Mini project using Unity Catalog

1. **Data Discovery**
2. **Data Audit**
3. **Data Lineage**
4. **Data Access Control**

We'll create a mini project that mimics a **retail data platform** where you:

1. Set up a **Unity Catalog** with schemas and tables.
2. Insert, update, and manage data in the catalog.
3. Implement **Access Control** to limit user permissions.
4. Explore **Data Lineage** and **Audit Logs** for a set of operations.

Mini Project: Retail Sales Data Governance Platform

Project Goals:

1. **Setup a Unity Catalog Metastore**
2. **Create a Sales Data Schema**
3. **Create and Manage Tables in the Catalog**
4. **Set Up Views and Perform Operations on the Data**
5. **Control Access to the Data**
6. **Explore Data Lineage and Auditing**

Step 1: Setup Unity Catalog Metastore

1. Create a metastore from the Databricks admin console.
2. Assign the metastore to your workspace.

Step 2: Create a Retail Catalog and Sales Schema

1. **Create the `retail_data` catalog:**

```
CREATE CATALOG retail_data;
```

2. **Create a `sales` schema in the catalog:**

```
CREATE SCHEMA retail_data.sales;
```

Step 3: Create Tables in the Sales Schema

1. **Create the `product_sales` table to store transactional sales data:**

```
CREATE TABLE retail_data.sales.product_sales (
  SaleID INT,
  ProductName STRING,
  Quantity INT,
  SaleDate DATE
);
```

2. **Insert sample data into the `product_sales` table:**

```
INSERT INTO retail_data.sales.product_sales
VALUES
  (1, 'Product A', 10, '2024-01-01'),
  (2, 'Product B', 5, '2024-02-01'),
  (3, 'Product C', 20, '2024-03-01');
```

3. **Create the `customer_data` table to store customer information:**

```
CREATE TABLE retail_data.sales.customer_data (
  CustomerID INT,
  CustomerName STRING,
  Email STRING,
  JoinDate DATE
);
```

4. **Insert sample data into the `customer_data` table:**

```
INSERT INTO retail_data.sales.customer_data
VALUES
  (1, 'Abdullah Khan', 'abdullah@example.com', '2023-01-01'),
  (2, 'John Smith', 'john@example.com', '2023-02-01'),
  (3, 'Sharma', 'sharma@example.com', '2023-03-01');
```

Step 4: Create Views and Manage Data

1. **Create a View for recent sales (last 30 days):**

```
CREATE VIEW retail_data.sales.recent_sales AS
SELECT *
FROM retail_data.sales.product_sales
WHERE SaleDate >= current_date() - INTERVAL 30 DAYS;
```

2. **Create a View to join customer and sales data:**

```
CREATE VIEW retail_data.sales.customer_sales AS
SELECT c.CustomerID, c.CustomerName, p.ProductName, p.Quantity, p.SaleDate
FROM retail_data.sales.customer_data c
JOIN retail_data.sales.product_sales p
ON c.CustomerID = p.SaleID;
```

Step 5: Implement Data Access Controls

1. **Grant read access to a user (e.g., an analyst) to the `recent_sales` view:**

```
GRANT SELECT ON VIEW retail_data.sales.recent_sales TO `analyst@example.com`;
```

2. **Grant full access to the sales data to a manager:**

```
GRANT ALL PRIVILEGES ON TABLE retail_data.sales.product_sales TO
`manager@example.com`;
```

3. **Revoke access from a user (if needed):**

```
REVOKE SELECT ON VIEW retail_data.sales.recent_sales FROM `analyst@example.com`;
```

Step 6: Explore Data Lineage and Auditing

1. **Lineage**: Navigate to the Databricks UI under **Catalog Explorer** to check the **lineage** of the `product_sales` table and `recent_sales` view.
    - Verify that you can track where the data is coming from and where it is used.
2. **Audit Logs**: In the Databricks admin console, view the **Audit Logs** for the operations performed.
    - Confirm that logs show actions such as table creation, data insertion, and access control modifications.

Step 7: Explore Advanced Capabilities (Optional)

Data Retention (Vacuum):

1. **Vacuum the `product_sales` table to remove files older than 7 days:**

```
VACUUM retail_data.sales.product_sales RETAIN 168 HOURS;
```

Time Travel:

2. **View the history of the `product_sales` table:**

```
DESCRIBE HISTORY retail_data.sales.product_sales;
```

3. **Query the table as it existed at a previous version:**

```
SELECT *
FROM retail_data.sales.product_sales VERSION AS OF 2;
```

Conclusion:

This project showcases the use of **Unity Catalog** for setting up **data governance** in a retail environment. You can:

- Create and manage **catalogs**, **schemas**, **tables**, and **views**.
- Control **data access** using SQL commands.
- Explore **data lineage** and **audit logs** to ensure the governance of your data.
- Apply advanced capabilities like **Vacuum** and **Time Travel** to manage data retention and versioning.