**1. Unity Catalog**

**Definition**

Unity Catalog is Databricks’ **centralized data governance and management layer**.  
It lets you control and track all your data assets (tables, files, ML models, dashboards, etc.) **across multiple Databricks workspaces** from a single, unified interface.

It’s essentially a **centralized metadata service + fine-grained access control system**.  
Instead of each workspace having its own isolated Hive Metastore, Unity Catalog provides one source of truth.

**Core Benefits**

* **Centralized Governance** → One place to manage permissions and policies
* **Three-Level Namespace** → Organize data as catalog.schema.table
* **Audit & Lineage** → See who accessed what data and when
* **Cross-Workspace Sharing** → Data accessible across all linked workspaces
* **Support for Multiple Data Sources** → S3, ADLS, GCS, etc.

**Main Components**

* **Catalog** → Top-level container of data assets
* **Schema** → Logical grouping of tables/views within a catalog
* **Table** → The actual dataset
* **External Locations** → Links to raw data in cloud storage
* **Functions & Views** → For processing and presenting data

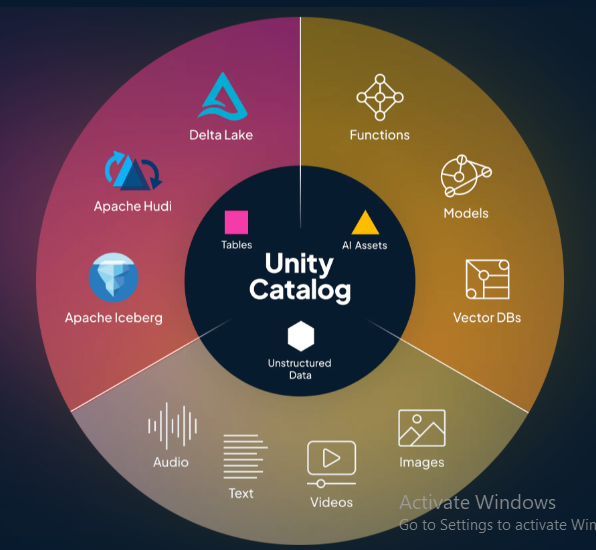
**Example**

# Create Spark session

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("UnityCatalogExample").getOrCreate()

print("Unity Catalog simulated: Ready to manage catalogs, schemas, and tables.")

****

**2. Create Unity Catalog Metastore and Enable a Databricks Workspace with Unity Catalog**

**Definition**

The **Metastore** is the **brain** of Unity Catalog — a central metadata repository that:

* Stores definitions of all catalogs, schemas, and tables
* Records ownership and access permissions
* Knows the physical storage locations of the data
* Tracks lineage and audit logs

When you **enable Unity Catalog in a Databricks workspace**, you are:

1. Linking that workspace to a **single metastore** (per cloud region).
2. Giving it access to all catalogs, schemas, and tables defined in that metastore.

**Types of Metastore Setup**

1. **Regional Metastore** → One per cloud region (recommended)
2. **Shared Metastore** → Linked to multiple workspaces
3. **Dedicated Metastore** → Linked to one workspace only

**Example**

# Simulated metastore

metastore = {

"name": "MainMetastore",

"region": "us-east-1",

"storage\_root": "gs://fake-metastore-bucket/",

"catalogs": {}

}

# Linking workspace

workspace = {

"name": "AnalyticsWorkspace",

"linked\_metastore": metastore["name"]

}

print("Metastore Created:", metastore)

print("Workspace Enabled:", workspace)

**3. Overview of 3-Level Namespace**

**Definition**

Unity Catalog uses a **three-level hierarchical naming convention**:

catalog.schema.table

This ensures global uniqueness of object names across workspaces.

**Levels**

1. **Catalog**
   * The highest-level container (e.g., sales\_data)
   * Contains multiple schemas
   * Controlled by Unity Catalog permissions
2. **Schema** (like a database in SQL)
   * Logical grouping of tables/views inside a catalog (e.g., january)
3. **Table**
   * The actual dataset (e.g., transactions)

**Example (Real Unity Catalog SQL)**

CREATE CATALOG sales\_data;

CREATE SCHEMA sales\_data.january;

CREATE TABLE sales\_data.january.transactions (

id INT, amount DOUBLE, date DATE

);

**Simulation in PySpark**

# Add catalog

metastore["catalogs"]["sales\_data"] = {}

# Add schema

metastore["catalogs"]["sales\_data"]["january"] = {}

# Add table

metastore["catalogs"]["sales\_data"]["january"]["transactions"] = {

"columns": ["id", "amount", "date"],

"rows": []

}

print("3-Level Namespace:", metastore)

**4. Creating Unity Catalog Objects**

**Definition**

Unity Catalog objects are the actual entities you create and manage in Databricks under governance rules.

**Main Types of Objects**

1. **Catalogs** → Top-level container for schemas
2. **Schemas** → Grouping for related tables/views
3. **Tables**
   * **Managed Tables** → Unity Catalog stores both metadata and data
   * **External Tables** → Metadata in Unity Catalog, data stored externally
4. **Views** → Saved queries over tables
5. **Functions** → User-defined reusable logic

**Example (Real Unity Catalog SQL)**

CREATE CATALOG marketing;

CREATE SCHEMA marketing.campaigns;

CREATE TABLE marketing.campaigns.ads (

id INT,

campaign\_name STRING,

budget DOUBLE

);

from pyspark.sql import Row

# Create PySpark DataFrame for a table

ads\_data = [

Row(id=1, campaign\_name="Summer Sale", budget=5000.0),

Row(id=2, campaign\_name="Winter Sale", budget=7000.0)

]

df\_ads = spark.createDataFrame(ads\_data)

# Show the table data

df\_ads.show()

# Register as temporary view to simulate Unity Catalog table

df\_ads.createOrReplaceTempView("marketing\_campaigns\_ads")

# Query the view

spark.sql("SELECT \* FROM marketing\_campaigns\_ads").show()

