Unity Catalog

Unity Catalog is a centralized data governance solution within Databricks that provides a single place to manage data access, auditing, lineage, and discovery across all Databricks workspaces. It offers a unified view of data assets, including tables, files, and machine learning models, allowing for consistent security and access policies across an organization.

Key Features and Benefits:

- Centralized Governance:

Unity Catalog provides a single point of control for managing data access and security policies, making it easier to administer and enforce consistent rules across all workspaces.

- Standards-Compliant Security:

It uses standard ANSI SQL for defining permissions, allowing administrators to leverage their existing knowledge and tools for managing data access.

- Built-in Auditing and Lineage:

Unity Catalog automatically captures user-level audit logs for data access and tracks lineage information, showing how data assets are created and used.

- Data Discovery:

It enables data consumers to easily find and understand data assets through tagging, documentation, and a search interface.

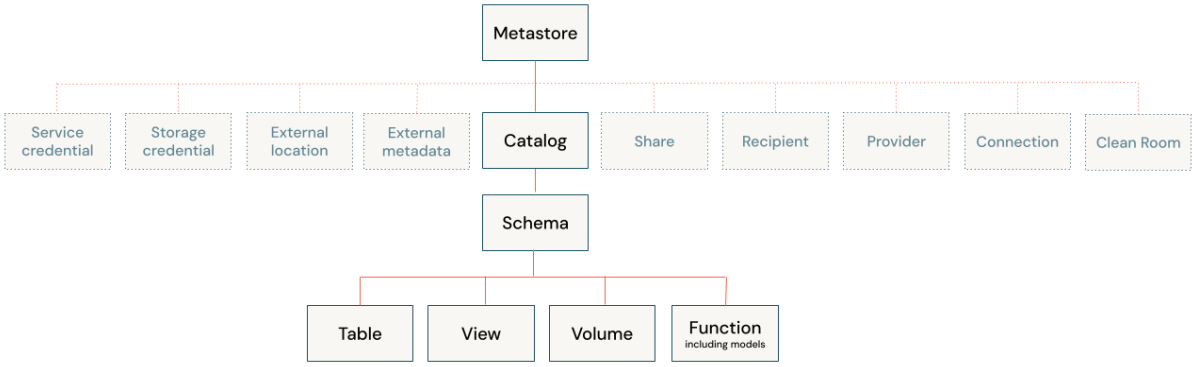
Metastore

The metastore is the top-level container for metadata in Unity Catalog. It registers metadata about data and AI assets and the permissions that govern access to them. For a workspace to use Unity Catalog, it must have a Unity Catalog metastore attached. You should have one metastore for each region in which you have workspaces.

Unlike Hive metastore, the Unity Catalog metastore is not a service boundary: it runs in a multi-tenant environment and represents a logical boundary for the segregation of data by region for a given Azure Databricks account.

The Unity Catalog object model:

Unity Catalog metastore, the three-level database object hierarchy



Level one:

Catalogs are used to organize your data assets and are typically used as the top level in your data isolation scheme. Catalogs often mirror organizational units or software development lifecycle scopes.

Non-data securable objects, such as storage credentials and external locations, are used to manage your data governance model in Unity Catalog. These also live directly under the metastore. They are described in more detail in Securable objects that Unity Catalog uses to manage access to external data sources.

Level two:

Schemas (also known as databases) contain tables, views, volumes, AI models, and functions. Schemas organize data and AI assets into logical categories that are more granular than catalogs. Typically a schema represents a single use case, project, or team sandbox.

Level three:

Tables are collections of data organized by rows and columns. Tables can be either managed, with Unity Catalog managing the full lifecycle of the table, or external, with Unity Catalog managing access to the data from within Azure Databricks, but not managing access to the data in cloud storage from other clients.

Volumes represent logical volumes of data in cloud object storage. You can use volumes to store, organize, and access files in any format, including structured, semi-structured, and unstructured data. Typically they are used for non-tabular data. Volumes can be either managed,

with Unity Catalog managing the full lifecycle and layout of the data in storage, or external, with Unity Catalog managing access to the data from within Azure Databricks, but not managing access to the data in cloud storage from other clients.

Creation of Unity Catalog Metastore:

You need **account admin** access in Databricks (not just workspace admin).

Steps:

1. Login to the Databricks Account Console
   Go to: https://accounts.databricks.com
2. Go to “Data” → Metastores
   Click “Create Metastore**”**.
3. Fill details:

   Name → MainMetastore (or any name)

   Cloud region → Match where your workspace is hosted (e.g., eastus for Azure)

   Storage Root → An external cloud storage location:

   Azure:
   abfss://<container>@<storageaccount>.dfs.core.windows.net/unitycatalog/

   Assign default data access credentials for the storage account.
4. Click Create.

3. Step 2 – Assign Metastore to Workspace

1. In Account Console → Workspaces, select your workspace.
2. Click “Assign Metastore” and choose the one you just created.
3. Confirm.

4. Step 3 – Enable Unity Catalog in Workspace

1. Go to Workspace Admin Settings → Unity Catalog.
2. Make sure it's enabled.
3. Restart the cluster or SQL warehouse.

Overview of three name spaces:

In Unity Catalog (Databricks' centralized data governance system), the 3-level namespace is the way Databricks organizes and references the data.

It follows this hierarchy:

catalog_name. schema_name. table_name

1. Catalog

- Top-level container for data assets in Unity Catalog.
- A catalog belongs to a metastore.
- Purpose: Segregate data at the highest level (e.g., for different business units, projects, or environments).
- Example:
    - sales_data
    - finance_catalog
    - research_catalog

2. Schema (similar to a "database" in traditional RDBMS)

- Second-level container inside a catalog.
- Holds related tables, views, and functions.
- Purpose: Organize data within a catalog by subject area or use case.
- Example:
    - In sales_data catalog, you might have:
        - raw_data
        - analytics
        - archive

3. Table / View

- Third-level object, contains the actual data or query definition.
- Can be:
    - Managed table – Unity Catalog stores data and metadata.

    - External table – Unity Catalog stores metadata, data is in external storage.
- Example:
    - In sales_data.analytics, you might have:
        - monthly_revenue
        - top_customers

Example Full Path

finance_catalog.q1_reports.profit_summary

- finance_catalog → Catalog
- q1_reports → Schema
- profit_summary → Table

Create Unity Catalog Objects:

Create a Catalog

```
CREATE CATALOG retail_catalog
COMMENT 'Catalog for retail analytics data';
```

Create a Schema

```
CREATE SCHEMA retail_catalog.sales_data
COMMENT 'Sales department schema';
```

Create a Table

```
CREATE TABLE retail_catalog.sales_data.transactions (
  id INT,
  date DATE,
  amount DOUBLE
)
COMMENT 'Retail transactions table';
```