

Unity Catalog

-Sarthak Niranjan Kulkarni (Maverick)

- sarthakkul2311@gmail.com

- (+91) 93256 02791

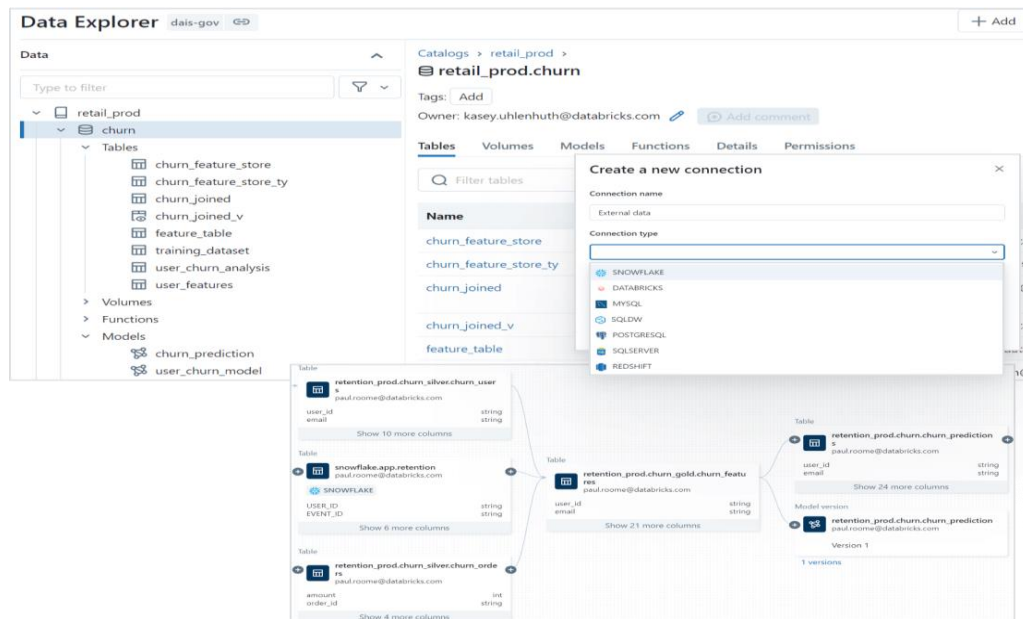
09/12/2024 (Monday)

Databricks Unity Catalog Architecture Breakdown

Here's an architecture breakdown of Databricks Unity Catalog:

1) Unified Governance Layer

Databricks Unity Catalog offers a unified governance layer for both structured and unstructured data, tables, machine learning models, notebooks, dashboards and files across any cloud or platform. This layer enables organizations to govern their data and AI assets seamlessly, ensuring regulatory compliance and accelerating data initiatives.



Unified Governance Layer of Databricks Unity Catalog (Source:databricks.com)

2) Data Discovery

Finding the right data for your needs is simple with Databricks Unity Catalog's discovery features. You can tag and document your data assets, and then use the search interface to locate the specific data you need based on keywords, tags, or other metadata.

Catalogs

[Create catalog](#)

demo-databricks

×

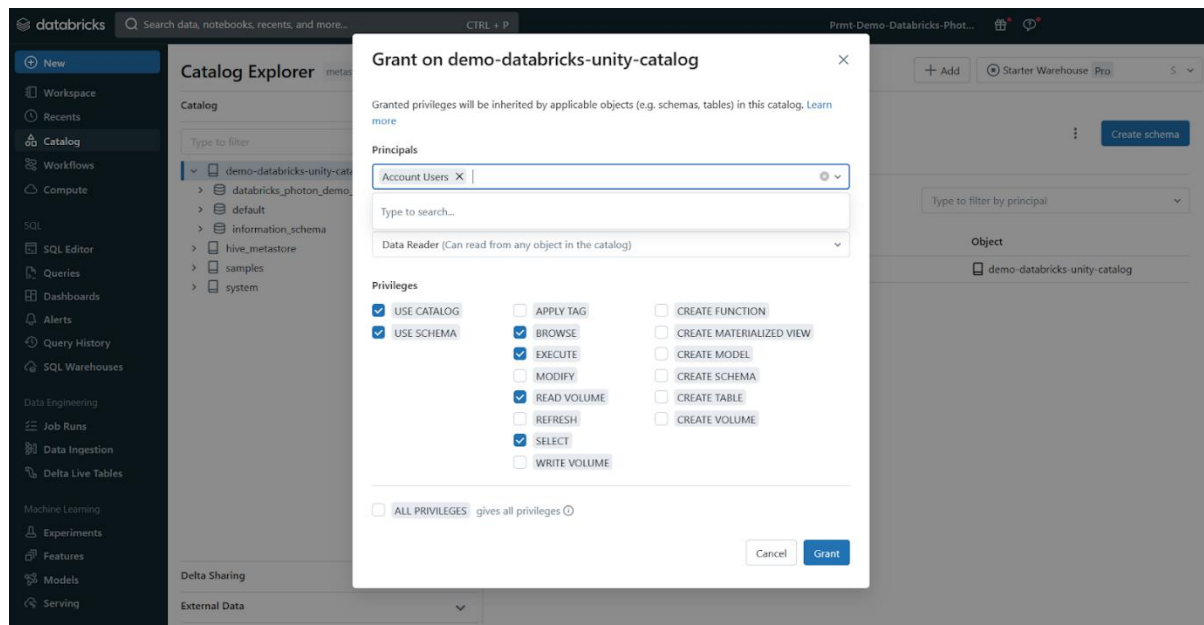
1 catalog

Name	Owner	Created at
demo-databricks-unity-catalog		2024-03-09 23:59:58

Databricks Unity Catalog's data discovery features.

3) Access Control and Security

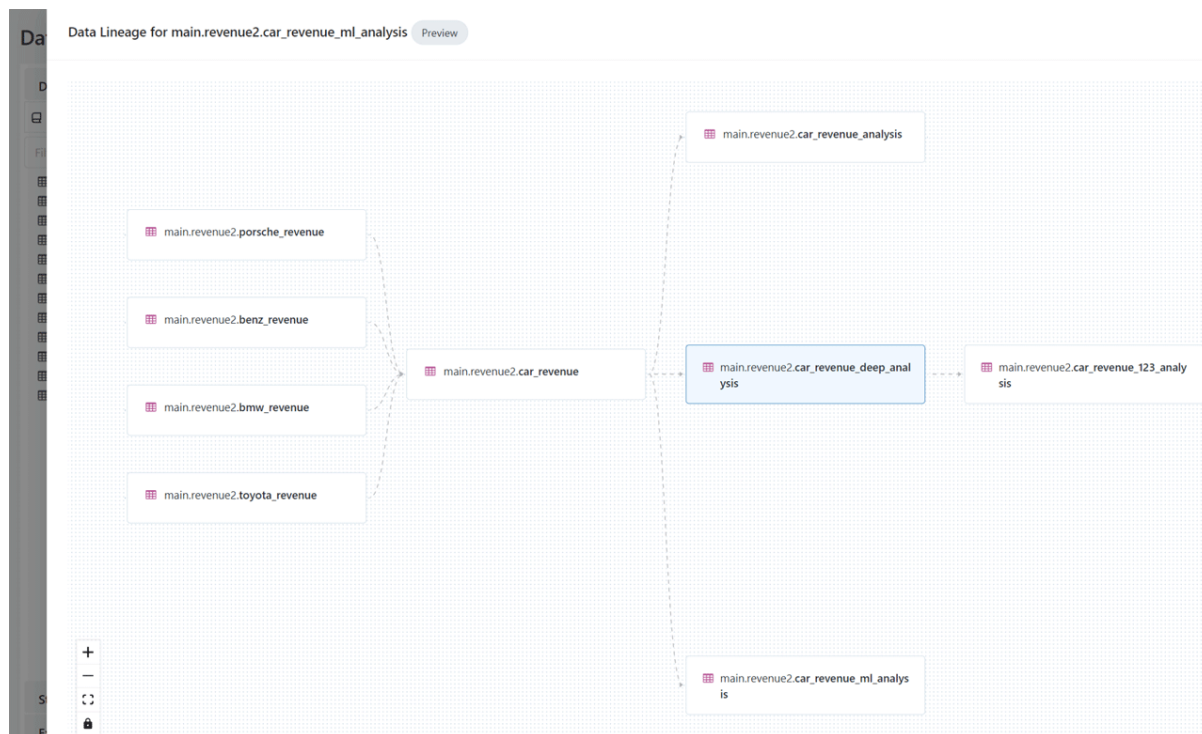
Databricks Unity Catalog simplifies access management by providing a single interface to define access policies on data and AI assets. It supports fine-grained control on rows and columns and manages access through low-code attribute-based access policies that scale seamlessly across different clouds and platforms.



Access Control and Security - Databricks Unity Catalog

4) Auditing and Lineage

Databricks Unity Catalog automatically captures audit logs that record who accessed which data assets and when. It also tracks the lineage of your data, so you can see how assets were created and how they're being used across different languages and workflows. This lineage information is crucial for understanding data flows and dependencies.



Auditing and Lineage - Databricks Unity Catalog

5) Open Data Sharing

Databricks Unity Catalog integrates with open source Delta Sharing, which allows you to securely share data and AI assets across clouds, regions, and platforms without relying on proprietary formats or complex ETL processes.



Databricks Delta Sharing - Databricks Unity Catalog

6) Object Model

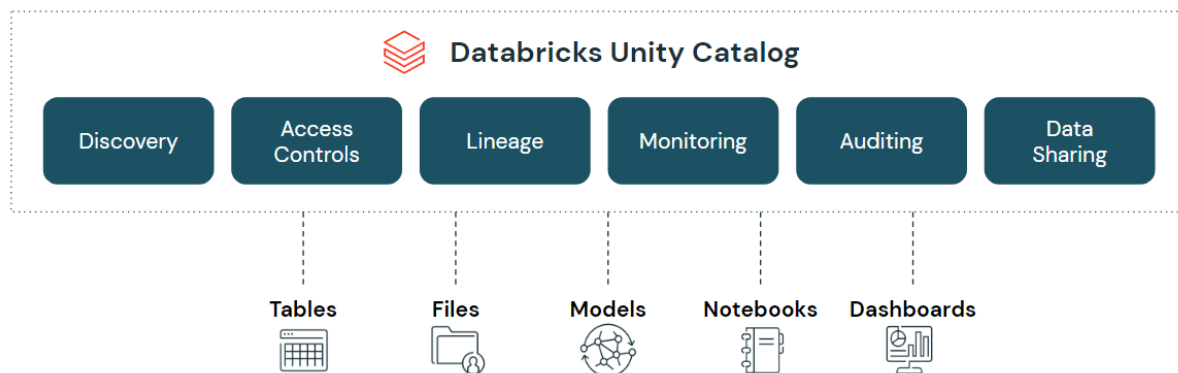
Databricks Unity Catalog organizes your data and AI assets into a hierarchical structure:

Metastore ► Catalog ► Schema ► Tables, Views, Volumes, and Models. At the top level, you have the metastore, which contains your schemas. Within each schema, you can have tables, views, or volumes (for unstructured data). To reference any asset, you use a three-part naming

convention: <catalog>.<schema>.<asset>. We will dive deeper into the object model in Unity Catalog in the next section.

7) Operational Intelligence

Databricks Unity Catalog provides AI-powered monitoring and observability capabilities that give you deep insights into your data and AI assets. You can set up active alerts, track data lineage at the column level, and gain comprehensive visibility into how your assets are being used and managed.



Databricks Unity Catalog (Source: databricks.com)

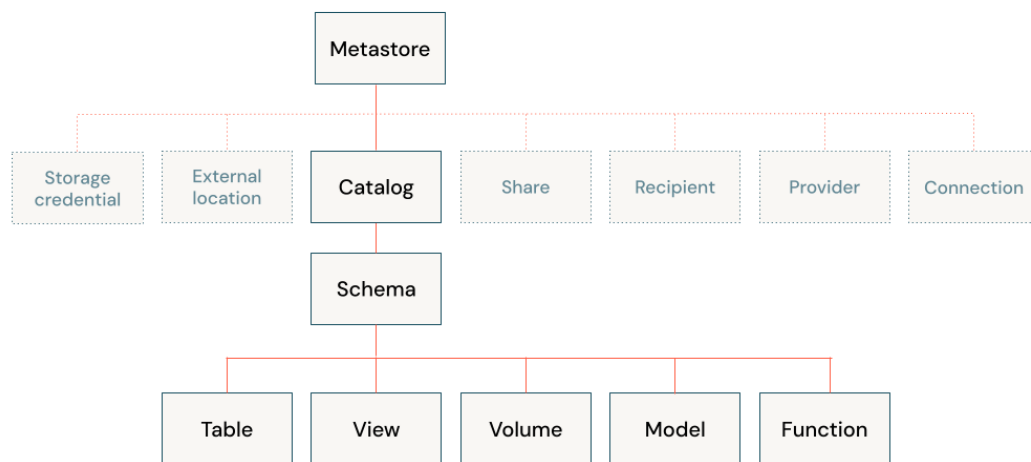
Databricks Unity Catalog Object Model

Databricks Unity Catalog follows a hierarchical architecture with the following components: Metastore ► Catalog ► Schema ► Tables, Views, Volumes, and Models.

- **Metastore:** The top-level container for metadata, exposing a three-level namespace (catalog.schema.table) to organize data assets.
- **Catalog:** The first layer of the object hierarchy, used to organize data assets logically, often aligned with organizational units or data domains.
- **Schema:** Also known as databases, schemas are the second layer of the object hierarchy, containing tables, views, and volumes.
- **Tables, Views, and Volumes:** The lowest level in the data object hierarchy, where:

- **Table:** A structured data asset that represents a collection of rows with a defined schema.
- **View:** A virtual table that is defined by a query.
- **Volume:** A container for unstructured (non-tabular) data files.
- **Models:** Machine learning models registered in the MLflow Model Registry can also be managed within Databricks Unity Catalog.

Databricks Unity Catalog Object Model: -



What Are the Supported Compute and Cluster Access Modes of Databricks Unity Catalog?

To fully leverage the capabilities of Unity Catalog, clusters must operate on compatible Databricks Runtime versions and be configured with appropriate access modes.

Databricks Unity Catalog is supported on clusters running Databricks Runtime 11.3 LTS or later. All SQL warehouse compute versions inherently support Unity Catalog, ensuring seamless integration with the latest data governance features. Clusters running on earlier Databricks Runtime versions may not provide support for all Databricks Unity Catalog features and functionality.

Unity Catalog is secure by default, meaning that if a cluster is not configured with one of the Unity Catalog-capable access modes, it cannot access data in Unity Catalog. Not all access modes are compatible with Unity Catalog. Here's a breakdown of the supported modes:

Supported Access Modes:

- **Shared Access Mode:** This is recommended for sharing clusters among multiple users. It provides a balance between isolation and collaboration.
- **Single User Access Mode:** This mode is ideal for automated jobs and machine learning workloads where a single user is responsible for the computations. It offers the highest level of isolation.

Unsupported Modes:

- **No-Isolation Shared Mode:** This is a legacy mode that doesn't meet Unity Catalog's security requirements.

Databricks recommends using compute policies to simplify configuration for managing Unity Catalog access on clusters. There are certain limitations associated with using Databricks Unity Catalog in different access modes. These limitations can affect aspects like init scripts, libraries, network access, and file system access.

Check out this [article](#) for compute access mode limitations for Databricks Unity Catalog.

What Are the Supported Regions and Data File Formats of Databricks Unity Catalog?

Databricks Unity Catalog is designed to be region-agnostic, meaning it adapts to the specific region where your Databricks workspace is located. Given that Databricks provides workspaces across a range of cloud service providers, Unity Catalog can be conveniently leveraged wherever your workspace is deployed.

Check out this [official documentation](#) on the list of supported Databricks clouds and regions.

When it comes to data formats, Databricks Unity Catalog offers distinct options for managed and external tables:

Here are the Supported Data File Formats of Databricks Unity Catalog:

- **Managed Tables:** Managed tables in Databricks Unity Catalog must use the Delta table format.

- **External (Unmanaged) Tables:** External tables in Databricks Unity Catalog can use various file formats, including Delta, CSV, JSON, Avro, Parquet, ORC, and Text.
-

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 to grant and revoke access to data assets across different environments and teams

In Hive Metastore data sharing is typically achieved through Hadoop 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

Unity catalog setup and hands on: -

1. Basic overview of Unity Catalog like its important features and architecture.
2. How to create a unity catalog.

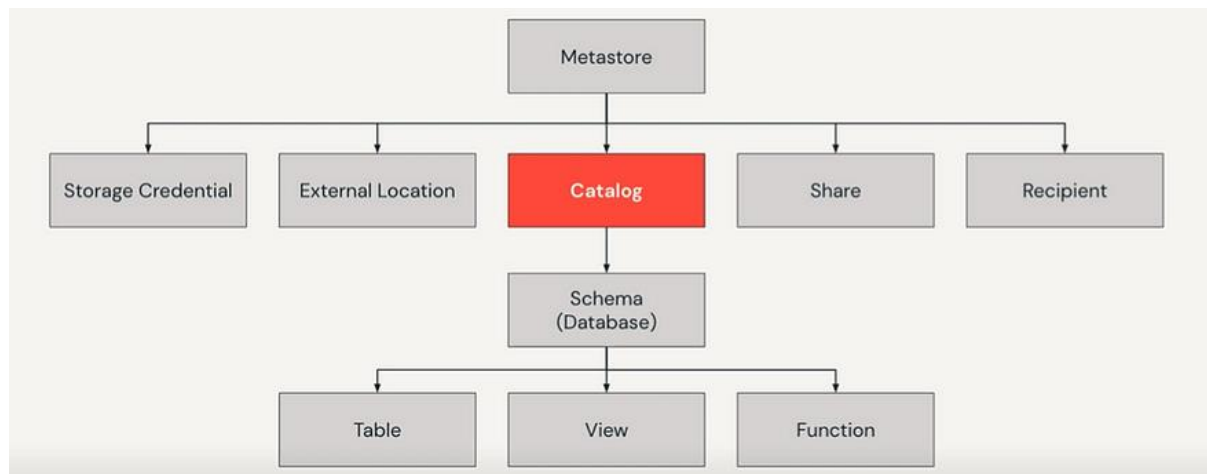
Lets first understand about Unity Catalog, it is unified governance solution for data and AI assets on the Lakehouse.

Four key functional areas of Unity Catalog:

1. **Data Access Control:** It controls who has access to which data.
2. **Data Access Audit:** It captures and record all access to data.
3. **Data Lineage:** Captures upstream sources and downstream consumers.
4. **Data Discovery:** It provides ability to search for and discover authorized assets.

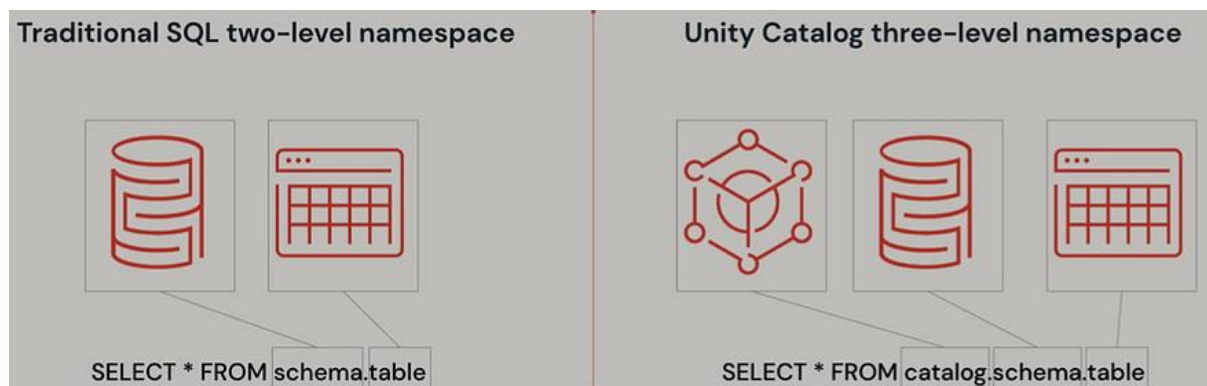
Unity Catalog Three Level Namespace:

Metastore → Catalog → Schema(Database) → Table/View/Volume

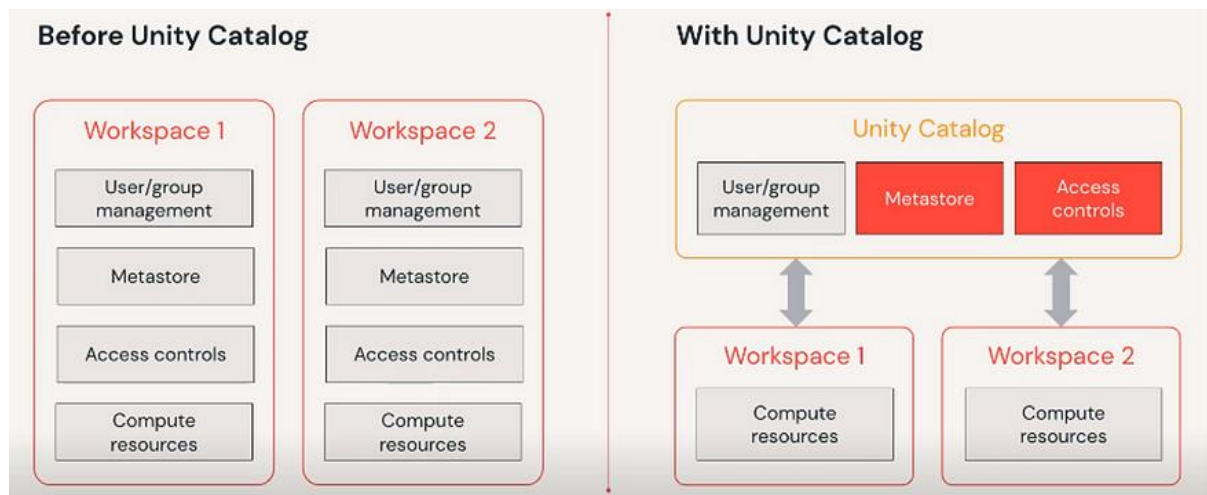


We can reference all data in Unity Catalog using a three level namespace: -

catalog.schema.table

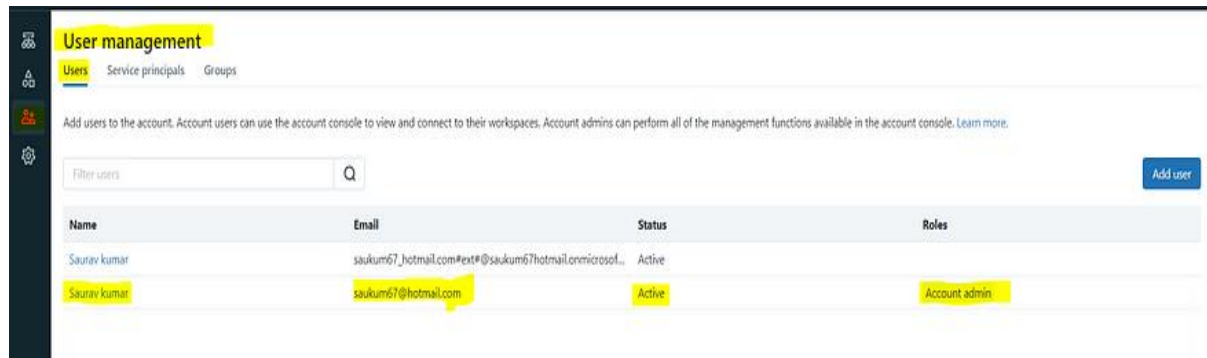


Let us compare the Architecture before Unity Catalog and with Unity Catalog:



Requirements:

To create a Unity catalog: You must be an Azure Databricks account admin.

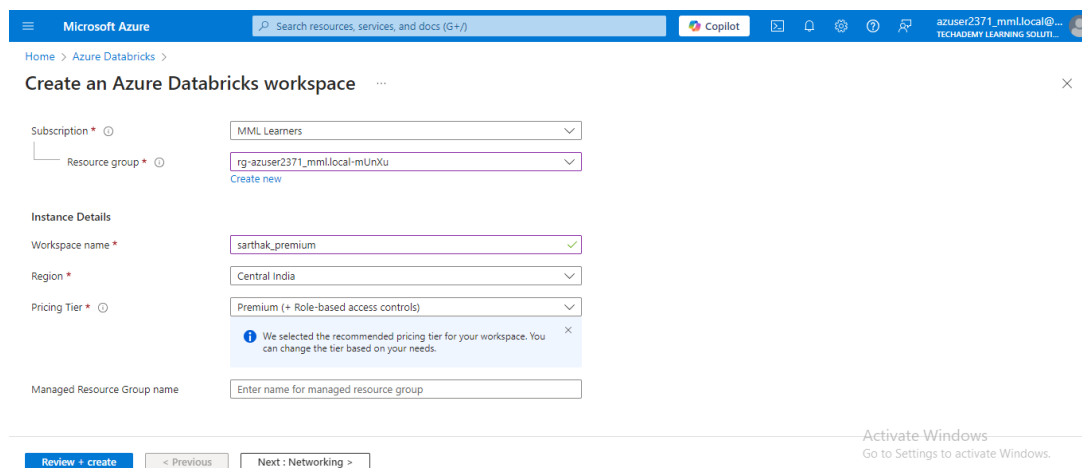


The screenshot shows the 'User management' interface in the Azure Databricks console. It has tabs for 'Users', 'Service principals', and 'Groups'. Below the tabs, there's a message: 'Add users to the account. Account users can use the account console to view and connect to their workspaces. Account admins can perform all of the management functions available in the account console. [Learn more.](#)' There is a search bar labeled 'Filter users' and an 'Add user' button. A table lists the current users:

Name	Email	Status	Roles
Saurev kumar	saikum67@hotmail.com#ext#@saikum67hotmail.onmicrosoft...	Active	
Saurev kumar	saikum67@hotmail.com	Active	Account admin

Steps to create Unity Catalog: -

Step 1. Create an Azure Databricks workspace with Premium pricing tier.

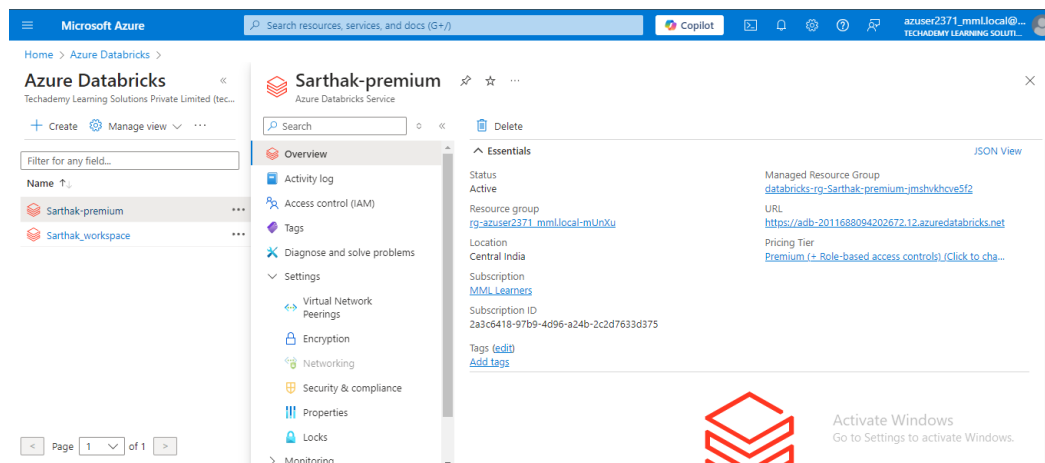


The screenshot shows the 'Create an Azure Databricks workspace' form. The fields are filled as follows:

- Subscription: MML Learners
- Resource group: rg-azuser2371_mml.local-mUnXu
- Instance Details:
 - Workspace name: sarthak_premium
 - Region: Central India
 - Pricing Tier: Premium (+ Role-based access controls)
- Managed Resource Group name: Enter name for managed resource group

At the bottom, there are buttons for 'Review + create', '< Previous', and 'Next: Networking >'. A message states: 'We selected the recommended pricing tier for your workspace. You can change the tier based on your needs.'

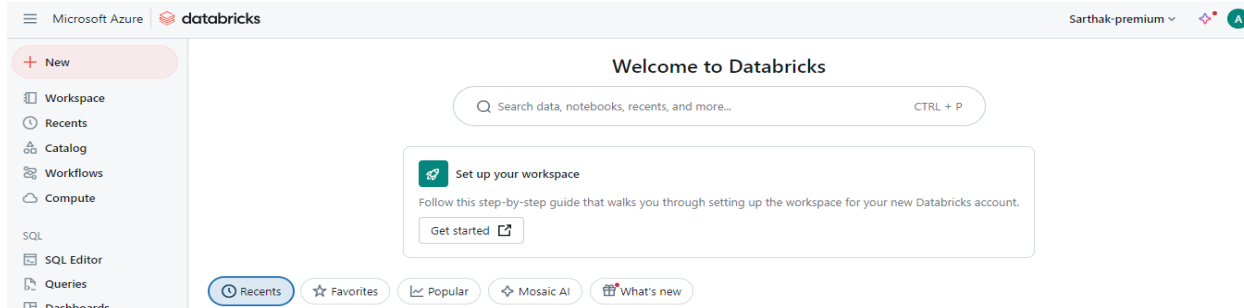
Step 2. Login into your Databricks Azure Account



The screenshot shows the 'Sarthak-premium' workspace overview in the Azure Databricks console. The left sidebar shows the workspace name 'Sarthak-premium' and a list of resources including 'Sarthak-workspace'. The main area displays the workspace details:

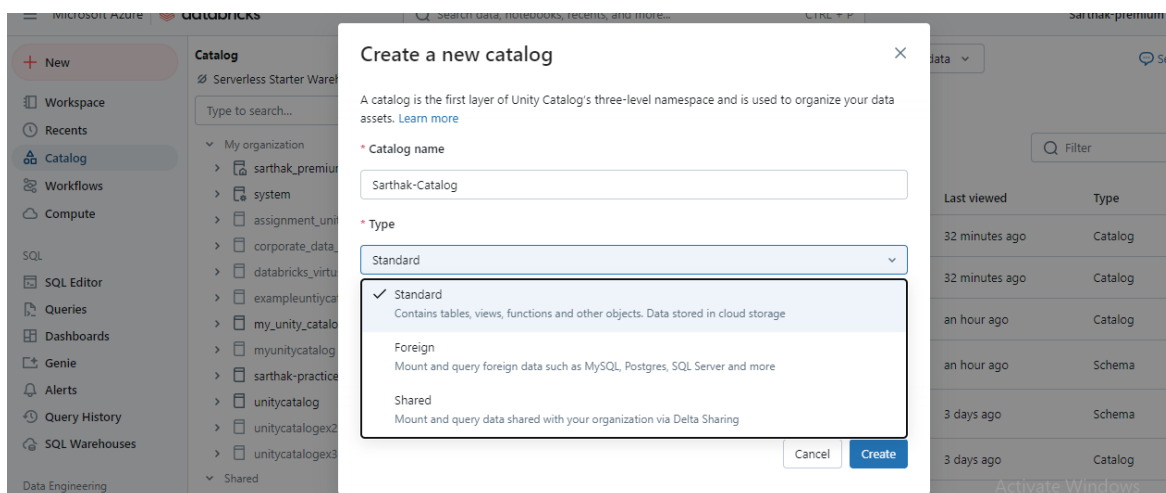
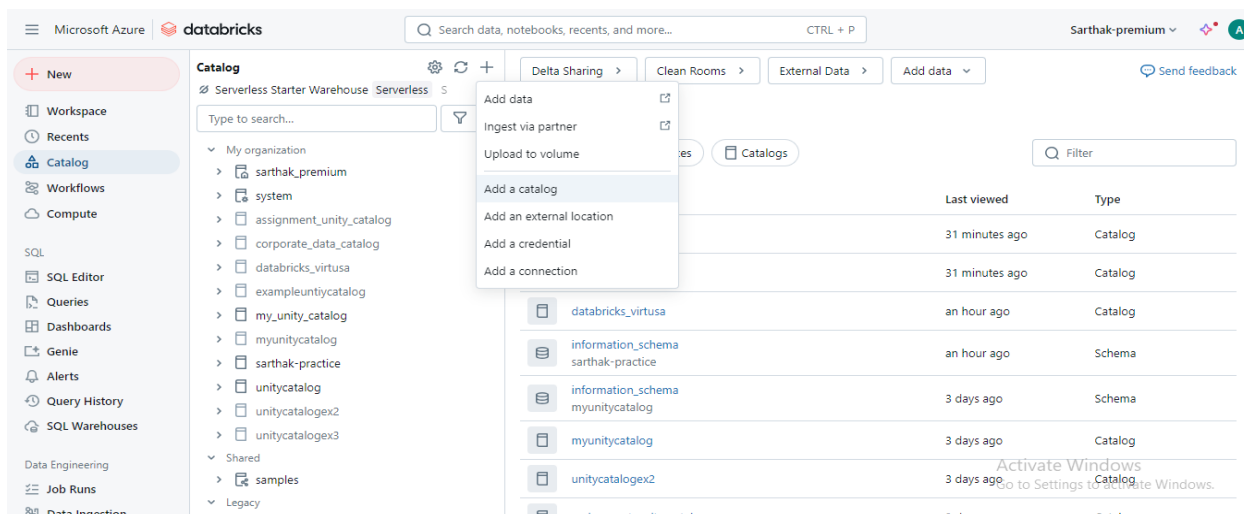
- Status: Active
- Access control (IAM): rg-azuser2371_mml.local-mUnXu
- Location: Central India
- Subscription: MML Learners
- Subscription ID: 2a3c6418-97b9-4d96-a24b-2c2d7633d375
- Tags: (edit) Add tags

At the bottom, there is an 'Activate Windows' watermark and a message: 'Go to Settings to activate Windows.'



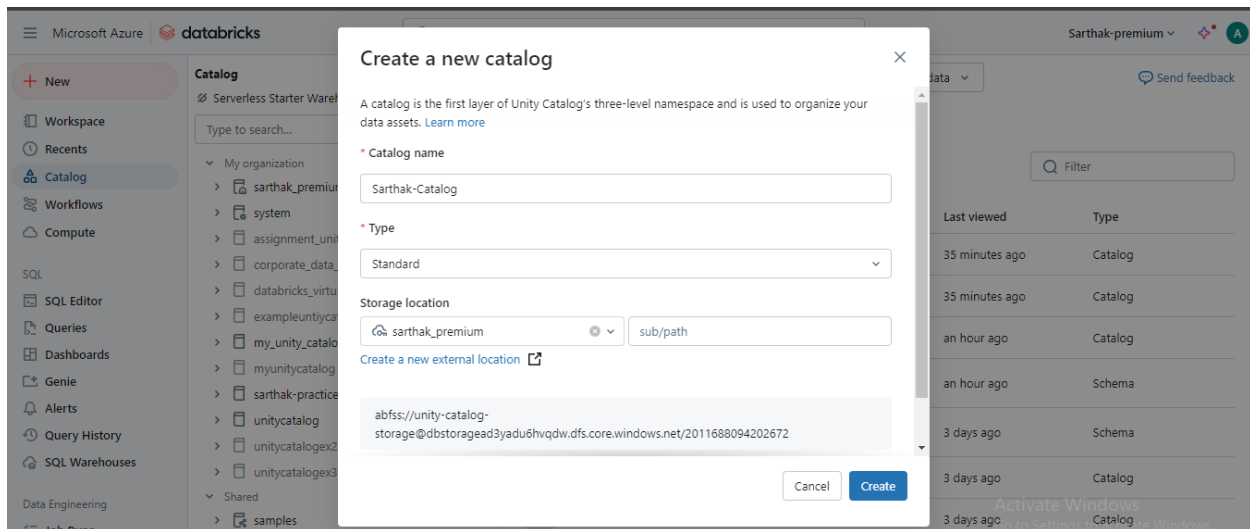
Step 3. Creating a catalog in Azure Databricks

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

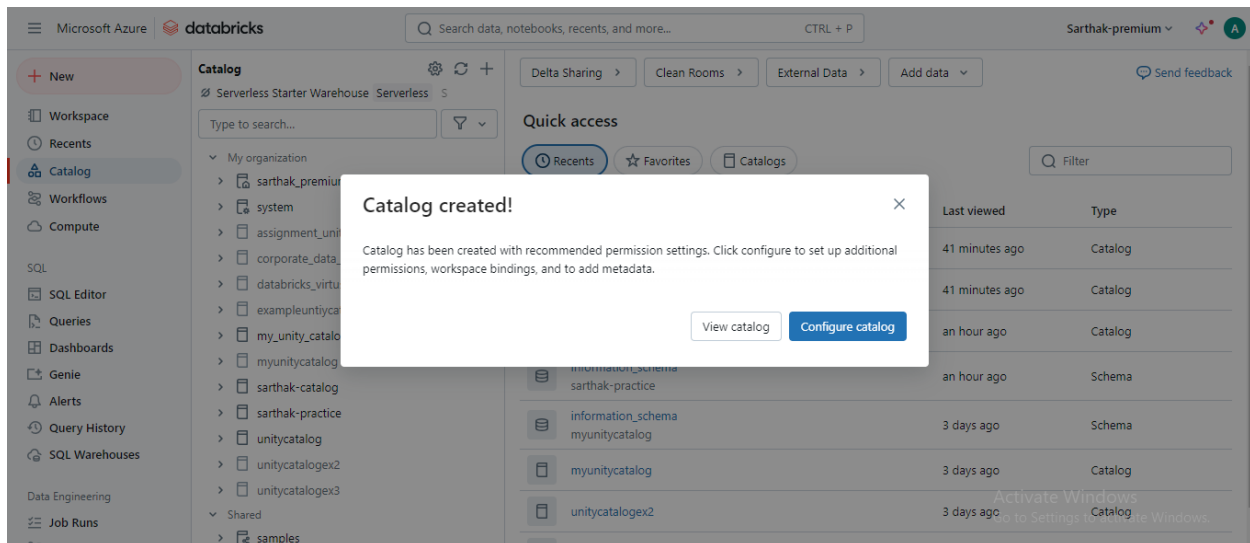


Step 4. Configure the catalog to create it:

- 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.

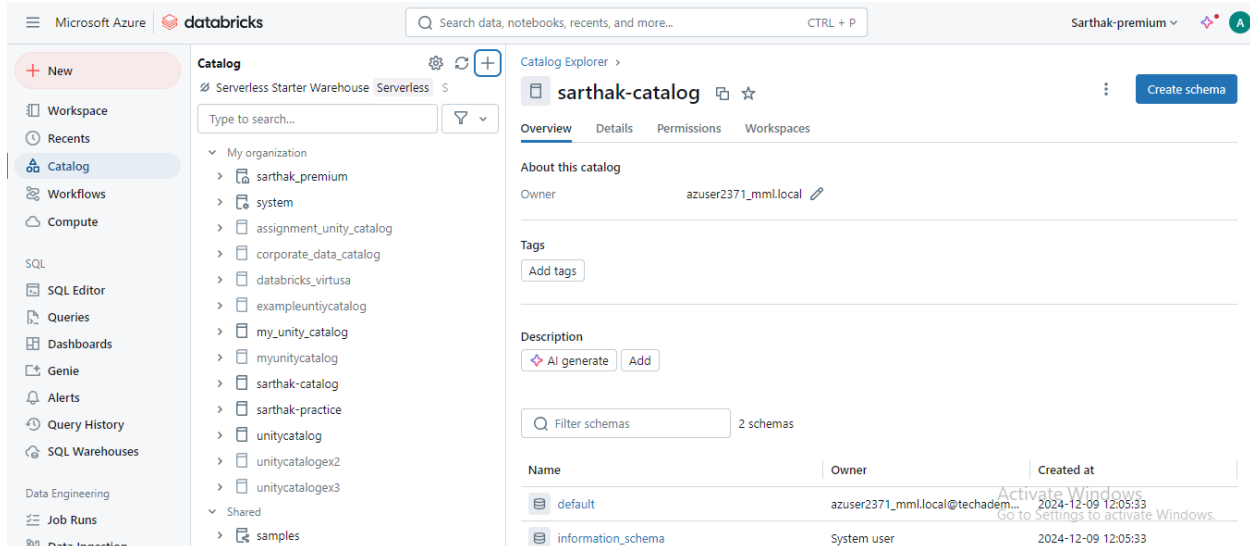


Step 5. After creating you need to view and configure the catalog for permissions



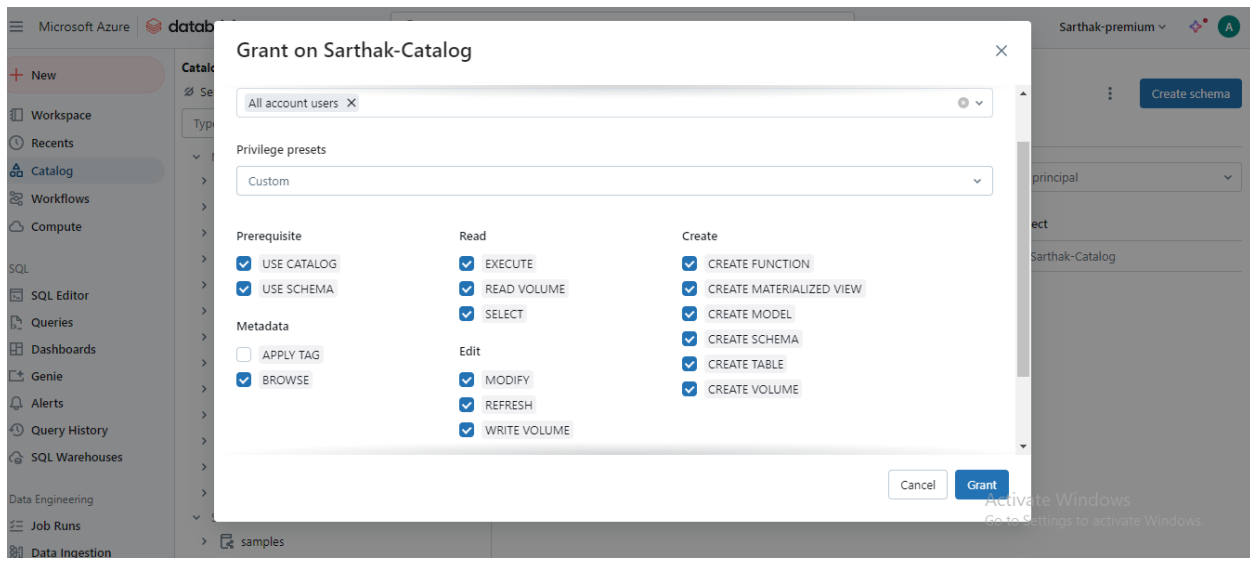
Step 6. View and Configure

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



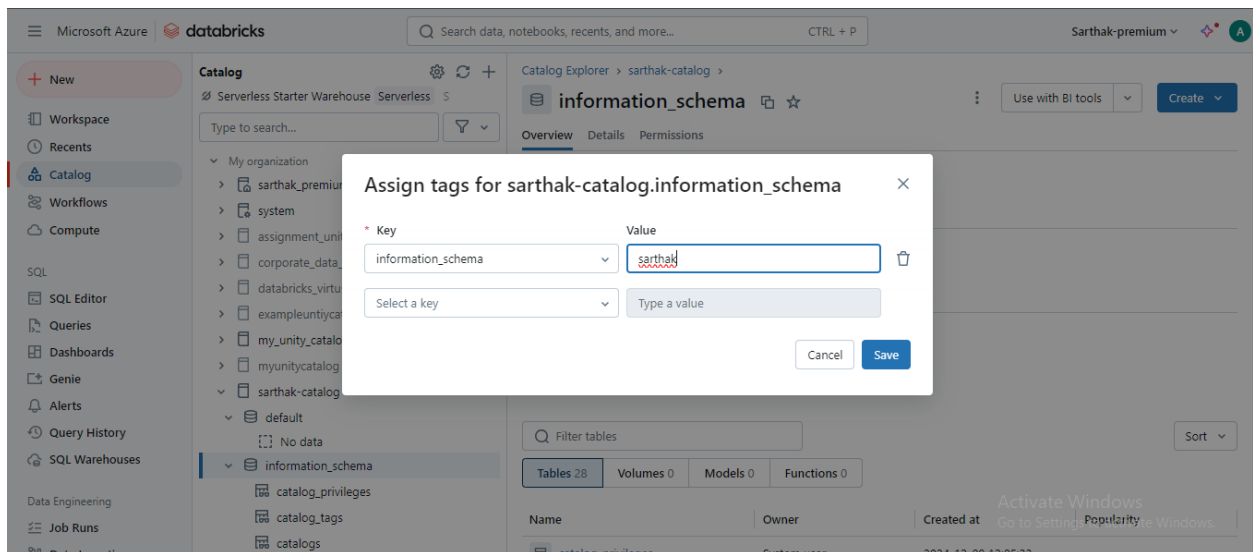
2) grant various permissions to any Groups/Users.

- Select Configure Catalog.
- Assign appropriate Permissions to users, groups, or roles.

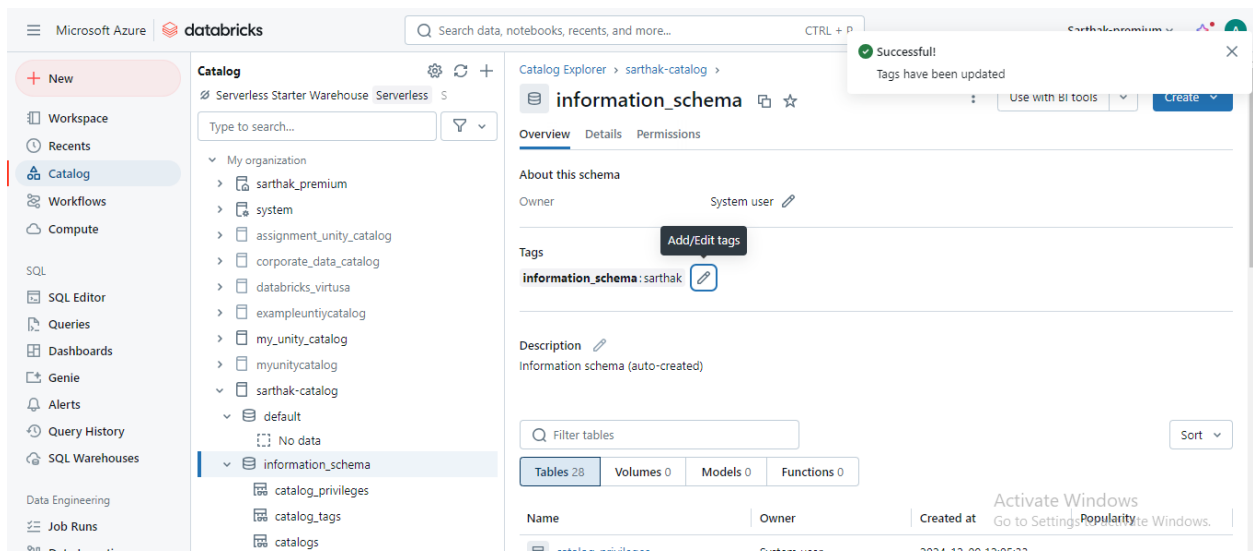


Step7. Additional metadata management.

1) Add or modify Keys and Values under the information_schema database for additional metadata management.



2) Save the changes to apply the configuration.



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>';
```

Databricks Unity Catalog Example

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`;

Databricks Unity Catalog Example

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>;

Databricks Unity Catalog Example

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>;
```

Databricks Unity Catalog Example

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>;
```

Databricks Unity Catalog Example

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>
```

Databricks Unity Catalog Example

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`;

Databricks Unity Catalog Example

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.

9) 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.

9) 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.
