DATABRICKS CODING ASSESMENT

[Jobaoushadancse2021@jerusalemengg.ac.in](mailto:Jobaoushadancse2021@jerusalemengg.ac.in)

# UNITY CATALOG

Unity Catalog is a centralized data catalog that provides access control, auditing, lineage, quality monitoring, and data discovery capabilities across Azure Databricks workspaces.

Key features of Unity Catalog include:

* **Define once, secure everywhere**: Unity Catalog offers a single place to administer data access policies that apply across all workspaces in a region.
* **Standards-compliant security model**: Unity Catalog's security model is based on standard ANSI SQL and allows administrators to grant permissions in their existing data lake using familiar syntax.
* **Built-in auditing and lineage**: Unity Catalog automatically captures user-level audit logs that record access to your data. Unity Catalog also captures lineage data that tracks how data assets are created and used across all languages.
* **Data discovery**: Unity Catalog lets you tag and document data assets, and provides a search interface to help data consumers find data.
* **System tables**: Unity Catalog lets you easily access and query your account's operational data, including audit logs, billable usage, and lineage.

## 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

In a Unity Catalog metastore, the three-level database object hierarchy consists of catalogs that contain schemas, which in turn contain data and AI objects, like tables and models. This hierarchy is represented as a three-level namespace (catalog.schema.table-etc) when you reference tables, views, volumes, models, and functions.

**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. See [What are catalogs in Azure Databricks?](https://learn.microsoft.com/en-us/azure/databricks/catalogs/).
* **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](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/#other-securables).

**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. See [What are schemas in Azure Databricks?](https://learn.microsoft.com/en-us/azure/databricks/schemas/).

**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. See [Introduction to Azure Databricks tables](https://learn.microsoft.com/en-us/azure/databricks/tables/) and [Managed versus external tables and volumes](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/#managed-vs-external).
* **Views** are saved queries against one or more tables. See [What is a view?](https://learn.microsoft.com/en-us/azure/databricks/views/).
* **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. See [What are Unity Catalog volumes?](https://learn.microsoft.com/en-us/azure/databricks/volumes/) and [Managed versus external tables and volumes](https://learn.microsoft.com/en-us/azure/databricks/data-governance/unity-catalog/#managed-vs-external).
* **Functions** are units of saved logic that return a scalar value or set of rows. See [User-defined functions (UDFs) in Unity Catalog](https://learn.microsoft.com/en-us/azure/databricks/udf/unity-catalog).
* **Models** are AI models packaged with MLflow and registered in Unity Catalog as functions. See [Manage model lifecycle in Unity Catalog](https://learn.microsoft.com/en-us/azure/databricks/machine-learning/manage-model-lifecycle/).

## ****Step 1 – Create Catalog and Schema****

**Code:**

-- Create catalog for the demo

CREATE CATALOG IF NOT EXISTS retail\_data

COMMENT 'Retail Sales Data Catalog for Governance Demo';

-- Switch to the catalog

USE CATALOG retail\_data;

-- Create schema for storing sales-related data

CREATE SCHEMA IF NOT EXISTS sales\_data

COMMENT 'Schema for storing sales transactions and customer details';

-- Use the schema

USE retail\_data.sales\_data;

**Expected Output:**

OK

OK

OK

## ****Step 2 – Create Sample Tables with Data****

**Code:**

-- Customers table

CREATE OR REPLACE TABLE retail\_data.sales\_data.customers (

customer\_id INT,

name STRING,

email STRING,

phone STRING,

region STRING

) USING DELTA;

INSERT OVERWRITE retail\_data.sales\_data.customers VALUES

(1, 'Asha Kumar', 'asha.kumar@example.com', '+91-90000-00001', 'west'),

(2, 'Ravi Patel', 'ravi.patel@example.com', '+91-90000-00002', 'east'),

(3, 'Neha Singh', 'neha.singh@example.com', '+91-90000-00003', 'west'),

(4, 'Arjun Mehta', 'arjun.mehta@example.com', '+91-90000-00004', 'east');

-- Transactions table

CREATE OR REPLACE TABLE retail\_data.sales\_data.transactions (

transaction\_id INT,

customer\_id INT,

transaction\_date DATE,

amount DECIMAL(10,2),

payment\_method STRING

) USING DELTA;

INSERT OVERWRITE retail\_data.sales\_data.transactions VALUES

(101, 1, DATE'2025-06-01', 1200.00, 'Card'),

(102, 1, DATE'2025-06-15', 450.00, 'UPI'),

(103, 2, DATE'2025-06-20', 999.99, 'Card'),

(104, 3, DATE'2025-07-03', 150.00, 'Cash'),

(105, 4, DATE'2025-07-07', 200.00, 'UPI'),

(106, 3, DATE'2025-07-15', 750.00, 'Card');

**Expected Output:**

OK

4 rows affected

OK

6 rows affected

## ****Step 3 – Data Discovery (Metadata & Tags)****

**Code:**

-- Add comments and tags

COMMENT ON TABLE retail\_data.sales\_data.transactions IS 'Retail transactions (demo)';

COMMENT ON TABLE retail\_data.sales\_data.customers IS 'Customer master with PII-like columns (demo)';

ALTER TABLE retail\_data.sales\_data.customers

SET TAGS ('PII' = 'true', 'owner' = 'data-eng');

ALTER TABLE retail\_data.sales\_data.transactions

SET TAGS ('domain' = 'retail', 'gold' = 'false');

-- Query information\_schema

SELECT table\_catalog, table\_schema, table\_name, table\_type, comment

FROM retail\_data.information\_schema.tables

WHERE table\_schema = 'sales\_data';

-- Show table properties

SHOW TBLPROPERTIES retail\_data.sales\_data.customers;

**Expected Output (table list):**

| **table\_catalog** | **table\_schema** | **table\_name** | **table\_type** | **comment** |
| --- | --- | --- | --- | --- |
| retail\_data | sales\_data | customers | BASE TABLE | Customer master with PII-like columns (demo) |
| retail\_data | sales\_data | transactions | BASE TABLE | Retail transactions (demo) |

**Expected Output (table properties):**

| **key** | **value** |
| --- | --- |
| PII | true |
| owner | data-eng |

## ****Step 4 – Data Access Control (GRANT/REVOKE)****

**Code:**

-- Grant SELECT on transactions

GRANT SELECT ON TABLE retail\_data.sales\_data.transactions TO `analyst\_group`;

-- Revoke SELECT on customers

REVOKE SELECT ON TABLE retail\_data.sales\_data.customers FROM `analyst\_group`;

**Expected Output:**

OK

OK

## ****Step 5 – Dynamic Data Masking****

**Code:**

CREATE OR REPLACE VIEW retail\_data.sales\_data.customers\_masked AS

SELECT

customer\_id,

name,

CASE

WHEN is\_member('pii\_readers') THEN email

ELSE concat(substr(email,1,3), '\*\*\*@\*\*\*.\*\*\*')

END AS email\_masked,

CASE

WHEN is\_member('pii\_readers') THEN phone

ELSE concat('+\*\*-\*\*\*\*\*-', substr(phone, -2))

END AS phone\_masked,

region

FROM retail\_data.sales\_data.customers;

**Expected Output (if NOT in pii\_readers):**

| **customer\_id** | **name** | **email\_masked** | **phone\_masked** | **region** |
| --- | --- | --- | --- | --- |
| 1 | Asha Kumar | ash\*\*\*@***.*** | +**-**\*\*\*-01 | west |
| 2 | Ravi Patel | rav\*\*\*@***.*** | +**-**\*\*\*-02 | east |

## ****Step 6 – Row-Level Security****

**Code:**

CREATE OR REPLACE VIEW retail\_data.sales\_data.transactions\_rls AS

SELECT t.\*

FROM retail\_data.sales\_data.transactions t

LEFT JOIN retail\_data.sales\_data.customers c

ON t.customer\_id = c.customer\_id

WHERE

(is\_member('west\_region') AND c.region = 'west')

OR (is\_member('east\_region') AND c.region = 'east')

OR (is\_member('account\_admin'));

**Expected Output (if in west\_region):**

| **transaction\_id** | **customer\_id** | **transaction\_date** | **amount** | **payment\_method** |
| --- | --- | --- | --- | --- |
| 101 | 1 | 2025-06-01 | 1200.00 | Card |
| 102 | 1 | 2025-06-15 | 450.00 | UPI |
| 104 | 3 | 2025-07-03 | 150.00 | Cash |
| 106 | 3 | 2025-07-15 | 750.00 | Card |

## ****Step 7 – Data Lineage****

**Code:**

-- Derived table 1

CREATE OR REPLACE TABLE retail\_data.sales\_data.monthly\_sales AS

SELECT customer\_id,

date\_format(transaction\_date, 'yyyy-MM') AS month,

SUM(amount) AS total\_sales

FROM retail\_data.sales\_data.transactions

GROUP BY customer\_id, date\_format(transaction\_date, 'yyyy-MM');

-- Derived table 2

CREATE OR REPLACE TABLE retail\_data.sales\_data.customer\_monthly\_sales AS

SELECT c.customer\_id, c.name, c.region,

m.month, m.total\_sales

FROM retail\_data.sales\_data.monthly\_sales m

JOIN retail\_data.sales\_data.customers c

ON m.customer\_id = c.customer\_id;

**Expected Output (monthly\_sales):**

| **customer\_id** | **month** | **total\_sales** |
| --- | --- | --- |
| 1 | 2025-06 | 1650.00 |
| 2 | 2025-06 | 999.99 |
| 3 | 2025-07 | 900.00 |
| 4 | 2025-07 | 200.00 |

## ****Step 8 – Data Audit****

**Code:**

SELECT event\_time,

user\_identity.email AS user\_email,

action\_name,

request\_params.full\_name AS object\_full\_name,

response.result

FROM system.access.audit

WHERE request\_params.full\_name LIKE 'retail\_data.sales\_data.%'

AND action\_name IN ('createTable','createView','select','grant','revoke')

ORDER BY event\_time DESC

LIMIT 20;

**Expected Output:**

| **event\_time** | **user\_email** | **action\_name** | **object\_full\_name** | **result** |
| --- | --- | --- | --- | --- |
| 2025-08-14T09:30Z | you@company.com | select | retail\_data.sales\_data.transactions | SUCCESS |
| 2025-08-14T09:28Z | you@company.com | createTable | retail\_data.sales\_data.customer\_monthly\_sales | SUCCESS |