Detailed methods to read data from both **Databricks Unity Catalog** and **Azure Blob Storage** in your Streamlit app:

Method 1: Read from Databricks Unity Catalog

Option A: Using Databricks SQL Connector (Recommended)

Step 1: Update requirements.txt

```
streamlit
databricks-sql-connector
pandas
```

Step 2: Get Databricks Connection Details

You need:

1. **Server hostname:** e.g., `adb-xxxxxxxxx.azuredatabricks.net`
2. **HTTP path:** e.g., `/sql/1.0/warehouses/xxxxx`
3. **Access token:** Personal Access Token (PAT)

To get these:

- Go to Databricks workspace
- Click your profile → **Settings** → **Developer** → **Access tokens** → Generate new token
- For SQL Warehouse: **SQL Warehouses** → Select warehouse → **Connection details**

Step 3: Create Streamlit App

```
import streamlit as st
from databricks import sql
import pandas as pd
import os

st.title("Databricks Unity Catalog Data Reader")

# Databricks connection details
```

```
DATABRICKS_SERVER_HOSTNAME = "adb-xxxxxxxxx.azuredatabricks.net"
DATABRICKS_HTTP_PATH = "/sql/1.0/warehouses/xxxxx"
DATABRICKS_TOKEN = os.environ.get("DATABRICKS_TOKEN")  # Store securely!

# Or use Streamlit secrets (recommended)
# DATABRICKS_TOKEN = st.secrets["databricks"]["token"]

@st.cache_data(ttl=600)  # Cache for 10 minutes
def load_data_from_databricks(query):
    """Load data from Databricks Unity Catalog"""
    try:
        with sql.connect(
            server_hostname=DATABRICKS_SERVER_HOSTNAME,
            http_path=DATABRICKS_HTTP_PATH,
            access_token=DATABRICKS_TOKEN
        ) as connection:
            with connection.cursor() as cursor:
                cursor.execute(query)
                # Fetch results as pandas DataFrame
                df = cursor.fetchall_arrow().to_pandas()
                return df
    except Exception as e:
        st.error(f"Error connecting to Databricks: {e}")
        return None

# Example queries
catalog = st.text_input("Catalog name", "main")
schema = st.text_input("Schema name", "default")
table = st.text_input("Table name", "my_table")

if st.button("Load Data"):
    query = f"SELECT * FROM {catalog}.{schema}.{table} LIMIT 1000"

    with st.spinner("Loading data from Databricks..."):
        df = load_data_from_databricks(query)

    if df is not None:
        st.success(f"Loaded {len(df)} rows!")
        st.dataframe(df)

        # Download option
        csv = df.to_csv(index=False)
        st.download_button(
            label="Download as CSV",
            data=csv,
            file_name="databricks_data.csv",
            mime="text/csv"
        )
```

Step 4: Store Credentials Securely

Option 1: Using Azure App Settings (Recommended for production)

In Azure Portal:

- Go to **sample7 → Configuration → Application settings**
- Add new setting:
  - Name: `DATABRICKS_TOKEN`
  - Value: Your Databricks token
- Click **Save**

Option 2: Using Streamlit Secrets (For local development)

Create `.streamlit/secrets.toml`:

```
[databricks]
server_hostname = "adb-xxxxxxxxx.azuredatabricks.net"
http_path = "/sql/1.0/warehouses/xxxxx"
token = "dapi-xxxxxxxxxxxxxxxxxxxxx"
```

Then in your app:

```
DATABRICKS_TOKEN = st.secrets["databricks"]["token"]
```

Option B: Using Databricks REST API

```
import streamlit as st
import requests
import pandas as pd
import os

st.title("Databricks Unity Catalog via REST API")

DATABRICKS_INSTANCE = "https://adb-xxxxxxxxx.azuredatabricks.net"
DATABRICKS_TOKEN = os.environ.get("DATABRICKS_TOKEN")

@st.cache_data(ttl=600)
def execute_query_api(query):
    """Execute SQL query via Databricks REST API"""
    url = f"{DATABRICKS_INSTANCE}/api/2.0/sql/statements"

    headers = {
        "Authorization": f"Bearer {DATABRICKS_TOKEN}",
```

```
        "Content-Type": "application/json"
    }

    payload = {
        "statement": query,
        "warehouse_id": "xxxxx",  # Your warehouse ID
        "wait_timeout": "30s"
    }

    try:
        response = requests.post(url, headers=headers, json=payload)
        response.raise_for_status()
        result = response.json()

        # Convert to DataFrame
        if result.get("status", {}).get("state") == "SUCCEEDED":
            columns = [col["name"] for col in result["manifest"]["schema"]["columns"]]
            data = result["result"]["data_array"]
            df = pd.DataFrame(data, columns=columns)
            return df
        else:
            st.error(f"Query failed: {result}")
            return None

    except Exception as e:
        st.error(f"API Error: {e}")
        return None

# Usage
query = st.text_area("Enter SQL Query", "SELECT * FROM main.default.my_table LIMIT
100")

if st.button("Execute Query"):
    df = execute_query_api(query)
    if df is not None:
        st.dataframe(df)
```

Method 2: Read from Azure Blob Storage

Option A: Using azure-storage-blob (Recommended)

Step 1: Update requirements.txt

```
streamlit
```

azure-storage-blob
pandas

Step 2: Get Azure Storage Credentials

You need:

1. **Storage Account Name:** e.g., `mystorageaccount`
2. **Container Name:** e.g., `mycontainer`
3. **Access Key** or **SAS Token** or **Managed Identity**

To get Access Key:

- Azure Portal → Your Storage Account → **Security + networking** → **Access keys**
- Copy **Key1** or **Key2**

Step 3: Create Streamlit App

```
import streamlit as st
from azure.storage.blob import BlobServiceClient, ContainerClient
import pandas as pd
import io
import os

st.title("Azure Blob Storage Data Reader")

# Azure Storage credentials
STORAGE_ACCOUNT_NAME = "mystorageaccount"
STORAGE_ACCOUNT_KEY = os.environ.get("AZURE_STORAGE_KEY")  # Store
securely!
CONTAINER_NAME = "mycontainer"

# Create connection string
connection_string =
f"DefaultEndpointsProtocol=https;AccountName={STORAGE_ACCOUNT_NAME};AccountK
ey={STORAGE_ACCOUNT_KEY};EndpointSuffix=core.windows.net"

@st.cache_data(ttl=600)
def list_blobs(container_name):
    """List all blobs in container"""
    try:
        blob_service_client = BlobServiceClient.from_connection_string(connection_string)
        container_client = blob_service_client.get_container_client(container_name)

        blob_list = []
        for blob in container_client.list_blobs():
            blob_list.append(blob.name)
```

```
        return blob_list
    except Exception as e:
        st.error(f"Error listing blobs: {e}")
        return []

@st.cache_data(ttl=600)
def read_csv_from_blob(container_name, blob_name):
    """Read CSV file from Azure Blob Storage"""
    try:
        blob_service_client = BlobServiceClient.from_connection_string(connection_string)
        blob_client = blob_service_client.get_blob_client(container=container_name,
blob=blob_name)

        # Download blob content
        blob_data = blob_client.download_blob()
        content = blob_data.readall()

        # Read as pandas DataFrame
        df = pd.read_csv(io.BytesIO(content))
        return df

    except Exception as e:
        st.error(f"Error reading blob: {e}")
        return None

@st.cache_data(ttl=600)
def read_parquet_from_blob(container_name, blob_name):
    """Read Parquet file from Azure Blob Storage"""
    try:
        blob_service_client = BlobServiceClient.from_connection_string(connection_string)
        blob_client = blob_service_client.get_blob_client(container=container_name,
blob=blob_name)

        blob_data = blob_client.download_blob()
        content = blob_data.readall()

        # Read as pandas DataFrame
        df = pd.read_parquet(io.BytesIO(content))
        return df

    except Exception as e:
        st.error(f"Error reading parquet: {e}")
        return None

@st.cache_data(ttl=600)
def read_excel_from_blob(container_name, blob_name):
    """Read Excel file from Azure Blob Storage"""
```

```
    try:
        blob_service_client = BlobServiceClient.from_connection_string(connection_string)
        blob_client = blob_service_client.get_blob_client(container=container_name,
blob=blob_name)

        blob_data = blob_client.download_blob()
        content = blob_data.readall()

        # Read as pandas DataFrame
        df = pd.read_excel(io.BytesIO(content))
        return df

    except Exception as e:
        st.error(f"Error reading excel: {e}")
        return None

# UI
st.subheader("Browse Files")

container_input = st.text_input("Container Name", CONTAINER_NAME)

if st.button("List Files"):
    with st.spinner("Loading file list..."):
        blobs = list_blobs(container_input)

    if blobs:
        st.success(f"Found {len(blobs)} files")

        # Select file
        selected_blob = st.selectbox("Select a file to read:", blobs)

        # Detect file type
        if selected_blob.endswith('.csv'):
            if st.button("Read CSV"):
                df = read_csv_from_blob(container_input, selected_blob)
                if df is not None:
                    st.dataframe(df)
                    st.info(f"Shape: {df.shape}")

        elif selected_blob.endswith('.parquet'):
            if st.button("Read Parquet"):
                df = read_parquet_from_blob(container_input, selected_blob)
                if df is not None:
                    st.dataframe(df)
                    st.info(f"Shape: {df.shape}")

        elif selected_blob.endswith(('.xlsx', '.xls')):
            if st.button("Read Excel"):
```

```
            df = read_excel_from_blob(container_input, selected_blob)
            if df is not None:
                st.dataframe(df)
                st.info(f"Shape: {df.shape}")
```

Option B: Using SAS Token (More Secure)

```
import streamlit as st
from azure.storage.blob import BlobServiceClient
import pandas as pd
import io

st.title("Azure Blob with SAS Token")

# Using SAS token instead of account key
STORAGE_ACCOUNT_NAME = "mystorageaccount"
SAS_TOKEN = os.environ.get("AZURE_SAS_TOKEN")  # e.g.,
"?sv=2021-06-08&ss=b&srt=sco&sp=rl..."
CONTAINER_NAME = "mycontainer"

# Create blob service client with SAS
account_url = f"https://{STORAGE_ACCOUNT_NAME}.blob.core.windows.net"
blob_service_client = BlobServiceClient(account_url=account_url, credential=SAS_TOKEN)

@st.cache_data(ttl=600)
def read_blob_with_sas(container_name, blob_name):
    """Read blob using SAS token"""
    try:
        blob_client = blob_service_client.get_blob_client(container=container_name,
blob=blob_name)
        blob_data = blob_client.download_blob()
        content = blob_data.readall()

        if blob_name.endswith('.csv'):
            df = pd.read_csv(io.BytesIO(content))
        elif blob_name.endswith('.parquet'):
            df = pd.read_parquet(io.BytesIO(content))
        else:
            st.error("Unsupported file format")
            return None

        return df

    except Exception as e:
        st.error(f"Error: {e}")
        return None
```

```
# Usage
blob_name = st.text_input("Blob name (e.g., data/myfile.csv)")
if st.button("Load Data"):
    df = read_blob_with_sas(CONTAINER_NAME, blob_name)
    if df is not None:
        st.dataframe(df)
```

Option C: Using Managed Identity (Most Secure for Azure)

```
import streamlit as st
from azure.storage.blob import BlobServiceClient
from azure.identity import DefaultAzureCredential
import pandas as pd
import io

st.title("Azure Blob with Managed Identity")

STORAGE_ACCOUNT_NAME = "mystorageaccount"
CONTAINER_NAME = "mycontainer"

# Use Managed Identity
credential = DefaultAzureCredential()
account_url = f"https://{STORAGE_ACCOUNT_NAME}.blob.core.windows.net"
blob_service_client = BlobServiceClient(account_url=account_url, credential=credential)

@st.cache_data(ttl=600)
def read_blob_with_managed_identity(container_name, blob_name):
    """Read blob using Managed Identity"""
    try:
        blob_client = blob_service_client.get_blob_client(container=container_name,
blob=blob_name)
        blob_data = blob_client.download_blob()
        content = blob_data.readall()

        df = pd.read_csv(io.BytesIO(content))
        return df

    except Exception as e:
        st.error(f"Error: {e}")
        return None

# Usage
blob_name = st.text_input("Blob path", "data/sales.csv")
```

```
if st.button("Load"):
    df = read_blob_with_managed_identity(CONTAINER_NAME, blob_name)
    if df is not None:
        st.dataframe(df)
```

To enable Managed Identity:

1. Azure Portal → **sample7** → **Identity**
2. Enable **System assigned** identity
3. Go to your Storage Account → **Access Control (IAM)**
4. Add role assignment → **Storage Blob Data Reader** → Select your web app

---

Complete Example: Combined App

Here's a complete app that reads from both sources:

```
import streamlit as st
from databricks import sql
from azure.storage.blob import BlobServiceClient
import pandas as pd
import io
import os

st.title("Data Reader: Databricks & Azure Blob")

# Sidebar for data source selection
data_source = st.sidebar.radio("Select Data Source", ["Databricks Unity Catalog", "Azure Blob Storage"])

if data_source == "Databricks Unity Catalog":
    st.header("📊 Databricks Unity Catalog")

    # Connection details from environment variables
    DATABRICKS_SERVER_HOSTNAME = os.environ.get("DATABRICKS_SERVER_HOSTNAME")
    DATABRICKS_HTTP_PATH = os.environ.get("DATABRICKS_HTTP_PATH")
    DATABRICKS_TOKEN = os.environ.get("DATABRICKS_TOKEN")

    @st.cache_data(ttl=600)
    def load_databricks_data(query):
        with sql.connect(
            server_hostname=DATABRICKS_SERVER_HOSTNAME,
            http_path=DATABRICKS_HTTP_PATH,
            access_token=DATABRICKS_TOKEN
```

```
        ) as connection:
            with connection.cursor() as cursor:
                cursor.execute(query)
                return cursor.fetchall_arrow().to_pandas()

    # Input
    catalog = st.text_input("Catalog", "main")
    schema = st.text_input("Schema", "default")
    table = st.text_input("Table", "my_table")
    limit = st.number_input("Limit", min_value=1, max_value=10000, value=100)

    if st.button("Load from Databricks"):
        query = f"SELECT * FROM {catalog}.{schema}.{table} LIMIT {limit}"
        with st.spinner("Loading..."):
            df = load_databricks_data(query)
        st.success(f"Loaded {len(df)} rows")
        st.dataframe(df)

else:
    st.header("☁️ Azure Blob Storage")

    # Connection details
    STORAGE_ACCOUNT_NAME = os.environ.get("AZURE_STORAGE_ACCOUNT")
    STORAGE_ACCOUNT_KEY = os.environ.get("AZURE_STORAGE_KEY")
    connection_string =
f"DefaultEndpointsProtocol=https;AccountName={STORAGE_ACCOUNT_NAME};AccountK
ey={STORAGE_ACCOUNT_KEY};EndpointSuffix=core.windows.net"

    @st.cache_data(ttl=600)
    def load_blob_data(container, blob_name):
        blob_service_client = BlobServiceClient.from_connection_string(connection_string)
        blob_client = blob_service_client.get_blob_client(container=container,
blob=blob_name)
        blob_data = blob_client.download_blob()
        content = blob_data.readall()

        if blob_name.endswith('.csv'):
            return pd.read_csv(io.BytesIO(content))
        elif blob_name.endswith('.parquet'):
            return pd.read_parquet(io.BytesIO(content))
        else:
            return None

    # Input
    container = st.text_input("Container Name", "data")
    blob_path = st.text_input("Blob Path", "sales/sales_data.csv")

    if st.button("Load from Blob"):
```

```
with st.spinner("Loading..."):
    df = load_blob_data(container, blob_path)
if df is not None:
    st.success(f"Loaded {len(df)} rows")
    st.dataframe(df)
```

Update Your requirements.txt

```
streamlit
databricks-sql-connector
azure-storage-blob
azure-identity
pandas
pyarrow
openpyxl
```

Store Credentials in Azure

Go to Azure Portal → **sample7** → **Configuration** → **Application settings**

Add these:

For Databricks:

- `DATABRICKS_SERVER_HOSTNAME` = `adb-xxxxx.azuredatabricks.net`
- `DATABRICKS_HTTP_PATH` = `/sql/1.0/warehouses/xxxxx`
- `DATABRICKS_TOKEN` = `dapi-xxxxxxxx`

For Azure Blob:

- `AZURE_STORAGE_ACCOUNT` = `mystorageaccount`
- `AZURE_STORAGE_KEY` = `xxxxxxxx==`

Or use **SAS token:**

- `AZURE_SAS_TOKEN` = `?sv=2021...`

Click **Save** and restart your app!