

Govern Unity Catalog objects

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/govern-unity-catalog-objects/1-introduction>

Introduction

Completed

Effective data governance requires more than just controlling access—it demands visibility into your data assets, accountability for how they're used, and enforcement mechanisms that scale with your organization. As data platforms grow in complexity, managing table definitions, tracking data lineage, enforcing retention policies, and sharing data securely become increasingly challenging. **Unity Catalog** in Azure Databricks provides the foundation for comprehensive governance that addresses these challenges through centralized metadata management and policy enforcement.

When you govern Unity Catalog objects, you work across several interconnected capabilities. You document tables and columns with **comments and tags** that help data consumers discover and understand your assets. You implement **attribute-based access control (ABAC)** using governed tags and policies that automatically enforce fine-grained permissions. You configure **data retention policies** using Delta Lake's `VACUUM` and predictive optimization to manage storage costs and meet compliance requirements.

Beyond access control, governance extends to **data lineage tracking** that shows how data flows through your pipelines, enabling impact analysis and troubleshooting. **Audit logging** captures who did what and when, supporting security investigations and regulatory compliance. For external collaboration, **Delta Sharing** lets you share data with partners and customers while maintaining governance controls over what they can access.

This module guides you through implementing these governance capabilities in your Azure Databricks environment. You explore techniques for preserving metadata, enforcing policies at scale, managing data lifecycle, and sharing data securely. By mastering these concepts, you position yourself to build data platforms that balance accessibility with control, enabling your organization to derive value from data while meeting its governance obligations.

2. Create and preserve table definitions

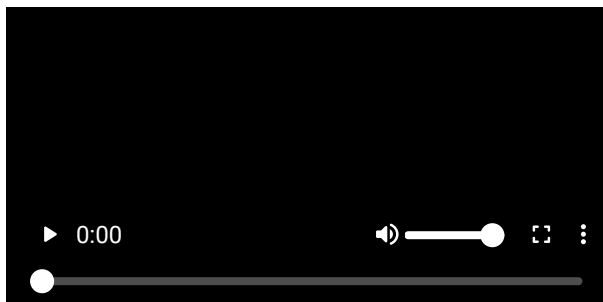
<https://learn.microsoft.com/en-us/training/modules/govern-unity-catalog-objects/2-preserve-table-column-definitions>

Create and preserve table definitions

Completed

Well-documented table and column definitions are the foundation of effective data governance. When you create and preserve comprehensive **metadata** for your data assets, you enable data consumers to discover, understand, and trust the data they need. This becomes especially important as AI-driven data discovery tools rely on high-quality metadata to surface relevant information.

In this unit, you learn how to create tables with descriptive comments and tags, and use discovery tools to explore your data assets in Unity Catalog.



Add comments to tables and columns

Comments provide human-readable descriptions that help data consumers understand the purpose and content of your data assets. You can add comments to any securable object in Unity Catalog, including catalogs, schemas, tables, views, and columns.

Consider the following example, where we add comments directly to columns as part of the `profiles` table definition:

```
CREATE TABLE sales.customers.profiles (
  customer_id  BIGINT COMMENT 'Unique identifier for each customer',
  email        STRING COMMENT 'Customer primary email address',
  created_date DATE   COMMENT 'Date when customer account was created',
  preferences  STRUCT<notifications: BOOLEAN, language: STRING>
                COMMENT 'Customer preference settings'
);
```

Add comments using SQL

Use the `COMMENT ON` statement to add or update comments on existing objects:

```
-- Add a comment to a table
COMMENT ON TABLE sales.customers.profiles
IS 'Contains customer profile information including contact details and preferences';

-- Add a comment to a column
ALTER TABLE sales.customers.profiles
ALTER COLUMN email COMMENT 'Primary email address used for account notifications';
```

You can also use markdown formatting in your comments to add structure:

```
COMMENT ON TABLE sales.orders.transactions IS
'## Order Transactions
Contains all customer orders with the following key fields:
- **order_id**: Unique order identifier
- **customer_id**: Foreign key to customers table
- **order_date**: Date order was placed';
```

Use AI-generated comments

Unity Catalog can automatically generate comments using AI-powered suggestions. This feature analyzes your table schema and column names to create initial descriptions. To generate AI comments in Catalog Explorer:

1. Open Catalog Explorer and select your table.
2. In the **About this table** panel, select **AI generate**.
3. Review the suggested comment and select **Accept** or **Edit** to modify it.

For column comments, select **AI generate** above the column list to generate suggestions for all columns.

Edit column comments



Column	Comment
customer_id int	A unique identifier assigned to each customer, allowing for easy tracking and analysis of individual customer data.
customer_name string	The full name of the customer, which can be used for personalized communication and marketing efforts.
region string	Indicates the geographical area where the customer is located, useful for regional analysis and targeted marketing strategies.
email_valid boolean	A flag indicating whether the customer's email address is valid, which is essential for effective communication and marketing campaigns.
total_orders bigint	Represents the total number of orders placed by the customer, providing insight into their purchasing behavior and loyalty.
total_spent double	The cumulative amount of money spent by the customer, which helps in assessing their overall value to the business.
avg_order_value double	Calculates the average value of orders placed by the customer, useful for understanding spending

8 comments unsaved

✓ Save all

Important

AI-generated comments are suggestions based on schema analysis. Always review these comments before saving, as AI models might generate inaccurate descriptions. Don't rely on AI comments for data classification tasks like detecting personally identifiable information (PII).

Required permissions for comments

To add or edit comments, you need:

- Ownership of the object, or
- `MODIFY` and `SELECT` privileges on the table
- `USE CATALOG` and `USE SCHEMA` on the parent catalog and schema

After adding comments, any user with the `BROWSE` privilege can view them in Catalog Explorer.

Apply tags for organization and discovery

Tags are key-value pairs that help you organize and categorize data assets in Unity Catalog. Unlike comments, which provide descriptive text, tags enable structured classification and improve search functionality.

Add tags using SQL

Use the `ALTER` command with `SET TAGS` to apply tags:

```
-- Add tags to a table
ALTER TABLE sales.customers.profiles
SET TAGS ('domain' = 'sales', 'data_classification' = 'internal');

-- Add tags to a specific column
ALTER TABLE sales.customers.profiles
ALTER COLUMN email
SET TAGS ('pii' = 'email', 'sensitivity' = 'high');
```

To remove tags, use `UNSET TAGS` :

```
ALTER TABLE sales.customers.profiles
UNSET TAGS ('domain');
```

Use governed tags

Governed tags provide centralized control over tag definitions. With governed tags, administrators define allowed keys and values at the account level, ensuring consistent classification across the organization. Governed tags appear with a lock icon in Catalog Explorer.

System tags are a special type of governed tag predefined by Azure Databricks for common use cases like data classification, ownership tracking, and lifecycle management.

Tag constraints

Keep these constraints in mind when working with tags:

- Maximum 50 tags per securable object
- Maximum 1,000 column tags per table
- Tag keys have a 255-character limit
- Tag values have a 1,000-character limit
- Tag keys are case-sensitive

Explore and discover database objects

After you create and document your data assets, you need ways to find and explore them. Unity Catalog provides several tools for data discovery.

Use Catalog Explorer

Catalog Explorer provides a visual interface for navigating your data assets. When you select a table, you can:

- View the **Columns** tab to see column names, data types, and comments
- Select the **Sample Data** tab to preview table contents
- Check the **Details** tab for table properties and metadata
- Review the **History** tab for Delta table version history

The **Insights** tab shows frequent queries and users who accessed the table in the past 30 days, helping you understand how data is being used.

Explore objects using SQL

Use SQL commands to programmatically explore your catalog:

```
-- List all catalogs available to you
SHOW CATALOGS;

-- List schemas in a catalog
SHOW SCHEMAS IN sales;

-- List tables in a schema
SHOW TABLES IN sales.customers;

-- View table details including comments
DESCRIBE TABLE EXTENDED sales.customers.profiles;

-- View table columns
SHOW COLUMNS IN sales.customers.profiles;

-- View table properties
SHOW TBLPROPERTIES sales.customers.profiles;
```

You can filter results using pattern matching:

```
-- Find tables matching a pattern
SHOW TABLES IN sales.customers LIKE 'profiles_*';
```

Search using tags

Use the workspace search bar to find tables and views by their tags. Enter tag keys or values to locate matching objects. Only objects you have at least the **BROWSE** privilege on appear in search results.

View table relationships

For tables with defined foreign keys, select **View relationships** in Catalog Explorer to display an Entity Relationship Diagram (ERD). This diagram visualizes how tables connect through primary and foreign key relationships.

Integrate with AI/BI tools

AI-powered discovery tools like **AI/BI Genie** use your table comments and tags to provide natural language search and question answering. High-quality metadata directly improves the accuracy of AI-assisted data exploration.

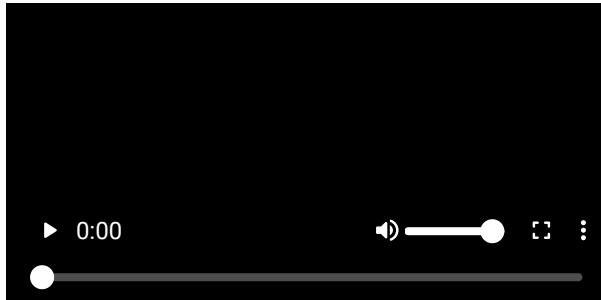
Maintaining comprehensive table and column definitions is an ongoing responsibility. As your data evolves, update your comments and tags to keep them accurate. This investment in metadata quality pays dividends through improved discoverability, governance compliance, and AI-readiness.

3. Configure ABAC with tags and policies

<https://learn.microsoft.com/en-us/training/modules/govern-unity-catalog-objects/3-configure-attribute-based-access-control-tags-policies>

Configure ABAC with tags and policies

Completed



Attribute-based access control (ABAC) provides a flexible, scalable approach to data governance in Unity Catalog. Instead of managing individual permissions for every table and user combination, you define policies based on attributes. When you tag data assets with meaningful classifications, ABAC policies automatically enforce the right access controls.

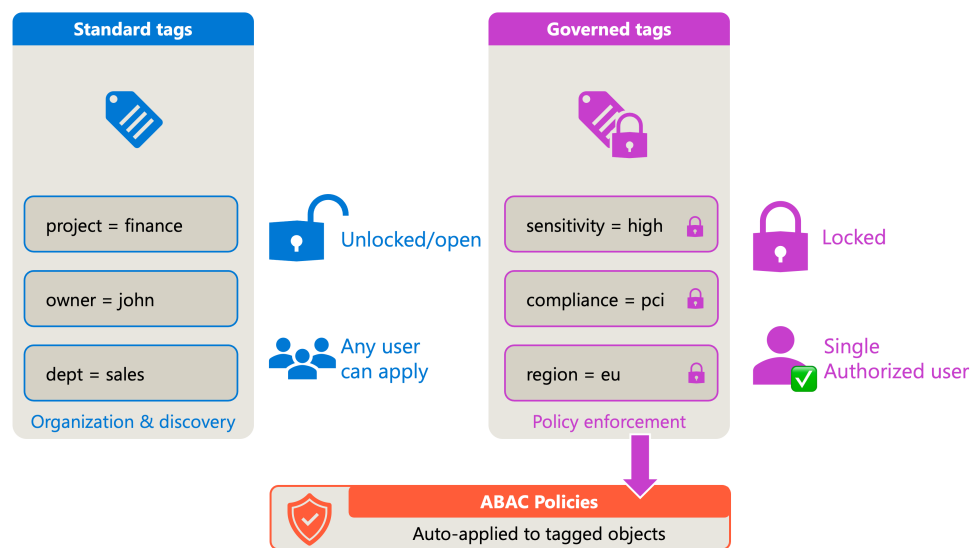
Important

ABAC in Unity Catalog is currently in **Public Preview**. Features may change before general availability.

In this unit, you learn how to use governed tags and ABAC policies to implement dynamic, centralized access control across your Unity Catalog environment.

Understand governed tags

Governed tags differ from the standard tags you've already encountered. While standard tags help with organization and discovery, governed tags add enforcement and consistency at the account level. Administrators define governed tags with specific allowed values, and only authorized users can assign them.



Consider the following differences between tag types:

Aspect	Standard tags	Governed tags
Scope	Any key-value pair	Predefined keys and allowed values
Control	Any user with <code>APPLY TAG</code> privilege	Only users with <code>ASSIGN</code> permission
Purpose	Organization and discovery	Policy enforcement and compliance
Visual indicator	None	Lock icon in Catalog Explorer

Governed tags serve as the foundation for ABAC policies. When you tag a table with `sensitivity=high`, you can then create a policy that masks certain columns for all tables with that tag. This approach scales efficiently because adding the tag to new tables automatically applies the policy.

Create governed tags

To create a governed tag, select **Catalog > Governance > Governed Tags**, then select **Create governed tag**. Enter a tag key, an optional description, and define the allowed values. Only these predefined values can be assigned to objects with this tag key.

For example, you might create a governed tag with key `pii` and allowed values `ssn`, `email`, and `address`. This ensures consistent classification of personally identifiable information across your organization.

Create governed tag



Creating a governed tag will define a tag policy with a set of allowed values and permissions for who can manage and apply the tag

* Tag key

Description

Allowed values

CancelCreate

Note

Tag data is stored as plain text. Don't use tag names or values that contain sensitive information.

Assign governed tags to objects

After creating a governed tag, apply it to columns, tables, schemas, or catalogs using the `SET TAGS` command:

```
-- Tag a column as containing SSN data
ALTER TABLE customers.profiles
ALTER COLUMN SSN
```

```
SET TAGS ('pii' = 'ssn');

-- Tag an entire table as containing sensitive data
ALTER TABLE customers.profiles
SET TAGS ('sensitivity' = 'high');
```

Governed tags applied to catalogs or schemas automatically inherit to all objects within them. This inheritance simplifies governance across large data estates.

Note

Tag inheritance applies only at the catalog and schema levels. Tags on individual table columns aren't inherited.

Create row filter policies

Row filter policies control which rows users can see based on governed tags. You define a user-defined function (UDF) that returns `TRUE` for rows the user should see and `FALSE` for rows to hide.

Before creating a policy, you need a UDF that implements your filtering logic. The following example creates a function that filters out European addresses:

```
CREATE OR REPLACE FUNCTION filter_non_eu(address STRING)
RETURNS BOOLEAN
RETURN NOT (
    LOWER(address) LIKE '%eu%' OR
    LOWER(address) LIKE '%europe%'
);
```

Now create a row filter policy that uses this function:

```
CREATE POLICY hide_eu_customers
ON CATALOG sales
ROW FILTER filter_non_eu
TO 'analysts'
FOR TABLES
MATCH COLUMNS
    hasTagValue('pii', 'address') AS addr
USING COLUMNS (addr);
```

This policy applies the `filter_non_eu` function to any table in the `sales` catalog that has a column tagged with `pii=address`. The `analysts` group only sees rows where the function returns `TRUE`.

You can also create policies using Catalog Explorer. Select the catalog or schema, choose the **Policies** tab, and select **New policy**. The visual interface guides you through selecting principals, scope, and conditions.

New policy

General

Configure which users and tables are governed by this policy.

Excluded users will remain unaffected and will still be able to use operations that requires full table access, such as cloning, Delta Sharing, and time travel.

Name*

hide_eu_customers

Description

Enter a description for this policy

Applied to...*

All account users X

Except for...

Search principals

Scope*

ms_learn X

All schemas

ms_learn (All schemas)

☒ Apply to tables that have specific tags

☒ Tags

pii : address X

Condition: hasTagValue('pii','address')

☐ Custom condition

Purpose

What should this policy do?

☐ Mask column data

Columns must have all selected tags to be masked.

☐ Hide table rows

Restrict access to individual rows in a table based on their content.

Create column mask policies

Column mask policies control what values users see in specific columns. Like row filters, these policies rely on UDFs to implement the masking logic.

Create a masking function that returns a redacted value:

```
CREATE OR REPLACE FUNCTION mask_ssn(ssn STRING)
RETURNS STRING
DETERMINISTIC
RETURN '***-**-*****';
```

Then create a column mask policy:

```
CREATE POLICY mask_sensitive_ssn
ON SCHEMA customers
COLUMN MASK mask_ssn
```

```
TO `all_users`  
EXCEPT `compliance_team`  
FOR TABLES  
MATCH COLUMNS  
  hasTagValue('pii', 'ssn') AS ssn_col  
ON COLUMN ssn_col;
```

This policy masks SSN values for all users except the compliance team. Any column in the `customers` schema tagged with `pii=ssn` displays `***-**-****` instead of the actual value.

Follow UDF best practices

The functions that power your ABAC policies run on every query against protected tables. Poorly designed UDFs can significantly impact query performance.

Follow these guidelines when writing UDFs for ABAC:

- **Keep logic simple:** Use basic `CASE` statements and boolean expressions
- **Stay deterministic:** The same input should always produce the same output
- **Avoid external calls:** Don't make API calls or lookups to other databases
- **Reference only table columns:** This enables query optimization and predicate pushdown
- **Test at scale:** Validate performance with at least 1 million rows

Avoid common performance pitfalls:

- Calling `is_account_group_member()` or `is_member()` inside UDFs
- Using complex subqueries or joins within the function
- Performing heavy regex operations on large text fields
- Including non-deterministic logic that prevents caching

Keep access control decisions in the policy definition, not the UDF. The policy specifies who the rules apply to; the UDF specifies how to transform the data.

Understand policy inheritance and scope

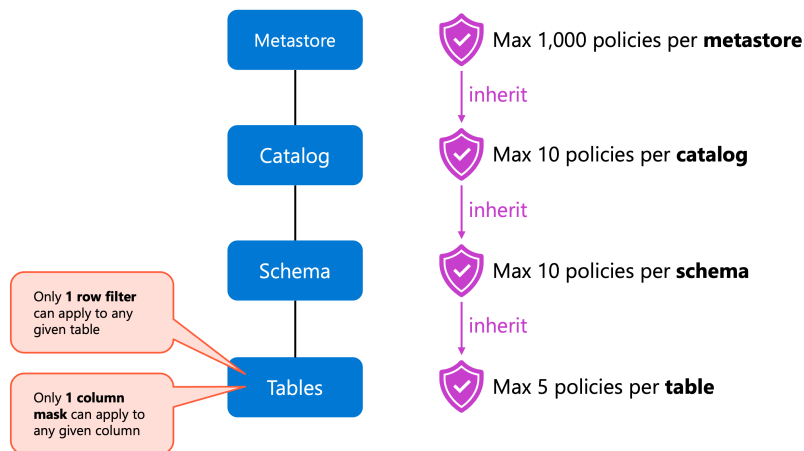
ABAC policies follow an inheritance model that simplifies governance at scale. A policy defined at the catalog level automatically applies to all schemas and tables within that catalog. Similarly, schema-level policies apply to all tables in the schema.

Policy quotas limit how many policies you can create:

- 10 policies per catalog
- 10 policies per schema
- 5 policies per table

Only one row filter can apply to any given table, and only one column mask can apply to any given column. If multiple policies would result in multiple filters or masks, Azure Databricks blocks access

and throws an error.



Important

You must use Databricks Runtime 16.4 or above, or serverless compute, to access tables secured by ABAC policies. Users not subject to the policy can use any runtime.

When you configure ABAC effectively, you create a governance model that scales with your organization. Adding the appropriate governed tag to a new table immediately brings it under the protection of existing policies, without requiring additional configuration.

4. Apply data retention policies

<https://learn.microsoft.com/en-us/training/modules/govern-unity-catalog-objects/4-apply-data-retention-policies>

Apply data retention policies

Completed

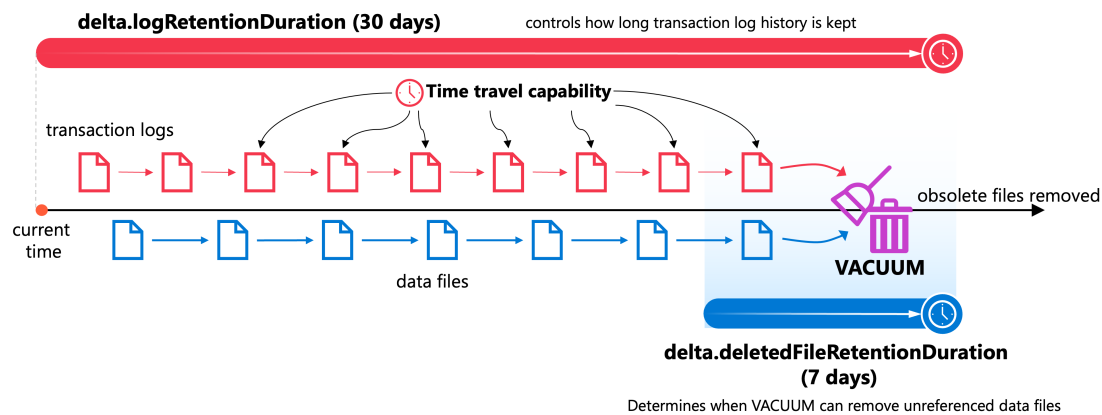
Every organization faces the challenge of balancing data availability with compliance requirements and storage costs. When you implement **data retention policies** in Azure Databricks, you ensure that data is kept only as long as necessary while remaining accessible for legitimate business needs. This becomes especially critical when regulations require you to delete personal data upon request.

In this unit, you learn how to configure retention settings, use **VACUUM** to remove obsolete data, handle deletion requests for compliance, and automate maintenance with **predictive optimization**.

Configure Delta Lake retention settings

Delta Lake maintains table history through **transaction logs** and **data files**. Two key properties control how long this historical data remains available:

Property	Default	Purpose
<code>delta.logRetentionDuration</code>	30 days	Controls how long transaction log history is kept
<code>delta.deletedFileRetentionDuration</code>	7 days	Determines when VACUUM can remove unreferenced data files



These settings work together to define your **time travel** capabilities. For example, if you need 30 days of historical data access, you must configure both properties accordingly:

```
ALTER TABLE sales.customers.transactions
SET TBLPROPERTIES (
  'delta.logRetentionDuration' = 'interval 30 days',
  'delta.deletedFileRetentionDuration' = 'interval 30 days'
);
```

Important

Increasing retention duration increases storage costs because more data files are preserved. Before adjusting these settings, evaluate your compliance requirements against storage budget constraints.

You must set both of these properties to ensure table history is retained for longer duration for tables with frequent `VACUUM` operations

To view current retention settings for a table, use the following command:

```
SHOW TBLPROPERTIES sales.customers.transactions;
```

Remove obsolete data with `VACUUM`

The `VACUUM` command removes data files that are no longer referenced by your Delta table and are older than the retention threshold. This operation permanently deletes historical data, so run it carefully.

```
-- Remove files older than the default retention period (7 days)
VACUUM sales.customers.transactions;

-- Remove files older than a specific retention period
VACUUM sales.customers.transactions RETAIN 168 HOURS;
```

When you run `VACUUM`, Delta Lake identifies files associated with versions older than the retention threshold and removes them from storage. After this operation, time travel queries to those older versions fail because the underlying data no longer exists.

For tables with **deletion vectors** enabled, you must also run `REORG TABLE ... APPLY (PURGE)` after deleting records to permanently remove the underlying data. `VACUUM` by itself does not remove data that deletion vectors have marked for deletion.

Note

Deletion vectors are a storage optimization feature you can enable on Delta Lake tables. By default, when a single row in a data file is deleted, the entire Parquet file containing the record must be rewritten. With deletion vectors enabled, it marks rows as deleted without rewriting the entire file.

Handle compliance deletion requests

Privacy regulations grant individuals the "**right to be forgotten**," requiring you to delete their personal data upon request. Delta Lake's **ACID transactions** make these deletions straightforward.

Delete individual records

Use the `DELETE` statement to remove specific records:

```
DELETE FROM bronze.users
WHERE user_id = 12345;
```

Process bulk deletion requests

For multiple deletion requests, use a `MERGE` statement with a control table that tracks pending requests:

```
MERGE INTO bronze.users AS target
USING (
  SELECT user_id
  FROM compliance.deletion_requests
  WHERE status = 'pending'
) AS source
ON target.user_id = source.user_id
WHEN MATCHED THEN DELETE;
```

After executing deletions, update your control table to mark requests as completed and run maintenance operations to permanently remove the data.

Propagate deletions through your data layers

When you delete data in your bronze layer, you must also remove it from silver and gold layers:

- **Materialized views** automatically reflect deletions after a refresh.
- **Streaming tables** require special handling because they expect append-only data. Add the `skipChangeCommits` option to ignore upstream deletions:

```
spark.readStream.option('skipChangeCommits', 'true').table("bronze.users")
```

After running deletion operations, execute `VACUUM` to permanently remove the data files from storage.

Automate retention with predictive optimization

Manually scheduling `VACUUM` and `OPTIMIZE` operations across many tables is time-consuming and error-prone. **Predictive optimization** in Unity Catalog automates these maintenance tasks for managed tables.

When enabled, predictive optimization automatically:

- Runs `VACUUM` to remove unreferenced files
- Runs `OPTIMIZE` to improve file layout and query performance
- Runs `ANALYZE` to update table statistics

Enable predictive optimization at the schema level:

```
ALTER SCHEMA sales.customers ENABLE PREDICTIVE OPTIMIZATION;
```

Tip

Before enabling predictive optimization, set your desired `delta.deletedFileRetentionDuration` on tables that require longer retention periods. The default `VACUUM` retention is 7 days, which might be shorter than your compliance requirements.

To check whether predictive optimization is enabled for a table:

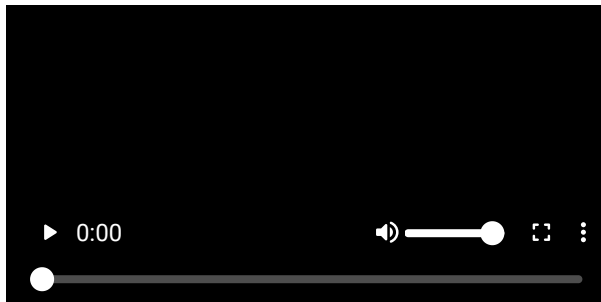
```
DESCRIBE TABLE EXTENDED sales.customers.transactions;
```

You can monitor predictive optimization operations through the system table `system.storage.predictive_optimization_operations_history`, which shows when maintenance ran, what operations completed, and associated costs.

Note

Predictive optimization is enabled by default if your account was created on or after November 11, 2024.

Integrate with Azure Storage lifecycle policies



For data stored in Azure Blob Storage, you can complement Delta Lake retention with **Azure Storage lifecycle management policies**. These policies automatically transition data to cooler storage tiers or delete blobs based on access patterns and age.

Consider using Azure lifecycle policies to:

- Move infrequently accessed historical data to **cool** or **archive** tiers
- **Delete** temporary files or staging data after a defined period
- Reduce storage costs for data that remains accessible but rarely queried

Lifecycle policies operate at the storage layer, while Delta Lake retention operates at the table layer. Use both together for a comprehensive data retention strategy.

Challenges with archiving Delta tables

When implementing storage lifecycle policies with Delta Lake, you must account for how Delta Lake manages files:

- **File creation time vs. modification time:** Azure Databricks typically relies on file creation time for lifecycle management. Operations like `UPDATE`, `MERGE`, `DELETE`, and `OPTIMIZE` rewrite Parquet files, which resets their creation time and can delay archival.
- **Querying archived data:**
 - **Offline tiers (Archive):** Querying files in offline tiers fails unless they're restored
 - **Online tiers (Cool, Cold):** Queries succeed but may incur higher access costs if data is frequently accessed.
- **Preventing frequent access:** It can be challenging to ensure users don't accidentally query archived data, which would trigger costly retrieval or failures.

Strategies for effective archiving

To overcome these challenges and implement a cost-effective archival strategy:

- **Partition data by date:** Partitioning tables by ingestion date or transaction date makes it easier to identify and manage older data files.
- **Define storage lifecycle policies:** Configure Azure Storage lifecycle management rules to move older files to cool or archive tiers based on their age (for example, move to Cool after 30 days, Archive after 365 days).
- **Set the `delta.timeUntilArchived` property:** Configure this table property to match your storage lifecycle policy. This informs Azure Databricks that files older than the specified period are archived, preventing the query optimizer from assuming they're immediately available.

```
ALTER TABLE sales_data
SET TBLPROPERTIES (delta.timeUntilArchived = '365 days');
```

- **Use views to restrict access:** Create views that filter out archived data to prevent users from accidentally querying it.

```
CREATE VIEW active_sales_data AS
SELECT * FROM sales_data
WHERE ingestion_date > CURRENT_DATE() - INTERVAL 365 DAYS;
```

- **Optimize and Vacuum with predicates:** When running maintenance commands, use predicates to avoid touching archived partitions, which would trigger retrieval.

```
OPTIMIZE sales_data
WHERE ingestion_date >= current_timestamp() - INTERVAL 30 days;
```

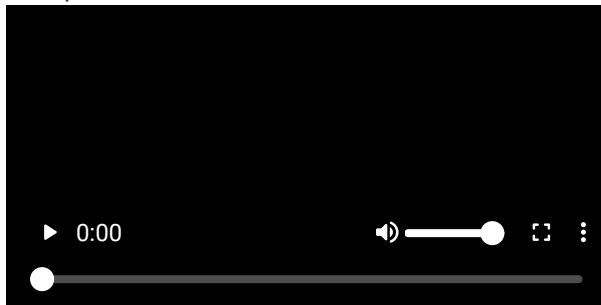
Implementing data retention policies requires ongoing attention as business requirements and regulations evolve. Establish clear processes for reviewing and updating your retention settings, and document your compliance procedures so your team can respond quickly to deletion requests.

5. Set up and manage data lineage

<https://learn.microsoft.com/en-us/training/modules/govern-unity-catalog-objects/5-manage-data-lineage-tracking>

Set up and manage data lineage

Completed

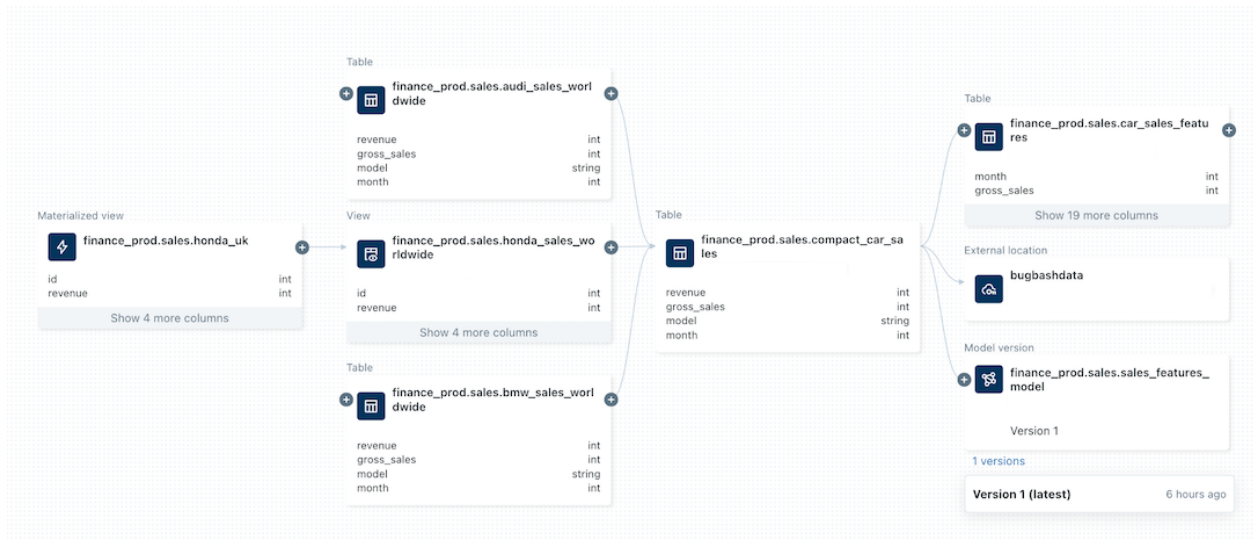


Understanding where your data comes from, how it transforms, and where it flows is essential for effective data governance. When you set up and manage **data lineage tracking** in Unity Catalog, you gain visibility into data dependencies, ownership, and history. This visibility helps you perform **impact analysis** before making changes, troubleshoot data quality issues, and meet compliance requirements.

In this unit, you learn how to use Catalog Explorer to view and manage data lineage, explore table history and ownership, and understand the connections between your data assets.

Understand data lineage in Unity Catalog

Unity Catalog automatically captures **runtime data lineage** across queries run on Azure Databricks. This lineage tracking works across all languages and captures relationships down to the **column level**. Lineage data includes the notebooks, jobs, and dashboards that interact with your tables.



Consider a scenario where a sales analytics dashboard suddenly shows incorrect revenue figures. With lineage tracking, you can trace back through the data flow to identify which upstream table or transformation caused the issue. This capability transforms troubleshooting from guesswork into a systematic investigation.

Configure compute to capture lineage

For Unity Catalog to capture lineage automatically, queries must run on **Unity Catalog-enabled compute**. This includes:

- **Compute clusters** configured with Unity Catalog access mode (shared, single user, or serverless)
- **SQL warehouses** that have Unity Catalog enabled
- **Jobs** running on Unity Catalog-enabled clusters

When you create a cluster with access to Unity Catalog, the compute automatically captures lineage for all queries that read from or write to Unity Catalog tables. This happens without requiring additional configuration or code changes, making lineage capture seamless and requiring minimal administrative effort.

Clusters that are not Unity Catalog-enabled, or queries that access tables by direct file path instead of catalog names, do not generate lineage data. To ensure comprehensive lineage coverage, configure all production workloads to use Unity Catalog-enabled compute and reference tables using their three-level namespace (`catalog.schema.table`).

Lineage is aggregated across all workspaces attached to a Unity Catalog metastore. When you capture lineage in one workspace, users in other workspaces that share the same metastore can view that lineage information. This **cross-workspace visibility** is particularly valuable for organizations with distributed data teams.

Note

Lineage data is retained for **one year**. You can filter lineage data by time frame within this window to focus on specific periods.

View data lineage using Catalog Explorer

Catalog Explorer provides a visual interface for exploring lineage relationships. To view lineage for a table:

1. In your Azure Databricks workspace, select **Catalog** in the sidebar.
2. Search or browse to locate your table.
3. Select the **Lineage** tab to display related tables in the lineage panel.
4. Select **See Lineage Graph** to open an interactive visualization of the data flow.

The lineage graph displays tables as nodes with arrows showing data flow direction. By default, one level of connections is visible. Select the **+** icon on any node to reveal more upstream or downstream connections.

When you select an arrow connecting two nodes, the **Lineage connection** panel opens. This panel shows details about the connection, including source and target tables, the notebooks that created the relationship, and the jobs that ran the transformations.

Explore column-level lineage

Column-level lineage shows how individual columns derive from source columns. This granularity helps you understand complex transformations and identify which source fields contribute to a calculated column.

To view column-level lineage, select a column in the lineage graph. The visualization highlights the upstream columns that contribute to your selected column and the downstream columns that depend on it.

View job and dashboard lineage

Beyond table relationships, Catalog Explorer shows which jobs and dashboards consume your data:

- To view **job lineage**, go to the table's **Lineage** tab, select **Jobs**, and choose **Downstream** to see jobs that consume the table.
- To view **dashboard lineage**, select **Dashboards** in the **Lineage** tab to see which dashboards query the table.

This information supports impact analysis. Before modifying a table schema, you can identify which downstream jobs and dashboards might be affected.

Manage table ownership

Every securable object in Unity Catalog has an **owner**. The owner has full privileges on the object and can grant permissions to other users. Understanding and managing ownership is essential for governance accountability.

To view or change ownership in Catalog Explorer:

1. Navigate to the table in Catalog Explorer.
2. On the **Overview** tab, locate the current owner.
3. Select the edit icon next to **Owner**.
4. Search for and select a user, group, or service principal.
5. Select **Save**.

Set owner for ms_learn.pipeline_lab.customers_bronze ✕

Type to search...

Cancel Save

Only the current owner or a **metastore admin** can transfer ownership. After transfer, the previous owner loses ownership privileges unless explicitly granted.

To view lineage, users need at least the **BROWSE** privilege on the parent catalog. Objects that a user doesn't have permission to access appear as **masked nodes** in the lineage graph. This security model ensures that lineage visibility respects your access control policies.

View table history and version changes

For **Delta tables**, Catalog Explorer displays a complete history of all operations. The **History** tab shows each version with timestamps, operation types, and the user who made the change.

You can also query table history using SQL:

```
-- View full table history
DESCRIBE HISTORY sales.customers.orders;

-- View only the most recent operation
DESCRIBE HISTORY sales.customers.orders LIMIT 1;
```

The history includes detailed information about each operation:

Column	Description
version	Table version number
timestamp	When the operation occurred
operation	Type of operation (WRITE, UPDATE, DELETE, MERGE)
userName	User who performed the operation
operationMetrics	Metrics like rows affected and files modified

This history supports auditing and compliance by providing a complete record of who changed what and when.

Query lineage data programmatically

For reporting and advanced analysis, you can query the **lineage system tables** directly. Unity Catalog provides two lineage tables:

- `system.access.table_lineage` records read and write events at the table level
- `system.access.column_lineage` tracks column-level dependencies

The following query finds the most frequently accessed tables in the past week:

```
SELECT
  source_table_full_name,
  COUNT(DISTINCT event_id) AS query_count
FROM system.access.table_lineage
WHERE event_date > CURRENT_DATE() - INTERVAL 7 DAYS
  AND source_table_full_name IS NOT NULL
GROUP BY source_table_full_name
ORDER BY query_count DESC
LIMIT 10;
```

You can also use *Databricks Assistant* to explore lineage interactively. From Catalog Explorer, select the Assistant icon and type:

- `/getTableLineages` to view upstream and downstream dependencies
- `/getTableInsights` to access metadata-driven insights like user activity patterns

These queries help answer questions such as "who queries this table most often" or "what are the downstream consumers of this data."

Understand lineage limitations

Lineage tracking has certain constraints you should be aware of:

- Lineage is **not preserved for renamed objects**. If you rename a table, the lineage connection to its previous name is lost.
- Tables accessed by path rather than name (for example, `delta.'abfss://...'`) show only the path in lineage, not the table name.
- Global temporary views and system tables under `system.information_schema` are not captured in lineage.
- Some user-defined functions can obscure column-level lineage by hiding the mapping between source and target columns.

Despite these limitations, lineage tracking provides comprehensive visibility for most data engineering workflows. By understanding these constraints, you can design your pipelines to maximize lineage capture.

Effective data lineage management transforms how you govern and troubleshoot your data platform. With the visibility that Catalog Explorer and system tables provide, you can confidently manage dependencies, assess change impact, and maintain accountability across your data assets.

6. Configure audit logging

<https://learn.microsoft.com/en-us/training/modules/govern-unity-catalog-objects/6-configure-audit-logging>

Configure audit logging

Completed



Tracking who did what, when, and to which data objects is essential for governance, security, and compliance. When you configure **audit logging** in Azure Databricks, you gain visibility into every significant action across your workspace and account. This visibility helps you meet regulatory requirements, investigate security incidents, and maintain accountability across your data platform.

Unity Catalog audit logging is **configured at the account level** and automatically applies to all workspaces within that account. This account-level configuration ensures consistent audit capture

across your entire Databricks environment without requiring per-workspace setup.

In this unit, you learn how to access and query audit logs, understand what events are captured, and use these logs for common governance scenarios.

Understand the audit log system table

Azure Databricks automatically captures audit events and stores them in the **audit log system table** located at `system.access.audit`. This table records actions across all workspaces attached to your Unity Catalog metastore, providing a **centralized view** of activity across your organization.

Consider a scenario where your security team needs to investigate unauthorized data access. Rather than searching through multiple log files, you query a single table that contains every tracked event. This centralized approach transforms compliance investigations from manual searches into straightforward SQL queries.

The audit log system table uses a consistent schema that includes the following key columns:

Column	Description
<code>event_time</code>	Timestamp when the action occurred
<code>event_date</code>	Calendar date of the action
<code>user_identity</code>	Identity of the user who initiated the action
<code>workspace_id</code>	The Azure Databricks workspace in which the audit event took place
<code>service_name</code>	Service category (for example, <code>unityCatalog</code> , <code>notebook</code> , <code>clusters</code>)
<code>action_name</code>	Specific action performed (for example, <code>getTable</code> , <code>createTable</code>)
<code>request_params</code>	Parameters included in the request
<code>response</code>	Response details including status codes and results
<code>source_ip_address</code>	IP address where the request originated

Account-level events record `workspace_id` as `0`, while workspace-level events include the actual workspace identifier. This distinction helps you filter events by scope when investigating issues.

Note

Azure Databricks retains audit logs for up to **one year** for security and fraud analysis purposes. Plan your long-term retention strategy if you need logs beyond this period.

Identify tracked events and services

The audit log captures events across numerous services within Azure Databricks. Understanding which actions are logged helps you design effective monitoring and compliance strategies.

Workspace-level services log events for actions taken within a specific workspace:

- **Unity Catalog:** Table creation, deletion, updates, and permission changes
- **Notebooks:** Commands executed, notebooks created, modified, or deleted
- **Clusters:** Cluster creation, resizing, and termination
- **Jobs:** Job creation, runs, and failures
- **Databricks SQL:** Query execution, warehouse management, and alert activity
- **Accounts:** User logins, token generation, and access control changes

Account-level services capture events that span **multiple workspaces**:

- **Account management:** User and group management across the account
- **Billable usage:** Access to usage reports and aggregated metrics
- **Delta Sharing:** Provider and recipient actions for shared data

For example, when a user queries a table in Unity Catalog, the `unityCatalog` service logs a `getTable` action. If the same user runs a command in a notebook, the `notebook` service logs a `runCommand` action. Each service provides specific action names that describe precisely what occurred.

Tip

For a comprehensive reference of audit log services and events, consult this article [Diagnostic log reference](#)

Query audit logs for common scenarios

You can write SQL queries against the audit log system table to answer common governance questions. The following examples demonstrate patterns you can adapt for your organization's needs.

Find who accessed a specific table

This query identifies users who accessed a particular table in the past seven days:

```
SELECT
  user_identity.email AS user_email,
  action_name AS access_type,
  event_time AS access_time
FROM system.access.audit
WHERE
  request_params.full_name_arg = 'catalog.schema.table_name'
  AND action_name IN ('getTable', 'createTable', 'deleteTable')
```

```
AND event_date > CURRENT_DATE() - INTERVAL 7 DAYS
ORDER BY event_time DESC;
```

Track permission changes across your metastore

This query shows all permission modifications, helping you audit access control changes:

```
SELECT
  event_time,
  user_identity.email AS changed_by,
  request_params.securable_type,
  request_params.securable_full_name,
  request_params.changes
FROM system.access.audit
WHERE
  service_name = 'unityCatalog'
  AND action_name = 'updatePermissions'
ORDER BY event_time DESC;
```

Identify failed login attempts

Security investigations often require finding authentication failures. This query surfaces denied access attempts:

```
SELECT
  event_time,
  user_identity.email AS user_email,
  source_ip_address,
  action_name,
  response.error_message
FROM system.access.audit
WHERE
  service_name = 'accounts'
  AND response.status_code != 200
  AND event_date > CURRENT_DATE() - INTERVAL 30 DAYS
ORDER BY event_time DESC;
```

These queries form a foundation for your monitoring strategy. You can extend them to create alerts, build dashboards, or export data to external security information and event management (SIEM) platforms.

Enable verbose audit logs

By default, Azure Databricks logs high-level events but doesn't capture the full text of commands executed in notebooks. When you need detailed visibility into exactly what code ran and when, enable **verbose audit logs**.

With verbose logging enabled, the audit log captures additional events:

Service	Action	Description
notebook	runCommand	Logs the full command text executed in a notebook cell
jobs	runCommand	Logs commands executed by job runs
databrickssql	commandSubmit	Logs SQL commands submitted to warehouses

To enable verbose audit logs:

1. Sign in to your Azure Databricks workspace as an administrator.
2. Navigate to **Admin Settings** by selecting your username and choosing **Admin Settings**.
3. Select the **Advanced** tab.
4. Locate **Verbose Audit Logs** and enable the feature.

Settings

 Workspace admin

Appearance

Identity and access

Security

Compute

Development

Notifications

Advanced

 User

Profile

Preferences

Developer

Linked accounts

Notifications

DBFS File Browser

Enable or disable DBFS File Browser

Off 

Databricks Autologging

Enable or disable Databricks Autologging for this workspace. When enabled, ML model training runs executed interactively on clusters with supported versions of the Databricks Runtime for Machine Learning will automatically be logged to MLflow.

On 

[Learn more](#) 

Legacy MLflow Model Serving

Enable or disable legacy MLflow model serving for this workspace. Disabling this option will not disable existing model serving.

On 

Verbose Audit Logs

Enable or disable verbose audit logs.

Off 

FileStore Endpoint

Enable or disable FileStore endpoint /files.

FileStore files are accessible at /files. When enabled, names of the files stored in FileStore have to be considered public.

On 

[Learn more](#) 

When you enable or disable verbose logging, Azure Databricks logs this configuration change as an auditable event. This creates accountability for who modified the logging configuration and when.

Important

Verbose logs capture command text, which might include sensitive information such as query parameters or hardcoded values. Ensure your security policies account for this additional data exposure.

Configure log delivery to external systems

While the audit log system table provides direct access within Databricks, many organizations route logs to external systems for centralized monitoring. Azure Databricks integrates with Azure diagnostic settings to deliver logs to multiple destinations, each optimized for different use cases:

- **Azure Log Analytics:** Provides near-real-time ingestion (typically within 15 minutes), centralized storage, and powerful querying using Kusto Query Language (KQL). This makes it ideal for fast analysis, interactive investigations, and building custom dashboards for auditing Unity Catalog access events.
- **Azure Storage accounts:** Offers low-cost, long-term archival and batch analysis. Best suited for compliance scenarios where logs are accessed infrequently but must be retained for extended periods.
- **Azure Event Hubs:** Enables real-time streaming to downstream consumers and processing systems. Ideal when you need to integrate audit logs with existing event-driven architectures or SIEM platforms.

Platform administrators configure log delivery at the account level through the Azure portal. When building monitoring solutions, account for typical ingestion latency of approximately **15 minutes** in your alerting thresholds.

Microsoft Azure

Search resources, services, and docs (G+)

[Home](#) >
[ms-learn](#) |
[Diagnostic settings](#) >

Diagnostic setting

Save

Discard

Delete

Feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name *

Logs

Category groups ⓘ

☐ allLogs

Categories

☐ Databricks File System

☐ Databricks Clusters

☐ Databricks Accounts

☒ Databricks Jobs

☒ Databricks Notebook

☐ Databricks SSH

Destination details

☒ Send to Log Analytics workspace

Subscription

ms-learn

Log Analytics workspace

No workspaces in this subscription.

☐ Archive to a storage account

☐ Stream to an event hub

☐ Send to partner solution

The combination of the system table for interactive queries and external delivery for operational monitoring provides comprehensive coverage. You can use the system table for ad-hoc investigations while relying on your SIEM platform for continuous security monitoring.

Understanding audit logging capabilities positions you to support compliance requirements and respond quickly to security incidents. With the audit log system table, you have a powerful tool for tracking activity, investigating issues, and demonstrating governance controls to stakeholders.

7. Design secure Delta Sharing strategy

<https://learn.microsoft.com/en-us/training/modules/govern-unity-catalog-objects/7-secure-delta-sharing-strategy>

Design secure Delta Sharing strategy

Completed

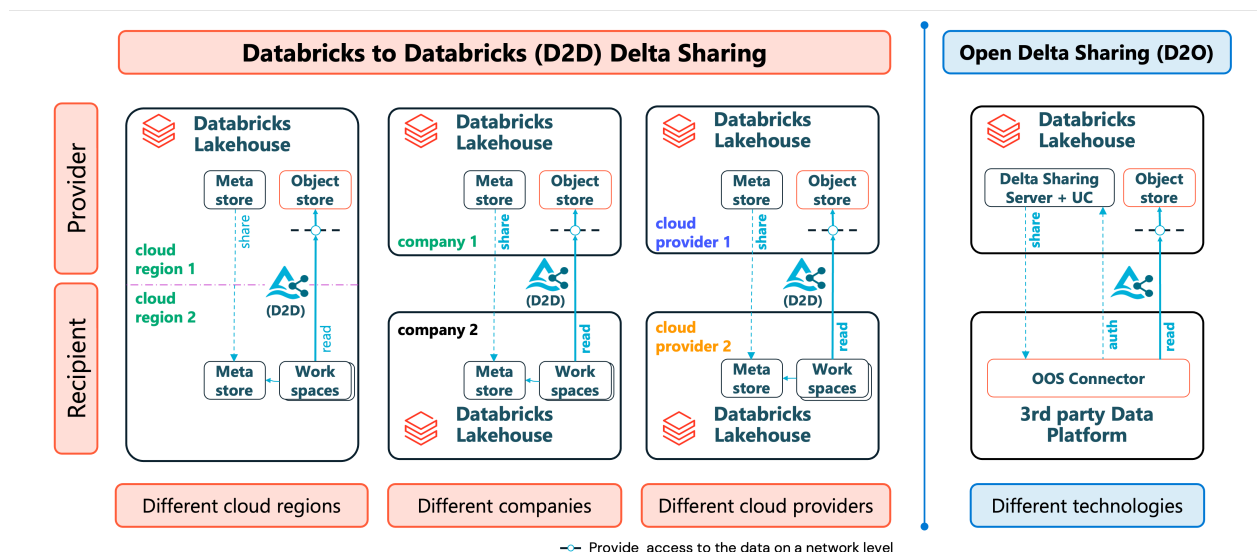


Sharing data securely with external partners, other teams, or across organizational boundaries requires a deliberate strategy. **Delta Sharing** in Azure Databricks lets you share live data without copying it, while maintaining full control over who accesses what. Designing a secure sharing strategy means choosing the right authentication method, applying appropriate access controls, and establishing monitoring practices that keep your data governance intact.

In this unit, you learn how to evaluate sharing protocols, implement security controls, and design fine-grained access patterns that align with your organization's compliance requirements.

Choose the right sharing protocol

Delta Sharing offers two primary protocols, each suited for different recipient scenarios. Your choice affects how authentication works and what assets you can share.



Databricks-to-Databricks sharing is the preferred option when recipients have access to a Unity Catalog-enabled Azure Databricks workspace. This protocol handles all security automatically through the platform. Recipients provide their **sharing identifier**—a string containing their metastore's cloud, region, and unique ID—and the connection establishes without token management. You can share tables, views, volumes, notebook files, and AI models through this approach.

Open sharing works when recipients use platforms other than Azure Databricks or lack access to Unity Catalog. You manage authentication by generating **bearer tokens** or configuring **OIDC**

(OpenID Connect) federation. While more flexible, this approach requires careful token lifecycle management and supports only tabular data.

Consider these factors when selecting your protocol:

Factor	Databricks-to-Databricks	Open sharing
Recipient platform	Databricks with Unity Catalog	Any platform
Authentication	Platform-managed	Token or OIDC
Shareable assets	Tables, views, volumes, notebook files, models	Tables only
Token management	Not required	Required
Performance	Optimized with history sharing	Standard

When recipients exist on both Databricks and non-Databricks platforms, you can create multiple recipient objects for the same logical organization—one for each access pattern.

Design authentication and access controls

A secure sharing strategy starts with proper authentication configuration and follows the principle of **least privilege**.

For **open sharing recipients**, configure **token expiration** to limit exposure if credentials are compromised. Set the default token lifetime at the metastore level:

1. Navigate to the account console and select your metastore.
2. Enable **Allow Delta Sharing with parties outside your organization**.
3. Set an expiration period rather than allowing tokens to live indefinitely.

For recipients who shouldn't rotate tokens frequently, consider **OIDC federation** instead. This approach grants short-lived Databricks OAuth tokens in exchange for JWT tokens from the recipient's identity provider, removing long-lived credentials entirely.

Restrict network access for sensitive shares by assigning **IP access lists** to open sharing recipients. You can limit access to specific IP addresses or CIDR ranges (Classless Inter-Domain Routing):

```
-- View recipient details including IP restrictions
DESCRIBE RECIPIENT finance_partner;
```

Each recipient supports up to 100 IP/CIDR values. When a request originates from an unlisted address, Delta Sharing denies access and logs the attempt.

Delegate sharing privileges carefully. Rather than granting metastore admin access broadly, use specific privileges:

- **CREATE SHARE** : Allows creating new shares
- **CREATE RECIPIENT** : Allows creating new recipients

- `USE SHARE` and `SET SHARE PERMISSION` : Together allow granting share access to recipients
- `USE RECIPIENT` : Allows viewing recipient details

Assign these privileges to groups representing your data stewardship team rather than individuals.

Configure shares and recipients

After designing your security approach, you need to create the actual share and recipient objects in Unity Catalog. This section walks through the configuration steps for both sharing protocols.

Create a share

A **share** is a securable object that contains a collection of tables, views, volumes, notebooks, and models you want to share. Create a share using SQL or Catalog Explorer:

```
-- Create a new share
CREATE SHARE IF NOT EXISTS partner_analytics
COMMENT 'Sales analytics data for partner integration';
```

Add assets to a share

Once created, add the data assets recipients should access. You can share individual tables, views, or entire schemas:

```
-- Add a table to the share
ALTER SHARE partner_analytics ADD TABLE catalog.schema.sales_summary;

-- Add a table with an alias (useful when internal naming differs from external)
ALTER SHARE partner_analytics ADD TABLE catalog.schema.internal_metrics
AS external_catalog.analytics.metrics;

-- Share a table with history for time travel queries
ALTER SHARE partner_analytics ADD TABLE catalog.schema.transactions
WITH HISTORY;

-- Add an entire schema (includes all current and future assets)
ALTER SHARE partner_analytics ADD SCHEMA catalog.analytics_schema;
```

For Databricks-to-Databricks sharing, you can also add volumes and views:

```
-- Add a volume
ALTER SHARE partner_analytics ADD VOLUME catalog.schema.documentation;

-- Add a view
ALTER SHARE partner_analytics ADD VIEW catalog.schema.aggregated_sales;
```

Note

To add notebook files to a share, use Catalog Explorer. Navigate to **Delta Sharing > Shared by me**, select your share, click **Manage assets > Add notebook file**, and browse for the notebook you want to share.

Create a recipient for Databricks-to-Databricks sharing

For recipients with Unity Catalog-enabled workspaces, request their **sharing identifier** first. This string contains their metastore's cloud, region, and UUID:

```
-- Create a recipient using their sharing identifier
CREATE RECIPIENT IF NOT EXISTS contoso_analytics
USING ID 'azure:westeurope:a1b2c3d4-e5f6-7890-abcd-ef1234567890'
COMMENT 'Contoso data analytics team';
```

The recipient can find their sharing identifier by running:

```
SELECT CURRENT_METASTORE();
```

Create a recipient for open sharing

For recipients without Databricks access, create a token-authenticated recipient:

```
-- Create an open sharing recipient
CREATE RECIPIENT IF NOT EXISTS external_partner
COMMENT 'External analytics platform integration';
```

After creation, retrieve the **activation link** to send to your recipient:

```
DESCRIBE RECIPIENT external_partner;
```

The activation link allows the recipient to download their credential file once. Share this link through a secure channel.

Grant recipients access to shares

After creating both the share and recipient, grant access:

```
-- Grant a recipient access to a share
GRANT SELECT ON SHARE partner_analytics TO RECIPIENT contoso_analytics;

-- View current grants on a share
SHOW GRANT ON SHARE partner_analytics;

-- View shares accessible to a recipient
SHOW GRANTS TO RECIPIENT contoso_analytics;
```

Manage shares and recipients

Use these commands to view and manage your sharing configuration:

```
-- List all shares
SHOW SHARES;

-- View share details including assets
DESCRIBE SHARE partner_analytics;
SHOW ALL IN SHARE partner_analytics;

-- List all recipients
SHOW RECIPIENTS;

-- Remove a table from a share
ALTER SHARE partner_analytics REMOVE TABLE catalog.schema.deprecated_table;

-- Revoke recipient access
REVOKE SELECT ON SHARE partner_analytics FROM RECIPIENT former_partner;

-- Delete a recipient
DROP RECIPIENT former_partner;
```

Implement fine-grained access with dynamic views

Standard table sharing grants recipients access to all rows and columns. When you need to restrict what specific recipients see, use **dynamic views** that filter data based on recipient properties.

Attach **custom properties** to recipients that correspond to your data segmentation:

```
-- Set country_region property for partition filtering
ALTER RECIPIENT regional_partner SET PROPERTIES ('country_region' = 'US');
```

Create a dynamic view that filters rows based on the recipient's property:

```
CREATE VIEW catalog.schema.regional_sales AS
SELECT * FROM catalog.schema.all_sales
WHERE region = CURRENT_RECIPIENT('country_region');
```

For column-level security, mask sensitive data for recipients who shouldn't see it:

```
CREATE VIEW catalog.schema.customer_data AS
SELECT
  customer_id,
  CASE
    WHEN CURRENT_RECIPIENT('access_level') = 'full' THEN email
    ELSE 'REDACTED'
  END AS email,
```

```
purchase_history
FROM catalog.schema.customers;
```

When you share this dynamic view, each recipient sees only the data their properties permit. The provider can't query views using `CURRENT_RECIPIENT()` directly—test by sharing with yourself as a recipient.

For simpler scenarios, **partition filtering** lets you share specific slices of a table without creating views:

```
ALTER SHARE partner_share ADD TABLE inventory
PARTITION (region = CURRENT_RECIPIENT('region'));
```

Monitor sharing activity through audit logs

Tracking who accesses shared data and when is essential for compliance and security investigations. Azure Databricks captures Delta Sharing events in the **audit log system table**.

Query the audit logs to understand sharing patterns:

```
-- Track share and recipient creation
SELECT
  event_time,
  user_identity.email AS performed_by,
  action_name,
  request_params
FROM system.access.audit
WHERE service_name = 'unityCatalog'
  AND action_name IN ('createShare', 'createRecipient', 'grantOnShare')
ORDER BY event_time DESC;
```

Monitor recipient access to shared tables:

```
-- View recipient data access events
SELECT
  event_time,
  request_params.recipient_name,
  request_params.share_name,
  action_name
FROM system.access.audit
WHERE service_name = 'unityCatalog'
  AND action_name IN ('deltaSharingQueriedTable', 'generateTemporaryTableCredential')
  AND event_date > CURRENT_DATE() - INTERVAL 7 DAYS;
```

The `deltaSharingQueriedTable` event appears for open sharing queries, while `generateTemporaryTableCredential` appears for Databricks-to-Databricks access with **history sharing**. Both events include details about what was accessed and by whom.

Establish regular review cycles for your sharing configuration:

- Audit active recipients monthly to confirm continued business need
- Review share contents to ensure only necessary data is exposed
- Verify token expiration settings haven't been modified
- Check for failed access attempts that might indicate credential compromise

Tip

Configure log delivery to **Azure Event Hubs** or **Log Analytics** if you need real-time alerting on suspicious sharing activity.

Apply security best practices

Building a robust Delta Sharing strategy requires combining technical controls with operational practices.

Document sharing agreements before configuring access. Establish what data is shared, for what purpose, and for how long. This documentation supports compliance audits and helps when reviewing whether shares should continue.

Share only what's necessary. When adding tables to a share, consider whether recipients need the entire table or just specific partitions. Use schema sharing with caution—it automatically includes future tables added to that schema.

Combine controls for defense in depth. A sensitive share might include:

- Databricks-to-Databricks authentication for platform-managed security
- Dynamic views for row-level filtering
- Recipient properties for parameterized access
- Regular audit log reviews

Plan for credential rotation. For open sharing recipients, implement processes to **rotate tokens** before expiration. Rotating a token invalidates the previous one after the old token's expiration, giving recipients time to update their configuration.

Review shares when personnel changes. When a recipient organization restructures, verify that the current sharing identifier still represents the appropriate metastore and that access should continue.

By designing your Delta Sharing strategy around these principles, you create a framework for secure collaboration that scales with your organization's needs while maintaining the governance controls your compliance requirements demand.

8. Exercise - Govern Unity Catalog Objects

<https://learn.microsoft.com/en-us/training/modules/govern-unity-catalog-objects/8-exercise-govern-unity-catalog-objects>

Exercise - Govern Unity Catalog Objects

Completed

Now it's your chance to govern Unity Catalog objects. In this lab, you apply Unity Catalog governance controls to a connected vehicle data platform built on Azure Databricks. You use SQL to tag tables and columns for PII classification, configure Delta Lake retention policies and run VACUUM to purge deleted data, and enable predictive optimization. You then query system tables to trace data lineage programmatically and analyze the audit log to answer compliance questions about who accessed data and when.

Note

To complete this lab, you need an [Azure subscription](#) in which you have administrative access.

Launch the exercise and follow the instructions.

[Launch Exercise](#)

9. Module assessment

<https://learn.microsoft.com/en-us/training/modules/govern-unity-catalog-objects/9-knowledge-check>

Module assessment

Completed

10. Summary

Summary

Completed

Governing Unity Catalog objects requires a comprehensive approach that spans metadata management, access control, compliance, and collaboration. Throughout this module, you explored the tools and techniques that enable effective data governance in Azure Databricks—from documenting your data assets with **comments and tags** to enforcing access policies through **attribute-based access control (ABAC)**. You learned how **governed tags** enable policies that scale automatically as new tables are added to your catalog.

Data lifecycle management emerged as a critical governance capability. You configured **Delta Lake retention settings** and used **VACUUM** to remove obsolete data files. You handled **compliance deletion requests**, propagating deletions through bronze, silver, and gold data layers. **Predictive optimization** automates these maintenance tasks, reducing operational overhead while ensuring storage efficiency.

Understanding your data's journey proved essential for both troubleshooting and compliance. **Data lineage tracking** in Catalog Explorer reveals how data flows through your pipelines, enabling impact analysis before making changes. **Audit logging** through the system table provides visibility into who accessed what data and when, supporting security investigations and regulatory audits. For external collaboration, **Delta Sharing** offers secure data exchange with partners and customers, with options for both Databricks-to-Databricks sharing and open protocol sharing.

Apply these governance practices incrementally as you build your data platform. Start by documenting tables and columns with meaningful comments and tags. Implement ABAC policies for sensitive data classifications. Configure retention policies that balance compliance requirements with storage costs. Review lineage and audit logs regularly to maintain visibility into your data operations. These foundational practices create a governance framework that scales with your organization's data needs.