Unity Catalog in Azure Databricks

1. Introduction to Unity Catalog

Imagine having multiple data lakes, databases, and workspaces, all storing different pieces of your organization's data. Without a central point of control, managing who can see what, where the data lives, and how it's used can quickly become a mess.

Unity Catalog is Databricks' unified governance solution for data and AI assets. It centralizes metadata, access control, and audit logging across all workspaces in an organization, regardless of the underlying storage system.

Some of the key benefits include:

- A **single place** to define and enforce data access policies.
- **Consistent governance** across workspaces, regardless of underlying storage systems.
- Detailed **audit logging** for compliance and investigations.
- **Data lineage tracking**, so you know exactly where each dataset comes from and how it's used.

Unity Catalog also introduces a **three-level namespace** (catalog.schema.table) that standardizes how data is organized and accessed across Databricks.

2. Unity Catalog Metastore and Workspace Enablement

The **metastore** is the brain of Unity Catalog. It's the top-level container that holds all catalogs, schemas, and tables. You can think of it as the "registry" where Databricks keeps track of your data assets and their permissions.

Before you can use Unity Catalog in a workspace, you need to **create and assign** a metastore. This is typically done once and then shared by multiple workspaces in the same region.

Each workspace in Databricks must be linked to exactly **one** Unity Catalog metastore.

2.1 Creating a Unity Catalog Metastore

Creating a metastore involves a few steps:

1. **Prepare Cloud Storage**
    - In Azure, the most common choice is **Azure Data Lake Storage Gen2**.
    - When you create the storage account, make sure you enable the **Hierarchical Namespace (HNS)** option — this allows Databricks to manage files and directories more efficiently.
    - Create a container in the storage account to hold Unity Catalog's managed data.

2. **Set Permissions for Databricks**
   - Identify the **managed identity** of your Databricks workspace.
   - In the Azure Portal, grant this identity the **Storage Blob Data Contributor** role on your storage account.
   - This ensures Databricks can read from and write to your container.
3. **Create the Metastore in the Account Console**
   - Go to the Databricks **Account Console**.
   - Under **Data → Metastores**, click **Create Metastore**.
   - Provide a **name**, select the **region**, and enter the **default storage location** in abfss:// format.
4. **Assign the Metastore to Your Workspace**
   - Still in the Account Console, go to **Workspaces**.
   - Select your workspace and click **Assign Metastore**.
   - Once assigned, the workspace can start using Unity Catalog.

3. Unity Catalog's 3-Level Namespace

Unity Catalog uses a **three-level namespace** to organize data assets:

catalog.schema.table

This structure standardizes how objects are referenced, making it easier to manage and secure data.

3.1 Catalogs

A catalog is the top-level container inside a metastore. Think of it like a department in your company. Each catalog can contain multiple schemas.

- Example: A finance catalog might contain all the company's financial data.
- Access control at the catalog level determines who can even see the schemas inside.

3.2 Schemas

Inside each catalog, you have schemas (also called databases in other systems). Schemas group related tables together.

- Example: In the finance catalog, you might have transactions and reports schemas.
- Permissions at the schema level decide who can create, modify, or view tables inside it.

3.3 Tables

Tables hold the actual data. They come in two main types:

- Managed tables – Unity Catalog controls both metadata and the underlying data files.
- External tables – Unity Catalog only stores metadata; the data itself lives in an external location like another ADLS Gen2 container.

Example fully qualified table name:

finance.transactions.customer_orders

Here:

- finance → catalog
- transactions → schema
- customer_orders → table

4. Creating Unity Catalog Objects

Once a workspace is linked to a Unity Catalog metastore, you can create objects in SQL or via the Databricks UI.

4.1 Catalogs

Catalogs are created by administrators to separate data by business unit, environment, or project. This helps with both security and organization. For example:

- A sales catalog for sales-related data.
- A dev catalog for development and testing purposes.

4.2 Schemas

Schemas are created within a specific catalog to group related datasets. For example:

- In the sales catalog, you might have raw_data and analytics schemas.
- This separation lets you apply different permissions to raw vs. processed data.

4.3 Tables

Tables are created inside schemas and can be:

- **Managed** – stored in Unity Catalog's default location, making lifecycle management easier.
- **External** – stored in a different storage path, giving you more control over the files.

The process of creating these objects can be done either:

- Through **Databricks SQL commands** in a notebook or SQL editor.
- Using the **Databricks UI**, which is simpler for new users.

5. Permissions and Governance

Unity Catalog provides **fine-grained access control** through SQL-style grants.

5.1 Access Control Levels

- **Catalog Level**: Control which schemas a user can see.
- **Schema Level**: Control which tables a user can create or query.
- **Table/View Level**: Control read, write, and ownership rights.

5.2 Security Features

- **Row-Level Security**: Filter data at query time based on user identity.
- **Column-Level Security**: Restrict access to specific sensitive columns.
- **Auditing**: All actions are logged for compliance.