

Bootcamp Project 2 — Transactions and Loan Data for a Customer

Project Objective

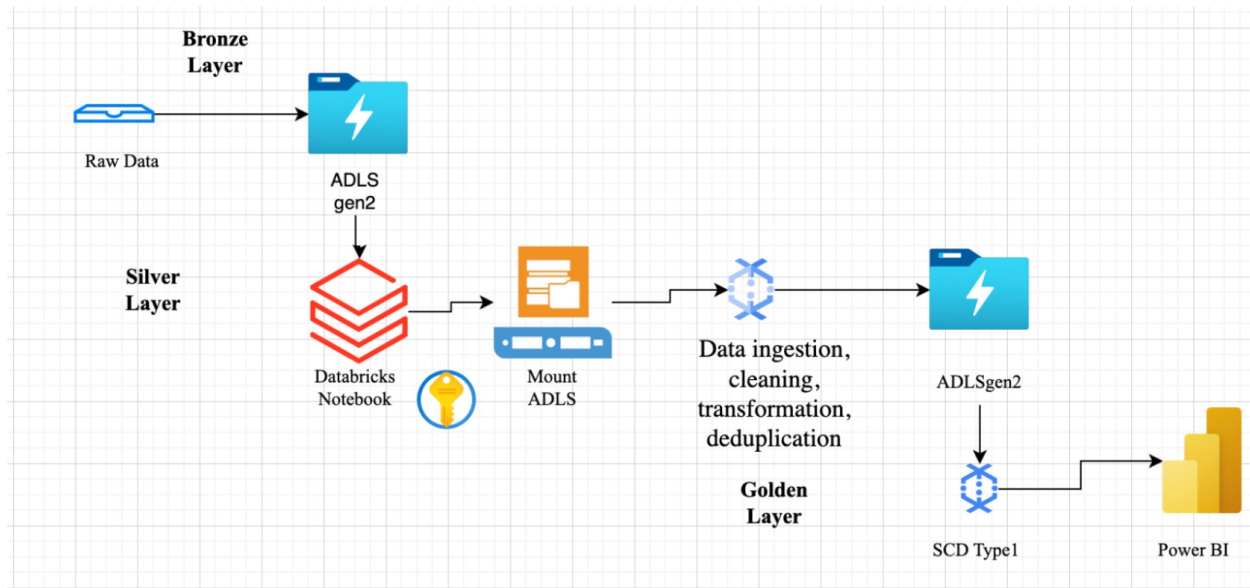
Design and implement a robust, scalable data pipeline for processing customer account and loan data using modern Azure services:

- Ingest data from **ADLS Gen2 (Bronze Layer)**
- Mount storage using **Service Principal Authentication**
- Clean and transform using **Databricks Notebooks (Silver Layer)**
- Create **Delta Tables with SCD Type 1 (Gold Layer)**
- Enable downstream analytics and **visualization in Power BI**

Tools and Technologies Used

- **Azure Data Lake Storage Gen2 (ADLS Gen2)**
- **Service Principal** for secure authentication.
- **Azure Key Vault**
- **Azure Databricks (PySpark and SQL)**
- **Delta Lake**
- **Power BI**

ARCHITECTURE DIAGRAM



Project Steps

Step 1: Data Ingestion (Bronze Layer)

- **Source Files:**
 - accounts.csv
 - customers.csv
 - loan_payments.csv
 - loans.csv
 - transactions.csv
- **Sink:**
 - ADLS Gen2 Raw (Bronze) Container
- **Reference:**
 - [Kaggle AI Bank Dataset](#)

Step 2: Mounting Storage & Data Cleaning (Silver Layer)

- **Mount ADLS Gen2 Storage** to Databricks using Service Principal credentials stored in Azure Key Vault

Microsoft Entra ID → Manage → App Registration → Create a service Principle → copy application ID and Tenant ID → go to clien services and create a new secrete

Home > Default Directory

Default Directory | App registrations

× < + New registration Endpoints Troubleshoot Refresh Download Preview features Got feedback?

Overview
Preview features
Diagnose and solve problems
Manage
Users
Groups
External Identities
Roles and administrators
Administrative units
Delegated admin partners
Enterprise applications
Devices
App registrations

1 applications found

Display name ↑↓	Application (client) ID	Created on ↑↓	Certificates & secrets
DA databricks	e05b0790-03c0-4c29-bd6c-0aa06989...	4/24/2025	⚠ Expiring soon

Add or remove favorites by pressing Cmd+Shift+F

< << [Got feedback?](#)

- Overview
- Quickstart
- Integration assistant
- Diagnose and solve problems
- Manage
 - Branding & properties
 - Authentication
 - Certificates & secrets**
 - Token configuration
 - API permissions
 - Expose an API
 - App roles
 - Owners

Credentials enable confidential applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTPS scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Application registration certificates, secrets and federated credentials can be found in the tabs below.

Certificates (0) **Client secrets (1)** Federated credentials (0)

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

+ New client secret

Description	Expires	Value ⓘ	Secret ID
connectadls	4/27/2025 ⚠	bXe*****	13fd79cc-1f10-4821-b1de-8fe2e9185791 📄 🗑️

AdHf n7 rmnuw faurvtet hu rnsccin FmI+Shift+F

copy secret value and create secret (app ID and app Value) in keyvalue

 < << [+ Generate/Import](#) [🔄 Refresh](#) [⬆️ Restore Backup](#) [🔗 Manage deleted secrets](#) [</> View sample code](#)

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Access policies
- Resource visualizer
- Events
- Objects
 - Keys
 - Secrets**
 - Certificates
- Settings

Name	Type	Status	Expiration date
appid1		✓ Enabled	
appValue1		✓ Enabled	
password		✓ Enabled	

Create a scope in databricks

adb-448833214983206.6.azuredatabricks.net/?o=448833214983206#secrets/createScope

Microsoft Azure databricks

Search data, notebooks, recents, and more... % + P

+ New

- Workspace
- Recents
- Catalog
- Workflows
- Compute
- Marketplace

SQL

- SQL Editor
- Queries
- Dashboards
- Genie
- Alerts
- Query History
- SQL Warehouses

Data Engineering

HomePage / Create Secret Scope

Create Secret Scope

Cancel Create

A store for secrets that is identified by a name and backed by a specific store type. [Learn more](#)

Scope Name [?]

avconnections

Manage Principal [?]

Creator

Azure Key Vault [?]

DNS Name

https://shubhkv.vault.azure.net/

Resource ID

/subscriptions/1a38e9a8-7466-4086-b2d7-49d34e933135/resourceGroups/sf

Now create a mount point in adls gen2

```

1 configs = {
2     "fs.azure.account.auth.type": "OAuth",
3     "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
4     "fs.azure.account.oauth2.client.id": dbutils.secrets.get(scope="avconnections", key="appid"),
5     "fs.azure.account.oauth2.client.secret": dbutils.secrets.get(scope="avconnections", key="appValue1"),
6     "fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/
7     f9087d13-42f9-4a25-8d93-312ee56664c2/oauth2/token"
8 }
9 dbutils.fs.mount(
10     source="abfss://edw@shubhaadlsgen2.dfs.core.windows.net/",
11     mount_point="/mnt/edw",
12     extra_configs=configs
13 )

```

Read the raw CSV files into Spark DataFrames.

Remove poor quality data and prepare consistent Parquet files.

Actions Taken:

- Dropped all **null values** (.na.drop()) to avoid incomplete records.
- Removed **duplicate records** (.dropDuplicates()).
- Saved each cleaned dataset as **Parquet** into the **Silver layer** (/mnt/edw/silver/...).

```
▶ ✓ 21 hours ago (23s) 5

# Define function to clean and write a DataFrame
def clean_and_write_csv(input_path, output_path):
    df = spark.read.format("csv").option("header", "true").load(input_path)
    df = df.na.drop().dropDuplicates()
    df.write.mode("overwrite").format("parquet").save(output_path)

# Process each dataset from Bronze → Silver
clean_and_write_csv("/mnt/edw/bronze_folder/accounts.csv", "/mnt/edw/silver/Accounts")
clean_and_write_csv("/mnt/edw/bronze_folder/customers.csv", "/mnt/edw/silver/Customers")
clean_and_write_csv("/mnt/edw/bronze_folder/transactions.csv", "/mnt/edw/silver/Transactions")
clean_and_write_csv("/mnt/edw/bronze_folder/loans.csv", "/mnt/edw/silver/Loans")
clean_and_write_csv("/mnt/edw/bronze_folder/loan_payments.csv", "/mnt/edw/silver/Loan_Payments")

▶ (15) Spark Jobs
```

Joining Datasets and Creating a Master Data Frame

Create a combined view containing necessary customer + account + transaction + loan + payment information.

Actions Taken:

- Performed multiple joins:
 - Joined Accounts and Customers on customer_id
 - Joined the result with Transactions on account_id
 - Joined further with Loans on customer_id
 - Finally joined with Loan Payments on loan_id
- Selected only required columns:
 - account_id, transaction_id, customer_id, loan_id, payment_id, amount, date
- Removed duplicates on the combination of selected fields.

```

from pyspark.sql.functions import col
# Step 1: Read cleaned data from Silver (Parquet)
df_account = spark.read.parquet("/mnt/edw/silver/Accounts")
df_customer = spark.read.parquet("/mnt/edw/silver/Customers")
df_transactions = spark.read.parquet("/mnt/edw/silver/Transactions")
df_loans = spark.read.parquet("/mnt/edw/silver/Loans")
df_loan_payments = spark.read.parquet("/mnt/edw/silver/Loan_Payments")

# Step 2: Join DataFrames (use proper keys based on schema)
df_join1 = df_account.join(df_customer, on="customer_id", how="left")
df_join2 = df_join1.join(df_transactions, on="account_id", how="left")
df_join3 = df_join2.join(df_loans, on="customer_id", how="left")
final_df = df_join3.join(df_loan_payments, on="loan_id", how="left")

# Step 3: Select required columns and remove duplicates
final_selected = final_df.select(
    col("account_id").cast("int"),
    col("transaction_id").cast("int"),
    col("customer_id").cast("int"),
    col("loan_id").cast("int"),
    col("payment_id").cast("int"),
    col("balance").cast("float"),
    col("transaction_date").cast("timestamp"),
    col("transaction_amount").cast("float"),
    col("loan_amount").cast("float"),
    col("payment_amount").cast("float"),
    col("payment_date").cast("timestamp")
).dropDuplicates()

```

Store the enriched, trusted master dataset into the **Gold Layer** in **Delta** format.

```

# Step 4: Write final DataFrame to Gold layer in Delta format
final_selected.write.mode("overwrite").format("delta").save("/mnt/edw/silver/Customer_Account_Loan_Data")

```

SCD Type 1 Implementation (Slowly Changing Dimension)

Goal:

Maintain the Gold Delta Tables by handling changes (update old records, insert new ones).

Actions Taken:

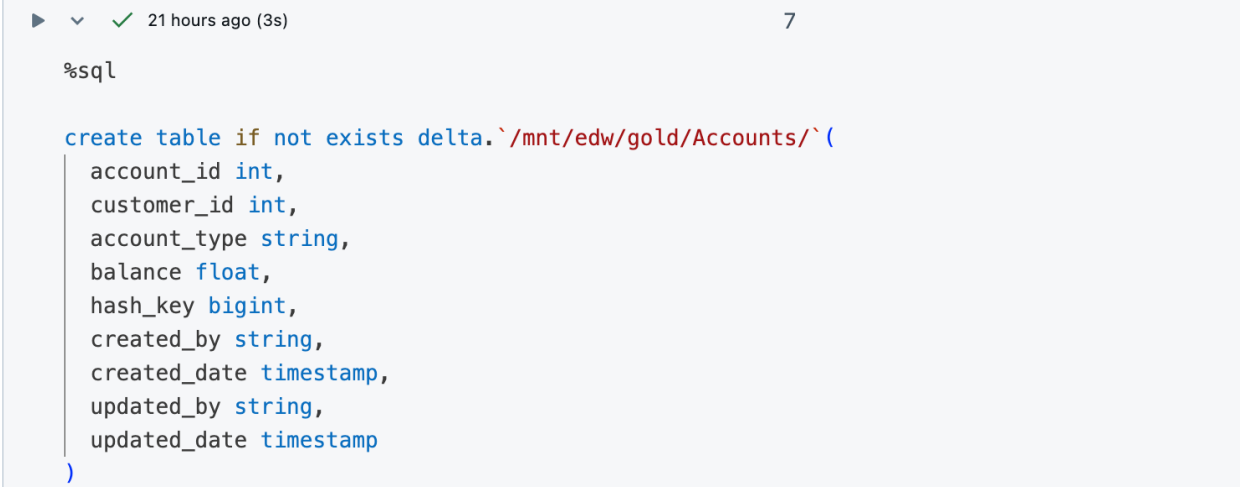
- Created a hash_key (using CRC32) on all columns to detect changes easily.
- Used **Delta Lake MERGE** operation:
 - **When Matched** → Update the record if changed.
 - **When Not Matched** → Insert as new record.
- Managed audit columns:
 - created_date, created_by

- updated_date, updated_by

Create a Delta Table (if not exists)

- Creates a **Delta table** for customer data.

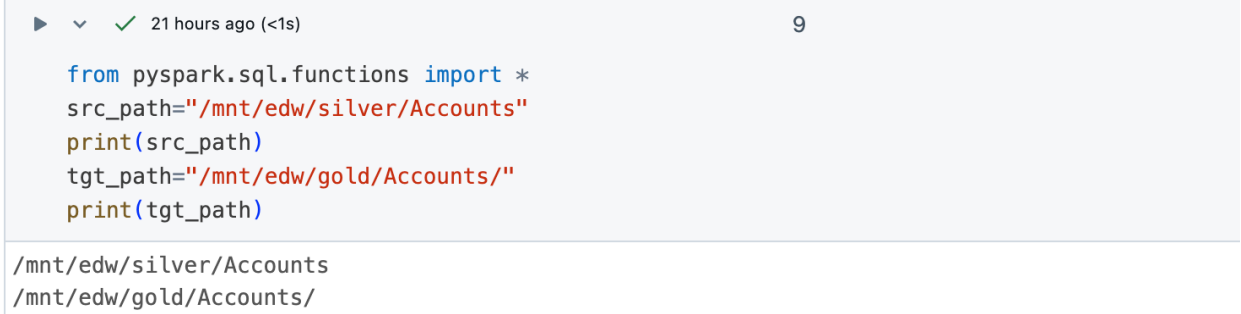
Stores data in **Delta format**, enabling ACID transactions & time travel.

- The screenshot shows a Databricks SQL query editor interface. At the top, it indicates the query was executed 21 hours ago (3s) and shows a progress bar at 7%. The query is a SQL statement to create a Delta table if it does not exist. The table is named 'delta' and is located at the path '/mnt/edw/gold/Accounts/'. The table has the following columns: account_id (int), customer_id (int), account_type (string), balance (float), hash_key (bigint), created_by (string), created_date (timestamp), updated_by (string), and updated_date (timestamp).

```
%sql

create table if not exists delta.`/mnt/edw/gold/Accounts/`(
  account_id int,
  customer_id int,
  account_type string,
  balance float,
  hash_key bigint,
  created_by string,
  created_date timestamp,
  updated_by string,
  updated_date timestamp
)
```

Define Source and Target path

The screenshot shows a Databricks Python script editor interface. It indicates the script was executed 21 hours ago (<1s) and shows a progress bar at 9%. The script imports all functions from pyspark.sql.functions and defines two paths: src_path as '/mnt/edw/silver/Accounts' and tgt_path as '/mnt/edw/gold/Accounts/'. The paths are printed to the console. Below the code, the output of the print statements is shown: '/mnt/edw/silver/Accounts' and '/mnt/edw/gold/Accounts/'.

```
from pyspark.sql.functions import *
src_path="/mnt/edw/silver/Accounts"
print(src_path)
tgt_path="/mnt/edw/gold/Accounts/"
print(tgt_path)
```

```
/mnt/edw/silver/Accounts
/mnt/edw/gold/Accounts/
```

read the data

The screenshot shows a Databricks Python script editor interface. It indicates the script was executed 21 hours ago (1s) and shows a progress bar at 10%. The script reads data from a Delta table located at src_path using the read.format("parquet").option("header", "true").option("inferSchema", "true").load(src_path) method. The resulting DataFrame is displayed. Below the code, it shows '(2) Spark Jobs'.

```
df_src=spark.read.format("parquet").option("header", "true").option("inferSchema", "true").load(src_path)
display(df_src)
```

▶ (2) Spark Jobs

Generate hashkey

▶

✓

21 hours ago (<1s)

11

P

```
df_src1=df_src.withColumn("hash_key",crc32(concat(*df_src.columns)))
display(df_src1)
```

▶ (1) Spark Jobs

▶

df_src1: pyspark.sql.dataframe.DataFrame = [account_id: string, customer_id: string ... 3 more fields]

Table ▼

+

	^A _C account_id	^A _C customer_id	^A _C account_type	^A _C balance	¹ ₂ ³ hash_key
1	1	45	Savings	1000.50	4261402674
2	48	6	Checking	4900.00	3216528439
3	21	53	Savings	300.25	2800704470
4	29	58	Savings	75.25	3637748452
5	34	41	Checking	3500.50	868746038
6	35	62	Savings	175.75	3427479153
7	5	56	Savings	500.00	2866974084
8	78	4	Checking	7900.50	2650882798

load data and perform anti join

▶

✓

21 hours ago (1s)

13

Python

⌵

```
df_src1=df_src1.alias("src").join(dhtable.toDF().alias("tgt"), ((col("src.account_id") == col("tgt.account_id")) & (col("src.hash_key") == col("tgt.hash_key"))), "anti").select(col("src.*"))
df_src1.show()
```

▶ (1) Spark Jobs

Merge Changes into the Delta Table

- **Merges new and updated records into Delta table.**
- **Updates existing records where ID matches.**
- **Inserts new records where no match is found.**

▶

✓

21 hours ago (6s)

14

Python

🗑️

✳️

⌵

⋮

```
dhtable.alias("tgt").merge(df_src1.alias("src"),"tgt.account_id = src.account_id")\
    .whenMatchedUpdate(set={"tgt.account_id":"src.account_id","tgt.customer_id":"src.customer_id","tgt.account_type":"src.account_type","tgt.balance":"src.balance","tgt.hash_key":"src.hash_key","tgt.updated_date":current_timestamp(),"tgt.updated_by":lit("databricks_Updated")})\
    .whenNotMatchedInsert(values={"tgt.account_id":"src.account_id","tgt.customer_id":"src.customer_id","tgt.account_type":"src.account_type","tgt.balance":"src.balance","tgt.hash_key":"src.hash_key","tgt.created_date":current_timestamp(),"tgt.created_by":lit("databricks"),"tgt.updated_date":current_timestamp(),"tgt.updated_by":lit("databricks")}).execute()
```

▶ (7) Spark Jobs

display the updated record

21 hours ago (1s)

15

Python

```
display(spark.read.format("delta").option("header", "true").load(tgt_path))
```

(1) Spark Jobs

Table

+

	pr_id	account_type	balance	hash_key	created_by	created_date	update
1	45	Savings	1000.5	4261402674	databricks	2025-04-26T00:08:59.087+00:00	databricks
2	6	Checking	4900	3216528439	databricks	2025-04-26T00:08:59.087+00:00	databricks
3	53	Savings	300.25	2800704470	databricks	2025-04-26T00:08:59.087+00:00	databricks
4	58	Savings	75.25	3637748452	databricks	2025-04-26T00:08:59.087+00:00	databricks
5	41	Checking	3500.5	868746038	databricks	2025-04-26T00:08:59.087+00:00	databricks
6	62	Savings	175.75	3427479153	databricks	2025-04-26T00:08:59.087+00:00	databricks
7	56	Savings	500	2866974084	databricks	2025-04-26T00:08:59.087+00:00	databricks
8	4	Checking	7900.5	2650882798	databricks	2025-04-26T00:08:59.087+00:00	databricks

databricks

Search data, notebooks, recents, and more...

ShubhaDB

project2

Python

Tabs: OFF

File

Edit

View

Run

Help

Last edit was 22 hours ago

Run all

murali krishna's Cluster

Schedule

Share

2 days ago (4s)

#dbutils.secrets.listScops

2 days ago (<1s)

dbutils.secrets.list("ask

Last execution failed

1 configs = {

2 "fs.azure.account.

3 "fs.azure.account.

4 "fs.azure.account.

5 "fs.azure.account.

6 "fs.azure.account.

7 f9087d13-42f9-4a25

8 }

9 dbutils.fs.mount(

Job name*

project2

Simple

Advanced

Schedule

Every

Day

at

15

:

27

Show cron syntax

Timezone

(UTC+00:00) UTC

Compute*

murali krishna's Cluster 16 GB · 4 Cores · DBR 15.4 LTS · Photon · Spark 3.5.0 · Scala 2.12

Jobs running on all-purpose clusters are considered all-purpose compute. Learn more

Cancel

Create