**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:**

**# Schema for transaction data**

CREATE SCHEMA finance\_data\_catalog.transaction\_data

COMMENT 'Schema for storing transaction data';

**#Schema for customer data**

CREATE SCHEMA finance\_data\_catalog.customer\_data

COMMENT 'Schema for storing customer data';

**3. Create Tables in Each Schema:**

**1.For transaction\_data** schema, create the table with the required columns:

CREATE TABLE finance\_data\_catalog.transaction\_data.transactions (

TransactionID STRING,

CustomerID STRING,

TransactionAmount DECIMAL(10, 2),

TransactionDate DATE)

COMMENT 'Table for storing transaction records';

**2.For customer\_data** schema, create the table with the required columns:

CREATE TABLE finance\_data\_catalog.customer\_data.customers (

CustomerID STRING,

CustomerName STRING,

Email STRING,

Country STRING

)

COMMENT 'Table for storing customer information';

**Task 2: Data Discovery Across Schemas**

**1. Explore Metadata:**

**Search for tables across both schemas and retrieve metadata using SQL**

**commands.**

**# List tables in the transaction\_data schema**

SHOW TABLES IN finance\_data\_catalog.transaction\_data;

**#List tables in the customer\_data schema**

SHOW TABLES IN finance\_data\_catalog.customer\_data;

DESCRIBE TABLE finance\_data\_catalog.transaction\_data.transactions;

DESCRIBE TABLE finance\_data\_catalog.customer\_data.customers;

**2. Data Profiling**

**Run SQL queries to perform data profiling on both tables, discovering**

**trends in transaction amounts and customer locations.**

SELECT

MIN(TransactionAmount) AS MinAmount,

MAX(TransactionAmount) AS MaxAmount,

AVG(TransactionAmount) AS AvgAmount,

COUNT(\*) AS TotalTransactions

FROM finance\_data\_catalog.transaction\_data.transactions;

**Group transactions by date** to observe trends

SELECT

TransactionDate,

SUM(TransactionAmount) AS TotalAmount,

COUNT(\*) AS NumberOfTransactions

FROM finance\_data\_catalog.transaction\_data.transactions

GROUP BY TransactionDate

ORDER BY TransactionDate;

**Profiling Customer Data** (e.g., discovering trends in customer locations)

-- Count customers by country

SELECT

Country,

COUNT(\*) AS NumberOfCustomers

FROM finance\_data\_catalog.customer\_data.customers

GROUP BY Country

ORDER BY NumberOfCustomers DESC;

**3. Tagging Sensitive Data:**

**Apply tags to sensitive columns such as Email and TransactionAmount**

**for better governance tracking.**

ALTER TABLE finance\_data\_catalog.customer\_data.customers

ADD TAG ('sensitive' = 'Email');

ALTER TABLE finance\_data\_catalog.transaction\_data.transactions

ADD TAG ('sensitive' = 'TransactionAmount');

**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.**

CREATE OR REPLACE TABLE finance\_data\_catalog.merged\_data.customer\_transactions AS

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;

**2. Audit User Actions:**

**Enable audit logs for operations performed on the tables and track who**

**accessed or modified the data**.

SELECT

user\_name,

action\_name,

object\_name,

timestamp,

details

FROM

catalog\_audit\_logs

WHERE

object\_name IN ('finance\_data\_catalog.transaction\_data.transactions',

'finance\_data\_catalog.customer\_data.customers')

ORDER BY

timestamp DESC;

**Task 4: Access Control and Permissions**

**1. Set Up Roles and Groups:**

**Create two groups: DataEngineers and DataAnalysts .**

**Assign appropriate roles:**

Create Group DataEngineers;

Create Group DataAnalysts;

**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.**

-- Give full access (SELECT, INSERT, UPDATE, DELETE) on both schemas and tables to DataEngineers

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`;

-- Grant read-only access to the customer\_data schema and table

GRANT SELECT ON SCHEMA finance\_data\_catalog.customer\_data TO `DataAnalysts`;

GRANT SELECT ON TABLE finance\_data\_catalog.customer\_data.customers TO `DataAnalysts`;

-- Grant restricted (read-only) access to the transaction\_data schema

GRANT USAGE ON SCHEMA finance\_data\_catalog.transaction\_data TO `DataAnalysts`;

GRANT SELECT ON TABLE finance\_data\_catalog.transaction\_data.transactions TO `DataAnalysts`;

**2. Row-Level Security:**

**Implement row-level security for the transaction\_data schema, allowing**

**only certain users to view 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 current\_user() IN ('admin\_user', 'high\_value\_role');

GRANT SELECT ON VIEW finance\_data\_catalog.transaction\_data.secure\_transactions TO `DataAnalysts`;

REVOKE SELECT ON TABLE finance\_data\_catalog.transaction\_data.transactions FROM `DataAnalysts`;

**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.**

ALTER TABLE finance\_data\_catalog.transaction\_data.transactions

ADD CONSTRAINT non\_negative\_transaction CHECK (TransactionAmount >= 0);

SELECT \* FROM finance\_data\_catalog.transaction\_data.transactions WHERE TransactionAmount < 0;

SELECT \* FROM finance\_data\_catalog.customer\_data.customers

WHERE Email NOT RLIKE '^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$';

**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.**

SELECT user\_name, action\_name, object\_name, timestamp, details

FROM

catalog\_audit\_logs

WHERE

object\_name IN ('finance\_data\_catalog.transaction\_data.transactions', 'finance\_data\_catalog.customer\_data.customers', 'finance\_data\_catalog.merged\_data.customer\_transactions')

ORDER BY

timestamp DESC;

**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.**

DESCRIBE HISTORY finance\_data\_catalog.transaction\_data.transactions;

SELECT \* FROM finance\_data\_catalog.transaction\_data.transactions VERSION AS OF 3;

SELECT \* FROM finance\_data\_catalog.transaction\_data.transactions TIMESTAMP AS OF '2024-09-20T12:00:00';

RESTORE TABLE finance\_data\_catalog.transaction\_data.transactions TO VERSION AS OF 3;

**2. Run a Vacuum Operation:**

**Run a vacuum operation on the tables to clean up old files and ensure**

**the Delta tables are optimized.**

VACUUM finance\_data\_catalog.transaction\_data.transactions RETAIN 168 HOURS;

VACUUM finance\_data\_catalog.transaction\_data.transactions RETAIN 24 HOURS;

OPTIMIZE finance\_data\_catalog.transaction\_data.transactions;

**2.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**

- - For Sales data

CREATE TABLE corporate\_data\_catalog.sales\_data.sales\_table(

SalesID STRING,

CustomerID STRING,

SalesAmount DECIMAL(10,2),

SalesDate DATE

);

- - For HR Data

CREATE TABLE corporate\_data\_catalog.hr\_data.hr\_table(

EmployeeID STRING,

EmployeeName STRING,

Department STRING,

Salary DECIMAL(10,2)

);

- - For Finance Data

CREATE TABLE corporate\_data\_catalog.finance\_data.finance\_table(

InvoiceID STRING,

VendorID STRING,

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.hr\_table

SET TAG 'sensitive' ON COLUMN Salary;

ALTER TABLE corporate\_data\_catalog.finance\_data.finance\_table

SET TAG 'sensitive' ON COLUMN InvoiceAmount;

**3. Data Profiling**

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

corporate\_data\_catalog.sales\_data.sales\_table;

SELECT AVG(Salary), MIN(Salary), MAX(Salary) FROM

corporate\_data\_catalog.hr\_data.hr\_table;

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

corporate\_data\_catalog.finance\_data.finance\_table;

**Task 3: Implement Data Lineage and Data Auditing**

**1. Track Data Lineage**

- - creating a reporting table that merges the sales and finance data.

CREATE TABLE corporate\_data\_catalog.reporting.sales\_finance\_report AS

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

f.InvoiceAmount

FROM corporate\_data\_catalog.sales\_data.sales\_table s

JOIN corporate\_data\_catalog.finance\_data.finance\_table f

ON s.CustomerID = f.VendorID;

**Task 4: Data Access Control and Security**

**1. Set Up Roles and Permissions**

CREATE GROUP SalesTeam;

GRANT USAGE ON SCHEMA corporate\_data\_catalog.sales\_data TO SalesTeam;

CREATE GROUP FinanceTeam;

GRANT USAGE ON SCHEMA corporate\_data\_catalog.sales\_data TO FinanceTeam;

GRANT USAGE ON SCHEMA corporate\_data\_catalog.finance\_data TO FinanceTeam;

CREATE GROUP HRTeam;

GRANT USAGE ON SCHEMA corporate\_data\_catalog.hr\_data TO HRTeam;

GRANT UPDATE ON TABLE corporate\_data\_catalog.hr\_data.hr\_table TO HRTeam;

**2. Implement Column-Level Security**

GRANT SELECT ON COLUMN Salary TO HRManager;

**3. Row-Level Security**

CREATE ROW ACCESS POLICY sales\_rep\_policy ON

corporate\_data\_catalog.sales\_data.sales\_table

FOR EACH ROW

WHEN current\_user = sales\_rep\_id;

**Task 5: Data Governance Best Practices**

**1. Define Data Quality Rules**

- - Sales amounts are positive in the sales\_data table.

SELECT \* FROM corporate\_data\_catalog.sales\_data.sales\_table

WHERE SalesAmount <= 0;

- - Employee salaries are greater than zero in the hr\_data table

SELECT \* FROM corporate\_data\_catalog.hr\_data.hr\_table

WHERE Salary <= 0;

- - Invoice amounts in the finance\_data table match payment records.

SELECT \* FROM corporate\_data\_catalog.finance\_data.finance\_table

WHERE InvoiceAmount <> PaymentAmount;

**2. Apply Time Travel for Data Auditing**

RESTORE TABLE corporate\_data\_catalog.finance\_data.finance\_table

TO VERSION AS OF 5;

**Task 6: Optimize and Clean Up Delta Tables**

**1. Optimize Delta Tables**

OPTIMIZE corporate\_data\_catalog.sales\_data.sales\_table;

OPTIMIZE corporate\_data\_catalog.finance\_data.finance\_table;

**2. Vacuum Delta Tables**

VACUUM corporate\_data\_catalog.sales\_data.sales\_table RETAIN 168 HOURS;

VACUUM corporate\_data\_catalog.finance\_data.finance\_table RETAIN 168 HOURS;

**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 Schemas for Each Department**

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**

- - For Marketing data

CREATE TABLE enterprise\_data\_catalog.marketing\_data.marketing\_table(

CampaignID INT,

CampaignName STRING,

Budget DECIMAL(10,2),

StartDate DATE

);

- - For Operations Data

CREATE TABLE enterprise\_data\_catalog.operations\_data.operations\_table(

OrderID INT,

ProductID INT,

Quantity INT,

ShippingStatus STRING

);

- - For IT Data

CREATE TABLE enterprise\_data\_catalog.it\_data.it\_table(

IncidentID STRING,

ReportedBy STRING,

IssueType STRING,

ResolutionTime INT

);

**Task 2: Data Discovery and Classification**

**1. Search for Data Across Schemas:**

SHOW TABLES IN enterprise\_data\_catalog;

**2. Tag Sensitive Information**

ALTER TABLE enterprise\_data\_catalog.marketing\_data.marketing\_table

SET TAG 'sensitive' ON COLUMN Budget;

>> ALTER TABLE enterprise\_data\_catalog.it\_data.it\_table

SET TAG 'sensitive' ON COLUMN ResolutionTime;

**3. Data Profiling**

SELECT AVG(Budget), MIN(Budget), MAX(Budget) FROM

enterprise\_data\_catalog.marketing\_data.marketing\_table;

SELECT COUNT(ShippingStatus), ShippingStatus FROM

enterprise\_data\_catalog.operations\_data.operations\_table GROUP BY ShippingStatus;

**Task 3: Data Lineage and Data Auditing**

**1. Track Data Lineage Across Schemas:**

- - Link the marketing\_data with the operations\_data by joining campaign

performance with product orders.

CREATE TABLE enterprise\_data\_catalog.reporting.campaign\_orders\_report AS

SELECT m.CampaignID, m.CampaignName, m.Budget, 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;

**Task 4: Implement Fine-Grained Access Control**

**1. Create User Roles and Groups**

CREATE GROUP MarketingTeam;

GRANT USAGE ON SCHEMA enterprise\_data\_catalog.marketing\_data TO

MarketingTeam;

GRANT USAGE ON SCHEMA enterprise\_data\_catalog.marketing\_data TO

OperationsTeam;

CREATE GROUP OperationsTeam;

GRANT USAGE ON SCHEMA enterprise\_data\_catalog.operations\_data TO

OperationsTeam;

CREATE GROUP ITSupportTeam;

GRANT USAGE ON SCHEMA enterprise\_data\_catalog.it\_data TO ITSupportTeam;

GRANT UPDATE ON TABLE enterprise\_data\_catalog.it\_data.it\_data TO ITSupportTeam;

**2. Implement Column-Level Security**

GRANT SELECT ON COLUMN Budget TO MarketingTeam;

**3. Row-Level Security**

CREATE ROW ACCESS POLICY operations\_team\_policy ON

enterprise\_data\_catalog.operations\_data.orders

FOR EACH ROW

WHEN current\_user = operations\_rep;

**Task 5: Data Governance and Quality Enforcement**

**1. Set Data Quality Rules:**

- - Campaign budget greater than Zero(0).

SELECT \* FROM enterprise\_data\_catalog.marketing\_data.marketing\_table

WHERE Budget <= 0;

- - Shipping status is valid (e.g.,'Pending', 'Shipped', 'Delivered').

SELECT \* FROM enterprise\_data\_catalog.operations\_data.operations\_table

WHERE ShippingStatus NOT IN ('Pending', 'Shipped', 'Delivered');

- - Issue resolution times are recorded correctly and not negative..

SELECT \* FROM enterprise\_data\_catalog.it\_data.it\_table WHERE ResolutionTime < 0;

**2. Apply Delta Lake Time Travel**

RESTORE TABLE enterprise\_data\_catalog.operations\_data.operations\_table

TO VERSION AS OF 1;

**Task 6: Performance Optimization and Data Cleanup**

**1. Optimize Delta Tables**

OPTIMIZE enterprise\_data\_catalog.operations\_data.operations\_table;

OPTIMIZE enterprise\_data\_catalog.it\_data.it\_table;

**2. Vacuum Delta Tables**

VACUUM enterprise\_data\_catalog.operations\_data.operations\_table

RETAIN 168 HOURS;

VACUUM enterprise\_data\_catalog.it\_data.it\_table RETAIN 168 HOURS;

**1.**

**Task 1: Raw Data Ingestion**

from pyspark.sql import SparkSession

from pyspark.sql.types import StructType, StructField, StringType, DateType, FloatType,

IntegerType

from pyspark.sql.functions import col

import os

spark = SparkSession.builder \

.appName("Weather Data Ingestion") \

.config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \

.config("spark.sql.catalog.spark\_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog")\

.getOrCreate()

# Define schema for the weather data

weather\_schema = StructType([

StructField("City", StringType(), True),

StructField("Date", DateType(), True),

StructField("Temperature", FloatType(), True),

StructField("Humidity", IntegerType(), True)

])

# Define file path for the raw data CSV

file\_path = "/path/to/weather\_data.csv"

# Check if the file exists

if not os.path.exists(file\_path):

print(f"File not found: {file\_path}")

# Log the error

with open("/path/to/logs/ingestion\_logs.txt", "a") as log\_file:

log\_file.write(f"Error: Weather data file {file\_path} does not exist\n")

else:

# Proceed to load and process the data

print(f"File found: {file\_path}")

# Load the CSV data with the defined schema

raw\_weather\_data = spark.read.csv(file\_path, header=True, schema=weather\_schema)

# Show a few rows to verify

raw\_weather\_data.show(5)

# Define Delta table path

delta\_table\_path = "/path/to/delta/weather\_data"

# Write data to Delta table

raw\_weather\_data.write.format("delta").mode("overwrite").save(delta\_table\_path)

print(f"Raw data successfully saved to Delta table at {delta\_table\_path}")

**Task 2: Data Cleaning**

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("Weather Data Cleaning") \

.config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \

.config("spark.sql.catalog.spark\_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog")

\

.getOrCreate()

# Define the path to the Delta table

delta\_table\_path = "/path/to/delta/weather\_data"

# Load the raw data from the Delta table

raw\_weather\_data = spark.read.format("delta").load(delta\_table\_path)

# Show the raw data

raw\_weather\_data.show(5)

# Remove rows with missing or null values

cleaned\_weather\_data = raw\_weather\_data.dropna()

# Show the cleaned data

cleaned\_weather\_data.show(5)

# Define path for the cleaned Delta table

cleaned\_delta\_table\_path = "/path/to/delta/cleaned\_weather\_data"

# Save the cleaned data to a new Delta table

cleaned\_weather\_data.write.format("delta").mode("overwrite").save(cleaned\_delta\_table\_pat

h)

print(f"Cleaned data successfully saved to Delta table at {cleaned\_delta\_table\_path}")

**Task 3: Data Transformation**

from pyspark.sql import SparkSession

from pyspark.sql import functions as F

spark = SparkSession.builder \

.appName("Weather Data Transformation") \

.config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \

.config("spark.sql.catalog.spark\_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog")

\

.getOrCreate()

# Define the path to the cleaned Delta table

cleaned\_delta\_table\_path = "/path/to/delta/cleaned\_weather\_data"

# Load the cleaned data from the Delta table

cleaned\_weather\_data = spark.read.format("delta").load(cleaned\_delta\_table\_path)

# Show the cleaned data

cleaned\_weather\_data.show(5)

# Calculate average temperature and humidity for each city

transformed\_data = cleaned\_weather\_data.groupBy("City").agg(

F.avg("Temperature").alias("Average\_Temperature"),

F.avg("Humidity").alias("Average\_Humidity")

)

# Show the transformed data

transformed\_data.show()

# Define path for the transformed Delta table

transformed\_delta\_table\_path = "/path/to/delta/transformed\_weather\_data"

# Save the transformed data to a new Delta table

transformed\_data.write.format("delta").mode("overwrite").save(transformed\_delta\_table\_pat

h)

print(f"Transformed data successfully saved to Delta table at

{transformed\_delta\_table\_path}")

**Task 4: Create a Pipeline to Execute Notebooks**

import subprocess

import logging

import os

# Set up logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -

%(message)s')

# Define paths to the notebooks

notebooks = {

"Raw Data Ingestion": "/path/to/Raw\_Data\_Ingestion\_Notebook.ipynb",

"Data Cleaning": "/path/to/Data\_Cleaning\_Notebook.ipynb",

"Data Transformation": "/path/to/Data\_Transformation\_Notebook.ipynb"

}

# Function to execute a notebook

def execute\_notebook(notebook\_path):

try:

# Execute the notebook using a command-line tool (e.g., nbconvert or databricks-cli)

result = subprocess.run(['databricks', 'notebook', 'run', '--path', notebook\_path], check=True)

logging.info(f"Successfully executed {notebook\_path}")

return True

except Exception as e:

logging.error(f"Failed to execute {notebook\_path}: {e}")

return False

# Main pipeline execution

def run\_pipeline():

for name, path in notebooks.items():

# Check if the notebook file exists

if not os.path.exists(path):

logging.error(f"Notebook file not found: {path}")

break

# Execute the notebook

success = execute\_notebook(path)

if not success:

logging.error(f"Pipeline failed at step: {name}")

break

else:

logging.info("Pipeline executed successfully!")

if \_\_name\_\_ == "\_\_main\_\_":

run\_pipeline()

**2.**

**Task 1: Raw Data Ingestion**

from pyspark.sql.types import StructType, StructField, StringType, DateType, FloatType,

IntegerType

from pyspark.sql import SparkSession

import logging

# Set up logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -

%(message)s',

handlers=[logging.FileHandler("/dbfs/tmp/raw\_data\_ingestion\_errors.log"),

logging.StreamHandler()])

# Define the schema

schema = StructType([

StructField("City", StringType(), True),

StructField("Date", DateType(), True),

StructField("Temperature", FloatType(), True),

StructField("Humidity", IntegerType(), True)

])

# Define file path

file\_path = "/dbfs/tmp/weather\_data.csv"

try:

# Load the CSV file into a DataFrame

weather\_df = spark.read.csv(file\_path, schema=schema, header=True)

# Log success

logging.info("CSV file loaded successfully.")

# Display the DataFrame (optional)

display(weather\_df)

# Write the DataFrame to a Delta table

delta\_table\_path = "/mnt/delta/weather\_data"

weather\_df.write.format("delta").mode("overwrite").save(delta\_table\_path)

# Log success

logging.info("Data successfully written to Delta table.")

except Exception as e:

# Handle missing file or other errors

error\_message = f"Error loading CSV file from {file\_path}: {str(e)}"

logging.error(error\_message)

# save error logs to a Delta table or file

error\_log\_df = spark.createDataFrame([(error\_message,)], ["Error"])

error\_log\_df.write.format("delta").mode("append").save("/mnt/delta/error\_logs")

**Task 2: Data Cleaning**

from pyspark.sql import SparkSession

import logging

# Set up logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -

%(message)s',

handlers=[logging.FileHandler("/dbfs/tmp/data\_cleaning\_errors.log"),

logging.StreamHandler()])

# Step 1: Load the raw data from Delta table

delta\_table\_path = "/mnt/delta/weather\_data"

raw\_weather\_df = spark.read.format("delta").load(delta\_table\_path)

# Step 2: Remove rows with null values

cleaned\_weather\_df = raw\_weather\_df.na.drop(subset=["Temperature", "Humidity"])

# Step 3: Filter out rows with invalid Temperature and Humidity values

cleaned\_weather\_df = cleaned\_weather\_df.filter((cleaned\_weather\_df.Temperature >= -50)

&

(cleaned\_weather\_df.Temperature <= 60) &

(cleaned\_weather\_df.Humidity >= 0) &

(cleaned\_weather\_df.Humidity <= 100))

# Step 4: Save the cleaned data to a new Delta table

cleaned\_delta\_table\_path = "/mnt/delta/cleaned\_weather\_data"

cleaned\_weather\_df.write.format("delta").mode("overwrite").save(cleaned\_delta\_table\_path)

# Log success

logging.info("Cleaned weather data successfully saved to new Delta table.")

**Task 3: Data Transformation**

from pyspark.sql.functions import avg

# Calculate the average temperature and humidity for each city

transformed\_weather\_df = cleaned\_weather\_df.groupBy("City").agg(

avg("Temperature").alias("AverageTemperature"),

avg("Humidity").alias("AverageHumidity")

)

# Display the transformed data (optional)

display(transformed\_weather\_df)

# Define the path to the new Delta table for transformed data

transformed\_delta\_table\_path = "/mnt/delta/transformed\_weather\_data"

# Write the transformed data to a new Delta table

transformed\_weather\_df.write.format("delta").mode("overwrite").save(transformed\_delta\_ta

ble\_path)

# Log the successful save operation

import logging

logging.info("Transformed weather data (average temperature and humidity) successfully

saved to Delta table.")

**Task 4: Build and Run a Pipeline**

# Import necessary libraries

import logging

from databricks import notebook

# Set up logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s',

handlers=[logging.FileHandler("/dbfs/tmp/pipeline\_logs.log"), logging.StreamHandler()])

def run\_notebook(notebook\_path):

try:

# Run the notebook

notebook.run(notebook\_path, timeout\_seconds=600)

logging.info(f"Successfully executed notebook: {notebook\_path}")

except Exception as e:

logging.error(f"Error executing notebook {notebook\_path}: {str(e)}")

raise

# Step 1: Run Data Ingestion Notebook

try:

run\_notebook("/path/to/raw\_data\_ingestion\_notebook")

except Exception as e:

logging.error("Data Ingestion Failed. Terminating Pipeline.")

raise

# Step 2: Run Data Cleaning Notebook

try:

run\_notebook("/path/to/data\_cleaning\_notebook")

except Exception as e:

logging.error("Data Cleaning Failed. Terminating Pipeline.")

raise

# Step 3: Run Data Transformation Notebook

try:

run\_notebook("/path/to/data\_transformation\_notebook")

except Exception as e:

logging.error("Data Transformation Failed. Terminating Pipeline.")

raise

logging.info("Pipeline execution completed successfully.")

**3.**

**Task 1: Customer Data Ingestion**

from pyspark.sql import SparkSession

from pyspark.sql.utils import AnalysisException

spark = SparkSession.builder.appName("CustomerDataIngestion").getOrCreate()

# Path to the CSV file

file\_path = "/path/to/customer\_transactions.csv"

try:

# Load CSV file into a DataFrame

customer\_df = spark.read.option("header", True).csv(file\_path)

# Write DataFrame into a Delta table

customer\_df.write.format("delta").mode("overwrite").save("/delta/customer\_transactions")

except AnalysisException as e:

print(f"File not found or error while loading the file: {e}")

**Task 2: Data Cleaning**

# Drop duplicates

cleaned\_df = customer\_df.dropDuplicates()

# Handle nulls in the TransactionAmount column by filling with 0

cleaned\_df = cleaned\_df.fillna({'TransactionAmount': 0})

# Write cleaned data to a new Delta table

cleaned\_df.write.format("delta").mode("overwrite").save("/delta/cleaned\_customer\_transacti

ons")

**Task 3: Data Aggregation Aggregate**

aggregated\_df =

cleaned\_df.groupBy("ProductCategory").sum("TransactionAmount").alias("TotalTransaction

Amount")

# Save aggregated data to a Delta table

aggregated\_df.write.format("delta").mode("overwrite").save("/delta/aggregated\_customer\_tra

nsactions")

**Task 4: Pipeline Creation**

def ingest\_data(file\_path):

try:

customer\_df = spark.read.option("header", True).csv(file\_path)

customer\_df.write.format("delta").mode("overwrite").save("/delta/customer\_transactions")

return customer\_df

except Exception as e:

print(f"Error during ingestion: {e}")

return None

def clean\_data(df):

try:

df\_cleaned = df.dropDuplicates().fillna({'TransactionAmount': 0})

df\_cleaned.write.format("delta").mode("overwrite").save("/delta/cleaned\_customer\_transacti

ons")

return df\_cleaned

except Exception as e:

print(f"Error during cleaning: {e}")

return None

def aggregate\_data(df\_cleaned):

try:

df\_aggregated = df\_cleaned.groupBy("ProductCategory").sum("TransactionAmount")

df\_aggregated.write.format("delta").mode("overwrite").save("/delta/aggregated\_customer\_tra

nsactions")

except Exception as e:

print(f"Error during aggregation: {e}")

# File path to the raw data

file\_path = "/path/to/customer\_transactions.csv"

# Execute the pipeline

df\_raw = ingest\_data(file\_path)

if df\_raw is not None:

df\_cleaned = clean\_data(df\_raw)

if df\_cleaned is not None:

aggregate\_data(df\_cleaned)

**Task 5: Data Validation**

# Get the total transaction count from raw data

total\_transactions\_raw = df\_cleaned.count()

# Get the total transaction amount from the aggregated data

df\_aggregated = spark.read.format("delta").load("/delta/aggregated\_customer\_transactions")

total\_transactions\_aggregated =

df\_aggregated.selectExpr("sum(TransactionAmount)").collect()[0][0]

if total\_transactions\_raw == total\_transactions\_aggregated:

print("Data validation passed!")

else:

print("Data validation failed!")

**4.**

**Task 1: Product Inventory Data Ingestion**

from pyspark.sql import SparkSession

from pyspark.sql.utils import AnalysisException

spark = SparkSession.builder.appName("ProductInventoryIngestion").getOrCreate()

# Path to the CSV file

file\_path = "/path/to/product\_inventory.csv"

try:

# Load CSV into a DataFrame

inventory\_df = spark.read.option("header", True).csv(file\_path)

# Write DataFrame into a Delta table

inventory\_df.write.format("delta").mode("overwrite").save("/delta/product\_inventory")

except AnalysisException as e:

print(f"File not found or error loading file: {e}")

**Task 2: Data Cleaning**

# Remove rows with negative StockQuantity

cleaned\_inventory\_df = inventory\_df.filter(inventory\_df.StockQuantity >= 0)

# Fill null values in StockQuantity and Price columns

cleaned\_inventory\_df = cleaned\_inventory\_df.fillna({'StockQuantity': 0, 'Price': 0})

# Write cleaned data to a new Delta table

cleaned\_inventory\_df.write.format("delta").mode("overwrite").save("/delta/cleaned\_product\_

inventory")

**Task 3: Inventory Analysis**

from pyspark.sql.functions import col

# Calculate total stock value

inventory\_analysis\_df = cleaned\_inventory\_df.withColumn("TotalStockValue",

col("StockQuantity") \* col("Price"))

# Find products that need restocking

restock\_df = cleaned\_inventory\_df.filter(cleaned\_inventory\_df.StockQuantity < 100)

# Save analysis results to a Delta table

inventory\_analysis\_df.write.format("delta").mode("overwrite").save("/delta/inventory\_analys

is")

restock\_df.write.format("delta").mode("overwrite").save("/delta/restock\_products")

**Task 4: Build an Inventory Pipeline**

def ingest\_product\_data(file\_path):

try:

inventory\_df = spark.read.option("header", True).csv(file\_path)

inventory\_df.write.format("delta").mode("overwrite").save("/delta/product\_inventory")

return inventory\_df

except Exception as e:

print(f"Error during ingestion: {e}")

return None

def clean\_product\_data(df):

try:

cleaned\_df = df.filter(df.StockQuantity >= 0).fillna({'StockQuantity': 0, 'Price': 0})

cleaned\_df.write.format("delta").mode("overwrite").save("/delta/cleaned\_product\_inventory"

)

return cleaned\_df

except Exception as e:

print(f"Error during cleaning: {e}")

return None

def analyze\_inventory(df\_cleaned):

try:

analysis\_df = df\_cleaned.withColumn("TotalStockValue", col("StockQuantity") \*

col("Price"))

restock\_df = df\_cleaned.filter(df\_cleaned.StockQuantity < 100)

analysis\_df.write.format("delta").mode("overwrite").save("/delta/inventory\_analysis")

restock\_df.write.format("delta").mode("overwrite").save("/delta/restock\_products")

except Exception as e:

print(f"Error during analysis: {e}")

# File path to the raw data

file\_path = "/path/to/product\_inventory.csv"

# Execute the pipeline

df\_raw = ingest\_product\_data(file\_path)

if df\_raw is not None:

df\_cleaned = clean\_product\_data(df\_raw)

if df\_cleaned is not None:

analyze\_inventory(df\_cleaned)

**Task 5: Inventory Monitoring**

from pyspark.sql import DataFrame

# Load the cleaned product inventory Delta table

df\_inventory = spark.read.format("delta").load("/delta/cleaned\_product\_inventory")

# Find products that need restocking

restock\_alert\_df = df\_inventory.filter(df\_inventory.StockQuantity < 50)

# Send an alert if any product needs restocking

if restock\_alert\_df.count() > 0:

print("ALERT: Some products need restocking!")

else:

print("All products are sufficiently stocked.")

**5.**

**Task 1: Employee Attendance Data Ingestion**

from pyspark.sql import SparkSession

from pyspark.sql.utils import AnalysisException

spark = SparkSession.builder.appName("EmployeeAttendanceIngestion").getOrCreate()

# Path to the CSV file

file\_path = "/path/to/employee\_attendance.csv"

try:

# Load CSV into DataFrame

attendance\_df = spark.read.option("header", True).csv(file\_path)

# Write the DataFrame into a Delta table

attendance\_df.write.format("delta").mode("overwrite").save("/delta/employee\_attendance")

except AnalysisException as e:

print(f"File not found or error while loading the file: {e}")

**Task 2: Data Cleaning**

from pyspark.sql.functions import col, unix\_timestamp

# Remove rows with null or invalid CheckInTime or CheckOutTime

cleaned\_df = attendance\_df.filter((col("CheckInTime").isNotNull()) &

(col("CheckOutTime").isNotNull()))

# Calculate HoursWorked based on CheckInTime and CheckOutTime

cleaned\_df = cleaned\_df.withColumn("CalculatedHoursWorked",

(unix\_timestamp("CheckOutTime", 'HH:mm') - unix\_timestamp("CheckInTime", 'HH:mm'))

/ 3600)

# Write cleaned data to a new Delta table

cleaned\_df.write.format("delta").mode("overwrite").save("/delta/cleaned\_employee\_attendan

ce")

**Task 3: Attendance Summary**

from pyspark.sql.functions import month, sum

# Calculate total hours worked by each employee for the current month

current\_month = 3

summary\_df = cleaned\_df.filter(month(col("Date")) == current\_month) \

.groupBy("EmployeeID") \

.agg(sum("CalculatedHoursWorked").alias("TotalHoursWorked"))

# Find employees who worked overtime (more than 8 hours on any given day)

overtime\_df = cleaned\_df.filter(col("CalculatedHoursWorked") > 8)

# Save the summary and overtime data to Delta tables

summary\_df.write.format("delta").mode("overwrite").save("/delta/employee\_attendance\_sum

mary")

overtime\_df.write.format("delta").mode("overwrite").save("/delta/employee\_overtime")

**Task 4: Create an Attendance Pipeline**

def ingest\_attendance\_data(file\_path):

try:

attendance\_df = spark.read.option("header", True).csv(file\_path)

attendance\_df.write.format("delta").mode("overwrite").save("/delta/employee\_attendance")

return attendance\_df

except Exception as e:

print(f"Error during ingestion: {e}")

return None

def clean\_attendance\_data(df):

try:

cleaned\_df = df.filter((col("CheckInTime").isNotNull()) &

(col("CheckOutTime").isNotNull()))

cleaned\_df = cleaned\_df.withColumn("CalculatedHoursWorked",

(unix\_timestamp("CheckOutTime", 'HH:mm') - unix\_timestamp("CheckInTime", 'HH:mm'))

/ 3600)

cleaned\_df.write.format("delta").mode("overwrite").save("/delta/cleaned\_employee\_attendan

ce")

return cleaned\_df

except Exception as e:

print(f"Error during cleaning: {e}")

return None

def summarize\_attendance(df\_cleaned):

try:

summary\_df =

df\_cleaned.groupBy("EmployeeID").agg(sum("CalculatedHoursWorked").alias("TotalHours

Worked"))

overtime\_df = df\_cleaned.filter(col("CalculatedHoursWorked") > 8)

summary\_df.write.format("delta").mode("overwrite").save("/delta/employee\_attendance\_sum

mary")

overtime\_df.write.format("delta").mode("overwrite").save("/delta/employee\_overtime")

except Exception as e:

print(f"Error during summarization: {e}")

# File path to the raw data

file\_path = "/path/to

**Task 5: Time Travel with Delta Lake**

# Assuming you want to roll back the attendance logs

spark.sql("CREATE OR REPLACE TABLE attendance\_logs AS SELECT \* FROM

delta.`/mnt/delta/attendance\_logs` VERSION AS OF <version\_number>")

spark.sql("DESCRIBE HISTORY delta.`/mnt/delta/attendance\_logs`").show(truncate=False)