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**

```
- - For Transaction data:
>>  CREATE TABLE finance_data_catalog.transaction_data.transactions (
   TransactionID STRING,
   CustomerID STRING,
   TransactionAmount DECIMAL(10, 2),
   TransactionDate DATE
);

- - For customer_data:
>>  CREATE TABLE finance_data_catalog.customer_data.customers (
   CustomerID STRING,
   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**

```
- - Profiling TransactionAmount in transaction_data.transactions
>>  SELECT AVG(TransactionAmount) AS AvgTransactionAmount,
    MAX(TransactionAmount) AS MaxTransactionAmount,
    MIN(TransactionAmount) AS MinTransactionAmount
    FROM finance_data_catalog.transaction_data.transactions;

- - To find Transaction counts over time
>>  SELECT TransactionDate,
    COUNT(*) AS TotalTransactions
    FROM finance_data_catalog.transaction_data.transactions
    GROUP BY TransactionDate
    ORDER BY TransactionDate;
```

- - Profiling Country in customer_data.customers

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

3. Tagging Sensitive Data

```
>>  ALTER TABLE finance_data_catalog.customer_data.customers
    ADD TAG (sensitive='true') FOR COLUMN Email;

>>  ALTER TABLE finance_data_catalog.transaction_data.transactions
    ADD TAG (sensitive='true') FOR COLUMN TransactionAmount;
```

Task 3: Implement Data Lineage and Auditing

1. Track Data Lineage:

- - Merge data from both schemas to generate a comprehensive view.

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

- - Use Unity Catalog to trace the data lineage

>> To view data lineage, navigate to data explorer in databricks. In unity catalog we can view lineage and track changes.

2. Audit User Actions

- - Enable audit logs for operations performed on the tables

- Navigate to admin console in databricks
- Go to audit logs tab and enable audit logs

- - track who accessed or modified the data.

Once audit logging is enabled, you can monitor user actions such as:

- Who queried or accessed the tables.
- Who performed modifications (e.g., inserts, updates, deletes) on the tables

Task 4: Access Control and Permissions

1. Set Up Roles and Groups

- - Create two groups: DataEngineers and DataAnalysts

```
>>    CREATE GROUP DataEngineers;
>>    CREATE GROUP DataAnalysts;
```

- - Assign appropriate roles

- - For data engineers full access

```
>>    GRANT ALL PRIVILEGES ON SCHEMA finance_data_catalog.transaction_data
      TO `DataEngineers`;
>>    GRANT ALL PRIVILEGES ON SCHEMA finance_data_catalog.customer_data
      TO `DataEngineers`;

>>    GRANT ALL PRIVILEGES ON TABLE
      finance_data_catalog.transaction_data.transactions TO `DataEngineers`;
>>    GRANT ALL PRIVILEGES ON TABLE
      finance_data_catalog.customer_data.customers TO `DataEngineers`;
```

- - For data analysts Read-only access

```
>>    GRANT SELECT ON SCHEMA finance_data_catalog.customer_data TO
      `DataAnalysts`;
>>    GRANT SELECT ON TABLE finance_data_catalog.customer_data.customers TO
      `DataAnalysts`;

>>    GRANT SELECT ON TABLE finance_data_catalog.transaction_data.transactions
      TO `DataAnalysts`;
```

2. Row-Level Security:

- - row-level security for the users to view high-value transactions.

- - Create a Dynamic View for High-Value Transactions

```
>>    CREATE OR REPLACE VIEW
      finance_data_catalog.transaction_data.secure_transactions AS
      SELECT * FROM finance_data_catalog.transaction_data.transactions
      WHERE (TransactionAmount <= 10000)
      OR
      (TransactionAmount > 10000 AND CURRENT_USER() IN ('authorized_user1',
      'authorized_user2'));
```

- - Restrict Access to the Original Table

```
>>    REVOKE SELECT ON TABLE finance_data_catalog.transaction_data.transactions
      FROM `DataAnalysts`;

>>    GRANT SELECT ON VIEW finance_data_catalog.transaction_data.secure_transactions
      TO `DataAnalysts`;
```

Task 5: Data Governance Best Practices

1. Create Data Quality Rules

- - Transaction amounts are non-negative

```
>>   ALTER TABLE finance_data_catalog.transaction_data.transactions
     ADD CONSTRAINT check_non_negative_amount CHECK (TransactionAmount >= 0);
```

- - Customer emails follow the correct format.

```
>>   ALTER TABLE finance_data_catalog.customer_data.customers
     ADD CONSTRAINT check_email_format
     CHECK (Email RLIKE '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$');
```

2. Validate Data Governance

- - Validate Data Quality Rules

```
>>   INSERT INTO finance_data_catalog.transaction_data.transactions
     VALUES (10001, 1, -500.00, '2024-09-20');

>>   INSERT INTO finance_data_catalog.customer_data.customers
     VALUES (101, 'MailUser', 'mail123.in', 'India');
```

- - Check Data Lineage

To check data lineage is tracked correctly, we can use unity catalog built-in data lineage tracking.

- Go to data explorer
- Select a table
- Navigate to lineage tab to ensure that the flow of data between tables and views is captured

- - Verify Audit Logs

```
>>   SELECT eventName,userIdentity, objectName, actionName, timestamp
     FROM <audit_log_table>
     WHERE objectName IN
     ('finance_data_catalog.transaction_data.transactions',
     'finance_data_catalog.customer_data.customers')
     AND actionName IN ('INSERT', 'UPDATE');
```

Task 6: Data Lifecycle Management

1. Implement Time Travel

- - Access historical versions of the table

```
>>   SELECT * FROM finance_data_catalog.transaction_data.transactions

     VERSION AS OF 1;
```

```
    - - Restore the table to a Previous State
>> RESTORE TABLE finance_data_catalog.transaction_data.transactions
    TO VERSION AS OF 5;
```

2. Run a Vacuum Operation

```
>> VACUUM finance_data_catalog.transaction_data.transactions RETAIN 168 HOURS;
>> VACUUM finance_data_catalog.customer_data.customers RETAIN 168 HOURS;
```