

Mini Project: Data Governance Using Unity Catalog - Advanced Capabilities

Objective:

Participants will:

1. Create multiple schemas and tables using Unity Catalog.
 2. Set up data governance features like Data Discovery, Data Audit, Data Lineage, and Access Control.
 3. Build a secure environment with fine-grained control over data access and visibility.
-

Task 1: Set Up Unity Catalog Objects with Multiple Schemas

1. Create a Catalog:

- Create a catalog named `finance_data_catalog` for storing financial data.

2. Create Multiple Schemas:

- Create two schemas inside the catalog:
 - `transaction_data`
 - `customer_data`

3. Create Tables in Each Schema:

- For `transaction_data`, create a table with columns: `TransactionID`, `CustomerID`, `TransactionAmount`, `TransactionDate`.
 - For `customer_data`, create a table with columns: `CustomerID`, `CustomerName`, `Email`, `Country`.
-

Task 2: Data Discovery Across Schemas

1. Explore Metadata:

- Search for tables across both schemas and retrieve metadata using SQL commands.

2. Data Profiling:

- Run SQL queries to perform data profiling on both tables, discovering trends in transaction amounts and customer locations.

3. Tagging Sensitive Data:

- Apply tags to sensitive columns such as `Email` and `TransactionAmount` for better governance tracking.
-

Task 3: Implement Data Lineage and Auditing

1. Track Data Lineage:

- Merge data from both schemas (`transaction_data` and `customer_data`) to generate a comprehensive view.
- Use Unity Catalog to trace the data lineage and track changes between these two tables.

2. Audit User Actions:

- Enable audit logs for operations performed on the tables and track who accessed or modified the data.

Task 4: Access Control and Permissions

1. Set Up Roles and Groups:

- Create two groups: `DataEngineers` and `DataAnalysts`.
- Assign appropriate roles:
 - `DataEngineers` should have full access to both schemas and tables.
 - `DataAnalysts` should have read-only access to the `customer_data` schema and restricted access to the `transaction_data` schema.

2. Row-Level Security:

- Implement row-level security for the `transaction_data` schema, allowing only certain users to view high-value transactions.

Task 5: Data Governance Best Practices

1. Create Data Quality Rules:

- Implement basic data quality rules to ensure that:
 - Transaction amounts are non-negative.
 - Customer emails follow the correct format.

2. Validate Data Governance:

- Validate all data governance rules by running SQL queries and checking that the lineage and audit logs capture all operations correctly.

Task 6: Data Lifecycle Management

1. Implement Time Travel:

- Use Unity Catalog's Delta Time Travel feature to access historical versions of the `transaction_data` table and restore to a previous state.

2. Run a Vacuum Operation:

- Run a vacuum operation on the tables to clean up old files and ensure the Delta tables are optimized.

Mini Project: Advanced Data Governance and Security Using Unity Catalog

Objective:

Participants will:

1. Create a multi-tenant data architecture using Unity Catalog.
 2. Explore the advanced features of Unity Catalog, including data discovery, data lineage, audit logs, and access control.
-

Task 1: Set Up Multi-Tenant Data Architecture Using Unity Catalog

1. Create a New Catalog:

- Create a catalog named `corporate_data_catalog` for storing corporate-wide data.

2. Create Schemas for Each Department:

- Create three schemas:
 - `sales_data`
 - `hr_data`
 - `finance_data`

3. Create Tables in Each Schema:

- For `sales_data`: Create a table with columns `SalesID`, `CustomerID`, `SalesAmount`, `SalesDate`.
 - For `hr_data`: Create a table with columns `EmployeeID`, `EmployeeName`, `Department`, `Salary`.
 - For `finance_data`: Create a table with columns `InvoiceID`, `VendorID`, `InvoiceAmount`, `PaymentDate`.
-

Task 2: Enable Data Discovery for Cross-Departmental Data

1. Search for Tables Across Departments:

- Use the Unity Catalog interface to search for tables across the `sales_data`, `hr_data`, and `finance_data` schemas.

2. Tag Sensitive Information:

- Tag columns that contain sensitive data, such as `Salary` in the `hr_data` schema and `InvoiceAmount` in the `finance_data` schema.

3. Data Profiling:

- Perform basic data profiling on the tables to analyze trends in sales, employee salaries, and financial transactions.
-

Task 3: Implement Data Lineage and Data Auditing

1. Track Data Lineage:

- Track data lineage between the `sales_data` and `finance_data` schemas by creating a reporting table that merges the sales and finance data.
- Use Unity Catalog's data lineage feature to visualize how data flows between these tables.

2. Enable Data Audit Logs:

- Ensure that all operations (e.g., data reads, writes, and updates) on the `hr_data` and `finance_data` tables are captured in audit logs for regulatory compliance.
-

Task 4: Data Access Control and Security

1. Set Up Roles and Permissions:

- Create the following groups:
 - SalesTeam : Should have access to the `sales_data` schema only.
 - FinanceTeam : Should have access to both `sales_data` and `finance_data` schemas.
 - HRTeam : Should have access to the `hr_data` schema with the ability to update employee records.

2. Implement Column-Level Security:

- Restrict access to the `Salary` column in the `hr_data` schema, allowing only HR managers to view this data.

3. Row-Level Security:

- Implement row-level security on the `sales_data` schema to ensure that each sales representative can only access their own sales records.

Task 5: Data Governance Best Practices

1. Define Data Quality Rules:

- Set up data quality rules to ensure:
 - Sales amounts are positive in the `sales_data` table.
 - Employee salaries are greater than zero in the `hr_data` table.
 - Invoice amounts in the `finance_data` table match payment records.

2. Apply Time Travel for Data Auditing:

- Use Delta Time Travel to restore the `finance_data` table to a previous state after an erroneous update and validate the changes using data audit logs.

Task 6: Optimize and Clean Up Delta Tables

1. Optimize Delta Tables:

- Use the `OPTIMIZE` command to improve query performance on the `sales_data` and `finance_data` tables.

2. Vacuum Delta Tables:

- Run a `VACUUM` operation to remove old and unnecessary data files from the Delta tables, ensuring efficient storage.

Mini Project: Building a Secure Data Platform with Unity Catalog

Objective:

Participants will:

1. Set up a secure data platform using Unity Catalog.
 2. Explore key data governance features such as Data Discovery, Data Lineage, Access Control, and Audit Logging.
-

Task 1: Set Up Unity Catalog for Multi-Domain Data Management

1. Create a New Catalog:

- Create a catalog named `enterprise_data_catalog` to manage data across various domains.

2. Create Domain-Specific Schemas:

- Create the following schemas:
 - `marketing_data`
 - `operations_data`
 - `it_data`

3. Create Tables in Each Schema:

- In the `marketing_data` schema, create a table with columns: `CampaignID`, `CampaignName`, `Budget`, `StartDate`.
 - In the `operations_data` schema, create a table with columns: `OrderID`, `ProductID`, `Quantity`, `ShippingStatus`.
 - In the `it_data` schema, create a table with columns: `IncidentID`, `ReportedBy`, `IssueType`, `ResolutionTime`.
-

Task 2: Data Discovery and Classification

1. Search for Data Across Schemas:

- Use Unity Catalog's data discovery features to list all tables in the catalog.
- Perform a search query to retrieve tables based on data types (e.g., `Budget`, `ResolutionTime`).

2. Tag Sensitive Information:

- Tag the `Budget` column in `marketing_data` and `ResolutionTime` in `it_data` as sensitive for better data management and compliance.

3. Data Profiling:

- Perform basic data profiling to understand trends in marketing budgets and operational shipping statuses.
-

Task 3: Data Lineage and Auditing

1. Track Data Lineage Across Schemas:

- Link the `marketing_data` with the `operations_data` by joining campaign performance with product orders.
- Use Unity Catalog to track the lineage of the data from marketing campaigns to sales.

2. Enable and Analyze Audit Logs:

- Ensure audit logging is enabled to track all operations on tables within the `it_data` schema. Identify who accessed or modified the data.
-

Task 4: Implement Fine-Grained Access Control

1. Create User Roles and Groups:

- Set up the following groups:
 - `MarketingTeam`: Access to the `marketing_data` schema only.
 - `OperationsTeam`: Access to both `operations_data` and `marketing_data` schemas.
 - `ITSupportTeam`: Access to the `it_data` schema with permission to update issue resolution times.

2. Implement Column-Level Security:

- Restrict access to the `Budget` column in the `marketing_data` schema, allowing only the `MarketingTeam` to view it.

3. Row-Level Security:

- Implement row-level security in the `operations_data` schema to ensure that users from the `OperationsTeam` can only view orders relevant to their department.

Task 5: Data Governance and Quality Enforcement

1. Set Data Quality Rules:

- Define rules for each schema:
 - `marketing_data`: Ensure that the campaign budget is greater than zero.
 - `operations_data`: Ensure that shipping status is valid (e.g., 'Pending', 'Shipped', 'Delivered').
 - `it_data`: Ensure that issue resolution times are recorded correctly and not negative.

2. Apply Delta Lake Time Travel:

- Use Delta Lake Time Travel to explore different historical states of the `operations_data` schema, and revert to an earlier version if required.

Task 6: Performance Optimization and Data Cleanup

1. Optimize Delta Tables:

- Apply `OPTIMIZE` to the `operations_data` and `it_data` schemas to enhance performance for frequent queries.

2. Vacuum Delta Tables:

- Run a `VACUUM` operation on the Delta tables to clean up old and unnecessary data files.

Task 1: Raw Data Ingestion

- Create a notebook to ingest raw weather data.
- The notebook should read a CSV file containing weather data.

- Define a schema for the data and ensure that proper data types are used (e.g., City, Date, Temperature, Humidity).
 - If the raw data file does not exist, handle the error and log it.
 - Save the raw data to a Delta table.
-

Task 2: Data Cleaning

- Create a notebook to clean the raw weather data.
 - Load the data from the Delta table created in Task 1.
 - Remove any rows that contain missing or null values.
 - Save the cleaned data to a new Delta table.
-

Task 3: Data Transformation

- Create a notebook to perform data transformation.
 - Load the cleaned data from the Delta table created in Task 2.
 - Calculate the average temperature and humidity for each city.
 - Save the transformed data to a Delta table.
-

Task 4: Create a Pipeline to Execute Notebooks

- Create a pipeline that sequentially executes the following notebooks:
 - Raw Data Ingestion
 - Data Cleaning
 - Data Transformation
 - Handle errors such as missing files or failed steps in the pipeline.
 - Ensure that log messages are generated at each step to track the progress of the pipeline.
-

Bonus Task: Error Handling

- Add error handling to the pipeline to manage scenarios like missing files or corrupted data.
 - Ensure the pipeline logs errors to a file or table for further analysis.
-
-

Task 1: Raw Data Ingestion

- Use the following CSV data to represent daily weather conditions:

```
City,Date,Temperature,Humidity
New York,2024-01-01,30.5,60
Los Angeles,2024-01-01,25.0,65
Chicago,2024-01-01,-5.0,75
Houston,2024-01-01,20.0,80
Phoenix,2024-01-01,15.0,50
```

- Load the CSV data into a Delta table in Databricks.
- If the file does not exist, handle the missing file scenario and log the error.

Task 2: Data Cleaning

- Create a notebook to clean the ingested weather data.
- Handle null or incorrect values in the temperature and humidity columns.
- After cleaning, save the updated data to a new Delta table.

Task 3: Data Transformation

- Transform the cleaned data by calculating the average temperature and humidity for each city.
- Save the transformed data into a new Delta table.

Task 4: Build and Run a Pipeline

- Create a Databricks pipeline that executes the following notebooks in sequence:
 - Data ingestion (from Task 1)
 - Data cleaning (from Task 2)
 - Data transformation (from Task 3)
 - Ensure each step logs its status and any errors encountered.
-

Task 1: Customer Data Ingestion

- Use the following CSV data representing customer transactions:

```
CustomerID,TransactionDate,TransactionAmount,ProductCategory
C001,2024-01-15,250.75,Electronics
C002,2024-01-16,125.50,Groceries
C003,2024-01-17,90.00,Clothing
C004,2024-01-18,300.00,Electronics
C005,2024-01-19,50.00,Groceries
```

- Load the CSV data into a Delta table in Databricks.
- If the file is not present, add error handling and log an appropriate message.

Task 2: Data Cleaning

- Create a notebook to clean the ingested customer data.
- Remove any duplicate transactions and handle null values in the TransactionAmount column.
- Save the cleaned data into a new Delta table.

Task 3: Data Aggregation

- Aggregate the cleaned data by ProductCategory to calculate the total transaction amount per category.
- Save the aggregated data to a Delta table.

Task 4: Pipeline Creation

- Build a pipeline that:
 1. Ingests the raw customer data (from Task 1).
 2. Cleans the data (from Task 2).
 3. Performs aggregation (from Task 3).
- Ensure the pipeline handles missing files or errors during each stage and logs them properly.

Task 5: Data Validation

- After completing the pipeline, add a data validation step to verify that the total number of transactions matches the sum of individual category transactions.
-

Task 1: Product Inventory Data Ingestion

- Use the following CSV data to represent product inventory information:

```
ProductID,ProductName,StockQuantity,Price,LastRestocked
P001,Laptop,50,1500.00,2024-02-01
P002,Smartphone,200,800.00,2024-02-02
P003,Headphones,300,100.00,2024-01-29
P004,Tablet,150,600.00,2024-01-30
P005,Smartwatch,100,250.00,2024-02-03
```

- Load this CSV data into a Delta table in Databricks.
- Handle scenarios where the file is missing or corrupted and log the error accordingly.

Task 2: Data Cleaning

- Clean the ingested product data:
 - Ensure no null values in `StockQuantity` and `Price` columns.
 - Remove any records with `StockQuantity` less than 0.
 - Save the cleaned data to a new Delta table.

Task 3: Inventory Analysis

- Create a notebook to analyze the inventory data:
 - Calculate the total stock value for each product (`StockQuantity * Price`).
 - Find products that need restocking (e.g., products with `StockQuantity < 100`).
- Save the analysis results to a Delta table.

Task 4: Build an Inventory Pipeline

- Build a Databricks pipeline that:
 1. Ingests the product inventory data (from Task 1).
 2. Cleans the data (from Task 2).
 3. Performs inventory analysis (from Task 3).
- Ensure the pipeline logs errors if any step fails and handles unexpected issues such as missing data.

Task 5: Inventory Monitoring

- Create a monitoring notebook that checks the Delta table for any products that need restocking (e.g., `StockQuantity < 50`).
- The notebook should send an alert if any product is below the threshold.

Task 1: Employee Attendance Data Ingestion

- Use the following CSV data representing employee attendance logs:

```
EmployeeID,Date,CheckInTime,CheckOutTime,HoursWorked
E001,2024-03-01,09:00,17:00,8
E002,2024-03-01,09:15,18:00,8.75
E003,2024-03-01,08:45,17:15,8.5
```

```
E004,2024-03-01,10:00,16:30,6.5
E005,2024-03-01,09:30,18:15,8.75
```

- Ingest this CSV data into a Delta table in Databricks.
- Handle potential issues, such as the file being missing or having inconsistent columns, and log these errors.

Task 2: Data Cleaning

- Clean the ingested attendance data:
 - Remove any rows with null or invalid values in the `CheckInTime` or `CheckOutTime` columns.
 - Ensure the `HoursWorked` column is calculated correctly (`CheckOutTime - CheckInTime`).
 - Save the cleaned data into a new Delta table.

Task 3: Attendance Summary

- Create a notebook that summarizes employee attendance:
 - Calculate the total hours worked by each employee for the current month.
 - Find employees who have worked overtime (e.g., more than 8 hours on any given day).
 - Save the summary to a new Delta table.

Task 4: Create an Attendance Pipeline

- Build a pipeline in Databricks that:
 1. Ingests employee attendance data (from Task 1).
 2. Cleans the data (from Task 2).
 3. Summarizes the attendance and calculates overtime (from Task 3).
- Ensure the pipeline logs errors and handles scenarios like missing data.

Task 5: Time Travel with Delta Lake

- Implement time travel using Delta Lake:
 - Roll back the attendance data to a previous version (e.g., the day before a change was made).
 - Use the `DESCRIBE HISTORY` command to inspect the changes made to the Delta table.
-