Mini Project: Data Governance Using Unity Catalog - Advanced Capabilities

Task 1: Set Up Unity Catalog Objects with Multiple Schemas

1. Create a Catalog:

```
CREATE CATALOG finance_data_catalog;
```

2. Create Multiple Schemas:

```
CREATE SCHEMA finance_data_catalog.transaction_data;

CREATE SCHEMA finance_data_catalog.customer_data;
```

3. Create Tables in Each Schema:

```
CREATE TABLE finance_data_catalog.transaction_data.transactions (
  TransactionID INT,
  CustomerID INT,
  TransactionAmount DECIMAL(10, 2),
  TransactionDate DATE
);
CREATE TABLE finance_data_catalog.customer_data.customers (
  CustomerID INT,
  CustomerName STRING,
  Email STRING,
  Country STRING
);
```

Task 2: Data Discovery Across Schemas

1. **Explore Metadata:**

```
SHOW TABLES IN finance_data_catalog.transaction_data;

SHOW TABLES IN finance_data_catalog.customer_data;
```

2. Data Profiling:

## Analyze transaction trends

```
SELECT AVG(TransactionAmount), MIN(TransactionAmount)

FROM finance_data_catalog.transaction_data.transactions;
```

## Analyze customer locations

```
SELECT Country, COUNT(*) AS TotalCustomers
FROM finance_data_catalog.customer_data.customers
GROUP BY Country;
```

3. Tagging Sensitive Data:

```
ALTER TABLE finance_data_catalog.transaction_data.transactions SET TAG 'Sensitive' ON
COLUMN TransactionAmount;

ALTER TABLE finance_data_catalog.customer_data.customers SET TAG 'Sensitive' ON COLUMN
Email;
```

Task 3: Implement Data Lineage and Auditing

1. Track Data Lineage:

```
CREATE TABLE finance_data_catalog.merged_data AS
SELECT t.TransactionID, t.CustomerID, t.TransactionAmount, t.TransactionDate, c.CustomerName,
c.Email, c.Country
FROM finance_data_catalog.transaction_data.transactions t
JOIN finance_data_catalog.customer_data.customers c
ON t.CustomerID = c.CustomerID;
```

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:

```
GRANT ALL PRIVILEGES ON SCHEMA finance_data_catalog.transaction_data TO
DataEngineers;
GRANT SELECT ON SCHEMA finance_data_catalog.customer_data TO DataAnalysts;
GRANT SELECT ON TABLE finance_data_catalog.transaction_data.transactions TO
DataAnalysts;
```

2. Row-Level Security:

```
CREATE VIEW finance_data_catalog.transaction_data.high_value_transactions AS
SELECT * FROM finance_data_catalog.transaction_data.transactions
WHERE TransactionAmount > 10000;

GRANT SELECT ON VIEW finance_data_catalog.transaction_data.high_value_transactions TO
specific_user;
```

Task 5: Data Governance Best Practices:

1. **Create Data Quality Rules:**

## Check for negative transaction amounts

```
SELECT * FROM finance_data_catalog.transaction_data.transactions
WHERE TransactionAmount < 0;
```

## Validate email format

```
SELECT * FROM finance_data_catalog.customer_data.customers
WHERE Email NOT L
LIKE '%@%.%';
```

Task 6: Data Lifecycle Management

1. **Implement Time Travel:**
   ```
   SELECT * FROM finance_data_catalog.transaction_data.transactions VERSION AS OF
   TIMESTAMP '2024-09-15';
   ```

2. **Run a Vacuum Operation:**
   ```
   VACUUM finance_data_catalog.transaction_data.transactions;
   ```

Mini Project: Advanced Data Governance and Security Using Unity Catalog

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

1. **Create a New Catalog:**

```
CREATE CATALOG corporate_data_catalog;
```

2. Create Schemas for Each Department:

```
CREATE SCHEMA corporate_data_catalog.sales_data;
CREATE SCHEMA corporate_data_catalog.hr_data;
CREATE SCHEMA corporate_data_catalog.finance_data;
```

3. Create Tables in Each Schema:

```
CREATE TABLE corporate_data_catalog.sales_data.sales (
  SalesID INT,
  CustomerID INT,
  SalesAmount DECIMAL(10, 2),
  SalesDate DATE
);
```

```
CREATE TABLE corporate_data_catalog.hr_data.employees (
  EmployeeID INT,
  EmployeeName STRING,
  Department STRING,
  Salary DECIMAL(10, 2)
);

CREATE TABLE corporate_data_catalog.finance_data.invoices (
  InvoiceID INT,
  VendorID INT,
  InvoiceAmount DECIMAL(10, 2),
  PaymentDate DATE
);
```

Task 2: Enable Data Discovery for Cross-Departmental Data

1. Search for Tables Across Departments:

```
SHOW TABLES IN corporate_data_catalog.sales_data;

SHOW TABLES IN corporate_data_catalog.hr_data;

SHOW TABLES IN corporate_data_catalog.finance_data;
```

2. Tag Sensitive Information:

```
ALTER TABLE corporate_data_catalog.hr_data.employees SET TAG 'Sensitive' ON COLUMN
Salary;

ALTER TABLE corporate_data_catalog.finance_data.invoices SET TAG 'Sensitive' ON COLUMN
InvoiceAmount;
```

3. Data Profiling:

```
## sales trends

SELECT AVG(SalesAmount), MIN(SalesAmount), MAX(SalesAmount)

FROM corporate_data_catalog.sales_data.sales;


## employee salary distribution

SELECT AVG(Salary), MAX(Salary)

FROM corporate_data_catalog.hr_data.employees;


## financial transactions

SELECT AVG(InvoiceAmount), MIN(InvoiceAmount), MAX(InvoiceAmount)

FROM corporate_data_catalog.finance_data.invoices;
```

Task 3: Implement Data Lineage and Data Auditing

1. Track Data Lineage:

```
CREATE TABLE corporate_data_catalog.reports.sales_finance_report AS

SELECT s.SalesID, s.CustomerID, s.SalesAmount, s.SalesDate, f.InvoiceID, f.InvoiceAmount, f.PaymentDate

FROM corporate_data_catalog.sales_data.sales s

JOIN corporate_data_catalog.finance_data.invoices f

ON s.CustomerID = f.VendorID;
```

2. Enable Data Audit Logs:

Turn on the audit log

Task 4: Data Access Control and Security:

1. Set Up Roles and Permissions:

```
CREATE GROUP SalesTeam;

CREATE GROUP FinanceTeam;

CREATE GROUP HRTeam;
```

## Grant access to SalesTeam

```
GRANT SELECT ON SCHEMA corporate_data_catalog.sales_data TO SalesTeam;
```

##Grant access to FinanceTeam

```
GRANT SELECT ON SCHEMA corporate_data_catalog.sales_data TO FinanceTeam;

GRANT SELECT, INSERT, UPDATE ON SCHEMA corporate_data_catalog.finance_data TO FinanceTeam;
```

##Grant access to HRTeam

```
GRANT SELECT, UPDATE ON SCHEMA corporate_data_catalog.hr_data TO HRTeam;
```

2. Implement Column-Level Security:

```
CREATE VIEW corporate_data_catalog.hr_data.salary_restricted AS

SELECT EmployeeID, EmployeeName, Department FROM corporate_data_catalog.hr_data.employees;


GRANT SELECT ON VIEW corporate_data_catalog.hr_data.salary_restricted TO HRTeam;
```

3. Row-Level Security:

```
CREATE VIEW corporate_data_catalog.sales_data.sales_rep_view AS
```

SELECT * FROM corporate_data_catalog.sales_data.sales WHERE SalesRepID = current_user();

GRANT SELECT ON VIEW corporate_data_catalog.sales_data.sales_rep_view TO specific_sales_rep;

Task 5: Data Governance Best Practices

1. Define Data Quality Rules:

##Ensure sales amounts are positive

SELECT * FROM corporate_data_catalog.sales_data.sales WHERE SalesAmount < 0;

## Ensure employee salaries are greater than zero

SELECT * FROM corporate_data_catalog.hr_data.employees WHERE Salary <= 0;

##Ensure invoice amounts match payment records

SELECT * FROM corporate_data_catalog.finance_data.invoices WHERE InvoiceAmount <= 0;

2. Apply Time Travel for Data Auditing:

SELECT * FROM corporate_data_catalog.finance_data.invoices VERSION AS OF TIMESTAMP '2024-09-15';

Task 6: Optimize and Clean Up Delta Tables

OPTIMIZE corporate_data_catalog.sales_data.sales;

OPTIMIZE corporate_data_catalog.finance_data.invoices;

## vaccum

VACUUM corporate_data_catalog.sales_data.sales;

VACUUM corporate_data_catalog.finance_data.invoices;

Mini Project: Building a Secure Data Platform with Unity Catalog

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

1. Create a New Catalog:

CREATE CATALOG enterprise_data_catalog;

2. Create Domain-Specific Schemas:

CREATE SCHEMA enterprise_data_catalog.marketing_data;

CREATE SCHEMA enterprise_data_catalog.operations_data;

CREATE SCHEMA enterprise_data_catalog.it_data;

3. Create Tables in Each Schema:

```
CREATE TABLE enterprise_data_catalog.marketing_data.campaigns (
    CampaignID INT,
    CampaignName STRING,
    Budget DOUBLE,
    StartDate DATE
);
CREATE TABLE enterprise_data_catalog.operations_data.orders (
    OrderID INT,
    ProductID INT,
    Quantity INT,
    ShippingStatus STRING
);
CREATE TABLE enterprise_data_catalog.it_data.incidents (
    IncidentID INT,
    ReportedBy STRING,
    IssueType STRING,
    ResolutionTime DOUBLE
);
```

Task 2: Data Discovery and Classification

1. Search for Data Across Schemas:

```
SHOW TABLES IN enterprise_data_catalog;

SELECT * FROM enterprise_data_catalog.INFORMATION_SCHEMA.COLUMNS
WHERE column_name = 'Budget';
```

2. Tag Sensitive Information:

```
ALTER TABLE enterprise_data_catalog.marketing_data.campaigns
SET TAGS ('Budget' = 'Sensitive');
ALTER TABLE enterprise_data_catalog.it_data.incidents
SET TAGS ('ResolutionTime' = 'Sensitive');
```

3. Data Profiling:

```
SELECT AVG(Budget) FROM enterprise_data_catalog.marketing_data.campaigns;
```

Task 3: Data Lineage and Auditing

1. Track Data Lineage Across Schemas:

```
SELECT m.CampaignID, m.CampaignName, o.OrderID, o.ProductID, o.Quantity

FROM enterprise_data_catalog.marketing_data.campaigns m

JOIN enterprise_data_catalog.operations_data.orders o

ON m.CampaignID = o.ProductID;
```

2. Enable and Analyze Audit Logs:

```
SHOW AUDIT LOGS FOR enterprise_data_catalog.it_data;
```

Task 4: Implement Fine-Grained Access Control

1. Create User Roles and Groups:

```
GRANT USAGE ON SCHEMA enterprise_data_catalog.marketing_data TO GROUP
MarketingTeam;

GRANT USAGE ON SCHEMA enterprise_data_catalog.operations_data TO GROUP
OperationsTeam;

GRANT USAGE ON SCHEMA enterprise_data_catalog.it_data TO GROUP ITSupportTeam;
```

2. Implement Column-Level Security:

```
GRANT SELECT (CampaignID, CampaignName, StartDate) ON TABLE
enterprise_data_catalog.marketing_data.campaigns

TO GROUP OperationsTeam;

GRANT SELECT ON TABLE enterprise_data_catalog.marketing_data.campaigns TO GROUP
MarketingTeam;
```

3. Row-Level Security:

```
CREATE ROW ACCESS POLICY operations_team_policy

ON enterprise_data_catalog.operations_data.orders

USING (User() = 'operations_rep');
```

Task 5: Data Governance and Quality Enforcement

1. Set Data Quality Rules:

```
SELECT * FROM enterprise_data_catalog.marketing_data.campaigns WHERE Budget <= 0;
```

2. Apply Delta Lake Time Travel:

```
DESCRIBE HISTORY enterprise_data_catalog.operations_data.orders;

RESTORE enterprise_data_catalog.operations_data.orders

TO VERSION AS OF <version-number>;
```

Task 6: Performance Optimization and Data Cleanup

1. Optimize Delta Tables:

```
OPTIMIZE enterprise_data_catalog.operations_data.orders;

OPTIMIZE enterprise_data_catalog.it_data.incidents;
```

2. Vacuum Delta Tables:

```
VACUUM enterprise_data_catalog.operations_data.orders;

VACUUM enterprise_data_catalog.it_data.incidents;
```