Day 8 - Unity Catalog Governance

Catalog → Schema → Table Hierarchy

This is the foundation of Unity Catalog.

🧱 *What is the hierarchy?*

Think of it like a folder structure 👇

Catalog
└── Schema (Database)
└── Tables / Views / Functions

🧠 *Real-world analogy*

- *Catalog → Company / Project*
- *Schema → Department*
- *Table → Files inside the department*

1. Catalog

- *Top-level container in Unity Catalog*
- *Used to logically separate data*
- *Examples:*
  - *dev_catalog*
  - *prod_catalog*
  - *sales_catalog*

📌 *Why catalogs?*

- *Environment isolation (dev / prod)*
- *Better governance & security*
- *Easy access control*

```
CREATE CATALOG ecommerce_catalog;
```

2 .Schema (Database)

- *Inside a catalog*
- *Groups related tables*

```
CREATE SCHEMA
ecommerce_catalog.bronze;
CREATE SCHEMA
ecommerce_catalog.silver;
CREATE SCHEMA ecommerce_catalog.gold;
```

📌 *This fits perfectly with Medallion Architecture (we learned on Day 6.)*

3. Table

- *Actual data stored here*
- *Can be managed or external*

```
CREATE TABLE
ecommerce_catalog.bronze.orders (
  order_id INT,
  order_date DATE,
  amount DOUBLE
);
```

Full Table Name :

```
catalog.schema.table
```

Example :

```
ecommerce_catalog.bronze.orders
```

✅ Why this hierarchy is IMPORTANT?

✔ *Fine-grained access control*
✔ *Clear data ownership*
✔ *Data lineage tracking*
✔ *Enterprise-level governance*

Without Unity Catalog → Databricks only had database.table
With Unity Catalog → catalog.schema.table (much powerful)

Access Control (GRANT / REVOKE)

Unity Catalog gives fine-grained security — not just "who can see data", but who can do what.

🧠 *Core idea*
Access is controlled at multiple levels:

Catalog → Schema → Table → View → Column

You can:
- *Allow read-only*
- *Allow write*
- *Block completely*

🟢 Types of privileges (important ones)

Privilege	Meaning
USAGE	Can see the object
SELECT	Can read data
INSERT	Can add data
UPDATE	Can modify data
DELETE	Can delete data
CREATE	Can create objects
ALL PRIVILEGES	Full control

📌 *USAGE is mandatory*
Without USAGE, even SELECT won't work.

🟢 Grant access (examples)

1️⃣ *Grant catalog access*

```
GRANT USAGE ON CATALOG
ecommerce_catalog TO `data_analyst`;
```

2️⃣ *Grant schema access*

```
GRANT USAGE ON SCHEMA
ecommerce_catalog.gold TO
`data_analyst`;
```

3️⃣ *Grant table access*

```
GRANT SELECT ON TABLE
ecommerce_catalog.gold.sales_summary
TO `data_analyst`;
```

Now: ✔ Analyst can only read gold data
❌ Cannot modify / delete

🟢 Grant access (examples)

4️⃣ *Grant write access (engineers)*

```
GRANT ALL PRIVILEGES
ON SCHEMA ecommerce_catalog.silver
TO `data_engineer`;
```

🔴 Revoke access

```
REVOKE SELECT
ON TABLE
ecommerce_catalog.gold.sales_summary
FROM `data_analyst`;
```

Access removed ❌

Roles / Groups (Best Practice)

Instead of users → use groups

Examples:
- *data_analyst*
- *data_engineer*
- *admin*

Grant once → applies to all users in group.

Data Lineage

🧠 *What is Data Lineage?*

Data lineage shows:
Where data comes from → how it transforms → where it is used

In simple words:
👉 *"Who created this data, from what, and how?"*

🧩 Why Data Lineage is IMPORTANT?

✔ *Understand data flow*
✔ *Debug wrong numbers*
✔ *Impact analysis before changes*
✔ *Audit & compliance*
✔ *Trust in dashboards*

This is enterprise-grade feature of Unity Catalog 💎

Example (E-commerce scenario)

Raw CSV (DBFS)

↓

Bronze.orders

↓

Silver.orders_cleaned

↓

Gold.sales_summary

↓

Dashboard

Unity Catalog automatically tracks this.

How lineage works in Databricks

Lineage is captured when:

- *Tables are created using CTAS*
- *Data is transformed using SQL / Spark*
- *Views are created from tables*

Example:

```
CREATE TABLE
ecommerce_catalog.silver.orders_cleane
d AS
SELECT order_id, amount
FROM ecommerce_catalog.bronze.orders;
```

➡ *Databricks records:*

- *Source: bronze.orders*
- *Target: silver.orders_cleaned*

Lineage with Views

```
CREATE VIEW
ecommerce_catalog.gold.sales_view AS
SELECT SUM(amount) AS total_sales
FROM
ecommerce_catalog.silver.orders_cleane
d;
```

Now lineage shows:

```
bronze → silver → gold → view
```

👀 Where to SEE lineage?

In Databricks UI:

- *Go to Data Explorer*
- *Click on a table or view*
- *Open Lineage tab*

You'll see:

- *Upstream (source)*
- *Downstream (consumers)*

📌 *No manual work needed.*

Managed vs External Tables

🧠 *Core Difference (one line)*

Managed table → Databricks manages data + metadata

External table → Databricks manages only metadata

Managed Tables

What happens?

- *Data stored in Databricks-managed location*
- *If you DROP TABLE → data is deleted*

When to use Managed Tables?

- ✔ *Temporary data*
- ✔ *Bronze layer*
- ✔ *Experiments*
- ✔ *Databricks-only workloads*

External Tables

What happens?

- *Data stored in external storage ADLS / S3 / GCS*
- *If you DROP TABLE → data remains safe*

📌 *Only metadata is removed.*

When to use External Tables?

✔ *Production data*
✔ *Shared across tools*
✔ *Compliance & backup*
✔ *Long-term storage*