**Introduction to Unity Catalog**

**Unity Catalog** is Databricks unified governance layer for data and AI assets across all Databricks workspaces and clouds. It enables organizations to **centralize access control, auditing, lineage, and data discovery** in one place, across data lakes, Delta tables, machine learning models, notebooks, dashboards, and more.

Before Unity Catalog, governance and access controls were managed at the workspace level, which made cross-workspace and enterprise-wide data sharing complex. Unity Catalog solves this by introducing **account-level metadata governance**.

**Key Features of Unity Catalog**

* **Three-level namespace**: catalog.schema.table for organizing data.
* **Fine-grained access control** using ANSI SQL and data permissions (ACLs).
* **Centralized data governance** across all Databricks workspaces.
* **Audit logging** for all data access using cloud-native audit logs.
* **Native support for data lineage** (preview or GA depending on version).
* **Cloud-native integration** with object stores like S3, ADLS, GCS.

**Unity Catalog Architecture Overview**

**Components:**

* **Metastore**: Central metadata repository; must be created per region.
* **Catalogs**: Top-level containers in Unity Catalog, like main, ecommerce, etc.
* **Schemas**: Also known as databases; live inside catalogs.
* **Tables**: Delta Lake tables or views; live inside schemas.

**Unity Catalog Object Hierarchy:**

**Account Level**

**├── Metastore**

**│ ├── Catalogs**

**│ │ ├── Schemas**

**│ │ │ └── Tables, Views, Functions**

**select \* from ecommerce.sales.orders;**

* ecommerce = catalog
* sales = schema
* orders = table

This allows for scalable, modular, and well-structured data environments.

**2)** **Creating and Configuring a Unity Catalog Metastore**

**Step 1: Create a Unity Catalog Metastore**

The **metastore** is the central container that stores metadata about catalogs, schemas, and tables. It must be created in the **same region** as your Databricks workspaces.

**Requirements**

* **Admin access** to the Databricks **Account Console**
* A **cloud storage location** (S3 bucket, ADLS container, or GCS bucket)
* **Identity Federation** enabled on workspaces

**Option A: Using Databricks Account Console (UI)**

1. Go to: https://accounts.cloud.databricks.com
2. Navigate to **Data > Metastores**
3. Click **Create Metastore**
4. Provide:
   * **Name**: prod-metastore
   * **Region**: E.g., us-west-2
   * **Storage root**: E.g., s3://my-unity-catalog-bucket
5. Click **Create**

**Option B: Using Terraform**

resource "databricks\_metastore" "this" {

name = "prod-metastore"

region = "us-west-2"

storage\_root = "s3://my-unity-catalog/"

}

**📍 Option C: Using Databricks CLI**

databricks auth login --account

databricks metastore create --json '{

"name": "prod-metastore",

"storage\_root": "s3://my-unity-catalog/",

"region": "us-west-2"

}'

**Step 2: Assign the Metastore to a Workspace**

Once the metastore is created, it must be **assigned** to a Databricks workspace:

databricks metastore assignment create --json '{

"metastore\_id": "xxxxxxxxxxxx",

"workspace\_id": 123456789,

"default\_catalog\_name": "main"

}'

This ensures that the workspace can access Unity Catalog objects.

**Step 3: Enable Unity Catalog Features**

* **Enable identity federation** for your workspace in the Account Console.
* **Use Unity Catalog-aware clusters** or SQL warehouses.
* Ensure **IAM roles and permissions** are set up to access the object store.

**3) Using Unity Catalog – Creating Objects and Access Control**

**Creating Catalogs, Schemas, and Tables**

Once the workspace is connected to Unity Catalog, users can begin creating **catalogs**, **schemas**, and **tables** using SQL or notebooks.

SQL Example:

-- Create a new catalog

CREATE CATALOG ecommerce;

-- Create schema (database)

CREATE SCHEMA ecommerce.sales;

-- Create managed table

CREATE TABLE ecommerce.sales.orders (

order\_id STRING,

customer\_id STRING,

amount DOUBLE);

-- Insert records

INSERT INTO ecommerce.sales.orders VALUES

('O001', 'C001', 100.50),

('O002', 'C002', 85.00);

-- Query data

SELECT \* FROM ecommerce.sales.orders;

**Pyspark (databricks example)**

spark.sql("CREATE CATALOG IF NOT EXISTS ecommerce")

spark.sql("CREATE SCHEMA IF NOT EXISTS ecommerce.sales")

df = spark.createDataFrame([

("O001", "C001", 100.50),

("O002", "C002", 85.00)

], ["order\_id", "customer\_id", "amount"])

df.write.saveAsTable("ecommerce.sales.orders")

**Access Control with Unity Catalog**

Unity Catalog uses a **simple permission model** where you grant access at the catalog, schema, or table level using SQL GRANT statements.

-- Grant SELECT on a table to a user

GRANT SELECT ON TABLE ecommerce.sales.orders TO `user@databricks.com`;

-- Grant usage on schema

GRANT USAGE ON SCHEMA ecommerce.sales TO `analyst-group`;

-- Revoke access

REVOKE SELECT ON TABLE ecommerce.sales.orders FROM `user@databricks.com`;

**Unity Catalog in Production**

In a real-world setup, you would:

* Use **Terraform** to manage Unity Catalog objects (catalogs, schemas, policies)
* Implement **data classification** using tags
* Enforce **row-level or column-level security**
* Automate governance workflows using **Unity Catalog APIs**

**Conclusion**

Unity Catalog brings enterprise-grade governance, security, and manageability to your Databricks data platform. By centralizing metadata and access control, it simplifies governance at scale, improves data discovery, and enables secure data collaboration across teams and workspaces.

Whether you're building a data lakehouse for analytics or powering machine learning workloads, Unity Catalog is a foundational component for modern data platforms.