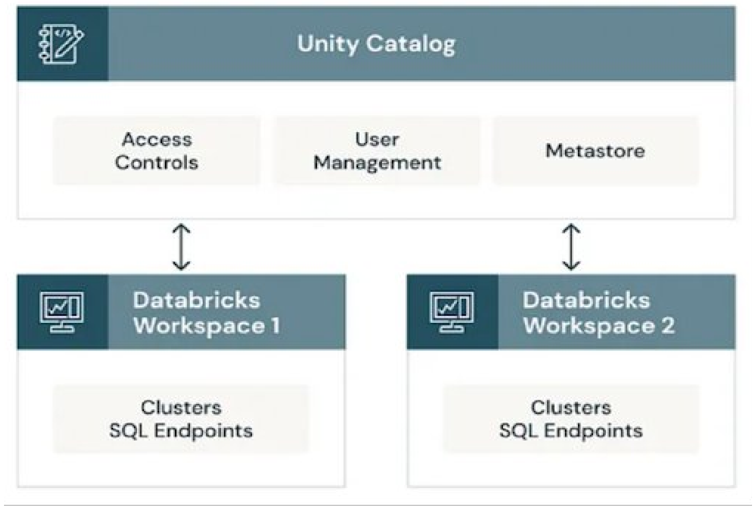
Unity Catalog

Unity Catalog is a centralized data governance solution within Databricks that provides a single place to manage data access, auditing, lineage, and discovery across all Databricks workspaces. It offers a unified view of data assets, including tables, files, and machine learning models, allowing for consistent security and access policies across an organization.



Syntax for creating catalog:

```
CREATE CATALOG IF NOT EXISTS retail_data
COMMENT 'This catalog contains all retail sales data';
```

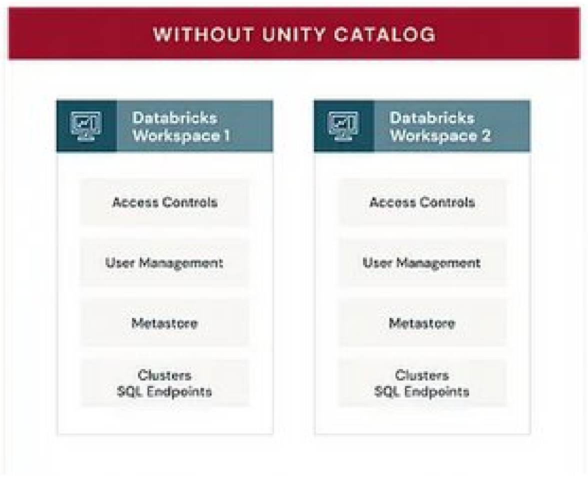
Key features of Unity Catalog include:

- **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

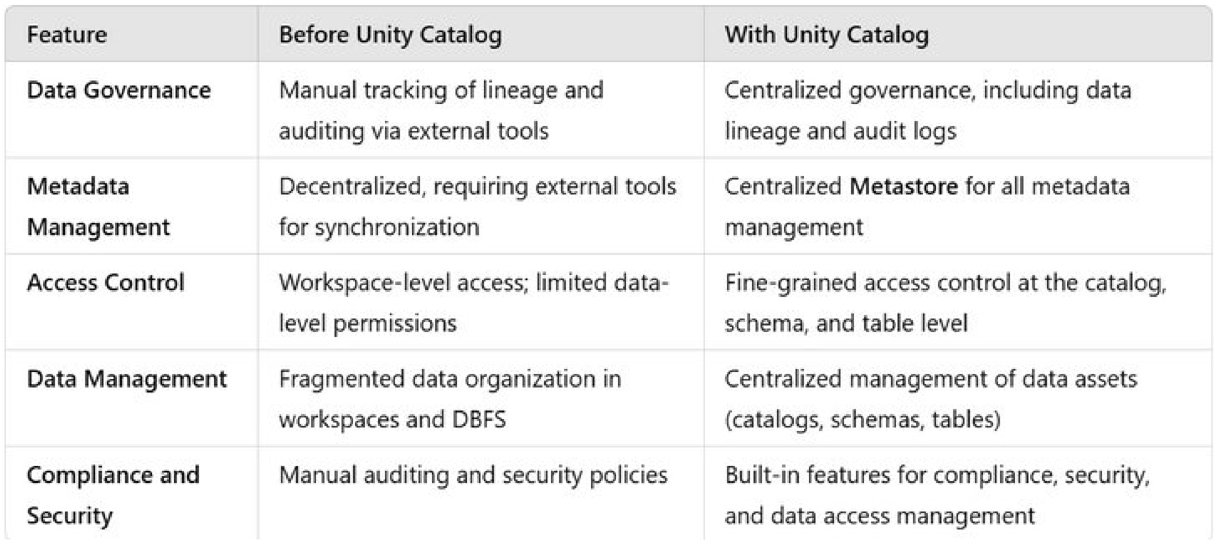
The metastore is the top-level container for metadata in Unity Catalog. It registers metadata about data and AI assets and the permissions that govern access to them. For a workspace to use Unity Catalog, it must have a Unity Catalog metastore attached. You should have one metastore for each region in 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 Databricks account.

Data Bricks without Unity Catalog:



- **Fragmented Governance**: There was no single, unified framework to ensure consistent data governance across the platform. Different tools were needed for lineage tracking, auditing, and managing metadata, often requiring significant overhead to integrate and maintain.
- **Complex Access Management**: Managing fine-grained data access at the level of individual tables, schemas, or data assets was difficult without Unity Catalog's centralized access controls. Permissions had to be manually configured for each workspace or cluster, which increased the potential for errors and security risks.
- **Lack of Metadata Centralization**: Storing and managing metadata across multiple workspaces was cumbersome. Users had to rely on external systems to manage data definitions, lineage, and schema information, leading to inconsistencies and challenges with synchronization.

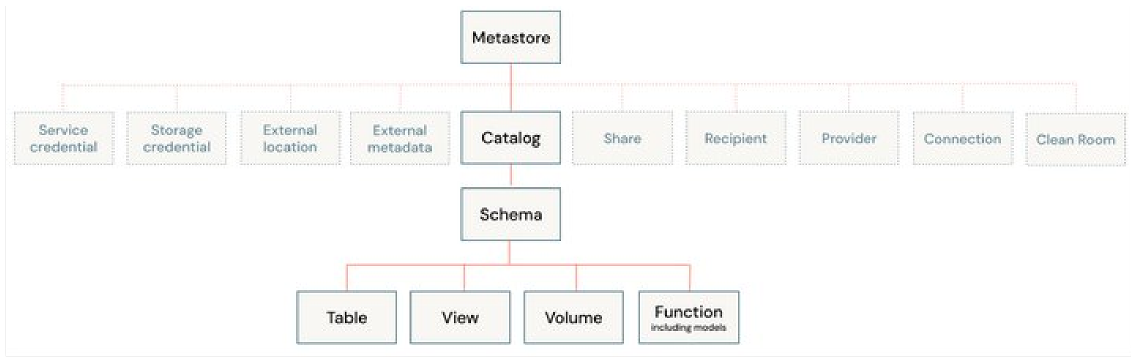
Differences



Feature	Before Unity Catalog	With Unity Catalog
Data Governance	Manual tracking of lineage and auditing via external tools	Centralized governance, including data lineage and audit logs
Metadata Management	Decentralized, requiring external tools for synchronization	Centralized **Metastore** for all metadata management
Access Control	Workspace-level access; limited data-level permissions	Fine-grained access control at the catalog, schema, and table level
Data Management	Fragmented data organization in workspaces and DBFS	Centralized management of data assets (catalogs, schemas, tables)
Compliance and Security	Manual auditing and security policies	Built-in features for compliance, security, and data access management

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:

Example:

```
CREATE CATALOG IF NOT EXISTS retail_catalog
COMMENT 'Retail sales data catalog';
```

- **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.

- **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.

Level two:

Example:

```
USE CATALOG retail_catalog;

CREATE SCHEMA IF NOT EXISTS sales_schema

COMMENT 'Sales data for all regions';
```

- **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.

Level three:

Example:

```
USE CATALOG retail_catalog;

USE SCHEMA sales_schema;

CREATE TABLE IF NOT EXISTS transactions (

 id INT,

 product STRING,

 amount DOUBLE

 )

 COMMENT 'Retail sales transactions table';
```

- **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 Databricks, but not managing access to the data in cloud storage from other clients.
- **Views** are saved queries against one or more tables.
- **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 Databricks, but not managing access to the data in cloud storage from other clients.

- **Functions** are units of saved logic that return a scalar value or set of rows.
- **Models** are AI models packaged with MLflow and registered in Unity Catalog as functions.

Unity Catalog in Databricks provides a centralized, secure, and scalable governance solution for data and AI assets. By introducing a three-level namespace (catalog → schema → table), it standardizes data organization across workspaces while enabling fine-grained access control, audit logging, and consistent data discovery. This unified approach streamlines collaboration, ensures compliance, and simplifies managing structured, unstructured, and machine learning assets in multi-cloud environments.