

Chapter 8. Implementing Data Governance

Databricks offers a robust data governance model designed to ensure the security, quality, and compliance of data throughout its lifecycle. This chapter delves into the key components of the Databricks data governance model, with a focus on data security. We will specifically examine data access management within the traditional Hive metastore and compare it with Databricks' governance solution, Unity Catalog.

What Is Data Governance?

Data governance is a strategic approach to managing data within an organization, ensuring that data is accurate, secure, and used responsibly. It involves the development and enforcement of policies and procedures to control data across various stages of its lifecycle—from ingestion and storage to processing and sharing. Data governance incorporates several key components, which are illustrated in [Figure 8-1](#).

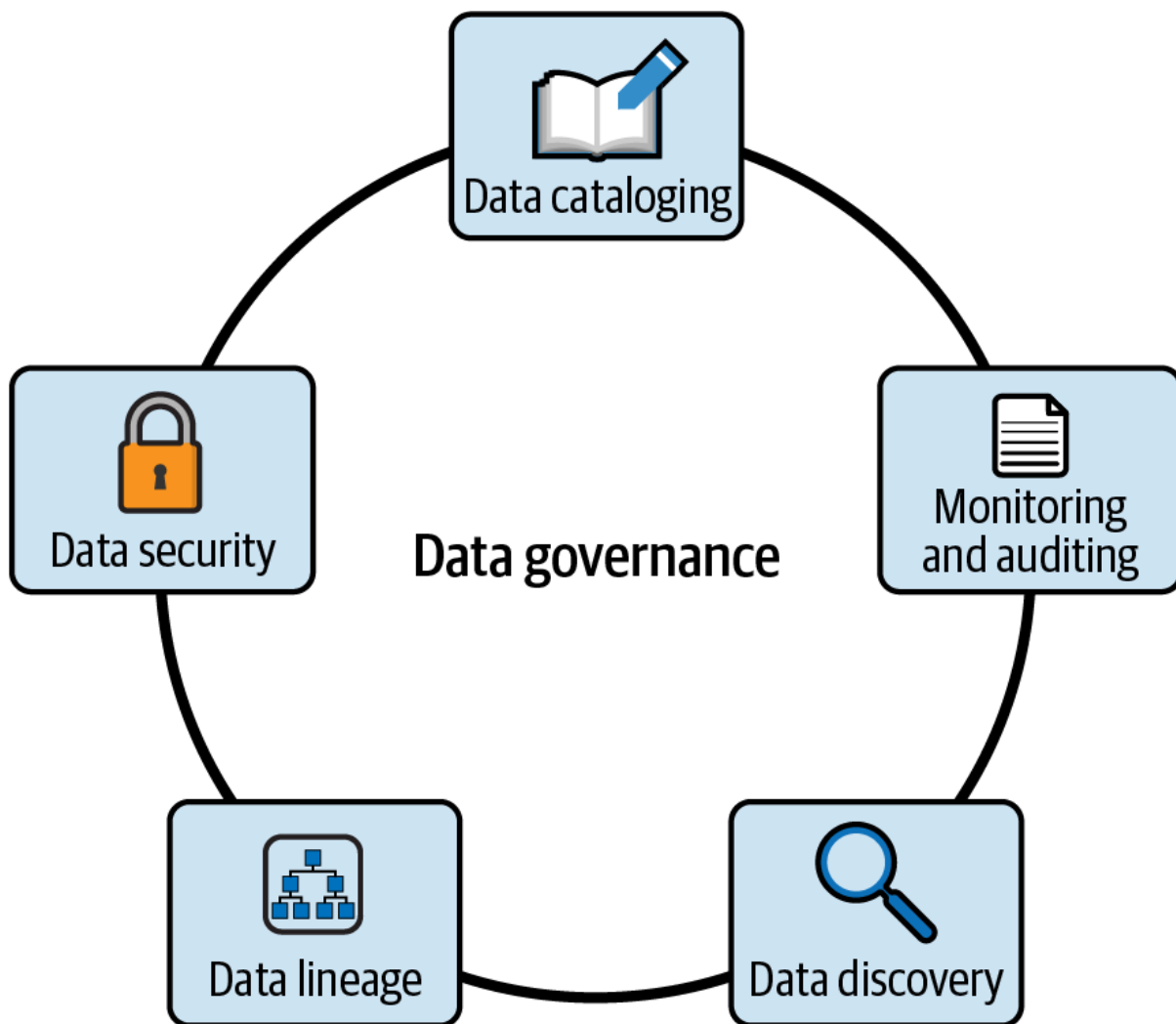


Figure 8-1. Components of data governance

Data cataloging

Effective data governance requires a comprehensive understanding of an organization's data assets. A data catalog plays a crucial role in this process by serving as a centralized repository for metadata, which facilitates efficient data discovery and access.

Data security

Robust data governance involves defining data access permissions to ensure that only authorized individuals or groups can access specific data. This practice is essential for maintaining data confidentiality and ensuring compliance with regulations such as [the European General Data Protection Regulation \(GDPR\)](#).

Monitoring and auditing

Comprehensive auditing of data access and usage is a fundamental aspect of strong data governance. It involves monitoring who is accessing the data, as well as when and how it is being used. This practice helps organizations ensure compliance with data protection regulations and reduce the risk of unauthorized access.

Data lineage

Data lineage refers to the tracking and visualization of the flow of data assets from their origin to their final destination. This is vital for data governance as it ensures transparency, data reliability, and compliance. By mapping dependencies between different resources, data lineage facilitates troubleshooting and identifying potential impacts of changes.

Data discovery

A key element of data governance is ensuring that data is easily discoverable, allowing data teams to efficiently locate data assets across the organization. This helps prevent data duplication and promotes better data utilization.

Databricks addresses these components with its new governance solution, Unity Catalog. Unity Catalog offers advanced governance capabilities, including fine-grained access control, data auditing, lineage tracking, and enhanced data discovery. Previously, Databricks relied on the Hive metastore, which was primarily focused on managing metadata for tables and schemas and had limited governance features.

In the following sections, we will explore how data governance is implemented within Databricks by examining two key aspects: managing data security in the existing Hive metastore and exploring advanced data governance with Unity Catalog. By understanding the transition from the Hive metastore to Unity Catalog, you will gain insights into how Databricks has evolved its data governance strategy to meet the demands of modern data management.

Managing Data Security in the Hive Metastore

In Databricks, the Hive metastore is the traditional, legacy solution for managing metadata and ensuring data governance. It serves as a local repository for metadata about tables, columns, partitions, and databases in each workspace, facilitating efficient querying and data management. Effective data security in the Hive metastore is essential to ensure that sensitive data is protected and accessed only by authorized users.

The data governance model of the Hive metastore focuses on controlling access to data objects within the `hive_metastore` catalog. This model enables administrators to perform key operations programmatically through Spark SQL, including granting, denying, and revoking access permissions. These

capabilities are used to manage who can view or manipulate data, thus supporting robust data governance.

Granting Permissions

The primary command used for managing data access is the `GRANT` statement. This command is used to provide a specific privilege on a data object to a user or group. The general syntax for this command is as follows:

```
GRANT <privilege> ON <object-type> <object-name> TO <user or group>
```

For example, to grant read access on a table named `product_info` to a user with the email `user_1@example.com`, the following command is used:

```
GRANT SELECT ON TABLE product_info TO user_1@example.com
```

In this example, `user_1` is given permission to perform `SELECT` operations on the `product_info` table, allowing them to read data from this object. Such precise control helps ensure that users have access only to the data they need, thus maintaining data security and integrity.

Besides tables, permissions can also be granted on various other objects. Let's explore these different types of securable objects available in the Hive metastore.

Data object types

In the Hive metastore, permission management extends across various types of data objects. The hierarchy of these objects is structured into three primary levels, as illustrated in [Figure 8-2](#).

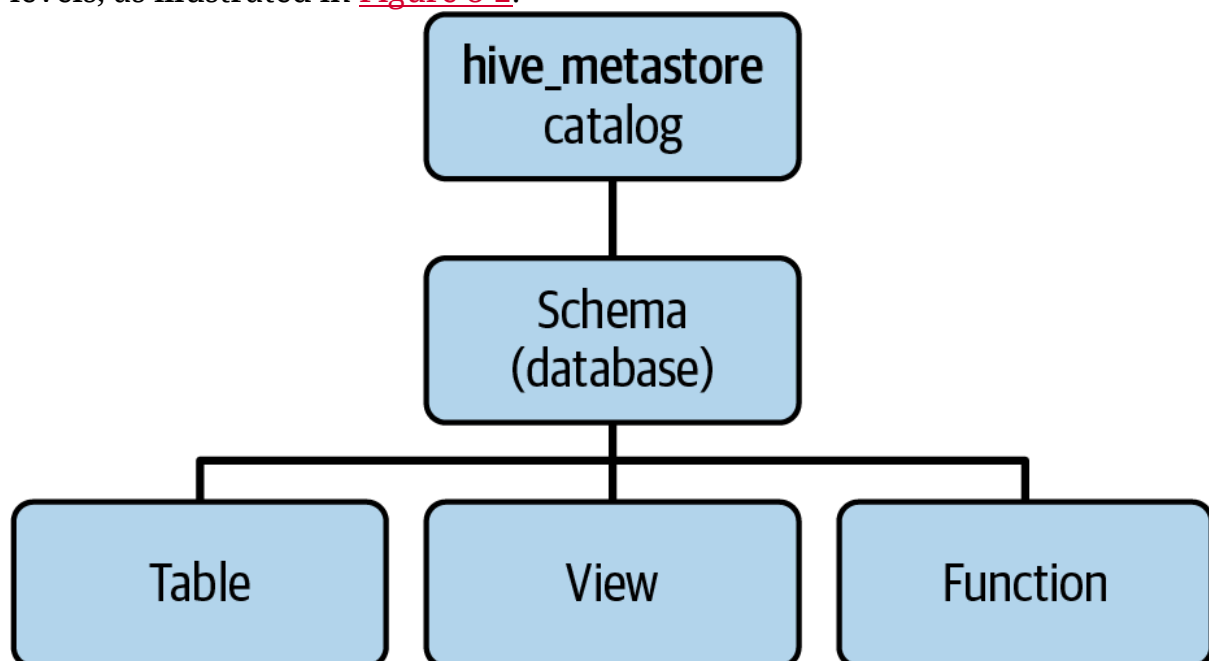


Figure 8-2. Data object hierarchy in the Hive metastore

The `hive_metastore` catalog

At the top of this hierarchy is the `hive_metastore` catalog. This catalog acts as a container for all data objects managed by the metastore. In Databricks, the Hive metastore is limited to this single default catalog, meaning that you cannot create additional catalogs within the Hive metastore.

Schemas (databases)

Within the `hive_metastore` catalog, there are schemas (also known as databases). A schema is a logical grouping of tables, views, and functions. Schemas facilitate granular access control, enabling administrators to manage permissions at a more detailed level than the catalog.

Tables, views, and functions

These objects are fundamental data objects within a schema. This level is ideal for situations where access needs to be restricted to particular entities within a database.

Databricks provides granular control over access to any of these data objects using the `GRANT` statement:

```
GRANT <privilege> ON <object-type> <object-name> TO <user or group>
```

In this context, `<object-type>` indicates the type of securable object, as outlined in [Table 8-1](#).

Object	Scope
CATALOG	Controls access to the entire data catalog
SCHEMA	Controls access to a specific database
TABLE	Controls access to a managed or external table
VIEW	Controls access to a SQL view
FUNCTION	Controls access to a specific named function
ANY FILE	Controls access to the underlying file system

Table 8-1. Types of securable objects in the Hive metastore

By specifying the appropriate object types, administrators can precisely manage permissions and ensure secure and efficient access control.

Having identified these types of data objects on which privileges can be configured, let’s now examine the specific privileges that can be granted.

Object privileges

Object privileges are permissions that can be granted to users or groups to perform various actions on data objects within the metastore. These privileges are granted using the `GRANT` statement:

```
GRANT <privilege> ON <object-type> <object-name> TO <user or group>
```

In this context, `<privilege>` represents the specific action being allowed. [Table 8-2](#) outlines the various privileges available in the Hive metastore and the capabilities they grant.

Privilege	Ability
SELECT	Read access to an object.
MODIFY	Add, delete, and modify data within an object.
CREATE	Create new objects.
READ_METADATA	View an object and its metadata.
USAGE	Required to perform any action on a database object, but has no direct effect on its own.
ALL PRIVILEGES	Grants all the above privileges simultaneously.

Table 8-2. Object privileges in the Hive metastore

Let’s dive deeper to gain a comprehensive understanding of these privileges.

SELECT privilege

This privilege grants read access to an object. Users with the `SELECT` privilege can query and retrieve data from the database object, such as a table, but cannot modify it:

```
GRANT SELECT ON TABLE product_info TO user_1@example.com;
```

MODIFY privilege

The `MODIFY` privilege allows users to add new data, delete existing data, and make modifications to the data within the object. This is translated by the ability to perform `INSERT`, `UPDATE`, and `DELETE` operations on a table:

```
GRANT MODIFY ON TABLE product_info TO user_1@example.com;
```

CREATE privilege

With the `CREATE` privilege, a user can create new objects. This is typically used for creating tables, views, or other schema objects within the database:

```
GRANT CREATE ON SCHEMA sales_db TO user_1@example.com;
```

READ_METADATA privilege

This privilege permits users to view an object along with its metadata. Metadata includes information about the structure of the object and its properties.

```
GRANT READ_METADATA ON TABLE product_info TO user_1@example.com;
```

USAGE privilege

The `USAGE` privilege, while not providing any direct access or ability on its own, is a prerequisite for performing any other actions on a database object. It is often combined with other privileges to enable their functionalities:

```
GRANT USAGE ON SCHEMA sales_db TO user_1@example.com;
```

ALL PRIVILEGES

This is a comprehensive privilege that grants all the individual privileges mentioned previously to a user or group. It simplifies the process of assigning multiple permissions by bundling them together.

To illustrate the application of these privileges, consider a scenario where we want to provide a user with full control over the `sales_db` schema. To achieve this, we would use the following command:

```
GRANT ALL PRIVILEGES ON SCHEMA sales_db TO user_1@example.com;
```

This command ensures that `user1` can read data, modify data, create new objects, and view metadata, and has the necessary `USAGE` privilege on the `sales_db` schema.

Granting privileges by role

In the Hive metastore, managing access to data objects is restricted to certain roles. To grant privileges on an object, one must be either a Databricks administrator or the owner of that particular object. [Table 8-3](#) illustrates the

different roles in the Hive metastore and the privileges that can be granted by each of them.

Role	Can grant access privileges for
Databricks administrator	All objects in the catalog and the underlying file system
Catalog owner	All objects in the catalog
Database owner	All objects in the database
Database object owner	Only the specific object within the database (table, view, ...)

Table 8-3. Roles in the Hive metastore

Let’s explore the details of each role and its abilities:

Databricks administrator

This role has the highest level of access, capable of granting privileges for all objects within the `hive_metastore` catalog as well as the underlying file system. This broad scope of authority enables comprehensive management and oversight of the data infrastructure.

Catalog owner

A catalog owner has the ability to grant access privileges for all objects contained within the `hive_metastore` catalog. This includes databases, tables, views, and functions, allowing for centralized control at the catalog level.

Database owner

Database owners are empowered to grant privileges for all objects within a single database. This includes tables, views, and functions within that particular database, ensuring that database-specific access control is managed efficiently.

Database object owner

Database object owners can grant privileges solely for the object they own within the database, such as a table, view, or function.

Understanding these roles and their associated privileges is crucial for maintaining a well-organized and secure data environment. Properly assigning and managing these roles ensures that access control is enforced at the appropriate levels, reducing the risk of unauthorized access while facilitating efficient data management.

Advanced Privilege Management

In the Hive metastore, managing object privileges extends beyond merely granting access. You can also explicitly deny access and revoke previously granted permissions, providing a comprehensive set of tools to ensure precise control over data access.

REVOKE operation

The `REVOKE` operation removes permissions that were previously granted to a user or group. This operation is essential for dynamically managing access as organizational roles and requirements change.

The syntax is as follows:

```
REVOKE <privilege> ON <object-type> <object-name> FROM <user or group>
```

Here is an example of the operation:

```
REVOKE SELECT ON TABLE product_info FROM user_1@example.com
```

In this example, the command removes the previously granted `SELECT` permission on the `product_info` table from `user1`.

DENY operation

The `DENY` operation is used to explicitly prevent a user or group from accessing specific resources or performing certain actions. This operation takes precedence over any other permissions that might otherwise allow access, ensuring that certain users or groups are definitively blocked from specific actions.

The syntax is as follows:

```
DENY <privilege> ON <object-type> <object-name> TO <user or group>
```

Here is an example:

```
DENY SELECT ON TABLE product_info TO user_1@example.com
```

This command will prevent `user1` from performing the `SELECT` action on the `product_info` table, regardless of other granted permissions.

SHOW GRANTS operation

The `SHOW GRANTS` operation allows administrators to view the current permissions assigned to a specific object. This command is useful for auditing and verifying access controls, ensuring that only authorized users have the necessary permissions.

The syntax is as follows:

```
SHOW GRANTS ON <object-type> <object-name>
```

Here is an example of the operation:

```
SHOW GRANTS ON TABLE product_info
```

This command will list all the permissions granted to each user or group for the `product_info` table, providing a clear overview of who has access and what actions they can perform.

With this clear understanding of object privileges, let's now turn our attention to the Databricks platform to see how these concepts can be applied in Databricks SQL.

Managing Permissions with Databricks SQL

In this section, we'll explore the practical steps for managing permissions for databases, tables, and views in the Hive metastore within the context of an HR example. We'll first cover the process of adding users and groups specific to HR personnel within the Databricks workspace. Then, we'll discuss assigning the appropriate permissions through Databricks SQL to ensure secure access to sensitive HR data.

Adding users

To begin managing permissions in Databricks SQL, we need to first add the designated users within our Databricks workspace. For this demonstration, we will add three fictional users: Alice, Bob, and Eve. Here's a step-by-step guide to adding them:

1. **Navigate to user settings:** In the upper-right corner of any Databricks page, click your username and select Settings from the drop-down menu.
2. **Access identity management:** From the left sidebar, select "Identity and access," and then click Manage next to Users, as displayed in [Figure 8-3](#).

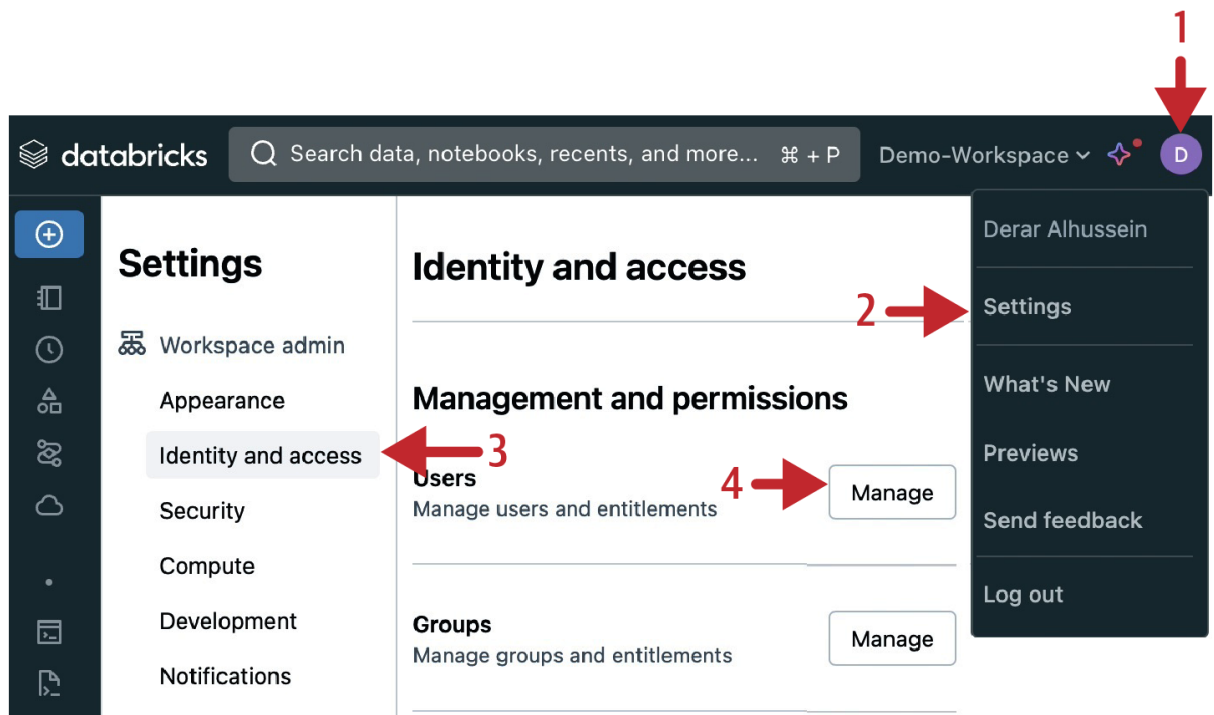


Figure 8-3. Identity and access settings in Databricks workspace

3. Add users: On the Users management page, click “Add user,” and enter the email of each new user:

alice@example.com

bob@example.com

eve@example.com

Figure 8-4 illustrates the final outcome of the user additions.

Workspace settings / [Identity and access](#) /

Users

<input type="text" value="Filter users"/> 3 total Add user		
Status	Email	Name
✓	alice@example.com	⋮
✓	bob@example.com	⋮
✓	eve@example.com	⋮
✓	derar@oreilly.com	Derar Alhussein ⋮
< Previous Next > 20 / page		

Figure 8-4. The result of user additions

With these users added to the workspace, we can now proceed to add a new group and assign permissions.

Adding groups

To streamline permission management and apply access controls across multiple users, groups can be utilized in Databricks. Here, we demonstrate how to add a group named `hr_team` and assign users to this group:

1. Access user settings: Navigate again to the Settings menu.
2. Access identity management: From the left sidebar, select “Identity and access,” and then click Manage next to Groups, as shown in [Figure 8-5](#).

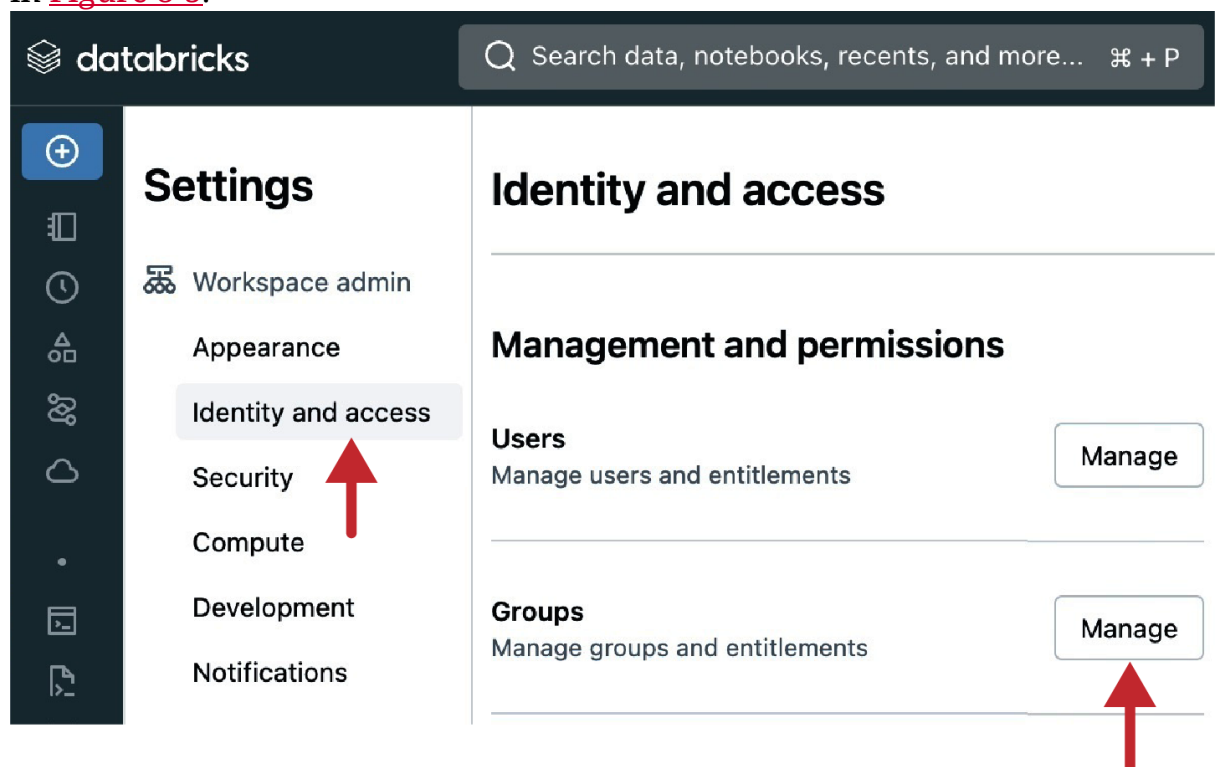


Figure 8-5. Identity and access management

3. Add group: On the Groups management page, click “Add group,” enter the group name as `hr_team`, and confirm the addition. [Figure 8-6](#) illustrates the result of the group addition.

Groups

<input type="text" value="Filter groups"/>	2 total	<button>Add group</button>
Name	Members	Source
admins	1	System ⓘ ⋮
hr_team	0	Account ⓘ ⋮
users	4	System ⓘ ⋮
< Previous Next >		20 / page ▼

Figure 8-6. The Groups management page after adding the `hr_team` group

4. Add members to the group: Click the group name `hr_team` from the groups list to open the group settings. Under the Members tab, click “Add member,” and add the users Alice (*alice@example.com*) and Bob (*bob@example.com*) to this group.

Figure 8-7 displays the final outcome of adding the `hr_team` group with its members.

Group details



hr_team		<button>Delete</button>
Group Information	Members	Entitlements Parent groups
<input type="text" value="Filter group mem..."/>	<button>Add members</button>	
Name	Email/ID	
 alice@example.com	alice@example.com	⋮
 bob@example.com	bob@example.com	⋮
< Previous Next >		20 / page ▼

Figure 8-7. The result of adding the `hr_team` group with its members

By grouping users, permissions can be managed more efficiently. Permissions granted to the group will automatically apply to all its members, simplifying access control administration.

Creating data objects

For this demonstration, we will create some data objects in the Hive metastore, specifically focusing on creating a database and defining a table and a stored view within this database.

To follow along with this section, you will need to use Databricks SQL. Begin by navigating to the SQL editor located in the left sidebar of the Databricks workspace. Before executing any commands, ensure that a SQL warehouse you can use is running.

Our first step is to create a new database called `hr_db` within the `hive_metastore` catalog. The command for creating this database is as follows:

```
CREATE DATABASE IF NOT EXISTS hive_metastore.hr_db
LOCATION 'dbfs:/mnt/demo/hr_db.db';
```

After creating the database, the next step is to define a table and populate it with sample data. We will create a table named `employees` with columns for ID, name, salary, and city. To do this, execute the following SQL commands:

```
CREATE TABLE hive_metastore.hr_db.employees
(id INT, name STRING, salary DOUBLE, city STRING);
```

```
INSERT INTO hive_metastore.hr_db.employees
VALUES (1, "Felipe", 3000, "London"),
       (2, "Sachin", 3400, "New York"),
       (3, "Anna", 3600, "London"),
       (4, "Hong-Thai", 3200, "London"),
       (5, "Charlotte", 3500, "New York"),
       (6, "Amine", 3400, "New York"),
       (7, "Emily", 3200, "London");
```

In addition, create a view named `london_employees_vw` to display employees located in London:

```
CREATE VIEW hive_metastore.hr_db.london_employees_vw
AS SELECT * FROM hive_metastore.hr_db.employees WHERE city = 'London';
```

After executing these commands, you can explore the created objects in the Catalog Explorer. [Figure 8-8](#) displays the structure and contents of the `hr_db` database within the `hive_metastore` catalog.

Now that we have created our database and its objects, the next step is configuring their permissions to control access. Properly setting permissions ensures that only authorized users and groups can access and manipulate data objects, thereby maintaining data security and integrity.

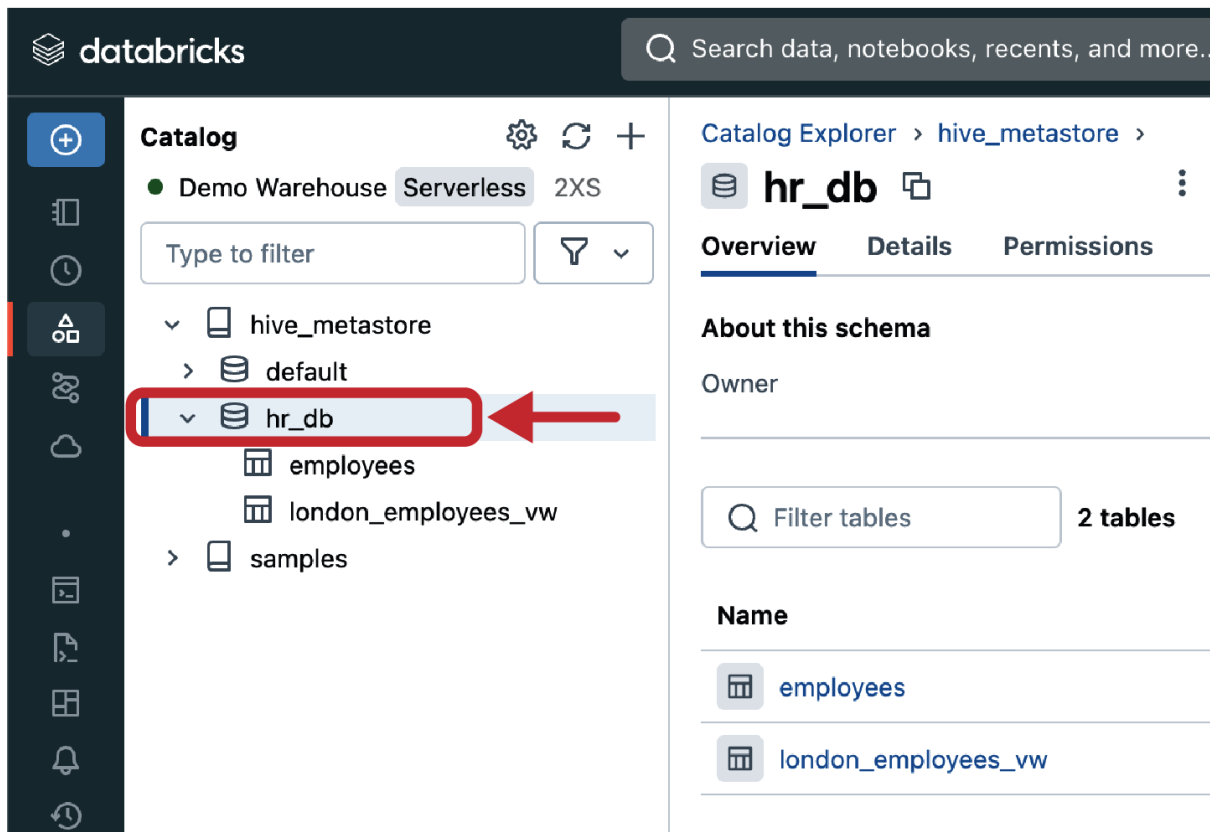


Figure 8-8. Reviewing the `hr_db` schema in the Catalog Explorer

Configuring object permissions

In this step, we will grant permissions to the necessary users and groups, allowing them to interact with our database and its objects effectively.

Granting privileges to a group

We begin by granting several privileges on the entire HR database (`hr_db`) to the `hr_team` group. This will enable all members of this group to read and modify the data, access metadata information, and create new objects such as tables and views within this database. The SQL command to achieve this is as follows:

```
GRANT SELECT, MODIFY, READ_METADATA, CREATE
ON SCHEMA hive_metastore.hr_db TO hr_team;
```

With this command, the HR team members will have the permissions to perform a range of actions on the `hr_db` schema, enhancing their ability to manage and utilize the HR data effectively.

In addition to the previously granted privileges, users must have the `USAGE` privilege to perform any action on database objects. Without this privilege, the objects within the database cannot be accessed or utilized by the group. The following command grants the `USAGE` privilege to the `hr_team` group:

```
GRANT USAGE ON SCHEMA hive_metastore.hr_db TO hr_team;
```

By executing this command, the `hr_team` group members will have the ability to use the database objects, ensuring they can interact with both data and metadata as needed.

Granting privileges to an individual user

In addition to configuring permissions for groups, it is often necessary to assign specific privileges to individual users for more customized access control. To illustrate this, we will grant read access on the `london_employees_vw` view to the user Eve (`eve@example.com`), who is not a member of the `hr_team`.

The SQL command to grant the specified permission to Eve is as follows:

```
GRANT SELECT  
ON VIEW hive_metastore.hr_db.london_employees_vw TO `eve@example.com`;
```

This selective permission allows Eve to access the information she needs without requiring her to be added to the `hr_team` group with broader permissions.

NOTE

While it is possible to assign privileges to individual users, it is generally recommended to grant permissions to groups. This practice simplifies permission management, especially in dynamic organizational environments where team structures frequently change.

Reviewing assigned permissions

After configuring the necessary permissions for groups and individual users, it is important to verify that these permissions have been applied correctly.

The `SHOW GRANTS` command is used to display the assigned privileges, ensuring transparency and accuracy in permission management.

To review the permissions assigned on the `hr_db` schema, we use the following SQL command:

```
SHOW GRANTS ON SCHEMA hive_metastore.hr_db;
```

Executing this command produces the list of granted privileges on our database, as displayed in [Figure 8-9](#). This confirms that the HR team has all the required permissions.

A^B_C Principal	A^B_C ActionType	A^B_C ObjectType	A^B_C ObjectKey
hr_team	CREATE	DATABASE	hr_db
hr_team	READ_METADATA	DATABASE	hr_db
hr_team	SELECT	DATABASE	hr_db
hr_team	MODIFY	DATABASE	hr_db
hr_team	USAGE	DATABASE	hr_db
derar@oreilly.com	OWN	DATABASE	hr_db

Figure 8-9. The output of the `SHOW GRANTS` command on the `hr_db` schema

Additionally, this figure shows that the user who created the database (in this case, me) is the owner of the database.

To review the permissions assigned to the `london_employees_vw` view, we use the following SQL command:

```
SHOW GRANTS ON VIEW hive_metastore.hr_db.london_employees_vw;
```

This command produces the list of granted privileges on our view, confirming that Eve has the `SELECT` privilege, as illustrated in [Figure 8-10](#).

A^B_C Principal	A^B_C ActionType	A^B_C ObjectType	A^B_C ObjectKey
eve@example.com	SELECT	TABLE	`hr_db`.`london_employees_vw`
derar@oreilly.com	OWN	TABLE	`hr_db`.`london_employees_vw`
hr_team	CREATE	DATABASE	hr_db
hr_team	READ_METADATA	DATABASE	hr_db
hr_team	SELECT	DATABASE	hr_db
hr_team	MODIFY	DATABASE	hr_db
hr_team	USAGE	DATABASE	hr_db

Figure 8-10. The output of the `SHOW GRANTS` command on the `london_employees_vw` view

Moreover, this figure shows that the HR team has inherited the relevant privileges on this view from the database level.

Managing permissions in Catalog Explorer

Beyond the SQL editor, we can also manage permissions through the Catalog Explorer. This simplifies the process of configuring access controls, making it easier for administrators who prefer a visual interface over writing SQL commands. To access the Catalog Explorer, click the Catalog tab in the left sidebar of your Databricks workspace.

In the Catalog Explorer, locate the database you previously created (`hr_db`). Clicking the database name will display a list of contained tables and views on

the left-hand side. On the right-hand side, you will see detailed information about the database, such as owner information, as displayed in [Figure 8-11](#).

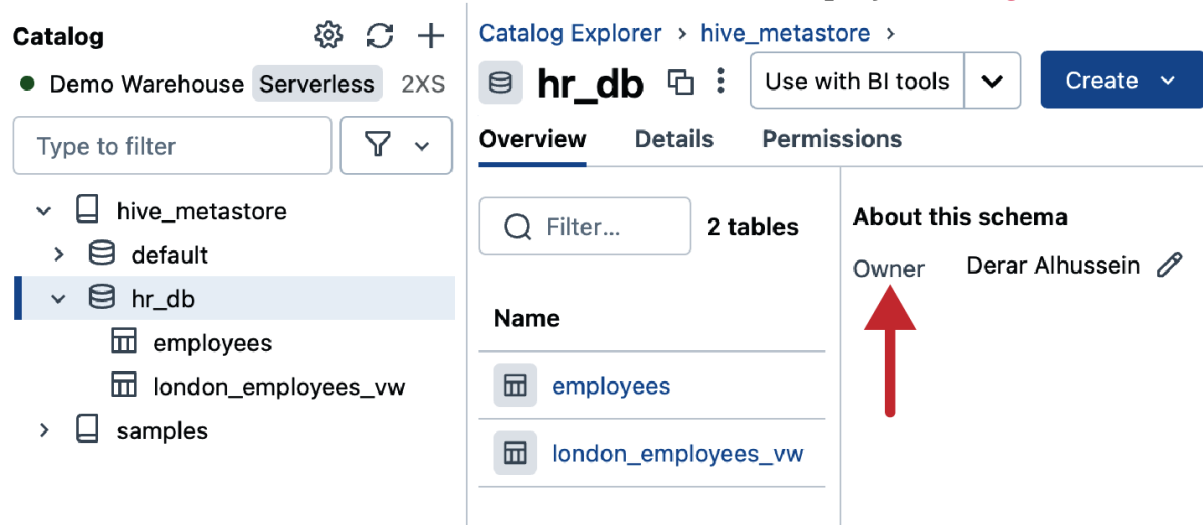


Figure 8-11. Reviewing the `hr_db` schema in the Catalog Explorer

From this interface, you can change the owner of the database by clicking the pencil icon next to the owner's name. It's worth noting that the owner of a database can be set as either an individual user or a group of users. This is necessary to ensure that the correct person or team has administrative control and access to the database, which is especially important when there are changes in personnel or when database management responsibilities need to be reassigned.

Reviewing and modifying permissions

Under the Permissions tab, you can review the current permissions for the database. This tab lists the granted privileges, such as those for the `hr_team` group, as illustrated in [Figure 8-12](#).

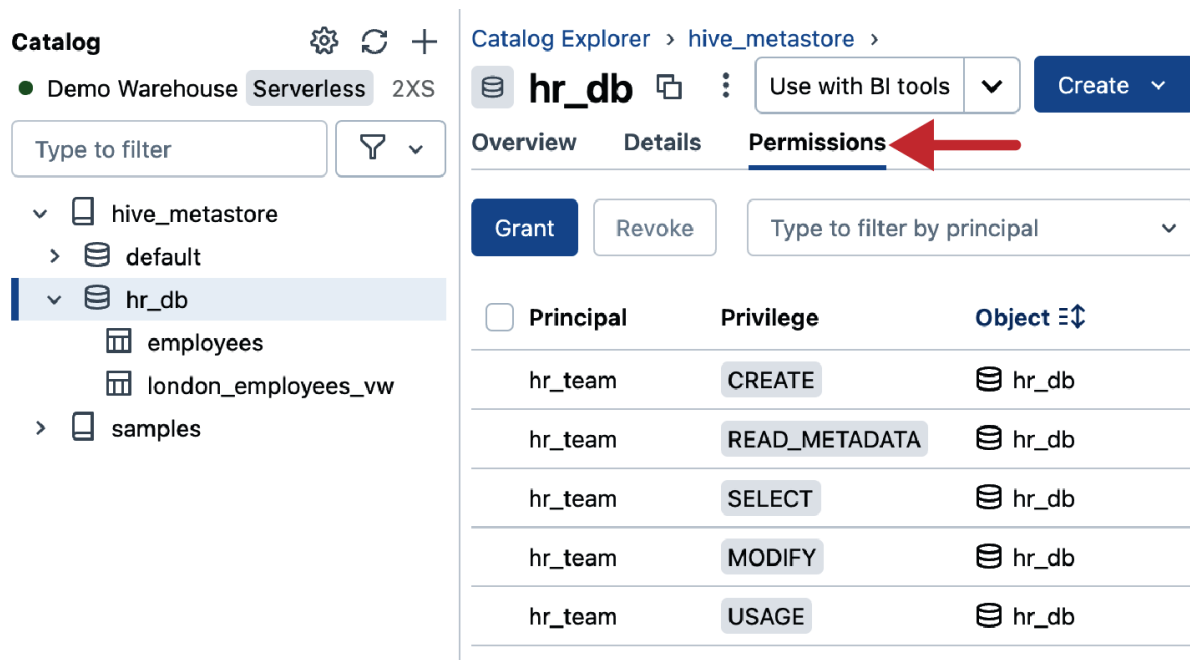


Figure 8-12. Reviewing the permissions of the `hr_db` schema in the Catalog Explorer

In the interface, you have the capability to manage access permissions, including granting and revoking privileges from users or groups. Next, we'll go through a step-by-step guide for each of these operations.

Granting new permissions

To grant new permissions, such as allowing all workspace users to review metadata about the database, use the following steps:

1. Click the Grant button: Begin by clicking the Grant blue button, which will open the grant permissions dialog, as shown in [Figure 8-13](#).
2. Select the group: In the dialog, search for and select the “All workspace users” group from the list of available groups.
3. Choose privileges: Next, select the specific privileges you want to grant. In this case, you will choose both `READ_METADATA` and `USAGE` privileges.
4. Confirm granting of permissions: Finally, click the Grant button to apply the permissions.

Grant on hr_db

All workspace users

Check the privileges you would like to grant.

☐ ALL gives all privileges

☐ SELECT gives read access to an object

☐ MODIFY gives ability to add, delete, and modify data to or from an object

☒ READ_METADATA gives ability to view an object and its metadata

☒ USAGE does not give any abilities, but is an additional requirement to perform any action on a schema object

☐ CREATE gives ability to create an object (for example, a table in a schema)

☐ CREATE_NAMED_FUNCTION gives ability to create a named UDF in an existing catalog or schema

CancelGrant

Figure 8-13. The grant permissions dialog in the Catalog Explorer

Figure 8-14 illustrates the interface where the two newly granted privileges to all the workspace users are displayed.

hr_db

Use with BI tools

Create

Overview

Details

Permissions

Grant

Revoke

Type to filter by principal

Principal	Privilege	Object
All workspace users	READ_METADATA	hr_db
All workspace users	USAGE	hr_db
hr_team	USAGE	hr_db
hr_team	CREATE	hr_db
hr_team	MODIFY	hr_db

Figure 8-14. The permissions interface of the hr_db schema showing the newly granted privileges

This interface simplifies the process of granting permissions, eliminating the need to write complex SQL commands. Users can easily manage permissions through intuitive steps, making it accessible even for those with limited technical knowledge.

Revoking permissions

To remove existing privileges from a user or group, follow these steps:

1. Select the privilege(s) you want to revoke.

2. Click the Revoke button.

This action will remove the selected privilege from the designated user or group. For instance, consider the example shown in [Figure 8-15](#). Here, the action of revoking the `CREATE` privilege from the `hr_team` is illustrated.

hr_db

Overview Details **Permissions**

Grant Revoke **2** Type to filter by principal

Principal	Privilege	Object
<input type="checkbox"/> All workspace users	READ_METADATA	hr_db
<input type="checkbox"/> All workspace users	USAGE	hr_db
<input type="checkbox"/> hr_team	USAGE	hr_db
<input checked="" type="checkbox"/> hr_team 1	CREATE	hr_db
<input type="checkbox"/> hr_team	MODIFY	hr_db
<input type="checkbox"/> hr_team	READ_METADATA	hr_db

Figure 8-15. Revoking permissions via the Catalog Explorer

By revoking the `CREATE` privilege, the `hr_team` will no longer be able to create new objects within the `hr_db` schema.

Managing permissions for database objects

The Catalog Explorer not only allows you to manage database-level permissions but also extends this capability to individual data objects, such as tables and views. To manage permissions for these objects, click the desired table or view name from the left-side navigator, and open its Permissions tab, as displayed in [Figure 8-16](#).

Catalog Explorer > hive_metastore > hr_db >

london_employees_vw

Overview Sample Data Details **Permissions** History Lineage Insights

Grant Revoke Type to filter by principal

Principal	Privilege	Object
eve@example.com	SELECT	hr_db.london_employees_vw
hr_team	CREATE	hr_db
hr_team	READ_METADATA	hr_db
hr_team	SELECT	hr_db
hr_team	MODIFY	hr_db
hr_team	USAGE	hr_db
All workspace users	READ_METADATA	hr_db
All workspace users	USAGE	hr_db

Figure 8-16. Reviewing the permissions of the `london_employees_vw` view in the Catalog Explorer

This figure illustrates the permissions interface for the `london_employees_vw` view, showing its current privileges and providing options to revoke and grant new permissions.

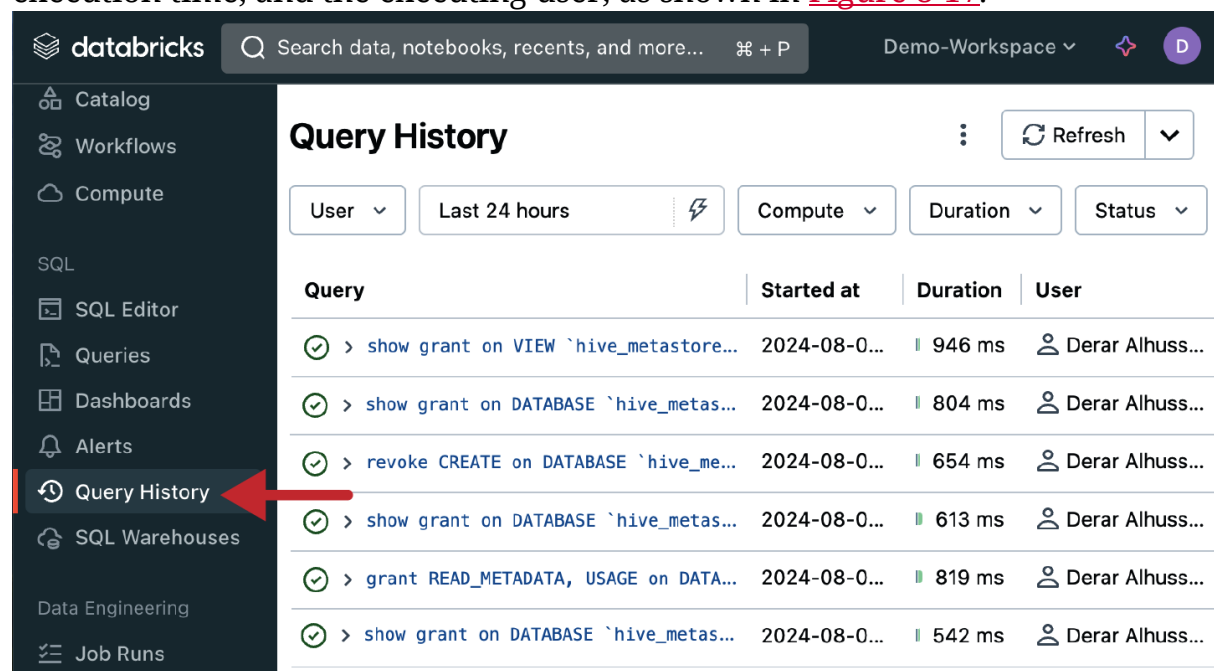
Limitations of the Catalog Explorer

While the Catalog Explorer is a versatile and powerful tool for managing data objects and permissions, there are some limitations. Specifically, the `ANY FILE` object cannot be managed through the Catalog Explorer and must be handled using the SQL editor.

Query History

One of the notable features of Databricks SQL is the query history functionality. This feature provides the ability to view all SQL queries executed in Databricks SQL, including those run behind the scenes by the Catalog Explorer.

To view these queries, simply select the Query History tab from the left sidebar of your Databricks workspace. This opens the query history interface, listing all executed queries along with details such as the query text, the execution time, and the executing user, as shown in [Figure 8-17](#).



Query	Started at	Duration	User
> show grant on VIEW `hive_metastore...	2024-08-0...	946 ms	Derar Alhuss...
> show grant on DATABASE `hive_metas...	2024-08-0...	804 ms	Derar Alhuss...
> revoke CREATE on DATABASE `hive_me...	2024-08-0...	654 ms	Derar Alhuss...
> show grant on DATABASE `hive_metas...	2024-08-0...	613 ms	Derar Alhuss...
> grant READ_METADATA, USAGE on DATA...	2024-08-0...	819 ms	Derar Alhuss...
> show grant on DATABASE `hive_metas...	2024-08-0...	542 ms	Derar Alhuss...

Figure 8-17. Query history in Databricks SQL

This transparency allows users to understand the exact commands being run and provides an opportunity to learn and replicate these commands manually if needed.

In conclusion, while the Hive metastore serves as a foundational component for metadata management, it offers only basic governance capabilities. It is clear that it lacks some advanced security and governance features required for modern data environments. This is where Unity Catalog comes into play, which will be the focus of our next section.

Governing Data with Unity Catalog

In the previous sections, we discussed the data governance model of Databricks with a default Hive metastore, highlighting its key features and limitations. Now, we turn our attention to Unity Catalog, the innovative governance solution introduced by Databricks. Unity Catalog represents a significant advancement in data management, offering a more robust and scalable architecture. This overview will delve into the key aspects of Unity Catalog, detailing its architecture, the hierarchical organization of its data objects, and its enhanced security model designed to meet the evolving needs of modern data governance.

What Is Unity Catalog?

[Unity Catalog \(UC\)](#) is an open source, centralized governance solution that spans across all your workspaces on any cloud. It unifies governance for all data and AI assets in your lakehouse, including files, tables, machine learning models, and dashboards. This centralization ensures consistent access controls, and simplified data management, which enhances overall data governance and security.

Unity Catalog Architecture

Before the introduction of Unity Catalog, metastore management, user and group definitions, as well as access control, were handled within individual workspaces. This approach required separate configurations for each workspace, which could lead to inconsistencies and inefficiencies.

With Unity Catalog, governance and access control have been centralized, significantly improving manageability and consistency across multiple workspaces, as illustrated in [Figure 8-18](#). Unity Catalog operates independently of individual workspaces and is managed via the account console, a user interface designed for administrative tasks.

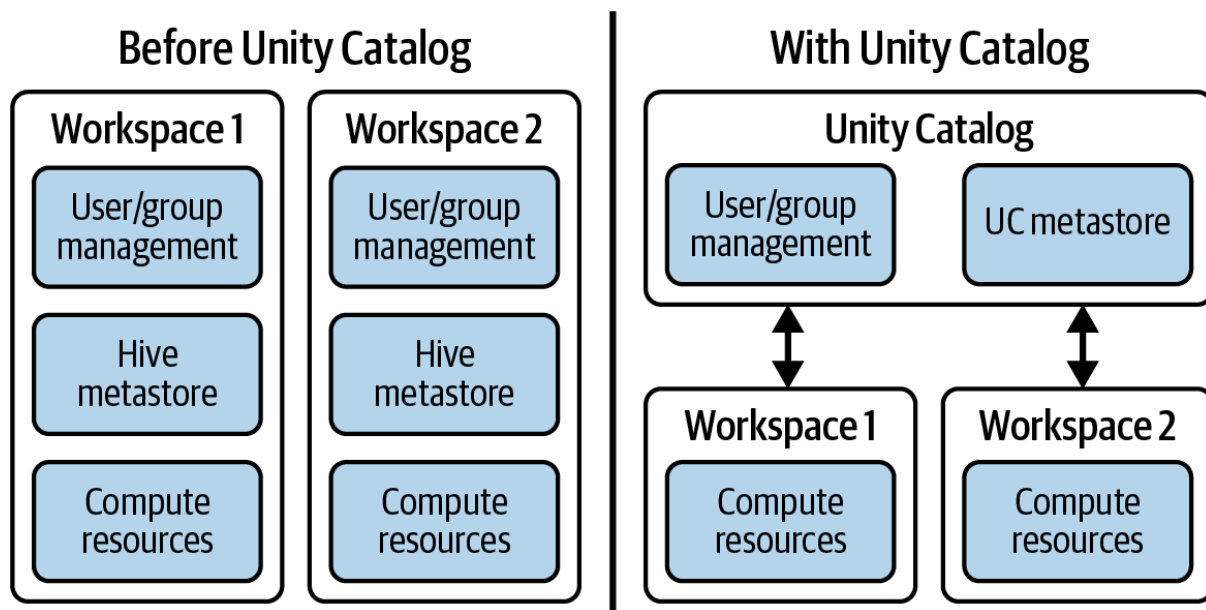


Figure 8-18. Databricks workspaces management before and after Unity Catalog

This figure illustrates how Unity Catalog decouples user, group, and metastore management from individual workspaces, offering a centralized approach to data governance.

Key Architectural Changes

There are three main components to the changes implemented by Unity Catalog:

Centralized user and group management

Unity Catalog utilizes the account console for managing users and groups, which can then be assigned to multiple workspaces. This approach means that user and group definitions are consistent across all workspaces assigned to Unity Catalog.

Separation of metastore management

Unlike the workspace-specific metastore used previously, Unity Catalog's metastores are managed centrally per cloud region through the account console. A single Unity Catalog metastore can serve multiple workspaces, allowing them to share the same underlying storage. This consolidation simplifies data management, improves data accessibility, and reduces data duplication, as multiple workspaces can access the same data without needing to replicate it across different environments.

Centralized access control

Access controls within Unity Catalog are controlled centrally and apply across all workspaces. This ensures that defined policies and permissions are enforced consistently across the organization, thereby enhancing overall security.

UC Three-Level Namespace

In the traditional Hive metastore, where there is only a single catalog, a two-level namespace (`schema.table`) was sufficient to address tables within schemas. While this structure is simple and effective for many use cases, it can become limiting as data volume and complexity increase, particularly in large-scale environments where a more granular level of organization is beneficial. To address these limitations, Unity Catalog introduces a three-level namespace that enhances data organization and management. This new structure incorporates an additional layer called *catalogs*, which sits above schemas in the hierarchy. The updated format for accessing tables becomes `catalog.schema.table`, as illustrated in [Figure 8-19](#).

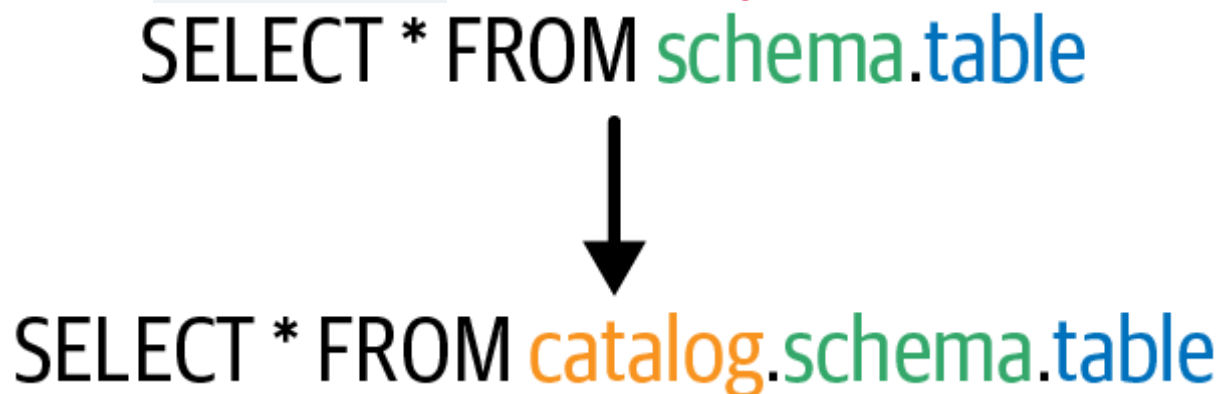


Figure 8-19. Transition to Unity Catalog's three-level namespace

To better understand the structure of Unity Catalog and its organization, let us explore the hierarchical model of its data objects in detail.

Data Object Hierarchy

The hierarchy of data objects in Unity Catalog begins with the metastore, which serves as the top-level logical container, as illustrated in [Figure 8-20](#). The metastore holds metadata, including information about the objects it manages and the access control lists (ACLs) that govern access to these objects.

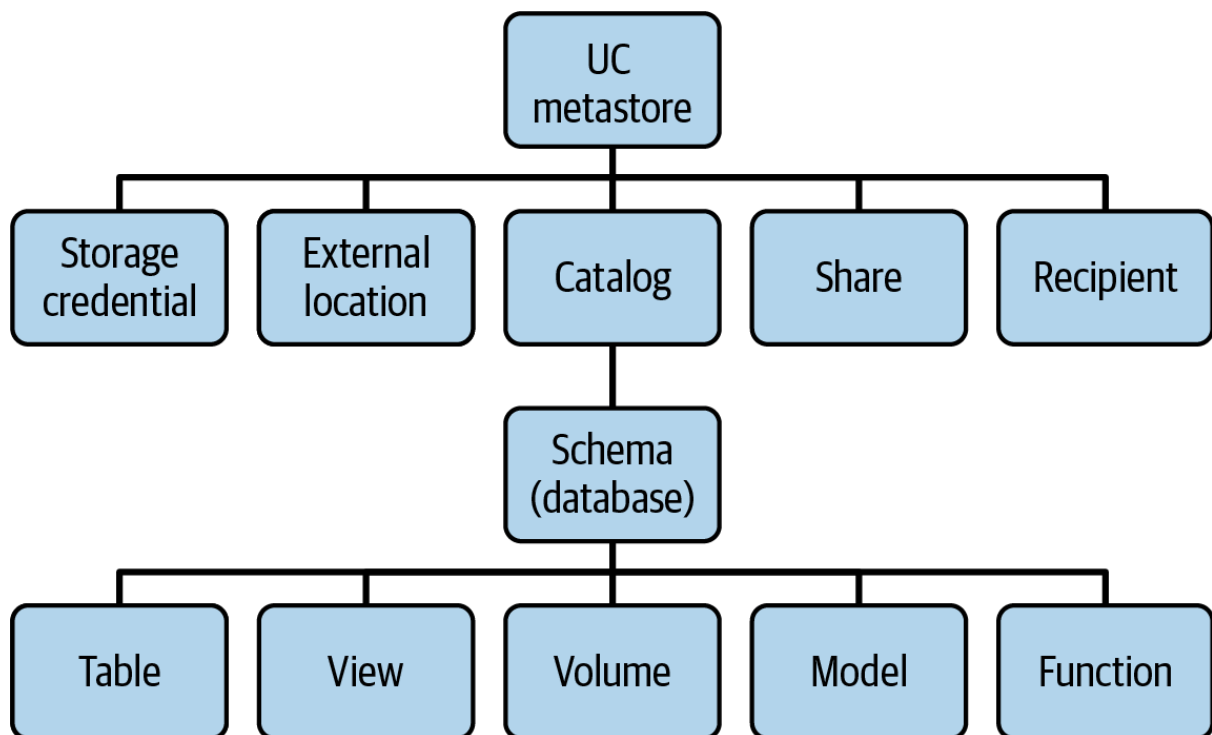


Figure 8-20. Unity Catalog hierarchy

Detailed Hierarchical Structure

Unity Catalog's hierarchical structure is based on relationships among a number of objects:

Metastore

The metastore is the top-level entity in Unity Catalog, containing catalogs and managing metadata and access control. It operates independently from workspaces, providing enhanced security and a unified governance model.

Catalogs

Within each metastore, catalogs act as high-level containers for organizing data objects. They represent the first part of the three-level namespace. A metastore can contain multiple catalogs, allowing for flexible and scalable data organization.

Schemas (databases)

Schemas reside within catalogs and represent the second part of the three-level namespace. They are commonly referred to as databases. Schemas group related data and AI assets, facilitating organized data management.

Data and AI assets

These are the third part of the three-level namespace. They reside within schemas and represent the actual data structures and AI artifacts

managed by Unity Catalog, such as tables, views, storage volume, registered machine learning models, and functions.

Storage access objects

Unity Catalog allows you to set custom locations in cloud storage for storing managed and external tables. This introduces new securable objects to govern data access to these locations: storage credentials and external locations. A storage credential abstracts long-term access keys from cloud storage providers, while an external location links a given storage location with a storage credential.

Delta sharing entities

Unity Catalog also supports Delta Sharing, an open protocol designed for the efficient exchange of large datasets. Within Unity Catalog, shares represent collections of assets that can be shared with designated consumers (Recipients). Please note that Delta Sharing is not included as a topic in the Databricks Data Engineer Associate certification exam.

By understanding the relationships among these objects, you can effectively manage your data infrastructure and leverage Unity Catalog's capabilities.

Identity Management

Unity Catalog categorizes identities into three types: users, service principals, and groups. These identities are referred to as principals and play a vital role in managing access and permissions.

Users

Users are individuals who interact with the Databricks environment. Each user is uniquely identified by their email address, which serves as a unique identifier. Users can be assigned various roles, including administrative roles, allowing them to perform advanced tasks such as managing metastores, assigning metastores to workspaces, and managing other users.

Service principals

Service principals are designed to represent automated tools or applications. They are identified by an application ID, which ensures that automated processes have a distinct identity separate from human users. Service principals can be assigned administrative roles similar to users, allowing them to perform essential tasks programmatically. This capability is particularly useful for facilitating automation and integration with other systems.

Groups

Groups are collections of users and service principals, treated as a single entity. Groups simplify the management of permissions by allowing administrators to assign roles and privileges to a group rather than to individual users. Groups can be nested within other groups. For example, a parent group named “Employees” could contain subgroups such as “HR” and “Finance.” This nesting capability mirrors organizational units or functional departments and allows for efficient delegation of access rights at different levels. Additionally, principals can belong to multiple non-nested groups.

By leveraging these identity types, administrators can streamline access control and ensure that both human and automated entities operate within their designated scopes.

Identity federation

In Databricks, identities are managed at two distinct levels: the account level and the workspace level. Unity Catalog introduces a feature called *identity federation*, which simplifies identity management by allowing identities to be created once at the account level and then assigned to multiple workspaces as needed. [Figure 8-21](#) visually demonstrates how identities created at the account level can be federated across multiple workspaces.

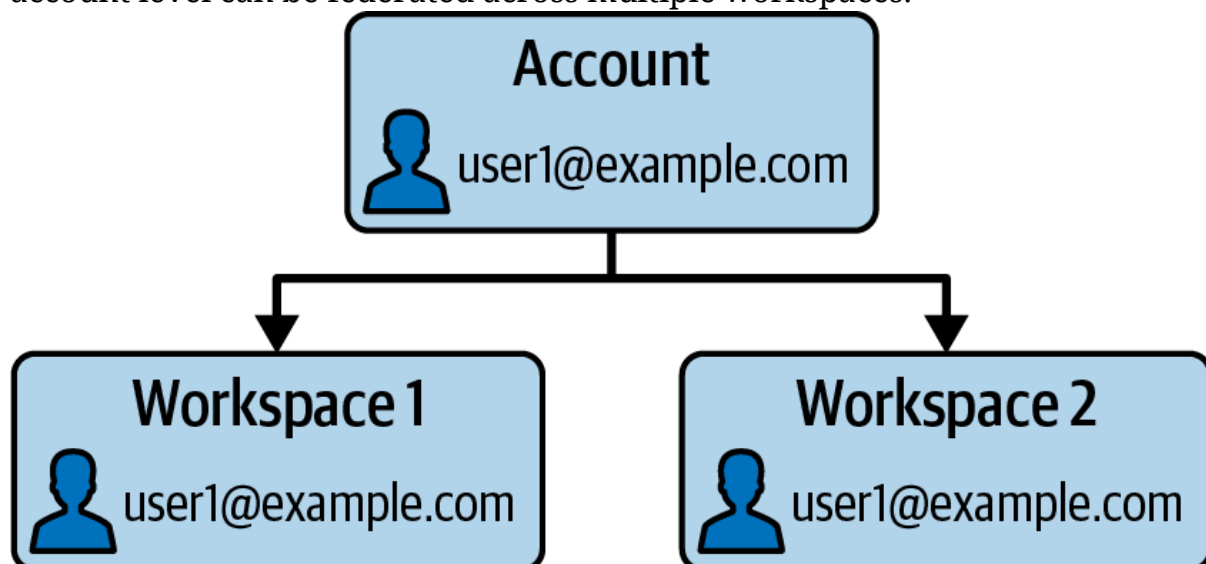


Figure 8-21. Identity federation in Unity Catalog

This approach is recommended for effective identity management across multiple workspaces. It significantly reduces the overhead associated with maintaining multiple copies of identities at workspace levels. As a result, it streamlines administrative tasks and enhances security and consistency across the Databricks environment.

UC Security Model

Unity Catalog offers a robust security model for permissions management based on standard ANSI SQL. It enhances data protection by offering granular access controls tailored to different types of operations and resources. This flexible and reliable model ensures efficient management and control over data access in the lakehouse.

Unity Catalog continues to use the `GRANT` statement for assigning privileges on securable objects to principals:

```
GRANT <privilege> ON <object-type> <object-name> TO <principal>
```

The UC security model provides a comprehensive set of privileges designed to efficiently manage access to various data and AI objects and underlying storage. Notably, two specific privileges now replace the legacy `ANY FILE` privilege from the Hive metastore, enhancing storage-related permissions. Here's a breakdown of the key categories:

Core privileges

`CREATE`

Allows users to create new objects, such as a catalog (`CREATE CATALOG`), a schema (`CREATE SCHEMA`), a table or view (`CREATE TABLE`), or a function (`CREATE FUNCTION`).

`USE`

Grants the ability to use a specified catalog (`USE CATALOG`) or schema (`USE SCHEMA`). Without this privilege, users cannot interact with the objects within the catalog or schema.

`SELECT`

Permits users to read data from tables or views.

`MODIFY`

Enables users to modify data within tables, including inserting, updating, and deleting records.

Storage-related privileges

`READ FILES`

Allows users to read files directly from the underlying storage linked to volumes and external locations.

`WRITE FILES`

Allows users to write files to the underlying storage.

Execution privilege

`EXECUTE`

Grants permission to invoke user-defined functions or load a machine learning model for inference.

[Figure 8-22](#) demonstrates the complete security model of Unity Catalog, highlighting its distinct approach compared to the Hive metastore's security model.

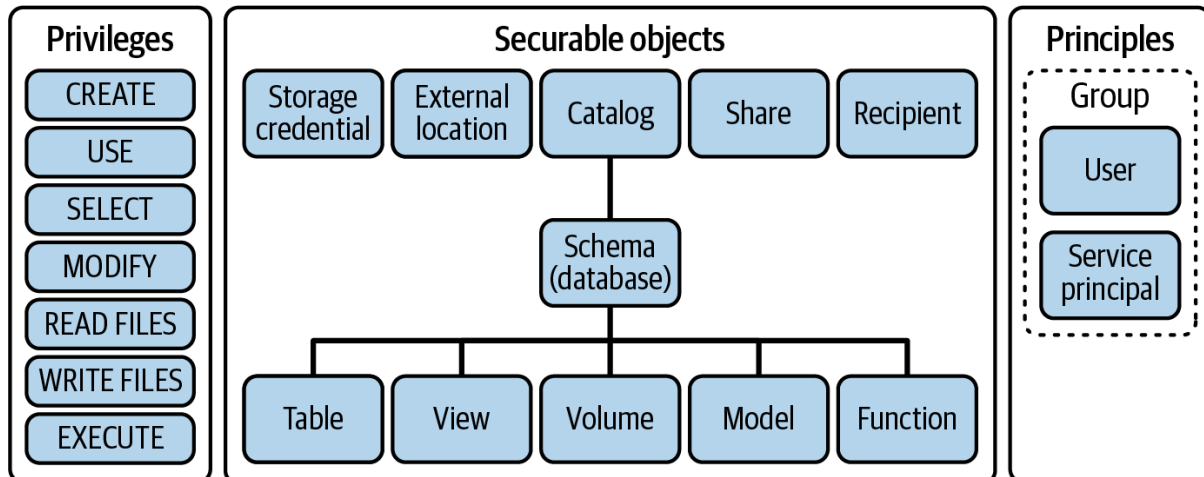


Figure 8-22. Security model of Unity Catalog

This comprehensive security model in Unity Catalog provides a clear and detailed framework for managing data access, supporting the needs of modern, data-driven organizations.

Accessing the Hive Metastore

Unity Catalog introduces advanced data governance and security features, but it is designed to be additive. This means that existing systems and data structures, such as the legacy Hive metastore, remain accessible and functional even after Unity Catalog is enabled.

When Unity Catalog is enabled in a Databricks workspace, the legacy Hive metastore remains available through its `hive_metastore` catalog, as illustrated in [Figure 8-23](#). This catalog provides seamless access to the Hive metastore that is local to the workspace, ensuring that users can continue to interact with their existing data without interruption.

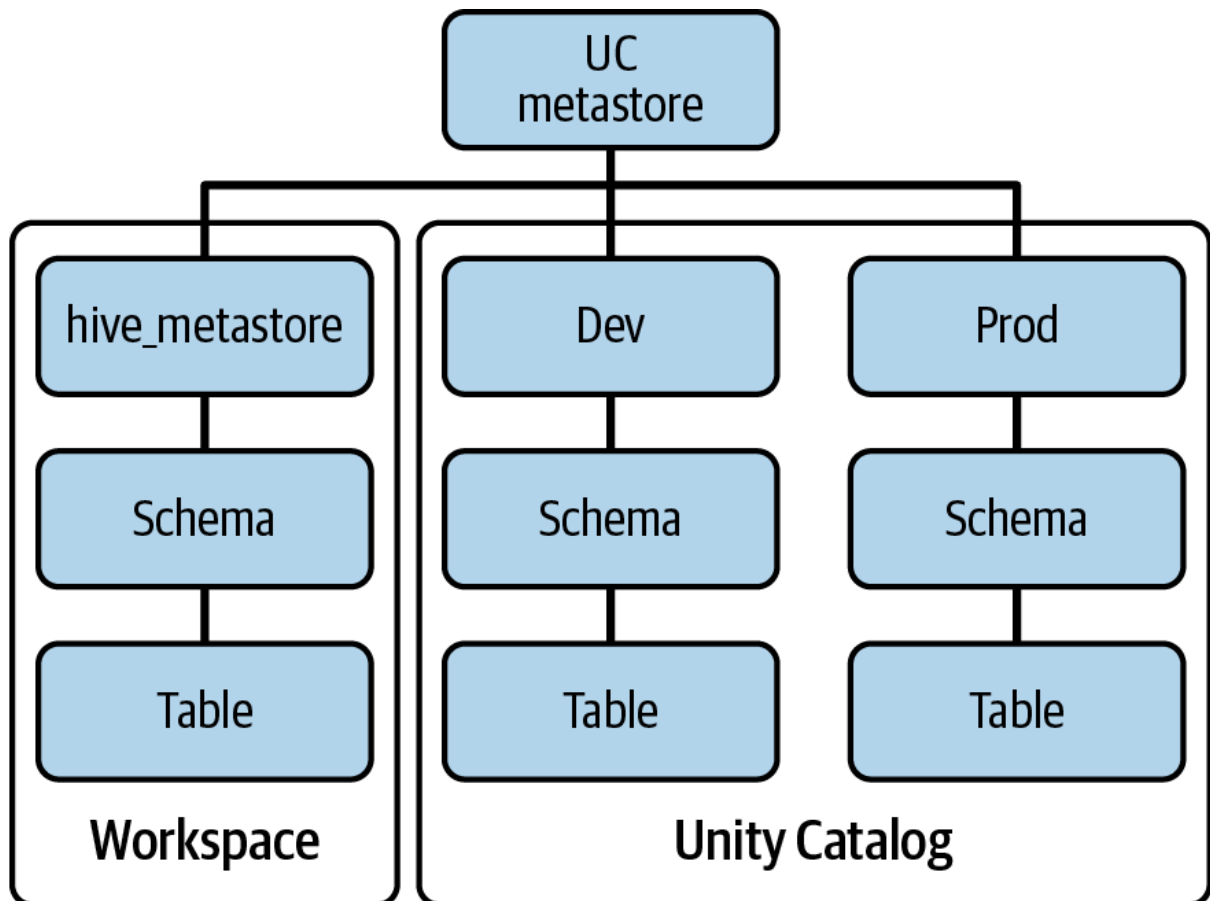


Figure 8-23. Coexistence of Unity Catalog and the legacy Hive metastore

This additive nature of Unity Catalog ensures a smooth transition to advanced data governance, and full compatibility with existing workflows and data assets.

Unity Catalog Features

In addition to its centralized and secure governance model, Unity Catalog offers the following features designed to enhance data management, accessibility, and traceability:

Automated lineage

This feature tracks and visualizes the origin and usage of data assets across notebooks, workflows, queries, and dashboards, providing transparency and traceability.

Data search and discovery

Unity Catalog offers capabilities for tagging and documenting data assets along with a powerful search interface, making it easier for users to find data.

System tables

Unity Catalog provides access to your account's operational data through system tables, including audit logs, billable usage, and lineage

information. This facilitates better monitoring and management of your data lakehouse.

Together, these features empower organizations to manage their data assets more efficiently, ensuring compliance and facilitating deeper insights.

Unity Catalog in Action

To leverage Unity Catalog, Databricks workspaces need to be properly configured and enabled. This process involves attaching the workspaces to a Unity Catalog metastore, which serves as a top-level container for all Unity Catalog metadata per cloud region.

Enabling workspaces for Unity Catalog

Databricks now automatically enables new workspaces for Unity Catalog. These workspaces are linked to a Unity Catalog metastore that is automatically provisioned in the same region. This simplifies the process for users and ensures that new workspaces are ready to take advantage of Unity Catalog's features right from the start.

Verifying Unity Catalog enablement

To verify whether your Databricks workspace is enabled for Unity Catalog, simply review the list of catalogs available in the Catalog Explorer. A UC-enabled workspace will display at least two additional catalogs besides the legacy `hive_metastore` and `samples` catalogs, as displayed in [Figure 8-24](#).

Catalog



● Demo Warehouse **Serverless** 2XS

Type to filter



- > demo_workspace
- > hive_metastore
- > samples
- > system

Figure 8-24. UC catalogs in the Catalog Explorer

These new catalogs include the following:

Main or workspace-named catalog

This is a local workspace catalog. Depending on whether Unity Catalog was enabled manually or automatically, this catalog might be named `main` or reflect the name of your workspace.

System catalog

This catalog hosts system tables that provide historical observability across your account, facilitating, among other things, operational insights and audit logging.

If you can see these catalogs, your workspace is all set up to utilize Unity Catalog's features. If not, you will need to manually enable your workspace for Unity Catalog.

Manual enabling of Unity Catalog

If your Databricks workspace was not automatically enabled for Unity Catalog, you can enable it by manually attaching it to a Unity Catalog metastore within the same region. If a Unity Catalog metastore is not already available in that region, you must create one through the Databricks account console.

Accessing account console

To manage your metastores and perform other administrative tasks in Unity Catalog, you need to access the Databricks account console as an account administrator. Depending on your cloud provider, you'll access the console via different URLs:

- AWS: <https://accounts.cloud.databricks.com>
- Azure: <https://accounts.azure.databricks.net>
- GCP: <https://accounts.gcp.databricks.com>

Upon logging in, you will be directed to the account console homepage, as displayed in [Figure 8-25](#).

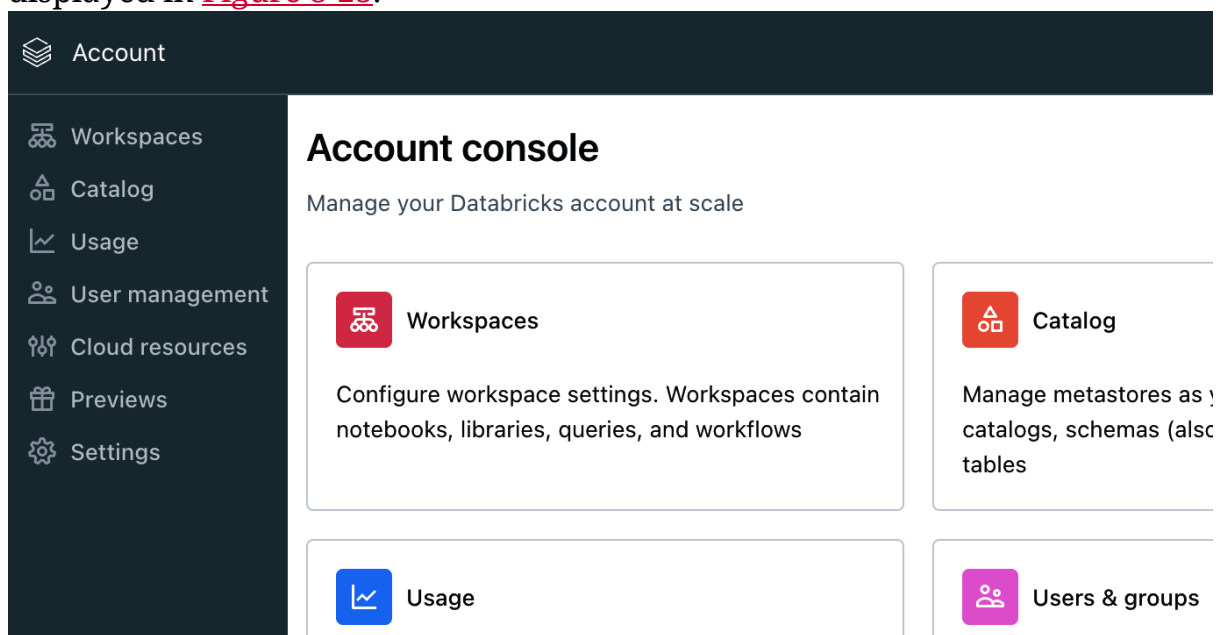


Figure 8-25. The Databricks account console

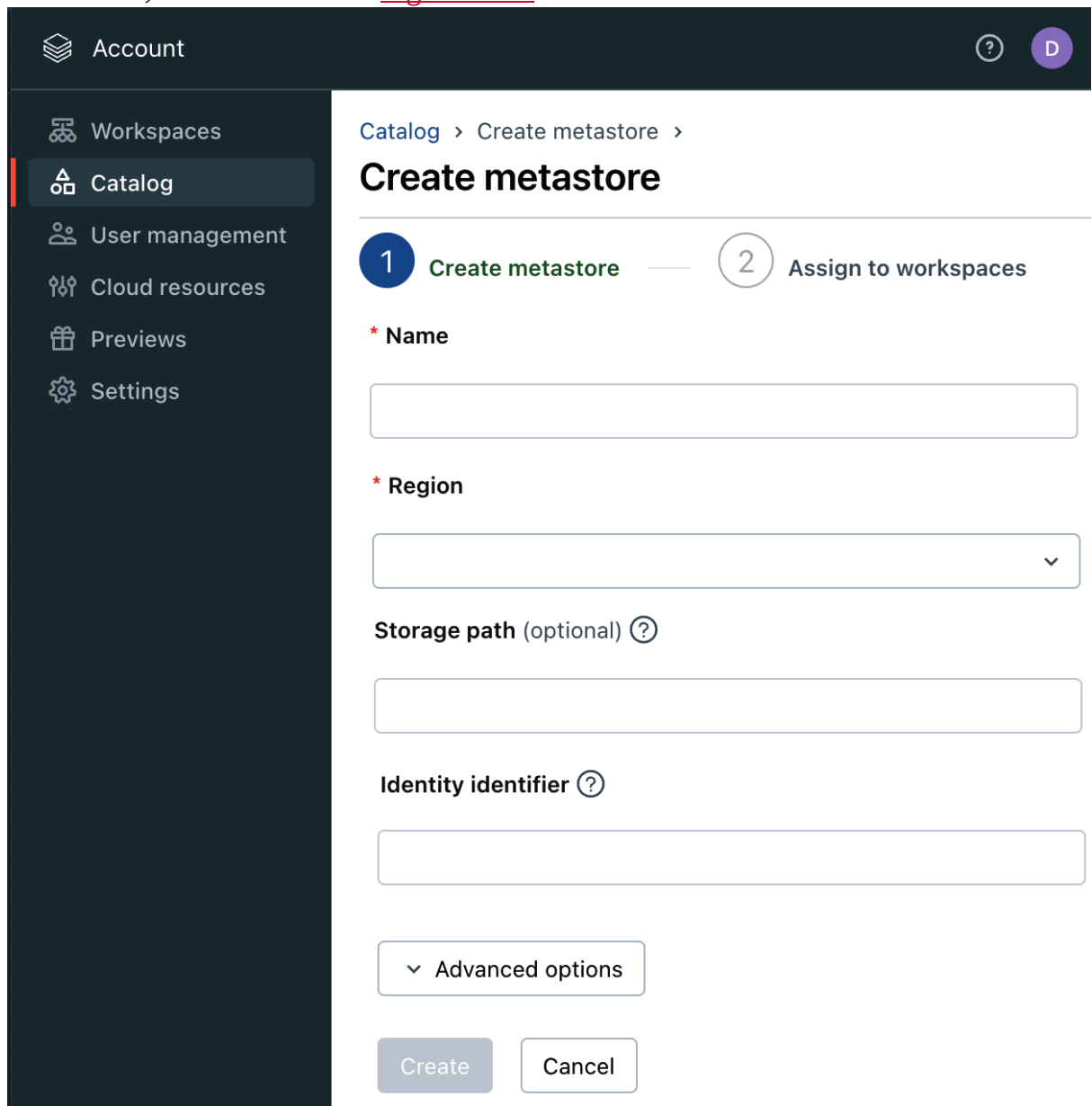
This interface provides tools for performing various administrative tasks at the account level, including managing workspaces, metastores, and identities, as well as monitoring account usage logs.

Creating a new metastore

In Databricks, each cloud region requires its own Unity Catalog metastore. This metastore can be linked to multiple workspaces within the same region, providing a unified view of the data across these workspaces. Data from other metastores can be accessed using Delta Sharing if needed to enable cross-regional federation.

To create a new metastore within a Databricks region, follow these steps:

1. Navigate to the account console and click Catalog from the left sidebar.
2. Click “Create metastore” to begin the setup process for your new metastore, as illustrated in [Figure 8-26](#).



The screenshot shows the 'Create metastore' configuration panel in the Unity Catalog account console. The left sidebar contains navigation links: Account, Workspaces, Catalog (selected), User management, Cloud resources, Previews, and Settings. The main panel has a breadcrumb 'Catalog > Create metastore >' and a title 'Create metastore'. Below the title are two steps: '1 Create metastore' (active) and '2 Assign to workspaces'. The form includes fields for 'Name' (required), 'Region' (required, dropdown), 'Storage path (optional)' (with a help icon), and 'Identity identifier' (with a help icon). There is an 'Advanced options' dropdown and 'Create' and 'Cancel' buttons at the bottom.

Figure 8-26. The configuration panel for creating a new metastore in Unity Catalog

3. Enter metastore details.
 1. Name: Provide a unique name for the new metastore.
 2. Region: Select the region where the metastore will be deployed. Ensure this is the same region as the workspaces that will access the data. Remember, only one metastore can be created per region.
 3. Root storage path (optional): Specify the path to the storage container or bucket that will serve as the default location for storing managed tables. If this is not provided, a storage

path must be specified at the catalog level each time a new catalog is created in the metastore.

4. Identity identifier: If a storage location is specified, provide the identifier of an identity that has the appropriate access permissions for that location. This information varies depending on your cloud provider:
 1. — AWS: Use an identity and access management (IAM) role.
 2. — Azure: Use an access connector for Azure Databricks resource.
 3. — GCP: A service account is automatically created for you. However, you must manually grant it access permissions to the specified storage location later.

A full discussion of all this information is beyond the scope of the certification exam. For detailed instructions, refer to the respective Databricks documentation for your cloud provider.

4. Click Create to finalize the creation of the metastore.

In the next step, select the workspaces you want to link to the new metastore, as displayed in [Figure 8-27](#). Remember, you can only select workspaces in the same region as the metastore.

[Catalog](#) > [Create metastore](#) >

Create metastore

1 Create metastore — 2 Assign to workspaces

Assign us_east1_demo_metastore to workspaces

<input checked="" type="checkbox"/> Name	Status	Pricing tier	Region
<input checked="" type="checkbox"/> Prod-Workspace	Running	Premium	us-east1
<input checked="" type="checkbox"/> Dev-Workspace	Running	Premium	us-east1
<input type="checkbox"/> Demo-Workspace	Running	Premium	europe-west2

2 workspaces selected

Figure 8-27. Metastore assignment to workspaces

Lastly, click Assign to activate Unity Catalog for the selected workspaces, or click Skip to proceed without linking any workspaces at this time. If you choose to skip, you can link workspaces to this metastore later.

Assigning existing metastore

Once a metastore has been created in Unity Catalog, you can assign it to workspaces at any time. Follow these steps to complete the assignment:

1. Navigate to the Databricks account console.
2. Click Catalog from the left sidebar.
3. Select the metastore you want to assign to workspaces.
4. Click the Workspaces tab within the metastore view, as illustrated in [Figure 8-28](#).

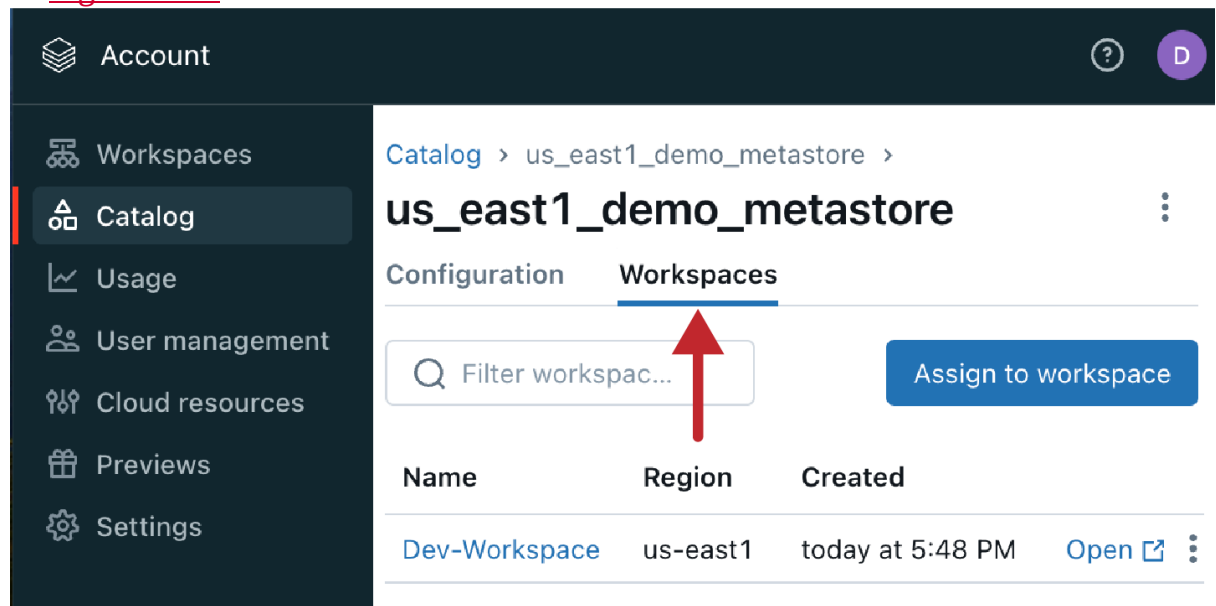


Figure 8-28. Exploring the workspaces of an existing metastore

5. Click “Assign to workspace.”
6. Choose one or more workspaces in the same region in which the metastore is provisioned, as shown in [Figure 8-29](#).

Assign us_east1_demo_metastore to workspaces



<input checked="" type="checkbox"/> Name	Status	Pricing tier	Region	Created
<input checked="" type="checkbox"/> Prod-Workspace	Running	Premium	us-east1	today at 5:49 PM
<input checked="" type="checkbox"/> Dev-Workspace	Running	Premium	us-east1	today at 5:48 PM
<input type="checkbox"/> Demo-Workspace	Running	Premium	eu-west-2 ⓘ	yesterday at 4:03 PM

1 workspace selected

Figure 8-29. Assigning the metastore to additional workspaces

7. Scroll to the bottom of the dialog and click Assign.
8. On the confirmation dialog, click Enable.

After the assignment is complete, the newly assigned workspace(s) will appear in the metastore's Workspaces tab, ensuring they now have access to the data and governance policies defined in that metastore.

Running Unity Catalog workloads

Running workloads in Unity Catalog requires that your compute resources meet specific security and compliance requirements. Notably, clusters that were created prior to enabling Unity Catalog in your workspace do not meet these security standards. As a result, these pre-existing clusters cannot be used to access data or other objects managed by Unity Catalog.

Creating a UC-compliant cluster

To run Unity Catalog workloads, it is essential to create a new cluster after enabling your workspace for Unity Catalog. When configuring a new cluster, the compliance with Unity Catalog can be verified through the summary card in the cluster configuration, as illustrated in [Figure 8-30](#).

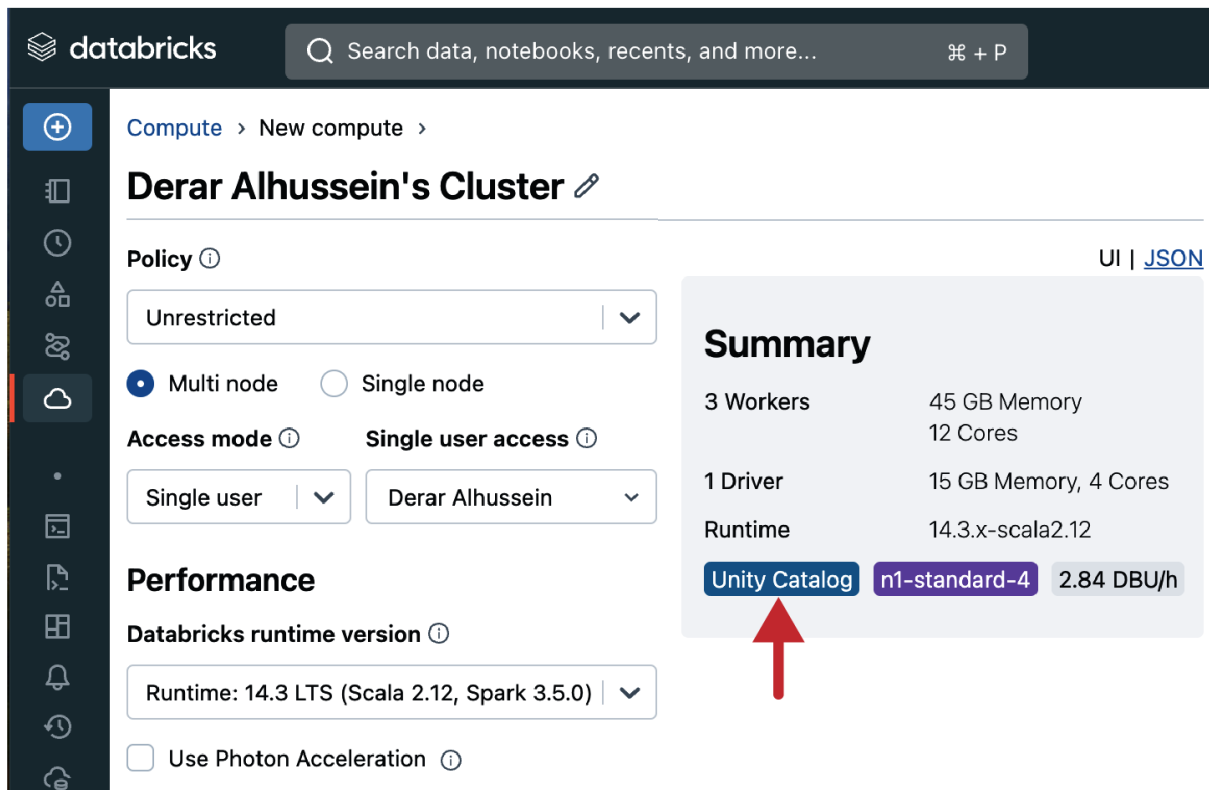


Figure 8-30. Creating a Unity Catalog-compliant cluster

Alternatively, you can use an SQL warehouse to run Unity Catalog workloads, as SQL warehouses are inherently compliant with Unity Catalog requirements.

Managing data catalogs

In Unity Catalog, managing data catalogs is an essential aspect that allows you to organize and secure your data across workspaces. You can create a new catalog in a metastore from any workspace linked to that metastore.

To follow along with this section, you will need to use Databricks SQL in a workspace enabled for Unity Catalog. Begin by navigating to the SQL editor located in the left sidebar of your Databricks workspace. Before executing any commands, ensure that your SQL warehouse is up and running.

Creating a new catalog

To create a new catalog, use the `CREATE CATALOG` command in Spark SQL. For example, to create a catalog named `hr_catalog`, execute the following command:

```
CREATE CATALOG IF NOT EXISTS hr_catalog;
```

This command will successfully create the catalog if your metastore is already configured with a root storage location. Otherwise, you will encounter the following error:

```
ERROR: Metastore storage root URL does not exist.  
Please provide a storage location for the catalog (for example 'CREATE  
CATALOG myCatalog MANAGED LOCATION '<location-path>').  
Alternatively set up a metastore root storage location to provide a  
storage location for all catalogs in the metastore.
```

This error indicates that a storage location is required for the managed tables in the catalog. To resolve this error, you have two options:

Set up a metastore-level storage location

Update the metastore configuration to define a root storage location for the entire metastore. This location will serve as the default storage path for all catalogs within the metastore.

Provide a catalog-level storage location

When creating the catalog, specify a default storage location by using the `MANAGED LOCATION` clause, as demonstrated in the following command:

```
CREATE CATALOG IF NOT EXISTS hr_catalog  
MANAGED LOCATION '<location-path>;
```

Each of these options requires pre-configuring the storage location path by creating an external location object and its associated storage credential object. This setup process is detailed in the Databricks documentation for your cloud provider and is beyond the scope of the certification exam.

Verifying the created catalog

After successfully executing the `CREATE CATALOG` command, you can verify the existence and proper creation of the catalog through the Catalog Explorer, as displayed in [Figure 8-31](#).

Catalog



● Demo Warehouse **Serverless** 2XS

Type to filter



- > demo_workspace
- > hive_metastore
- ✓ hr_catalog
 - > default
 - > information_schema
- > samples
- > system

Figure 8-31. List of UC catalogs in the Catalog Explorer after creating `hr_catalog`

As shown in the figure, each newly created catalog contains a default database named `default` and a system database named `information_schema`.

This `information_schema` database contains a set of views that reference system tables in the `system` catalog, offering catalog-level historical observability. Alternatively, you can run the following command to show all catalogs in the metastore:

```
SHOW CATALOGS;
```

[Figure 8-32](#) illustrates the output of this command, which lists all catalogs in the metastore, including our newly created catalog `hr_catalog`.

	A^B_C catalog
1	demo_workspace
2	hive_metastore
3	hr_catalog
4	samples
5	system

Figure 8-32. The output of the `SHOW CATALOGS` command after creating `hr_catalog`

What makes Unity Catalog particularly noteworthy is its ability to streamline access across various workspaces. Specifically, any workspace that is linked to this metastore will now have access to the `hr_catalog`. This unified access allows regional teams to seamlessly query and analyze the same datasets from a centralized repository whenever they have the appropriate permissions.

Granting permissions

Let's now grant permissions for creating schemas, creating tables, and using the catalog to all users on the account. These permissions can be combined in a single `GRANT` command, as shown here:

```
GRANT CREATE SCHEMA, CREATE TABLE, USE CATALOG ON CATALOG hr_catalog
TO `account` users`;
```

This command grants the specified permissions to the group `account`users``, which typically includes all users associated with the Databricks account. Similarly, you can assign permissions to other account-level groups or individual users.

After granting permissions, it is important to verify that the correct privileges have been assigned. This can be done using the `SHOW GRANT` statement, which

displays the permissions associated with a specific catalog, schema, or table. To check the grants on the `hr_catalog`, you can run the following command:

```
SHOW GRANT ON CATALOG hr_catalog;
```

This command produces the list of grants on our catalog, confirming that all account users have the granted privileges, as illustrated in [Figure 8-33](#).

A_C^B Principal	A_C^B ActionType	A_C^B ObjectType	A_C^B ObjectKey
account users	CREATE SCHEMA	CATALOG	hr_catalog
account users	CREATE TABLE	CATALOG	hr_catalog
account users	USE CATALOG	CATALOG	hr_catalog

Figure 8-33. The output of the `SHOW GRANT` command on the `hr_catalog`

Creating schemas

In addition to the `default` database in a catalog, we can create new databases using the `CREATE SCHEMA` command. To create a schema named `hr_db` within the `hr_catalog`, we execute the following command:

```
CREATE SCHEMA IF NOT EXISTS hr_catalog.hr_db;
```

After creating a new schema, we can verify its existence and ensure it has been properly added to the catalog using the Catalog Explorer. Alternatively, we can use the `SHOW SCHEMAS` command, which lists all schemas within a specified catalog. To list all schemas in the `hr_catalog`, we run the following command:

```
SHOW SCHEMAS IN hr_catalog;
```

[Figure 8-34](#) illustrates the results of executing this command. The output includes a list of all schemas within the `hr_catalog`, including the newly created schema `hr_db`.

	A_C^B databaseName
1	default
2	hr_db
3	information_schema

Figure 8-34. The output of the `SHOW SCHEMAS` command in the `hr_catalog`

Managing Delta tables

Unity Catalog integrates seamlessly with Delta Lake, enabling the creation and management of Delta Lake tables. To create a Delta Lake table in Unity Catalog, you use the `CREATE TABLE` command with the three-level namespace in the format `<catalog>.<schema>.<table>`. This approach ensures that the table is appropriately placed within the catalog and schema hierarchy.

For instance, to create a table named `jobs` in the `hr_db` schema of the `hr_catalog`, you can execute the following command:

```
CREATE TABLE IF NOT EXISTS hr_catalog.hr_db.jobs
(id INT, title STRING, min_salary DOUBLE, max_salary DOUBLE);
```

Once the table is created, data can be inserted using the `INSERT INTO` command, as in the following example:

```
INSERT INTO hr_catalog.hr_db.jobs
VALUES (1, "Software Engineer", 3000, 5000),
       (2, "Data Engineer", 3500, 5500),
       (3, "Web Developer", 2800, 4800);
```

Afterward, we can view advanced metadata information about the table using the `DESCRIBE TABLE EXTENDED` command:

```
DESCRIBE EXTENDED hr_catalog.hr_db.jobs;
```

The output of this command reveals that our table is a managed table created in the root storage of the metastore, as illustrated in [Figure 8-35](#).

A ^B col_name	A ^B data_type
Created by	spark
Type	MANAGED
Location	gs://metastores_root_storage/4330f82e-88f9-4bb9-a5ad-bcec594fdece/tables/32985eca-ee25-4753-8e20-19e29a010a74
Provider	delta
Owner	derar@oreilly.com

Figure 8-35. The output of the `DESCRIBE EXTENDED` command on the `jobs` table

Dropping tables

To drop a managed table, the `DROP TABLE` command is used, which effectively removes the table from the catalog. For example, to drop our `jobs` table, we run the following command:

```
DROP TABLE hr_catalog.hr_db.jobs;
```

Unlike traditional Hive metastore behavior, dropping a managed table in Unity Catalog does not immediately delete the table's directory from the underlying storage. Instead, Unity Catalog retains the data files for a period of 30 days before permanently deleting them. If you examine the root storage of the metastore in your cloud account, you'll find that the table directory and its data files remain present after dropping the table, as shown in [Figure 8-36](#).



Buckets > metastores_root_storage > 4330f82e-88f9-4bb9-a5ad-bcec594fdece > tables > 32985eca-ee25-4753-8e20-19e29a010a74						
UPLOAD FILES UPLOAD FOLDER CREATE FOLDER TRANSFER DATA MANAGE HOLDS EDIT RETENTION DOWNLOAD						
Filter by name prefix only ▾ Filter Filter objects and folders						
<input type="checkbox"/>	Name	Size	Type	Created ?	Storage class	Last modified
<input type="checkbox"/>	 _delta_log/	—	Folder	—	—	—
<input type="checkbox"/>	 part-00000-f79bbd73-7b66-43f0-b...	1.1 KB	application/octet-stream	Aug 5, 2024, 2:19:34 PM	Standard	Aug 5, 202

Figure 8-36. The content of the `jobs` table directory, in a GCP bucket, after executing the `DROP TABLE` command

During a seven-day period after deletion, it is possible to recover a recently dropped table using the `UNDROP TABLE` command. This feature is particularly useful in scenarios where a table is accidentally deleted or needs to be restored for further use, enhancing data safety and management. To recover the `jobs` table, we run the following command:

```
UNDROP TABLE hr_catalog.hr_db.jobs;
```

After executing this command, you can query the table to confirm that it has been successfully recovered. For example, use the following command to verify the contents of the recovered table:

```
SELECT * FROM hr_catalog.hr_db.jobs;
```

This query should return the data that was present in the table before it was dropped, confirming that the recovery process was successful.

Conclusion

In conclusion, implementing data governance within Databricks through Unity Catalog represents a significant advancement in managing and securing data assets. Unity Catalog offers a comprehensive suite of governance features that streamline data management, enhance security, and ensure regulatory compliance. For organizations still using the legacy Hive metastore, Databricks strongly recommends transitioning to Unity Catalog to fully leverage these advanced capabilities. Adopting Unity Catalog will significantly enhance your organization's ability to govern and secure data effectively, better positioning you to meet both regulatory requirements and operational demands.

Sample Exam Questions

Conceptual Question

1. Which of the following represents the hierarchy of relational entities in Unity Catalog?

1. Metastore → Catalog → Table → Schema (Database)
2. Schema (Database) → Metastore → Catalog → Table
3. Metastore → Catalog → Schema (Database) → Table
4. Catalog → Metastore → Schema (Database) → Table
5. Schema (Database) → Catalog → Table → Metastore

Code-Based Question

2. A data engineer uses the following SQL query:

```
GRANT MODIFY ON TABLE inventory TO supply_team
```

Which of the following describes the ability granted by the `MODIFY` privilege?

1. It gives the ability to add data to the table.
2. It gives the ability to delete data from the table.
3. It gives the ability to modify data in the table.
4. All of the above abilities are granted by the `MODIFY` privilege.
5. None of these options correctly describes the ability granted by the `MODIFY` privilege.

The correct answers to these questions are listed in [Appendix C](#).