# Unity Catalog in Azure Databricks Jothiswaran P

## 1. Unity Catalog Overview

Unity Catalog is a unified governance solution for all data and AI assets in Azure Databricks. It provides a central place to manage access controls, audit usage, and organize data assets across multiple workspaces. Unity Catalog enables organizations to implement fine-grained access control at the column and row level, ensuring compliance and secure collaboration.

## 2. Creating Unity Catalog Metastore and Enabling a Databricks Workspace

Steps to create and configure a Unity Catalog Metastore in Azure Databricks:

1. Sign in to the Azure portal and navigate to your Databricks workspace.  
2. In the Databricks admin console, go to the 'Data' section.  
3. Click 'Create Metastore' and provide details such as Name, Storage Root Path (e.g., an Azure Data Lake Storage Gen2 path), and Region.  
4. Assign the Metastore to your workspace by navigating to the 'Metastore Assignments' tab.  
5. Ensure the service principal or managed identity used by Databricks has Storage Blob Data Contributor access to the ADLS Gen2 container.  
6. Enable Unity Catalog by confirming the workspace assignment.

### Example:

In Azure Databricks CLI or REST API, you can create a metastore using:  
  
{  
 "name": "Sales",  
 "storage\_root": "dbfs://tmp/file",  
 "region": "eastus"  
}

## 3. Overview of 3-Level Namespace

Unity Catalog organizes data objects into a 3-level namespace:  
1. Metastore: The top-level container for all catalogs.  
2. Catalog: Contains schemas (databases) and is used for grouping related data.  
3. Schema: Contains tables, views, and functions.  
  
The fully qualified name of a table in Unity Catalog is in the format: <store>.<catalog>.<schema>.<table>.

### Example:

sales\_metastore.retail\_data.sales\_2024  
- store: sales\_store  
- Catalog: retail\_data  
- Schema: sales\_2024  
- Table: transactions

## 4. Creating Unity Catalog Objects

Examples of creating Unity Catalog objects in SQL:

CREATE CATALOG retail\_data;  
  
CREATE SCHEMA retail\_data.sales\_2024;  
  
CREATE TABLE retail\_data.sales\_2024.transactions (  
 transaction\_id STRING,  
 product\_id STRING,  
 quantity INT,  
 price DECIMAL(10,2),  
 transaction\_date DATE  
);

Examples in Azure Databricks notebooks can include using PySpark to interact with Unity Catalog tables.

### PySpark Example:

from pyspark.sql import SparkSession  
spark = SparkSession.builder.getOrCreate()  
  
# Reading from a Unity Catalog table  
df = spark.table('retail\_data.sales\_2024.transactions')  
df.show()  
  
# Writing to a Unity Catalog table  
df.write.format('delta').saveAsTable('retail\_data.sales\_2024.new\_transactions')

## 5. Granting Permissions in Unity Catalog

Unity Catalog uses ANSI SQL GRANT and REVOKE statements for permission management. You can grant permissions at the catalog, schema, or table level. Permissions can also be granted to Azure Active Directory groups for easier management.

Examples:  
-- Grant SELECT on table  
GRANT SELECT ON TABLE retail\_data.sales\_2024.transactions TO `data\_analyst`;  
  
-- Grant CREATE on schema  
GRANT CREATE ON SCHEMA retail\_data.sales\_2024 TO `etl\_engineer`;

## 6. Managing Data in Unity Catalog

Perform CRUD operations (Create, Read, Update, Delete) on Unity Catalog tables using PySpark or SQL. With Delta Lake integration, you can also perform UPSERT operations.

Example PySpark Update/Delete:  
from delta.tables import DeltaTable  
delta\_tbl = DeltaTable.forName(spark, 'retail\_data.sales\_2024.transactions')  
delta\_tbl.update(  
 condition="transaction\_id = 'T001'",  
 set={"price": "99.99"}  
)  
delta\_tbl.delete("quantity = 0")

Example UPSERT:  
delta\_tbl.alias('t').merge(  
 new\_data.alias('s'),  
 't.transaction\_id = s.transaction\_id'  
).whenMatchedUpdateAll()  
.whenNotMatchedInsertAll()  
.execute()

## 7. Best Practices

- Use separate catalogs for different business domains.  
- Assign permissions to AAD groups rather than individual users.  
- Use schemas to organize datasets by project or department.  
- Enable audit logging to track data access.  
- Store raw, clean, and aggregated data in separate schemas.  
- Always specify the fully qualified table name to avoid ambiguity.

## 8. Integrating with Azure Key Vault

Secure credentials for accessing external data sources by storing them in Azure Key Vault and referencing them in Databricks. Unity Catalog integrates seamlessly with Azure Key Vault for secret management.

Example:  
dbutils.secrets.get(scope='my-keyvault-scope', key='storage-key')

## 9. Viewing Data Lineage

Unity Catalog automatically tracks lineage for tables and views. You can view lineage in the Databricks UI to understand data flow across transformations.