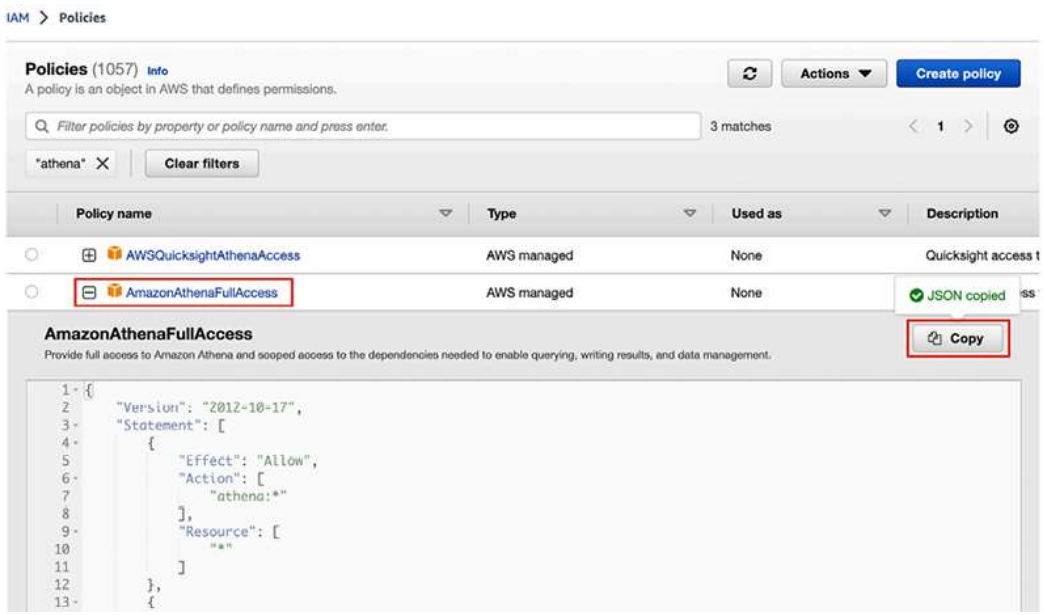

Figure 4.4: Copying the text of the AmazonAthenaFullAccess policy

5. At the top of the page, click on **Create policy**.

6. The visual editor is selected by default, but since we want to create a JSON policy directly, click on the **JSON** tab.
7. Paste the Athena Full Access policy that you copied to the clipboard in *step 4* into the policy, overwriting and replacing any text currently in the policy.
8. Look through the policy to identify the section that grants permissions for several Glue actions (`glue:CreateDatabase` , `glue:DeleteDatabase` , `glue:getDatabase` , and so on). This section currently lists the resource that it applies to as `*` , meaning that the user would have access to all databases and tables in the Glue catalog. In our use case, we want to limit permissions to just the `Glue CleanZoneDB` database (which was created in the hands-on section of *Chapter 3*, *The AWS Data Engineers Toolkit*). Replace the resource section of the section that provides Glue access with the following, which will limit access to the required DB only, although it also includes all tables in that database:

```
            "Resource": [
                "arn:aws:glue:*:*:catalog",
                "arn:aws:glue:*:*:database/cleanzonedb",
                "arn:aws:glue:*:*:database/cleanzonedb*",
                "arn:aws:glue:*:*:table/cleanzonedb/*"
            ]
```

   The following screenshot shows how this looks when applied to the policy:

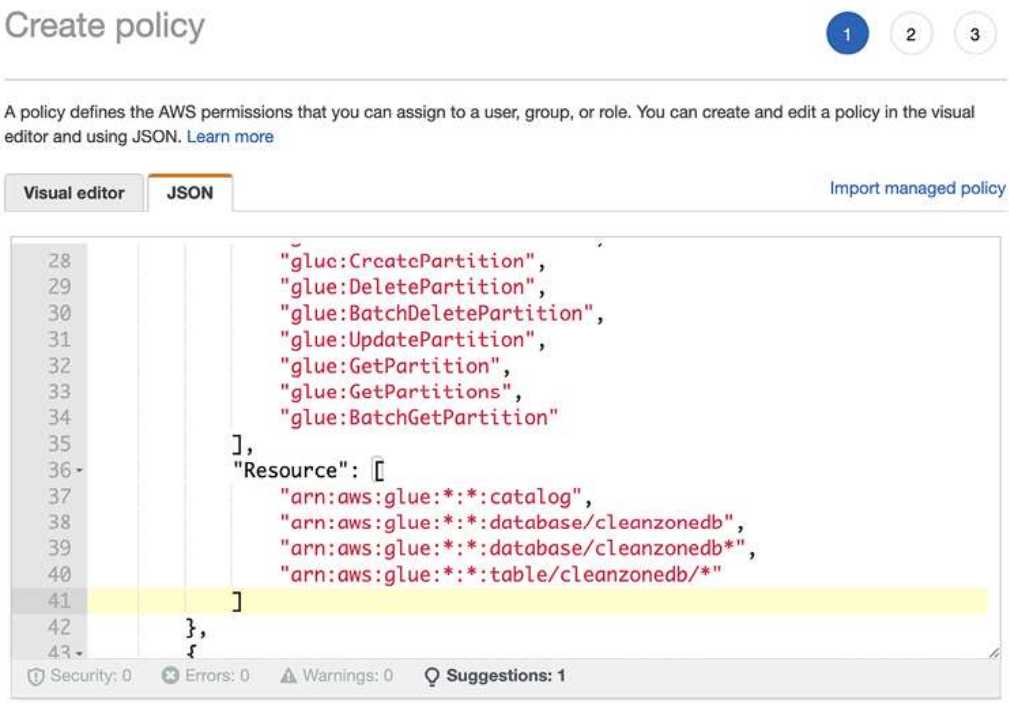

Figure 4.5: Updated policy with limited permissions for Glue resources

9. Immediately after the section that provides Glue permissions, we can add new permissions for accessing the S3 location where our `CleanZoneDB` data resides. Add the following section to provide these permissions, **making sure to replace `<initials>` with the unique identifier you used when creating the bucket** in *Chapter 2, Data Management Architectures for Analytics*:

```
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:PutObject"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::dataeng-clean-zone-<initials>/*"
            ]
        },
```

The following screenshot shows the S3 permissions added to the policy:

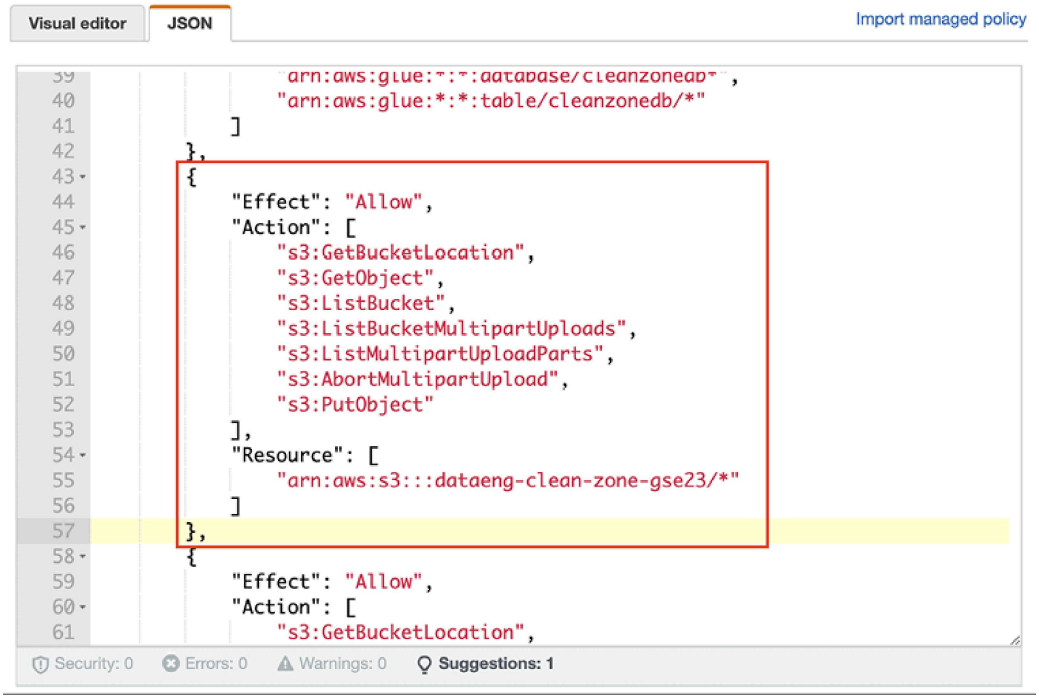

Figure 4.6: S3 permissions added to the policy

10. Once you have pasted in the new S3 permissions, click on **Next:Tags** at the bottom right of the screen.
11. Optionally, add any tags for this policy, and then click on **Next: Review**.
12. For `Name`, provide a policy name of `AthenaAccessCleanZoneDB` and click **Create policy**.

Now that we have created an IAM policy for providing the required permissions to the Glue catalog and S3 buckets, we can create a new data lake user and attach our new policy to the user.

> Note that AWS recommends the use of **IAM Identity Center** to manage users in your AWS account, and this should be strongly considered when creating users in production, and even development, accounts. However, for our purposes, where we are using a single-purpose sandbox-type account for performing the hands-on exercises in this book, we will create an IAM user as this is a simpler setup.

Follow these steps to create the new IAM user:

1. In the IAM console, on the left-hand side, click on **Users**, and then click **Add users**.
2. For **User name**, enter `datalake-user` .
3. Click the checkbox for **Provide user access to the AWS Management Console - optional**.
4. Select the **I want to create an IAM user** option.
5. Select your preferred options for creating a password, and then click **Next**.
6. On the **Set permissions** page, select **Attach policies directly**.
7. In the **Permission policies** search bar, search for **Athena** and select the **AthenaAccessCleanZoneDB** policy, which we created in the previous set of steps.
8. Click **Next**.
9. Review the new user details, and then click **Create user**.
10. Take note of the **Console sign-in URL**, which can be used to access the login screen for your account.

Now, let's create a new Amazon S3 bucket that we can use to capture the results of any Amazon Athena queries that we run:

1. In the AWS Management Console, use the top search bar to search for and select the **S3** service.
2. Click on **Create bucket**.

3. For **Bucket name**, enter `aws-athena-query-results-dataeng-book-<initials>`. Replace `<initials>` with your initials or some other unique identifier.
4. Ensure **AWS Region** is set to the region you have been using for the other exercises in this book.
5. Leave all other options with their defaults, and click on **Create bucket**.

We can now verify that our new `datalake-user` has access to `CleanZoneDB` and that the user can run Athena queries on the table in this database:

1. Sign out of the AWS Management Console, and then sign in again using the new user you just created, `datalake-user`. Use the **Console sign-in URL** you previously noted down to access the login page.
2. From the top search bar, search for and select the **Athena** service.
3. Before you can run an Athena query, you need to set up a query result location in Amazon S3. This is the S3 bucket and prefix where all the query results will be written to. From the top right of the Athena console, click on **Edit Settings**.
4. For **Query result location**, enter the S3 path you created in *the previous Step 3* (for example, `s3://aws-athena-query-results-dataengbook-<initials>/`).
5. Click on **Save**.
6. Change to the **Editor** tab, and in the query window, run the following SQL query: `select * from cleanzonedb.csvtoparquet`.
7. If all permissions have been configured correctly, the results of the query should be displayed in the lower window. The file we created shows names and ages.
8. Log out of the AWS Management Console since we need to be logged in as our regular user, not `datalake-user`, for the next steps.

We have now set up permissions for our data lake using IAM policies to manage fine-grained access control, as was always done before the launch of the AWS Lake Formation service. In the next section, we will transition to using Lake Formation to manage fine-grained permissions on data lake objects.

Transitioning to managing fine-grained permissions with AWS Lake Formation

In the initial setup, we configured permissions for our data lake user to be able to run SQL queries using Amazon Athena, and we restricted their access to just `cleanzonedb` using an IAM permissions policy.

In this section, we are going to modify `cleanzonedb` and the tables in that database to make use of the Lake Formation permissions model.

Activating Lake Formation permissions for a database and table

As a reminder, **Lake Formation** adds a layer of permissions that work in addition to the IAM policy permissions. By default, every database and table in the catalog has a special permission enabled that effectively tells Lake Formation to just use IAM permissions and to ignore any permissions that may have been granted in Lake Formation. This is sometimes called the **Pass-Through** permission, as it allows security checks to be validated at the IAM level, but then passes through Lake Formation without doing any additional permission checks.

With our initial setup, we granted Glue Data Catalog permissions to `datalake-user` in an IAM policy. This policy allowed the user to access the `cleanzonedb` database, as well as all the tables in that database. Let's have a look at how permissions are set up on the `cleanzonedb` database and tables in Lake Formation:

1. Log in to the **AWS Management Console** and search for the `Lake Formation` service in the top search bar. Make sure you are logged in as your regular user, and not as `datalake-user`, which you created earlier in this chapter.
2. The first time you access the Lake Formation service, a **pop-up box** will prompt you to choose initial users and roles to be Lake Formation data lake administrators. By default, **Add myself** should be selected. Click **Get started** to add your current user as a data lake admin.

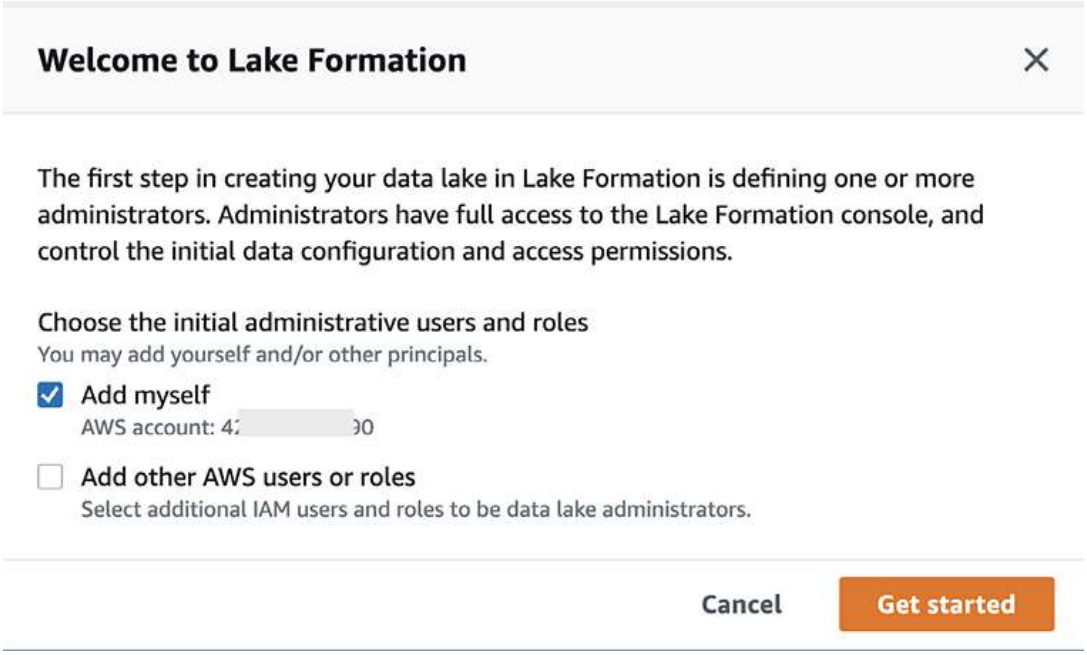

Figure 4.7: Adding your user as a Lake Formation administrator

3. Once selected, you should be taken to the Lake Formation **Data lake administrators** screen, where you can confirm that your user has been added as a data lake administrator.
4. On the left-hand side of the Lake Formation console, click on **Databases**. In the list of databases, click on the `cleanzonedb` database.
5. This screen displays details of `cleanzonedb`. Click on **Actions**, and then **View permissions**.

6. On the **View permissions** screen, we can see that two permissions have been assigned for this database. The first one is `DataEngLambdaS3CWGlueRole`, and this IAM role has been granted full permissions on the database. The reason for this is that `DataEngLambdaS3CWGlueRole` was the role that was assigned to the Lambda function that we used to create the database back in *Chapter 3*, *The AWS Data Engineers Toolkit*, so it is automatically granted these permissions.

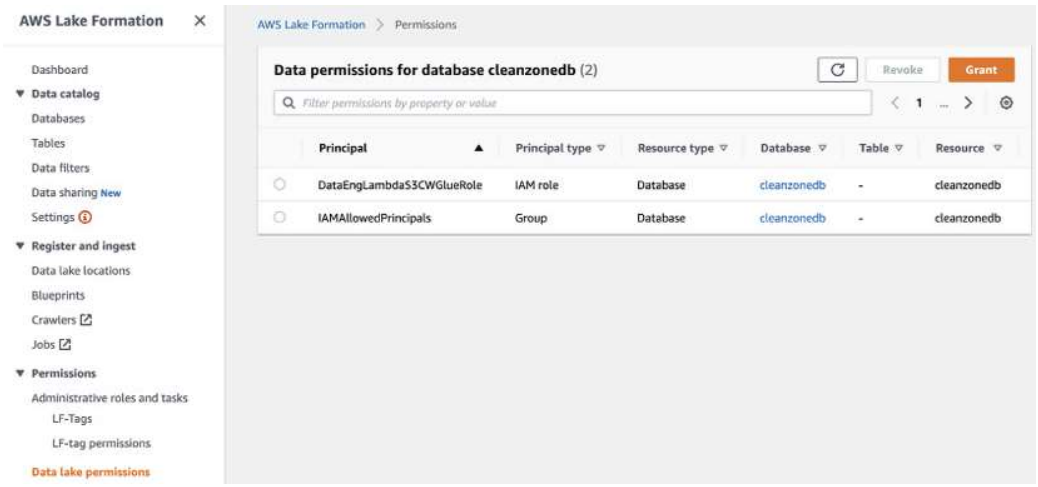

Figure 4.8: Lake Formation permissions for the cleanzonedb database

The other permission that we can see is for the `IAMAllowedPrincipals` group. This is the pass-through permission we mentioned previously, which effectively means that permissions at the Lake Formation layer are ignored. If this special permission was not assigned, only `DataEngLambdaS3CWGlueRole` would be able to access the database. However, because the permission has been assigned, any user who has been granted permissions to this database through an IAM policy, such as `datalake-user`, will be able to successfully access the database.

7. To enable Lake Formation permissions on this database, we can remove the `IAMAllowedPrincipals` permission from the database.

To do this, click the selector box for the `IAMAllowedPrincipals` permission and click **Revoke**. On the pop-up box, click on **Revoke**.

8. We now want to do the same thing for our `csvtoparquet` table in the database. To do this, click on **Databases** in the left-hand menu, then click on `cleanzonedb` . From the top right, click on **View tables**. Click the selector for the `csvtoparquet` table and click on **Actions/View Permissions**. Click the selector for `IAMAllowedPrincipals` and click on **Revoke**. On the pop-up window, click on **Revoke**. This removes the special **Pass-Through** permission from the table.

Optional - checking permissions

If you want to see what effect this has, you can log out of the AWS Console and log in again as `datalake-user` . Now, when you try to run a query on the `csvtoparquet` table using Athena, you will receive an error message as Lake Formation permissions are in effect, and your `datalake-user` has not been granted permissions to access the table yet via Lake Formation.

Granting Lake Formation permissions

By removing the `IAMAllowedPrincipals` permission from the `cleanzonedb` database and the `csvtoparquet` table, we have effectively enabled Lake Formation permissions on those resources. Now, if any principal needs to access that database or table, they need both IAM permissions, as well as Lake Formation permissions.

If we had enabled Lake Formation permissions on all databases and tables, then we could modify our user's IAM policy permissions to give them access to all data catalog objects. We can do this because we would know that they would only be able to access those databases

and tables where they had been granted specific Lake Formation permissions.

We previously created an edited copy of the `AmazonAthenaFullAccess` managed IAM policy to limit user access to specific data catalog databases and tables in the IAM policy. However, if all databases and tables had the `IAMAllowedPrincipals` permission removed and specific permissions granted to users instead, then we could apply the generic `AmazonAthenaFullAccess` policy.

We also previously provided access to the underlying S3 files using an IAM policy. However, when using Lake Formation permissions, compatible analytic tools are granted access to the underlying S3 data using temporary credentials provided by Lake Formation. Therefore, once Lake Formation permissions have been activated, we can remove permissions to the underlying S3 data from our user's IAM policy. Then, when using a compatible tool such as Amazon Athena, we know that Lake Formation will grant Athena temporary credentials to access the underlying S3 data.

Here, we will add specific Lake Formation permissions for our `datalake-user` to access the `CleanZoneDB` database and the `csvtoparquet` table:

1. Ensure you are logged in as your regular user (the one you made a data lake admin earlier) and access the Lake Formation console.
2. Click on **CleanZoneDB**, and then click **View tables**.
3. Click on the **csvtoparquet** table, and then click **Actions/Grant**.
4. From the **IAM users and roles** dropdown, click on the **datalake-user** principal.
5. For **Table permissions**, mark the permission for **Select**.
6. Under **Data permissions**, select **Column-based access**.
7. Select **Exclude columns**, and then under **Select columns**, select the `favorite_num` column.
8. Click on **Grant** at the bottom of the screen.

In the preceding steps, we granted our `datalake-user` **Select** permissions on the `csvtoparquet` table. However, we put in a column limitation, which means that `datalake-user` will not be able to access the `favorite_num` column. Enabling column-level permissions is not something that would be possible if we were just using IAM-level permissions, as column-level permissions is a Lake Formation-specific feature.

Now, if you log in to the AWS Management Console as `datalake-user` and run the same Athena query we ran previously (`select * from cleanzonedb.csvtoparquet`), your permissions will enable the required access.

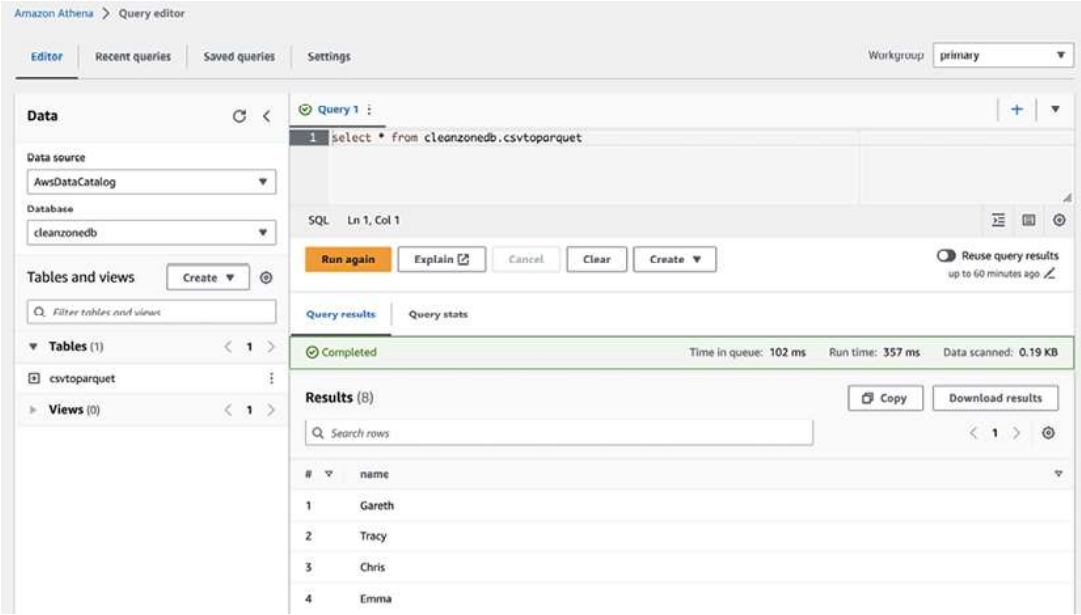

Figure 4.9: Running an Athena query with Lake Formation permissions

Note that in the results of the query, the `favorite_num` column is not included as we specifically excluded this column when granting permissions on this table to our `datalake-user` .

In this section, we transitioned to using Lake Formation for managing data lake permissions for the `cleanzonedb` database. We added fine-grained permissions in Lake Formation to limit `cleanzonedb` access to

just our `datalake-user`, and excluded the `favorite_num` column from the list of columns that our user could query.

Summary

In this chapter, we reviewed important concepts around the many different aspects of data governance, including how a data catalog can be used to help prevent your data lake from becoming a data swamp.

We reviewed data profiling, data quality, and data lineage as important aspects of data governance, as well as an introduction to how to ensure that your data is secured correctly and used in accordance with relevant regulations (such as GDPR). Ensuring that your data is used correctly and that it offers the most value to your organization does not just happen, but rather requires a strong data governance program and focus.

In the next chapter, we will take a step back and look at the bigger picture of how a data engineer can architect a data pipeline. We will begin exploring how to understand the needs of our data consumers, learn more about our data sources, and review the transformations that are required to transform raw data into useful data for analytics.

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