Unity Catalog and steps to create

Submitted By-

E.R Harish, DE-1

- ## What is Unity Catalog?

Unity Catalog is a unified data governance solution provided by **Databricks** to manage and secure data across the entire Databricks Lakehouse Platform. It is designed to streamline the management of data assets, improve collaboration, and ensure data security and compliance in multi-cloud and hybrid-cloud environments.

Key Features:

1. **Centralized Metadata Management**:
    - Provides a single interface to manage metadata and data lineage across all Databricks workspaces and cloud environments.
    - Organizes data into hierarchical namespaces: **Catalogs**, **Schemas**, and **Tables**.
2. **Fine-Grained Access Control**:
    - Implements row- and column-level permissions using ANSI SQL syntax, enabling precise control over data access.
    - Supports integration with existing identity providers (e.g., Azure Active Directory, AWS IAM).
3. **Data Lineage**:
    - Tracks the complete lineage of data from source to output, helping to understand dependencies and impact analysis.
4. **Audit Logging**:
    - Logs all access and modifications to ensure compliance with regulatory and security requirements.
5. **Collaboration**:
    - Allows multiple teams to securely access and share data across different workspaces.
6. **Multi-Cloud Support**:
    - Works seamlessly across multiple cloud providers like AWS, Azure, and GCP.
7. **Seamless Integration**:

  - Integrated with Databricks SQL, Delta Lake, and machine learning workflows.

Benefits:

- **Enhanced Security**: Centralized governance minimizes the risk of unauthorized data access.
- **Improved Compliance**: Simplifies auditing and ensures adherence to data regulations.
- **Increased Productivity**: Unified management makes data discovery and usage more efficient for teams.
- **Scalability**: Manages large-scale data operations across clouds effectively.

Unity Catalog is particularly useful for organizations adopting the **Lakehouse architecture**, where combining governance with analytics and AI on a unified platform is critical.

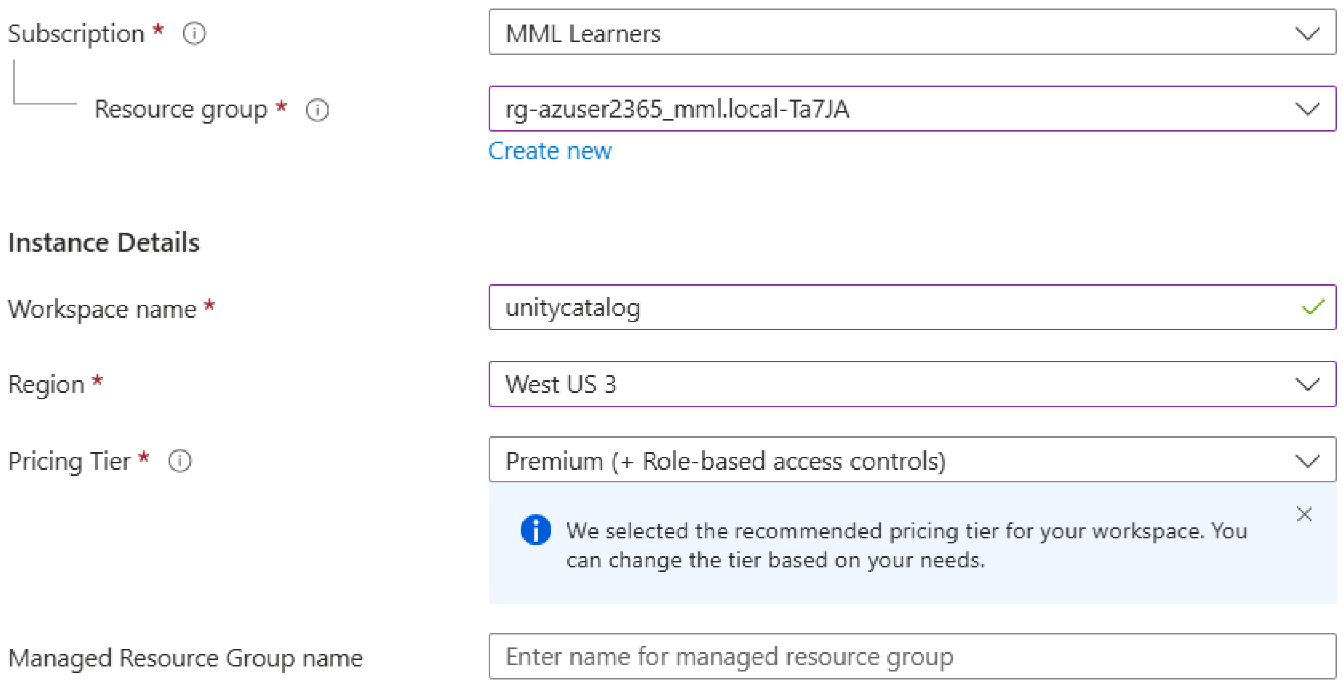
• Steps to create a Unity Catalog?

Requirements:

To create a metastore: You must be an Azure Databricks account admin**.**

Step 1:

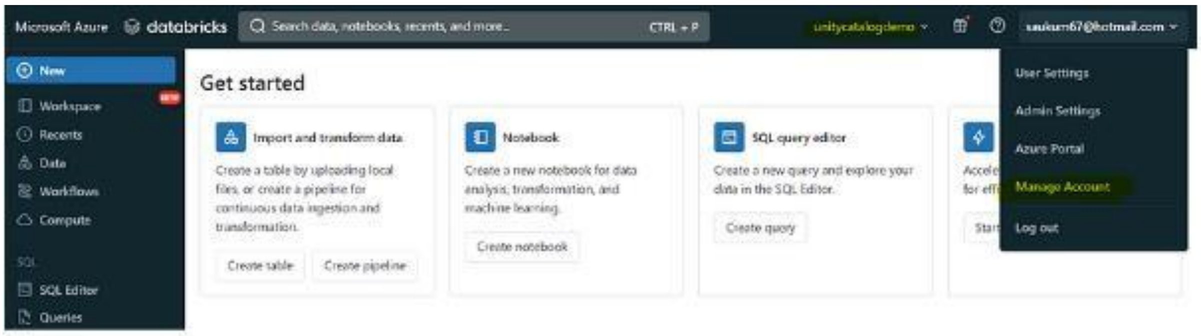
Create an Azure Databricks workspace with Premium pricing tier.



Subscription *	MML Learners
Resource group *	rg-azuser2365_mml.local-Ta7JA
	Create new
Instance Details	
Workspace name *	unitycatalog
Region *	West US 3
Pricing Tier *	Premium (+ Role-based access controls)
	We selected the recommended pricing tier for your workspace. You can change the tier based on your needs.
Managed Resource Group name	Enter name for managed resource group

Step 2:

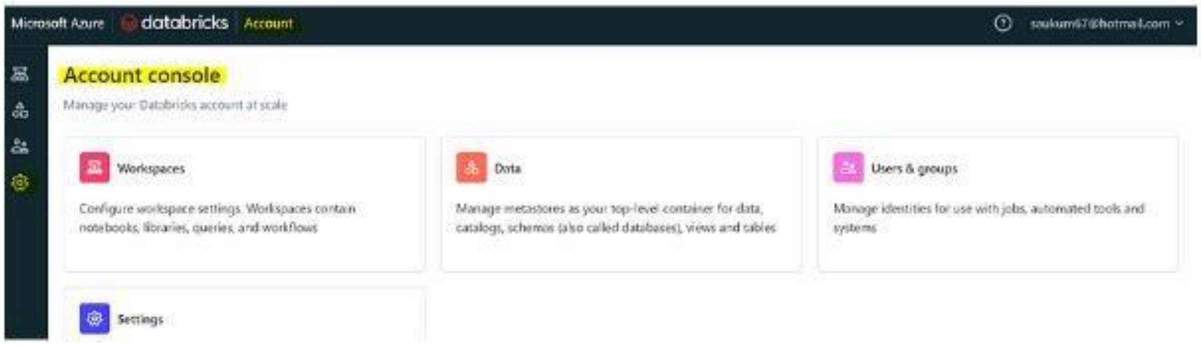
Click on Manage Account and login into Account console.



Create a table by uploading local files, or create a pipeline for continuous data ingestion and transformation.

Create a new notebook for data analysis, transformation, and machine learning.

Create a new query and explore your data in the SQL Editor.



Manage your Databricks account at scale

Configure workspace settings. Workspaces contain notebooks, libraries, queries, and workflows

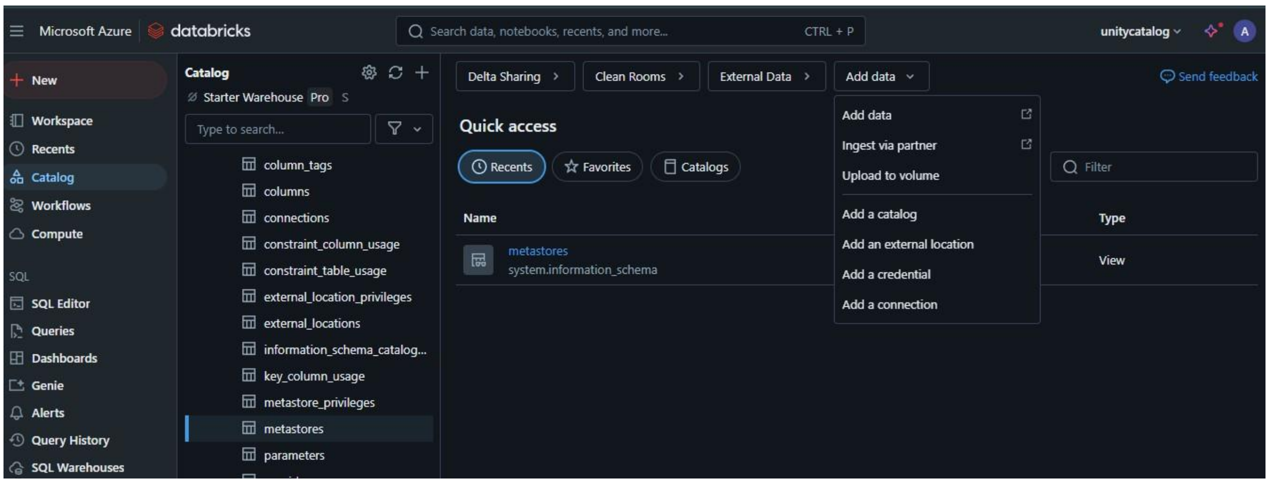
Manage metastores as your top-level container for data, catalogs, schemas (also called databases), views and tables

Manage identities for use with jobs, automated tools and systems

Step 3:

To create a catalog in Azure Databricks:

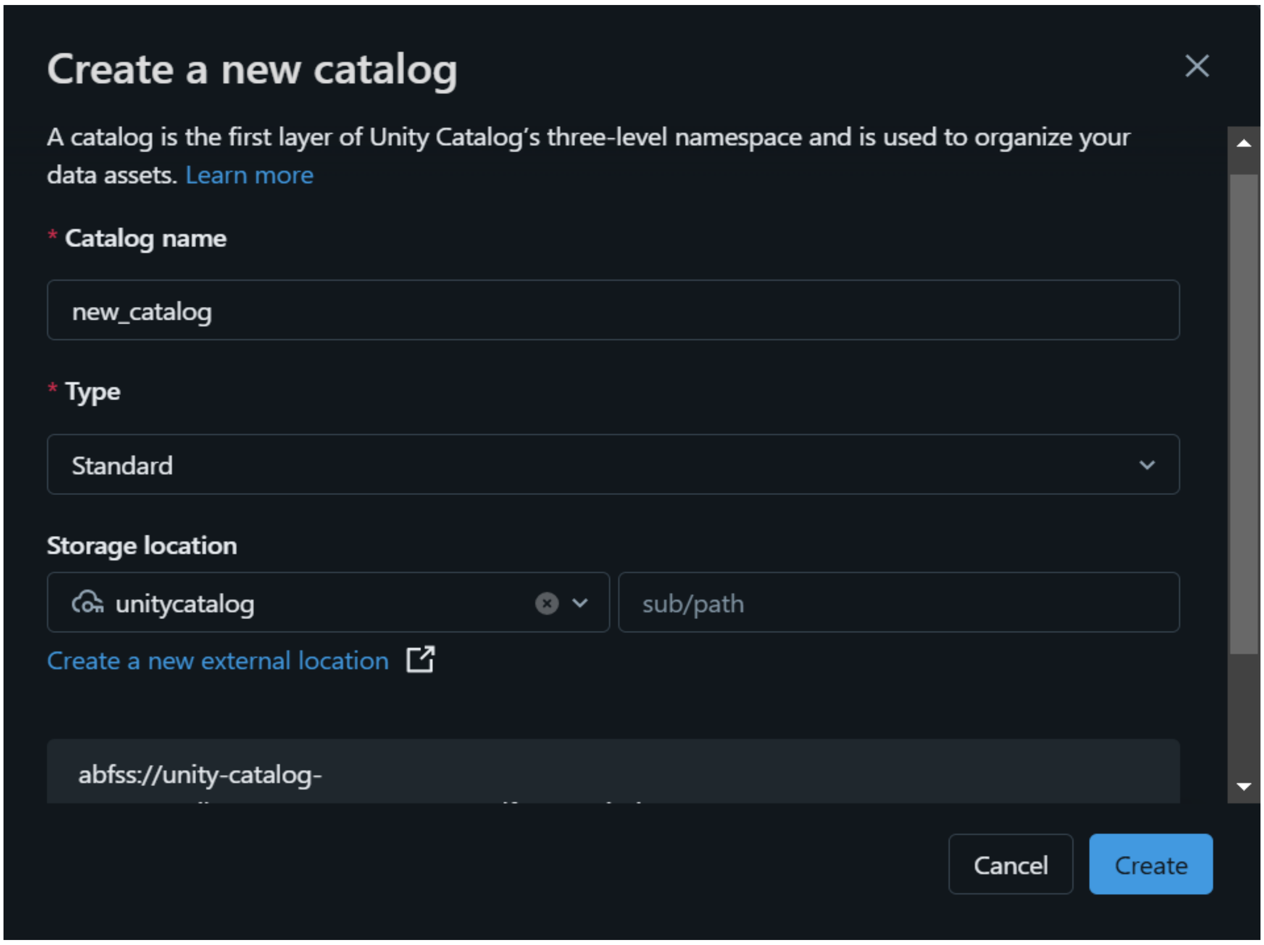
- **Open the workspace.**
- **Go to the Catalog tab from the left-hand pane.**
- **Click on Add Data (+) or Add a Catalog.**
- **Select the catalog type from the available options.**



Step 4:

Configure the catalog to create it:

- Provide a unique Catalog Name.
- Select the Type of catalog (Standard, Foreign, or Shared).
- Specify the Storage Location to define where data assets will be stored.
- Review the settings and click Save or Create to complete the catalog configuration.



Create a new catalog

A catalog is the first layer of Unity Catalog's three-level namespace and is used to organize your data assets. Learn more

* Catalog name

new_catalog

* Type

Standard

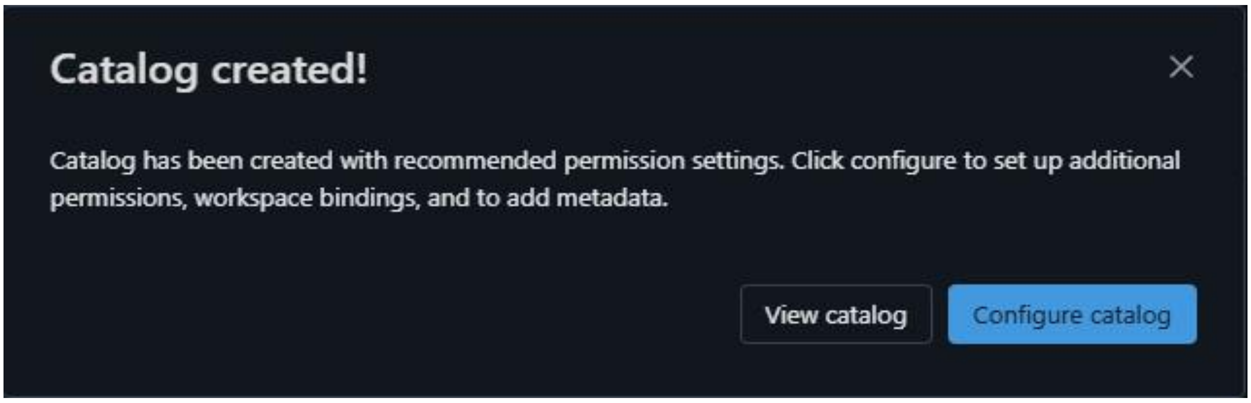
Storage location

unitycatalog

sub/path

Create a new external location

abfss://unity-catalog-



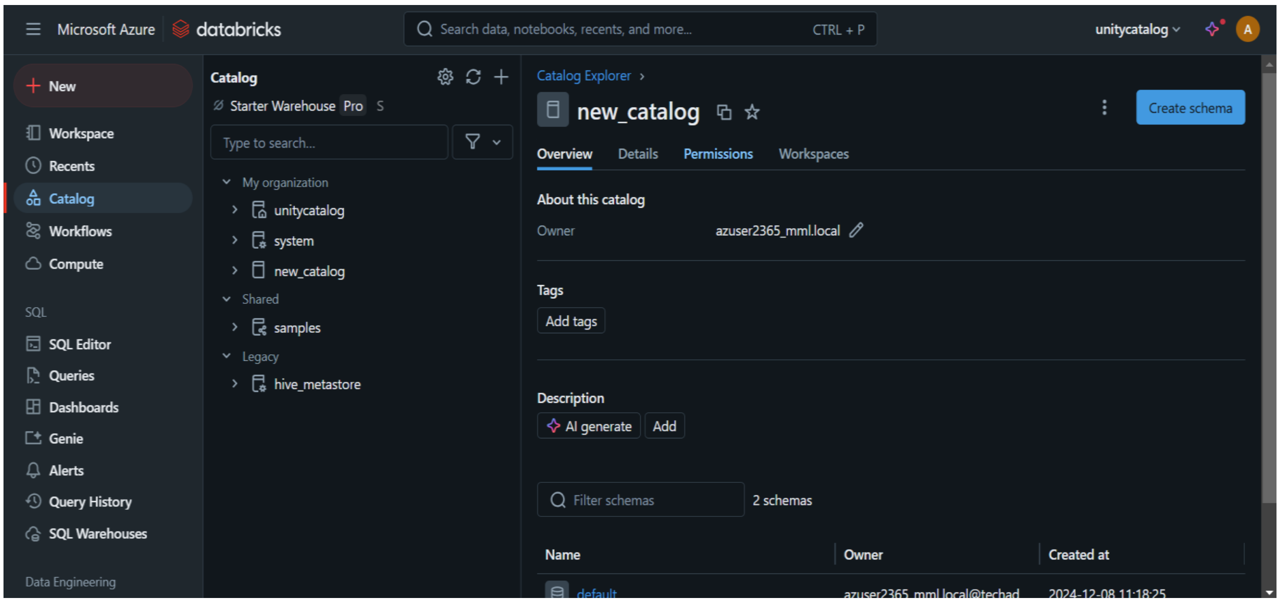
Catalog created!

Catalog has been created with recommended permission settings. Click configure to set up additional permissions, workspace bindings, and to add metadata.

Step 5:

View and configure the catalog:

- Navigate to the newly created catalog and click on **View Catalog** to review its structure and details.



new_catalog

About this catalog

Owner azuser2365_mml.local

Tags

Add tags

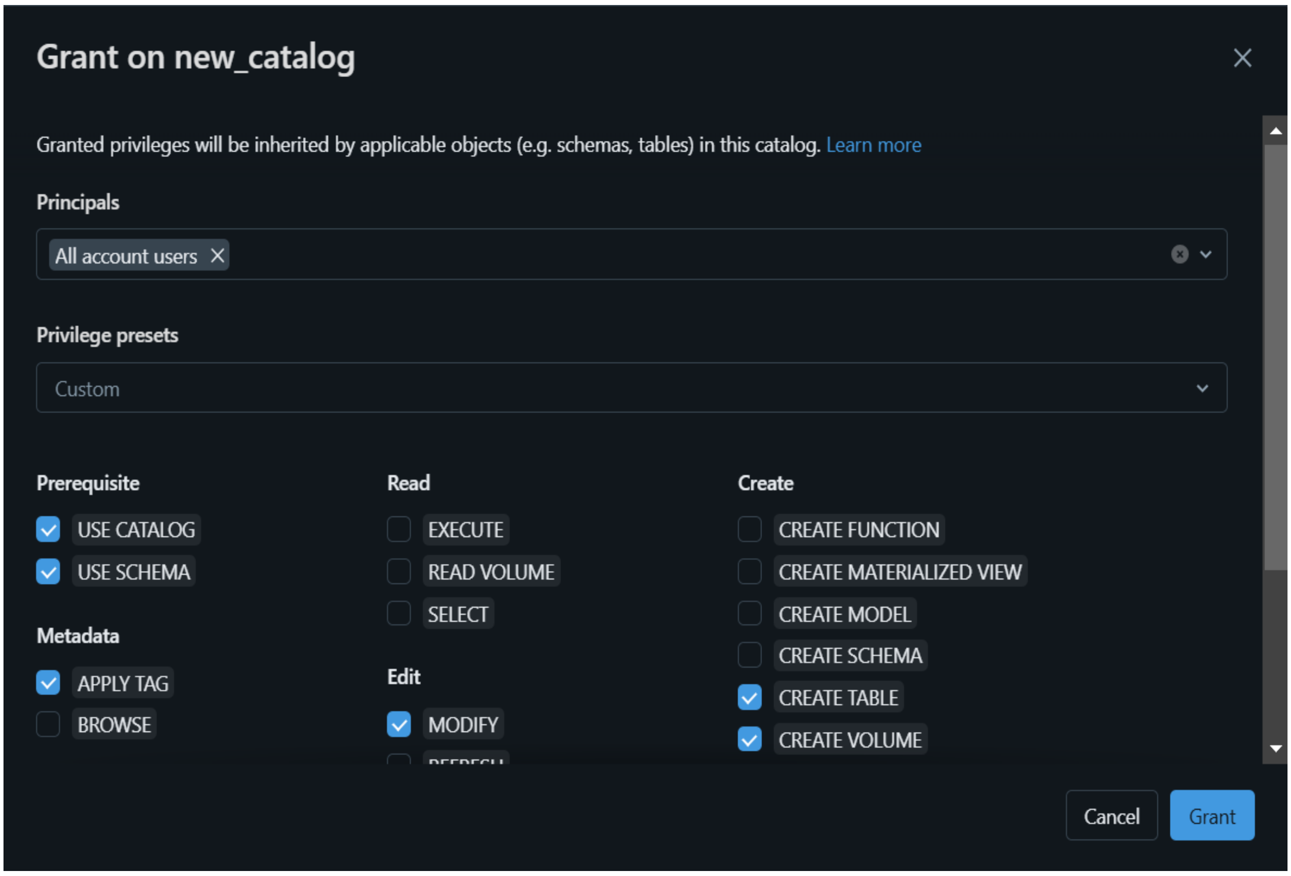
Description

AI generate Add

Filter schemas 2 schemas

Name	Owner	Created at
default	azuser2365_mml.local@techad	2024-12-08 11:18:25

- To configure permissions and metadata:
- We can also **grant various permissions** to any **Groups/Users.**
  - Select **Configure Catalog**.
  - Assign appropriate **Permissions** to users, groups, or roles.



Grant on new_catalog

Granted privileges will be inherited by applicable objects (e.g. schemas, tables) in this catalog. Learn more

Principals

All account users

Privilege presets

Custom

Prerequisite

- [x] USE CATALOG
- [x] USE SCHEMA

Metadata

- [x] APPLY TAG
- [ ] BROWSE

Read

- [ ] EXECUTE
- [ ] READ VOLUME
- [ ] SELECT

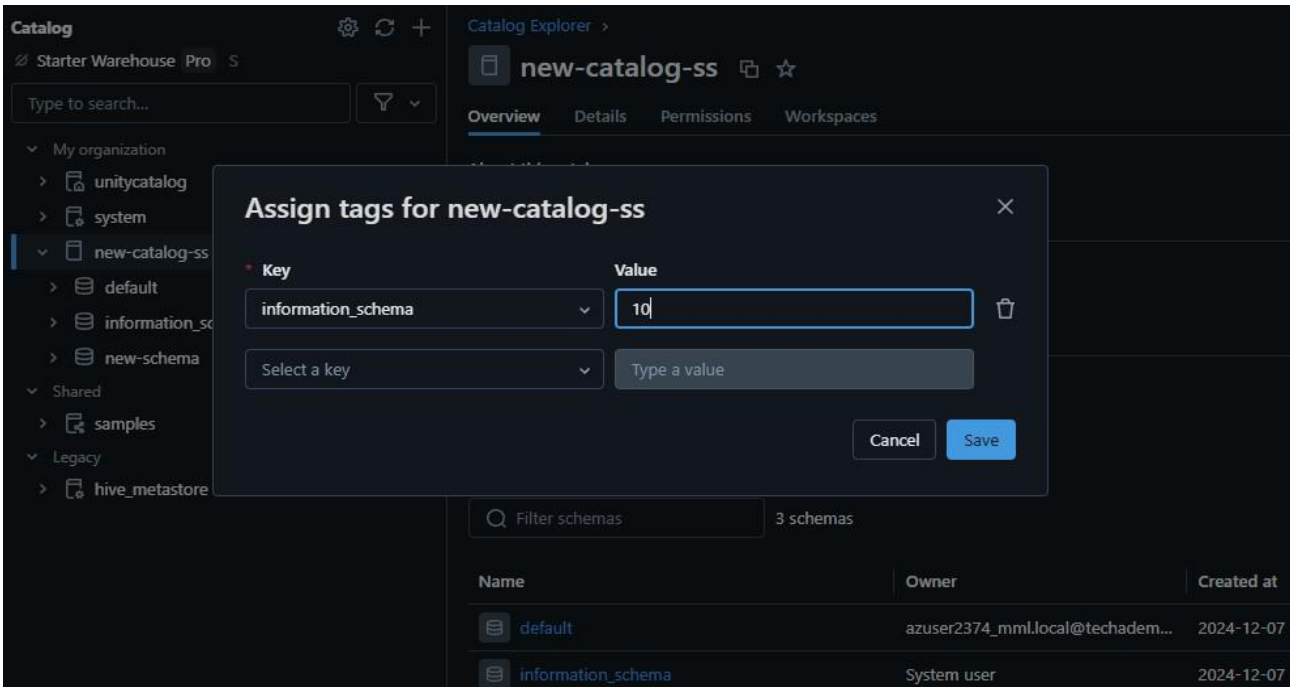
Edit

- [x] MODIFY
- [ ] REFRESH

Create

- [ ] CREATE FUNCTION
- [ ] CREATE MATERIALIZED VIEW
- [ ] CREATE MODEL
- [ ] CREATE SCHEMA
- [x] CREATE TABLE
- [x] CREATE VOLUME

- Add or modify **Keys and Values** under the information_schema database for additional metadata management.

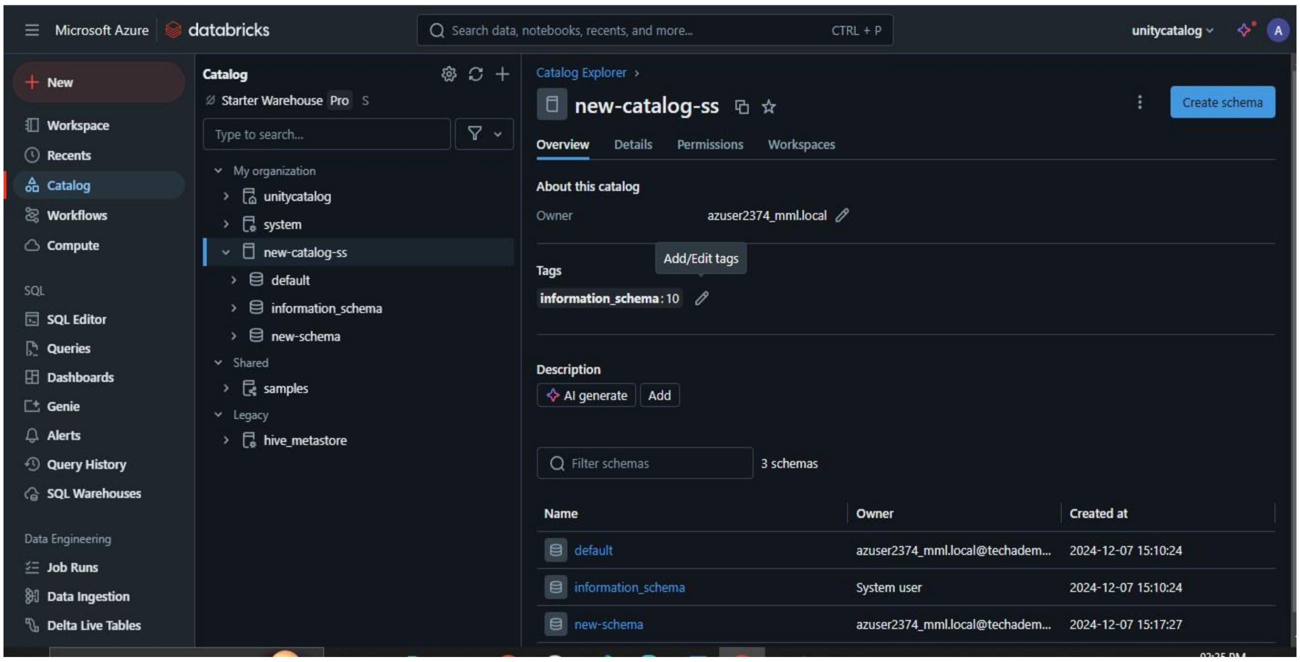


Assign tags for new-catalog-ss

Key	Value
information_schema	10

Name	Owner	Created at
default	azuser2374_mml.local@techadem...	2024-12-07
information_schema	System user	2024-12-07

- Save the changes to apply the configuration.



About this catalog

Owner azuser2374_mml.local

Tags

information_schema : 10

Description

Name	Owner	Created at
default	azuser2374_mml.local@techadem...	2024-12-07 15:10:24
information_schema	System user	2024-12-07 15:10:24
new-schema	azuser2374_mml.local@techadem...	2024-12-07 15:17:27

Click on **Create tab.**

Next steps:

- Create and manage schemas (databases)
- Create tables

- ## What Is the Difference Between Unity Catalog and Hive Metastore?

Databricks Unity Catalog and Hive Metastore are both metadata management systems, but they serve different purposes and have distinct functionalities within their respective ecosystems. Here's a table that highlights the key differences between Databricks Unity Catalog and Hive Metastore:

Databricks Unity Catalog	**Hive Metastore**
Databricks Unity Catalog is a centralized service for managing data governance and access control across workspaces in the Databricks	Hive Metastore is central repository for storing metadata about Hive databases, tables, partitions, and other objects in the Apache Hive data warehousing system
Databricks Unity Catalog supports a wide range of data sources, including Apache Spark tables, Delta Lake tables, AWS S3, Azure Blob Storage, HDFS, and more.	Hive Metastore is primarily designed for Hive tables and databases, but can also store metadata for external data sources like HDFS or cloud storage
Databricks Unity Catalog provides APIs and tools for managing and updating metadata, enabling automated metadata capture and synchronization with external metadata sources	Metadata management is primarily done through Hive commands or directly interacting with the underlying database
Databricks Unity Catalog offers fine-grained access control and data lineage tracking, allowing administrators to define and enforce policies for data access and modification	Access control is typically handled through Hadoop permissions or external tools like Apache Ranger
Databricks Unity Catalog is designed specifically for Databricks, offering seamless integration and collaboration within the platform	Hive Metastore is primarily designed for Hadoop-based environments, but can be used with other systems that support the Hive Metastore interface
Databricks Unity Catalog facilitates data sharing and collaboration by allowing users	In Hive Metastore data sharing is typically achieved through Hadoop

to grant and revoke access to data assets across different environments and teams	permissions or external tools like Apache Ranger
Databricks Unity Catalog is tightly integrated with the Databricks Unified Analytics Platform and other components of the Databricks ecosystem	Hive Metastore integrates with the Apache Hive ecosystem and can be used with other tools like Apache Spark, Apache Impala, and Apache Ranger
Databricks Unity Catalog is designed to handle large-scale data and metadata operations with high performance and scalability	In Hive meta store scalability and performance can vary depending on the underlying database and configuration
Databricks Unity Catalog provides a searchable interface for data discovery and exploration	Metadata management is typically done through Hive commands or directly interacting with the underlying database

• Step-By-Step Guide for Getting Started With Databricks Unity Catalog

Prerequisites

- Databricks workspace enabled for Unity Catalog.
- Access to compute resources (SQL warehouse or cluster) that support Unity Catalog.
- Appropriate privileges on Databricks Unity Catalog objects (USE CATALOG, USE SCHEMA, and CREATE TABLE).
- Users and groups added to the workspace.

Namespace Overview:

Now, let's talk about Unity Catalog's three-level namespace. It organizes your data into catalogs, schemas (databases), and tables or views. Think of it like a filing cabinet with drawers (catalogs), folders (schemas), and documents (tables or views).

When referring to a table, you'll use this format:

<catalog>.<schema>.<table>

If you have data in your Databricks workspace's local Hive metastore or an external Hive metastore, it becomes a catalog called **hive_metastore**, and you can access tables like this:

hive_metastore.<schema>.<table>

Step 1—Create a New Catalog

Create a new catalog using the CREATE CATALOG command with spark.sql. To create a catalog, you must be a metastore admin or have the CREATE CATALOG privilege on the metastore.

CREATE CATALOG IF NOT EXISTS <catalog>;

Databricks Unity Catalog Example

If your workspace was enabled for Databricks Unity Catalog by default, you might need to specify a managed location for the new catalog.

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

Step 2—Select and Grant Permissions on the Catalog

Once your catalog is created, select it as the current catalog and grant permissions to other users or groups as needed.

-- Set the current catalog

USE CATALOG

<catalog>;

-- Grant permissions to all users

GRANT CREATE SCHEMA, CREATE TABLE, USE CATALOG

ON CATALOG <catalog>

TO `account users`;

Step 3—Create and Manage Schemas

Next, let's create schemas (databases) to logically organize tables and views.

-- Create a new schema

CREATE SCHEMA IF NOT EXISTS <schema>

COMMENT "A new Unity Catalog schema called <schema>";

```
-- Show schemas in the selected catalog
SHOW SCHEMAS;
-- Describe a schema
DESCRIBE SCHEMA EXTENDED <schema>;
```

Step 4—Create a Managed Table

Managed tables are the default way to create tables with Unity Catalog. The table is created in the managed storage location configured for the metastore, catalog, or schema.

```
-- Set the current schema
USE <schema>;
-- Create a managed Delta table and insert records
CREATE TABLE IF NOT EXISTS <table>
  (columnA Int, columnB String) PARTITIONED BY (columnA);

INSERT INTO TABLE <table>
VALUES
  (1, "one"),
  (2, "two"),
  (3, "three"),
  (4, "four"),
  (5, "five"),
  (6, "six"),
  (7, "seven"),
  (8, "eight"),
  (9, "nine"),
  (10, "ten");

-- View all tables in the schema SHOW
TABLES IN <schema>;
```

-- Describe the table

DESCRIBE TABLE EXTENDED <table>;

Step 5—Query the Table

Access your tables using the three-level namespace:

-- Query the table using the fully qualified name SELECT

* FROM <catalog>.<schema>.<table>;

-- Set the default catalog and query using the schema and table name USE

CATALOG <catalog>;

SELECT * FROM <schema>.<table>;

-- Set the default catalog and schema, and query using the table name USE

CATALOG <catalog>;

USE <schema>;

SELECT * FROM <table>;

Step 6—Drop a Table

Drop a managed table using the DROP TABLE command. This removes the table and its underlying data files. For external tables, dropping the table removes the metadata but leaves the data files untouched.

-- Drop the managed table

DROP TABLE <catalog>.<schema>.<table>

Step 7—Manage Permissions on Data (Optional)

Lastly, use GRANT and REVOKE statements to manage access to your data. Unity Catalog is secure by default, so access isn't automatically granted. Metastore admins and data object owners can control access for users and groups.

-- Grant USE SCHEMA privilege on a schema GRANT

USE SCHEMA

ON SCHEMA <schema>

TO `account users`;

```
-- Grant SELECT privilege on a table

GRANT SELECT

ON TABLE <schema>.<table>

TO `account users`;

-- Show grants on a table

SHOW GRANTS

ON TABLE <catalog>.<schema>.<table>;

-- Revoke a privilege

REVOKE SELECT

ON TABLE <schema>.<table>

FROM `account users`;
```

Well, that's all for now! I hope this guide has given you a good starting point for exploring Databricks Unity Catalog. Don't forget to check the official documentation for more advanced topics and scenarios.

• Databricks Unity Catalog Best Practices

To maximize the benefits of Databricks Unity Catalog and ensure efficient and secure data governance, follow these best practices:

1) Catalogs as Isolation Units

Use catalogs as the primary unit of isolation in your data governance model. Separate catalogs for production vs non-production data, sensitive vs non-sensitive information, and data belonging to different organizational units or domains.

2) Storage Isolation

In addition to catalog isolation, you can further segregate data by configuring separate cloud storage locations at the catalog or schema level. This physical separation is crucial when regulatory or corporate policies mandate storage boundaries for specific data categories.

3) Workspace Boundaries

Bind catalogs to specific Databricks workspaces to restrict data access to authorized compute environments only.

4) Access Control Model

Utilize the flexible, inheritance-based model provided by Databricks Unity Catalog for granular permissions.

5) Use Group Ownership

Always designate groups, not individuals, as owners of catalogs, schemas, tables, and other objects. This enables consistent access management using your identity provider's group membership.

6) Leverage Inheritance

Grant coarse permissions at higher catalog and schema levels, allowing automatic propagation of privileges to child objects like tables and views.

7) Implement Attribute-Based Access

Use dynamic views with the is_account_member() function for advanced access scenarios, filtering data access based on a user's group membership or other attributes from the authentication provider.

8) Column and Row Security

Implement column-level and row-level data masking and filtering using dynamic views to secure sensitive data while providing higher-privileged users with full access when needed.

G) External Location Boundaries

Grant external location creation privileges only to a limited set of administrators. This prevents indiscriminate data access bypassing Unity Catalog controls while maintaining secure, audited bridges between cloud storage and Unity Catalog.

10) Cluster Security

Configure compute clusters with an appropriate access mode to integrate with Unity Catalog's security model.

11) Access Mode Policies

Implement cluster policies to enforce standardized cluster configurations, allowing only the necessary access modes based on the use case (e.g., shared access for multi- tenant workloads, single user for jobs/ML workloads).

12) Least Privilege Clusters

Create dedicated private clusters bound to your secure data catalogs via workspace- catalog bindings for highly sensitive data processing.

13) **Auditing and Monitoring**

Comprehensive auditing of data access events enables security reviews, troubleshooting of issues, and detection of abuse or data exfiltration attempts.

14) **Integrate Audit Logs**

Configure log delivery of Databricks audit logs (including Unity Catalog audit events) to your centralized security monitoring solution. Make sure to establish monitoring processes to analyze these logs for anomalies.

15) **Metadata Operations**

In addition to data access, monitor for excessive or suspicious levels of create/alter/delete operations on securables such as catalogs, schemas, and tables.

16) **Delta Sharing for Secure Collaboration**

Use Delta Sharing for secure data sharing between isolated domains or external entities, rather than direct access methods that bypass governance controls.

That's it! If you follow these best practices, you can ensure secure and efficient data governance within your organization using Databricks Unity Catalog.

• What Are the Limitations of Databricks Unity Catalog?

Databricks Unity Catalog offers comprehensive data management capabilities, but it's essential to understand its limitations to plan your implementation accordingly. Here are some key limitations:

1) **Compatibility with Older Databricks Runtimes**

Databricks Runtime versions below 11.3 LTS may not fully support Unity Catalog functionalities. Upgrade to Databricks Runtime 11.3 LTS or later for complete functionality.

2) **Row-level and Column-level Security for R Workloads**

Databricks Unity Catalog does not support dynamic views for row-level or column-level security with R workloads. Plan accordingly if your organization relies on R for data analysis.

3) **Shallow Clones Limitations**

Shallow clones for creating managed tables are supported in Databricks Runtime 13.1 and above, but not in Databricks Runtime 13.0 and below.

4) Bucketing Limitations

Databricks Unity Catalog does not support bucketing for its tables. Attempting to create a bucketed table will throw an exception.

5) Writing from Multiple Regions

Writing to the same path or Delta Lake table from workspaces in multiple regions can cause unreliable performance if some clusters access Unity Catalog and others do not. Always maintain consistency across workspaces to avoid issues.

6) Custom Partition Schemes

Unity Catalog does not support custom partition schemes created using commands like ALTER TABLE ADD PARTITION. However, you can still access tables that use directory-style partitioning.

7) Overwrite Mode for DataFrame Write Operations

Overwrite mode for DataFrame write operations is only supported for Delta tables and not other file formats. Users must have CREATE privileges on the parent schema and be the object owner or have MODIFY privileges.

8) Python UDFs Limitations

In Databricks Runtime 13.2 and above, Python scalar UDFs are supported. In Databricks Runtime 13.1 and below, Python UDFs (including UDAFs, UDTFs, and Pandas on Spark) are not supported.

G) Scala UDFs on Shared Clusters Limitations

Scala scalar UDFs are supported on shared clusters in Databricks Runtime 14.2 and above, but not in Databricks Runtime 14.1 and below.

10) Workspace-level Groups in GRANT Statements

Workspace-level groups cannot be used in Unity Catalog GRANT statements. Create groups at the account level for consistency, and update automation to reference account endpoints instead of workspace endpoints.

11) Object Name Limitations

There are several limitations regarding object names in Unity Catalog:

- Object names cannot exceed 255 characters.
- Special characters such as periods (.), spaces, forward slashes (/), ASCII control characters (00-1F hex), and the DELETE character (7F hex) are not allowed.
- Unity Catalog stores all object names in lowercase.
- Use backticks to escape names with special characters, such as hyphens (-), when referencing Unity Catalog names in SQL.

12) Column Name Limitations

Column names can use special characters, but the name must be escaped with backticks (`) in all SQL statements if special characters are used. Databricks Unity Catalog preserves column name casing, but queries against Unity Catalog tables are case-insensitive.

13) Write Privileges on External Locations

Grant write privileges on a table backed by an external location in S3 only if the external location is defined in a single metastore. Concurrent writes to the same S3 location from multiple metastores may cause consistency issues. Reading data from a single external S3 location using multiple metastores is safe.

-- Thank You!