Retail Inventory Management Project

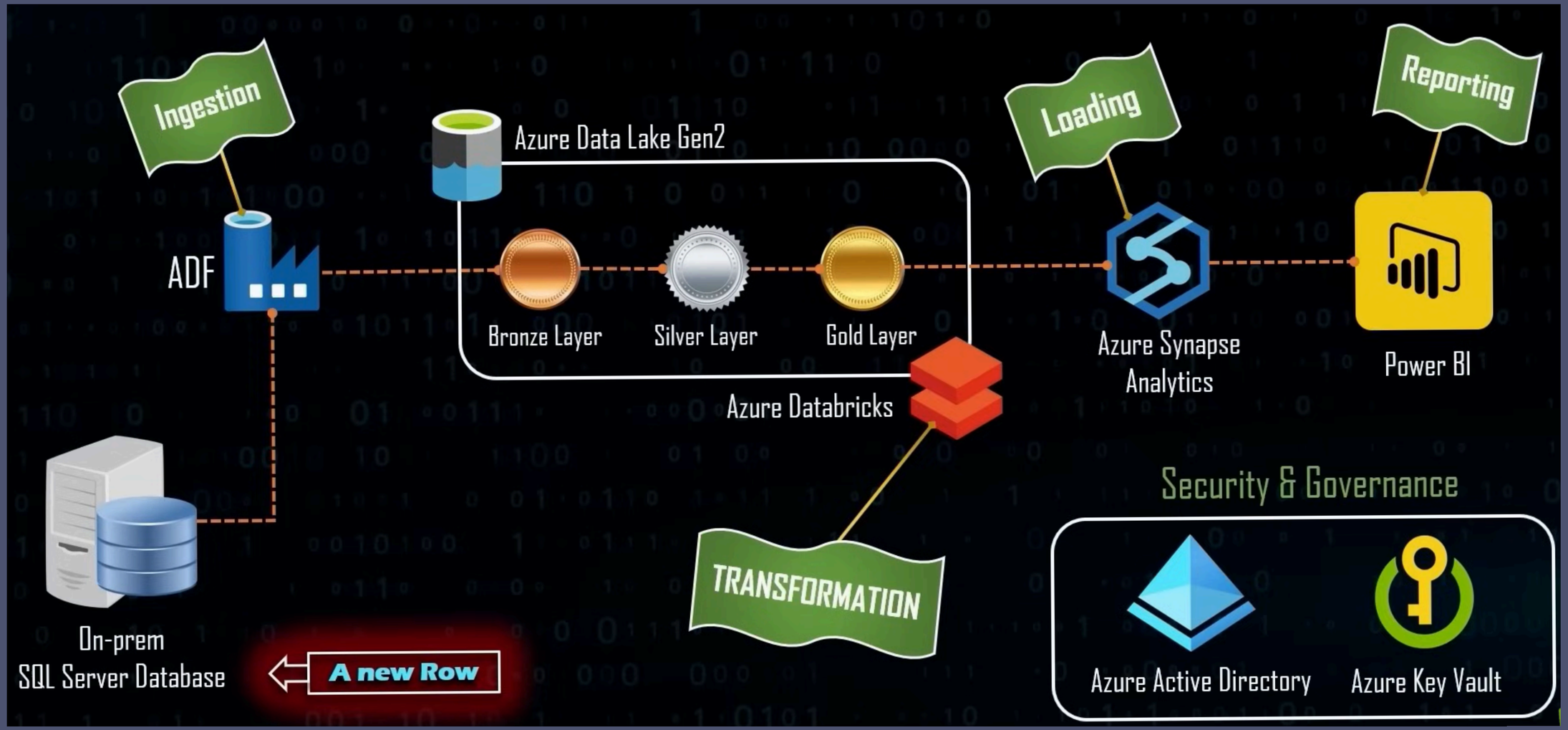
Goal

To create a comprehensive inventory management system that enables retail businesses to track stock, analyze sales data, and forecast demand.

Why It Matters?

- Reduces overstock and stockouts, leading to optimized storage costs.
- Improves customer satisfaction by ensuring stock availability.
- Provides insights for data-driven decision-making

PROJECT OVERVIEW ARCHTICTURE



Data Source

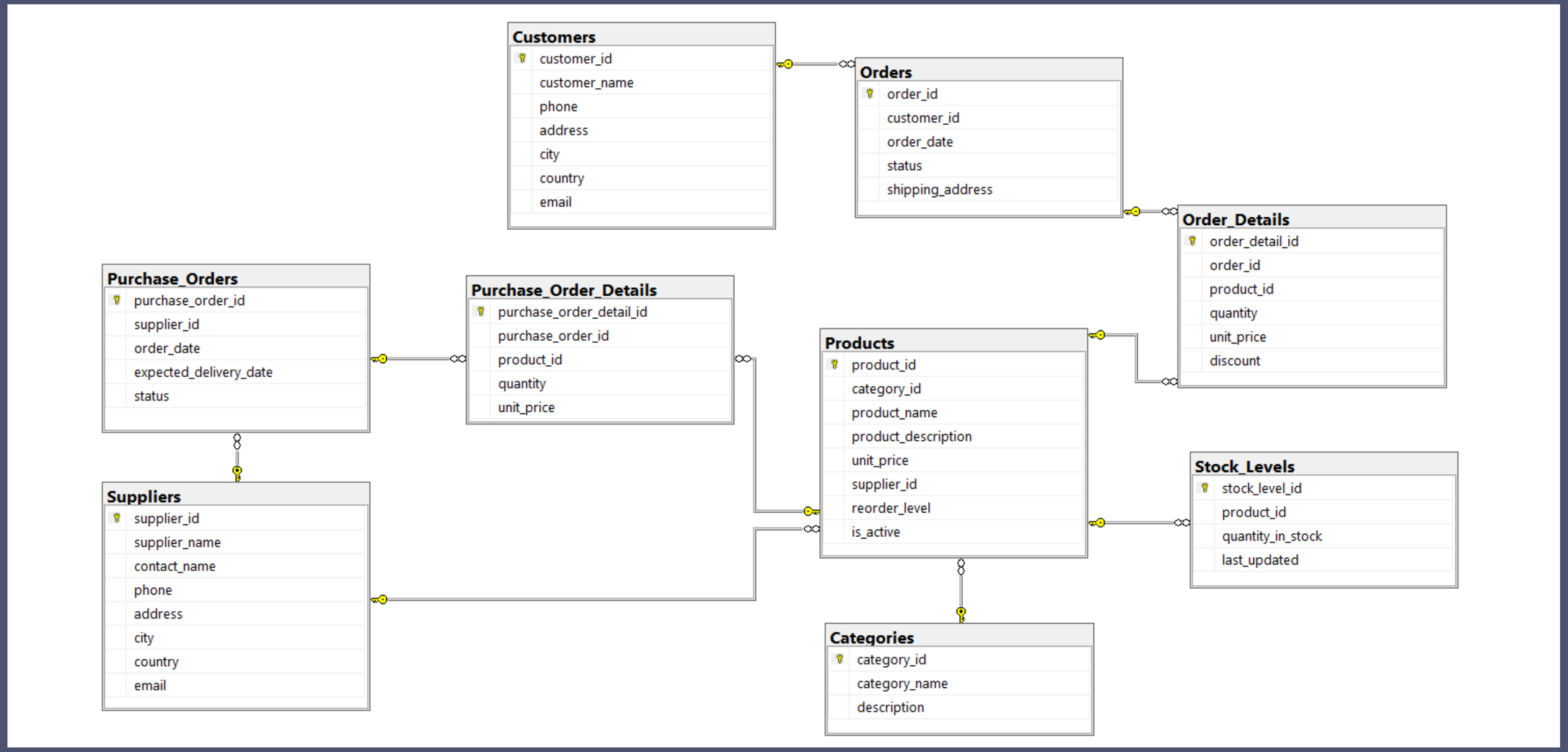
A Python script with the Faker library was used to generate realistic, randomized data such as names, addresses, and business details. This provided a scalable and reliable data source for testing and analysis.

```
populate_categories(500)
populate_suppliers(500)
populate_products(5000)
populate_customers(2500)
populate_orders(10000)
populate_order_details(25000)
populate_stock_levels(2500)
populate_purchase_orders(5000)
populate_purchase_order_details(25000)
```

```python
# Function to generate and insert data into Stock_Levels
def populate_stock_levels(n=1000):  1 usage
    product_ids = [row.product_id for row in cursor.execute("SELECT product_id FROM Products")]

    for _ in range(n):
        product_id = random.choice(product_ids)
        quantity_in_stock = random.randint( a: 0,  b: 500)
        last_updated = fake.date_time_between(start_date='-1y', end_date='now')
        cursor.execute( sql: """
            INSERT INTO Stock_Levels (product_id, quantity_in_stock, last_updated)
            VALUES (?, ?, ?)
        """,  *params: product_id, quantity_in_stock, last_updated)
    conn.commit()
    print(f"{n} stock levels inserted.")

# Function to generate and insert data into Purchase_Orders
def populate_purchase_orders(n=1000):  1 usage
    supplier_ids = [row.supplier_id for row in cursor.execute("SELECT supplier_id FROM Suppliers")]

    for _ in range(n):
        supplier_id = random.choice(supplier_ids)
        order_date = fake.date_between(start_date='-2y', end_date='today')
        status = random.choice(['Pending', 'Received', 'Cancelled'])
        cursor.execute( sql: """
            INSERT INTO Purchase_Orders (supplier_id, order_date, status)
            VALUES (?, ?, ?)
        """,  *params: supplier_id, order_date, status)
    conn.commit()
    print(f"{n} purchase orders inserted.")
```

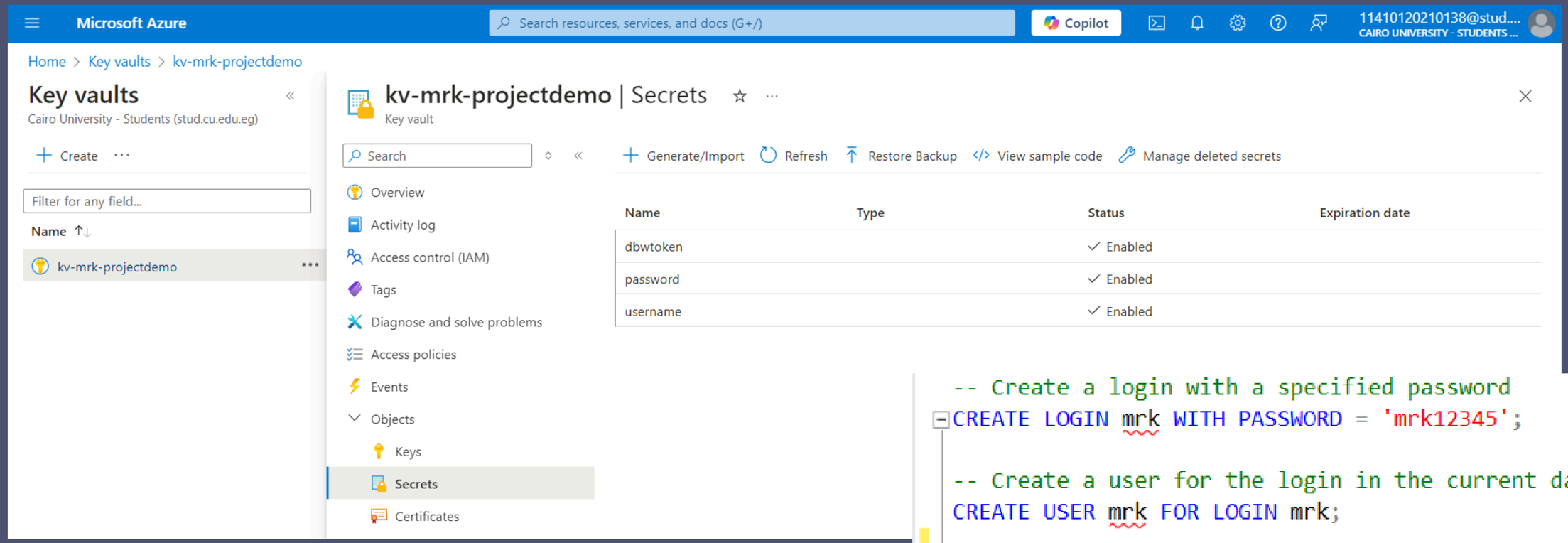
Database Diagram

The database follows a normalized format to minimize redundancy and ensure data integrity, optimizing storage and enhancing system efficiency.



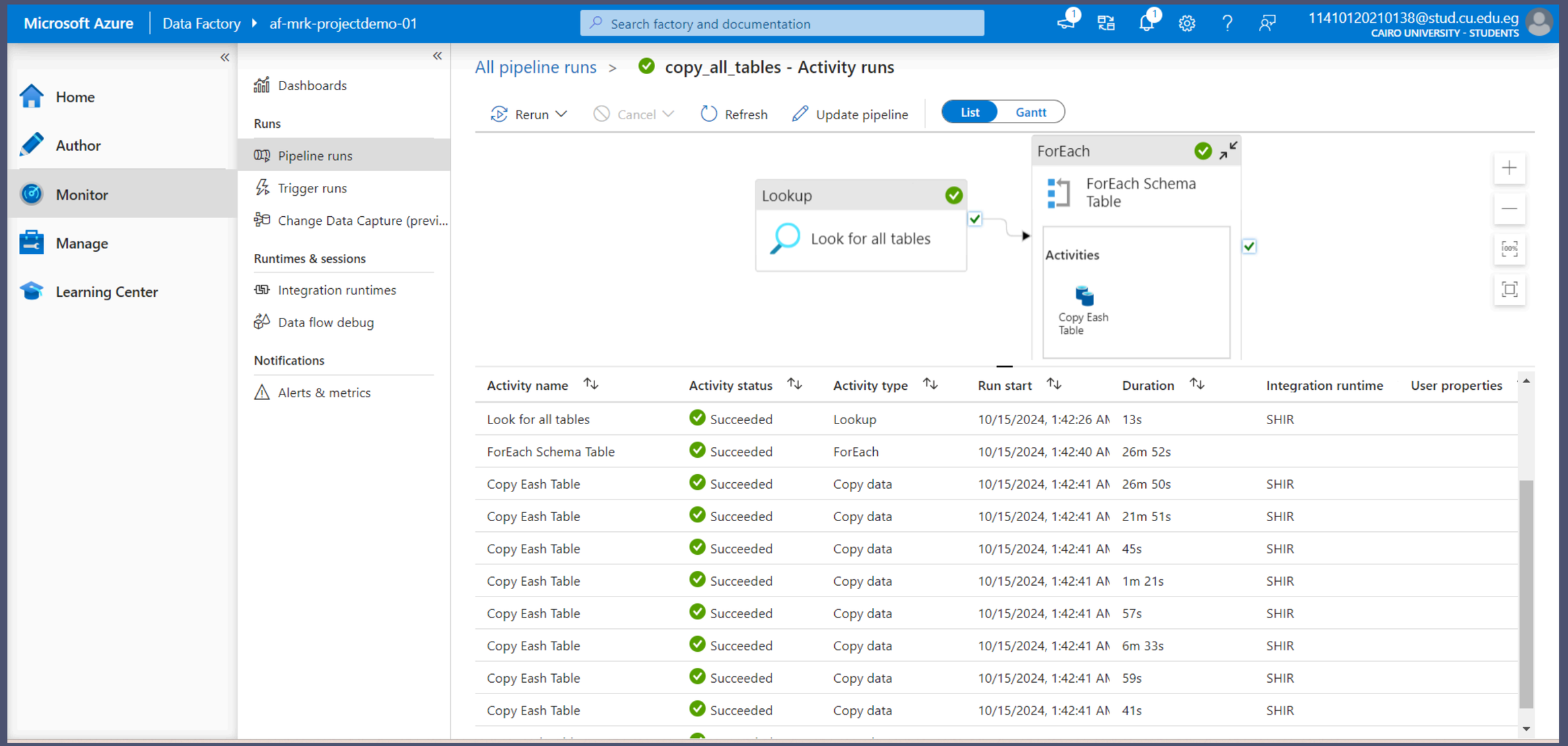
SECURE DATA EXTRACTION USING AZURE KEY VAULT

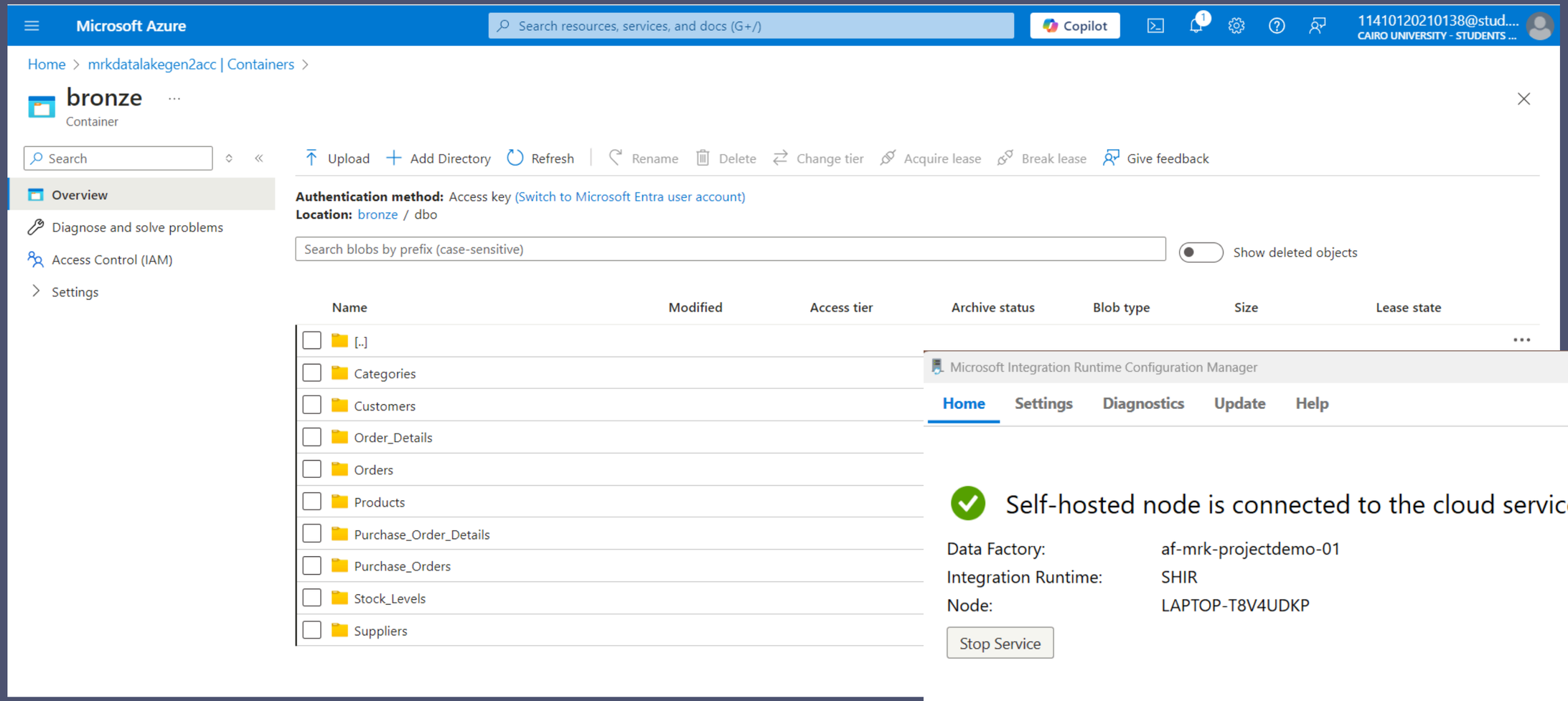
Azure Key Vault was utilized to securely store credentials, such as the username and password, ensuring safe authentication when accessing the SQL Server database. This approach ensures secure data extraction for processing.

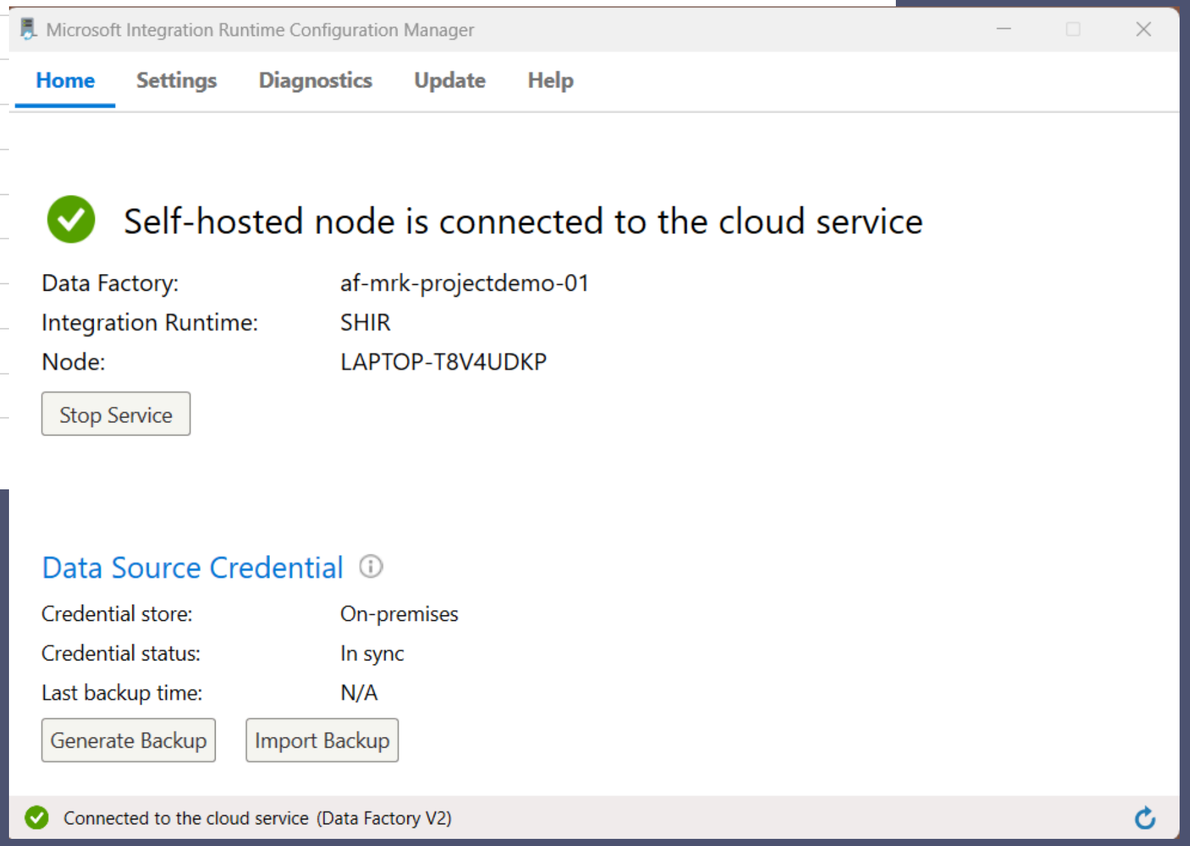



```
-- Create a login with a specified password
CREATE LOGIN mrk WITH PASSWORD = 'mrk12345';

-- Create a user for the login in the current database
CREATE USER mrk FOR LOGIN mrk;
```

DATA INGESTION USING AZURE DATA FACTORY




Microsoft Azure
Search resources, services, and docs (G+/)
Copilot
11410120210138@stud....
CAIRO UNIVERSITY - STUDENTS ...
Home > mrkdatalakegen2acc | Containers >
bronze
Container
Search
Overview
Diagnose and solve problems
Access Control (IAM)
Settings
Upload
Add Directory
Refresh
Rename
Delete
Change tier
Acquire lease
Break lease
Give feedback
Authentication method: Access key (Switch to Microsoft Entra user account)
Location: bronze / dbo
Search blobs by prefix (case-sensitive)
Show deleted objects
Name
Modified
Access tier
Archive status
Blob type
Size
Lease state
[..]
Categories
Customers
Order_Details
Orders
Products
Purchase_Order_Details
Purchase_Orders
Stock_Levels
Suppliers


Microsoft Integration Runtime Configuration Manager
Home
Settings
Diagnostics
Update
Help
Self-hosted node is connected to the cloud service
Data Factory: af-mrk-projectdemo-01
Integration Runtime: SHIR
Node: LAPTOP-T8V4UDKP
Stop Service
Data Source Credential
Credential store: On-premises
Credential status: In sync
Last backup time: N/A
Generate Backup
Import Backup
Connected to the cloud service (Data Factory V2)

Data Transformation using Databricks

Notebook 1: Storage Mount

- Purpose: Connects to external storage (e.g., Azure Data Lake) and mounts the storage to Databricks, allowing seamless access to raw data for further processing.

Notebook 2: Bronze to Silver

- Purpose: Transforms raw data from the Bronze layer into a more refined form in the Silver layer. This includes data cleaning, filtering, and standardizing data formats to ensure higher data quality.

Notebook 3: Silver to Gold

- Purpose: Converts the cleaned and structured data from the Silver layer into the Gold layer, which is optimized for analytics and reporting. This step includes data aggregation and further enrichment to support business insights.

Notebook 1: Storage Mount

Method 1: Using ADLS access key direct

Markdown

10/15/2024 (1s) 2

```
spark.conf.set("fs.azure.account.key.mrkdatalakegen2acc.dfs.core.windows.net", "JY2wFkg4u8dONTSBLUS0M1+M/DrvJe/5juZrr0Oh0iKMYhewj4uneV6PVO+H/Q6fqAJiGWZ7I1Zy+AStQmqKgQ==")
```

10/15/2024 (1s) 3

```
dbutils.fs.ls("abfss://bronze@mrkdatalakegen2acc.dfs.core.windows.net/")
```

```
Out[2]: [FileInfo(path='abfss://bronze@mrkdatalakegen2acc.dfs.core.windows.net/
```

Method 2: Creating Mount Point using ADLS Access Key

10/15/2024 (22s) 5

```
dbutils.fs.mount(
  source = "wasbs://bronze@mrkdatalakegen2acc.blob.core.windows.net/",
  mount_point = "/mnt/bronze",
  extra_configs = {"fs.azure.account.key.mrkdatalakegen2acc.blob.core.windows.net":"JY2wFkg4u8dONTSBLUS0M1+M/DrvJe/5juZrr0Oh0iKMYhewj4uneV6PVO+H/Q6fqAJiGWZ7I1Zy+AStQmqKgQ=="})
```

```
Out[4]: True
```

10/15/2024 (<1s) 6

```
dbutils.fs.ls("/mnt/bronze")
```

```
Out[5]: [FileInfo(path='dbfs:/mnt/bronze/dbo/', name='dbo/', size=0, modificationTime=0)]
```

10/15/2024 (<1s) 7

```
dbutils.fs.mounts()
```

```
Out[6]: [MountInfo(mountPoint='/databricks-datasets', source='databricks-datasets', encryptionType=''),
 MountInfo(mountPoint='/Volumes', source='UnityCatalogVolumes', encryptionType=''),
 MountInfo(mountPoint='/databricks/mlflow-tracking', source='databricks/mlflow-tracking', encryptionType=''),
 MountInfo(mountPoint='/databricks-results', source='databricks-results', encryptionType=''),
 MountInfo(mountPoint='/databricks/mlflow-registry', source='databricks/mlflow-registry', encryptionType=''),
 MountInfo(mountPoint='/mnt/bronze', source='wasbs://bronze@mrkdatalakegen2acc.blob.core.windows.net/', encryptionType=''),
 MountInfo(mountPoint='/Volume', source='DbfsReserved', encryptionType=''),
 MountInfo(mountPoint='/volumes', source='DbfsReserved', encryptionType=''),
 MountInfo(mountPoint='/', source='DatabricksRoot', encryptionType=''),
 MountInfo(mountPoint='/volume', source='DbfsReserved', encryptionType='')]
```

Notebook 2: Bronze to Silver

10/16/2024 (59s) 6

```python
from pyspark.sql.functions import col
from pyspark.sql.types import TimestampType

# Loop through each table in table_name
for i in table_name:
    # Define the path to the table
    path = '/mnt/bronze/dbo/' + i + '/' + i + '.parquet'
    # Read the table as a DataFrame
    df = spark.read.format('parquet').load(path)
    columns = df.columns

    # Loop through each column in the DataFrame
    for col_name in columns:
        # Check if the column is entirely null
        if df.filter(col(col_name).isNotNull()).count() == 0:
            # Drop the column if all values are null
            df = df.drop(col_name)

    # Define the output path for the processed table
    output_path = '/mnt/silver/ado/' + i + '/'
    # Write the DataFrame in delta format, overwriting any existing data
    df.write.format('delta').mode("overwrite").save(output_path)
```

(70) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [supplier_id: integer, supplier_name: string ... 6 more fields]

10/16/2024 (1s) 8

```python
from pyspark.sql.functions import from_utc_timestamp, date_format
from pyspark.sql.types import TimestampType

# Define the path to the Stock_Levels table in the datalake
input_path = '/mnt/silver/ado/Stock_Levels/'

# Read the Stock_Levels table as a DataFrame in Delta format
df = spark.read.format('delta').load(input_path)

# Convert the 'last_updated' column to a date format
df = df.withColumn(
    "last_updated",
    date_format(from_utc_timestamp(df["last_updated"].cast(TimestampType()), "UTC"), "yyyy-MM-dd")
)

# Display the processed DataFrame
display(df)
```

(1) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [stock_level_id: integer, product_id: integer ... 2 more fields]

Table

	stock_level_id	product_id	quantity_in_stock	last_updated
1	1	19154	146	2024-03-11
2	2	11257	169	2024-01-22

Notebook 3: Silver to Gold

10/16/2024 (29s) 2

```
from pyspark.sql import functions as F

# Define input and output paths
input_path = '/mnt/silver/ado/Order_Details/'
output_path = '/mnt/gold/ado/Order_Details/'

# Read the Order_Details table as a DataFrame in Delta format
order_details_df = spark.read.format('delta').load(input_path)

# Calculate total_amount for each order detail
fact_sales_df = order_details_df.withColumn(
    "total_amount",
    (F.col("quantity") * F.col("unit_price")) - F.col("discount")
)

# Display the transformed DataFrame (optional, for verification)
fact_sales_df.show()

# Write the transformed DataFrame back to Delta format, overwriting any existing data
fact_sales_df.write.format('delta').mode("overwrite").save(output_path)
```

(4) Spark Jobs

order_details_df: pyspark.sql.dataframe.DataFrame = [order_detail_id: integer, order_id: integer ... 4 more fields]

fact_sales_df: pyspark.sql.dataframe.DataFrame

order_detail_id: integer

order_id: integer

10/16/2024 (4s) 4

```
from pyspark.sql import functions as F

# Define input and output paths for the Purchase_Order_Details table
input_path = '/mnt/silver/ado/Purchase_Order_Details/'
output_path = '/mnt/gold/ado/Purchase_Order_Details/'

# Read the Purchase_Order_Details table as a DataFrame in Delta format
purchase_order_details_df = spark.read.format('delta').load(input_path)

# Calculate total_purchase_amount for each purchase order detail
fact_purchase_df = purchase_order_details_df.withColumn(
    "total_purchase_amount",
    F.col("quantity") * F.col("unit_price")
)

# Display the transformed DataFrame (optional, for verification)
fact_purchase_df.show()

# Write the transformed DataFrame back to Delta format, overwriting any existing data
fact_purchase_df.write.format('delta').mode("overwrite").save(output_path)
```

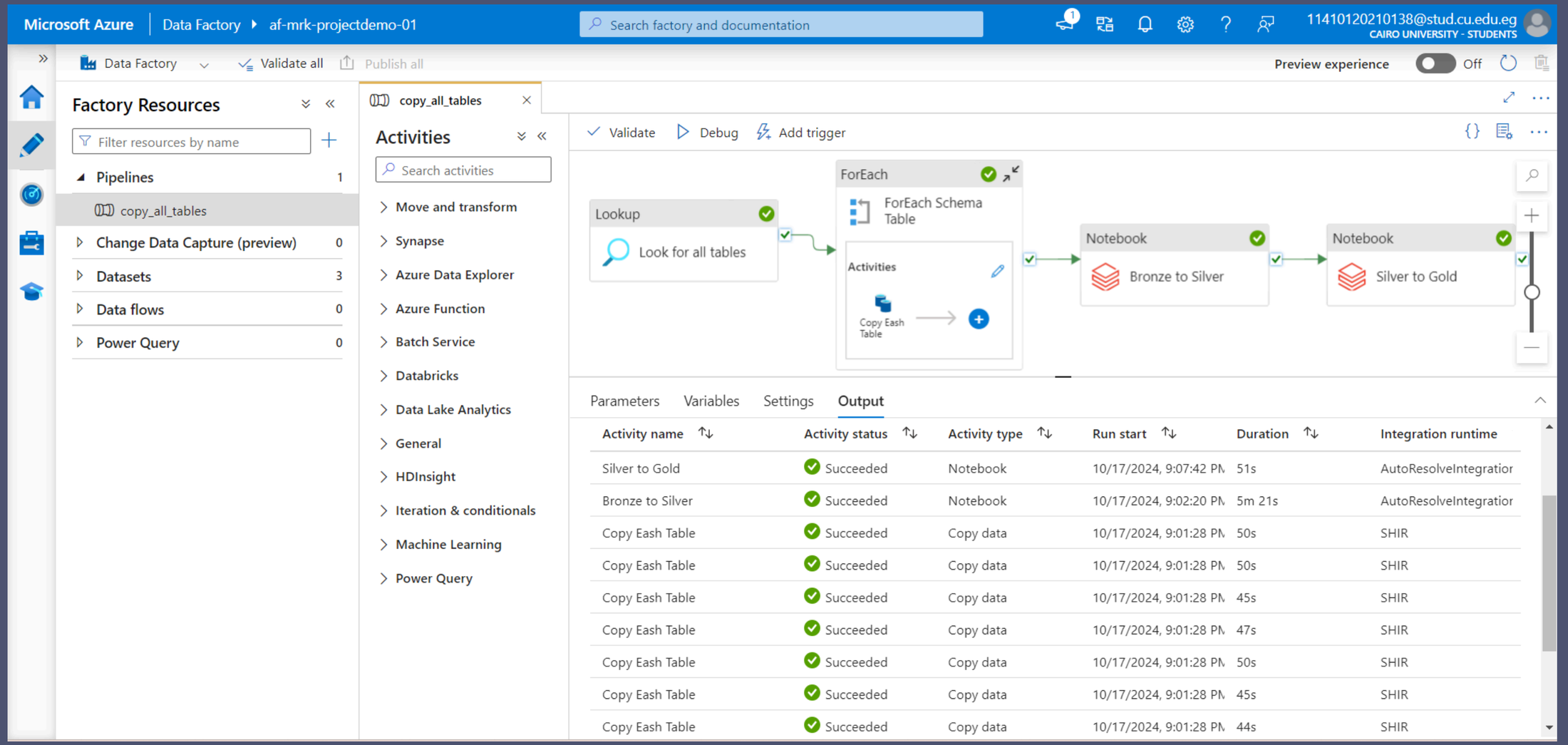
(4) Spark Jobs

purchase_order_details_df: pyspark.sql.dataframe.DataFrame = [purchase_order_detail_id: integer, purchase_order_id:

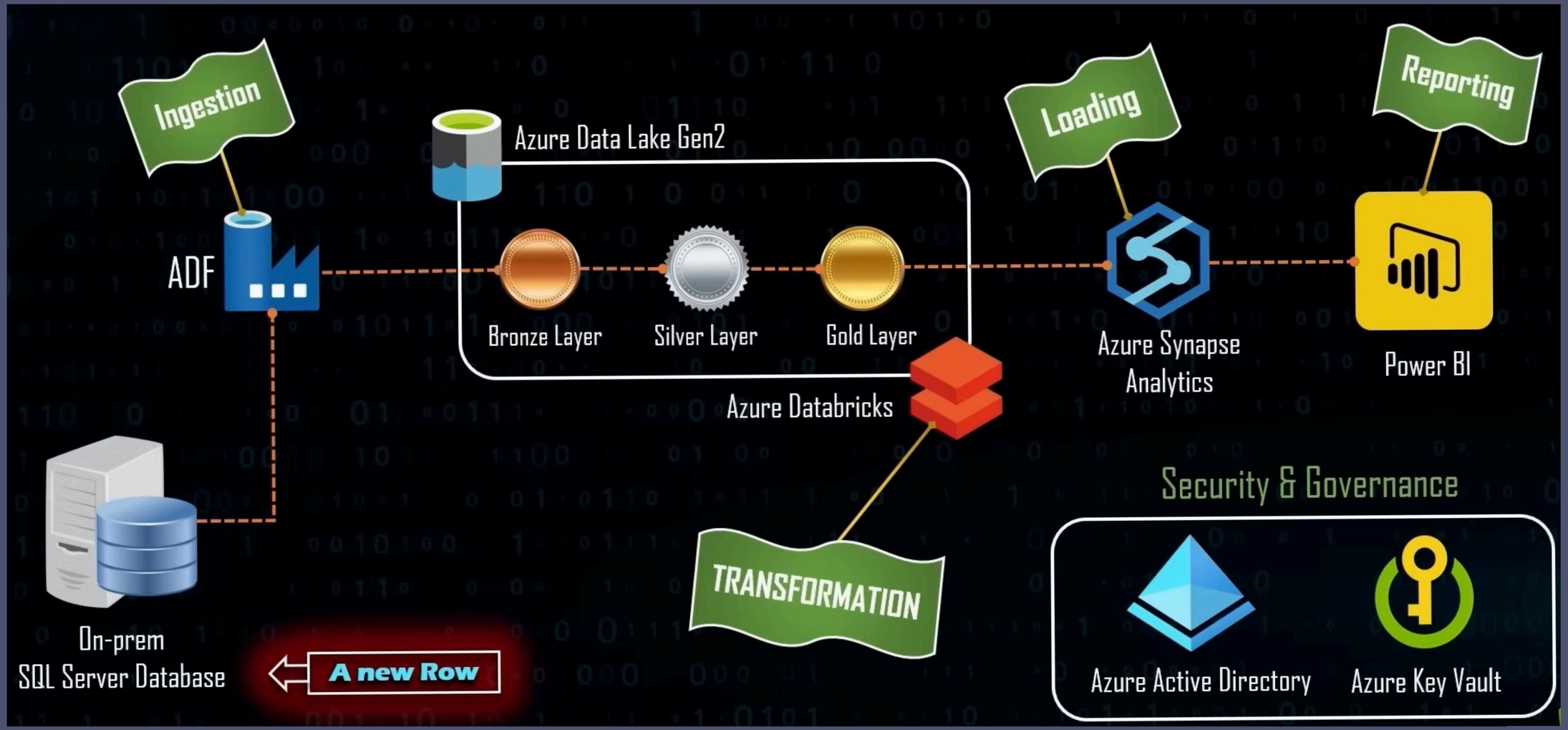
fact_purchase_df: pyspark.sql.dataframe.DataFrame = [purchase_order_detail_id: integer, purchase_order_id: integer ...

3	13912	14324	60	491.46	29487.60
4	18289	13949	26	44.05	1145.30
5	10053	16360	2	193.48	386.96

Data Factory Pipelines Enhanced by Databricks Notebooks

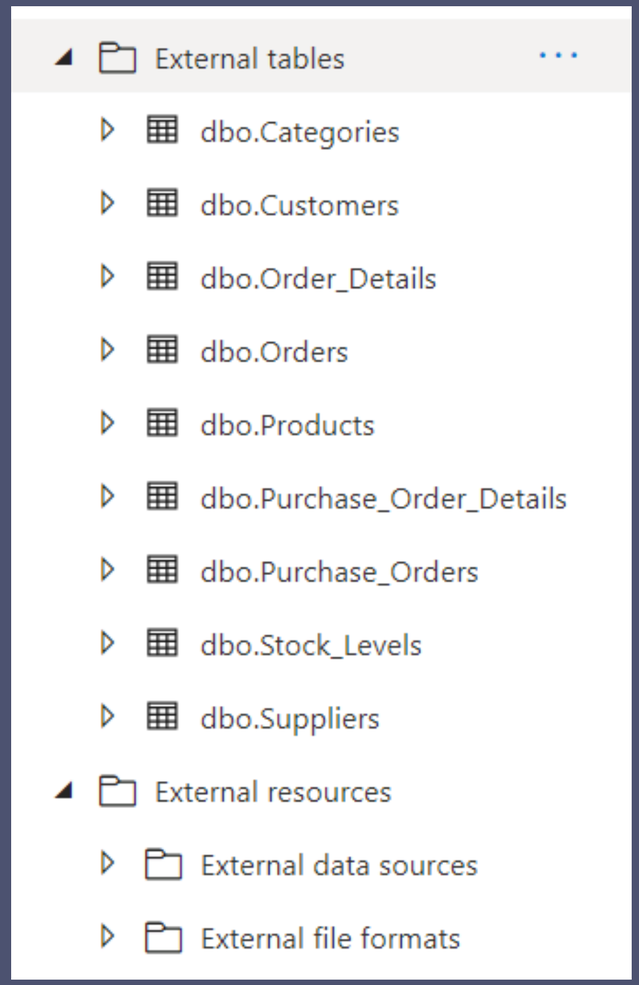


Project Overview articature



Creating a Data Warehouse Using External Tables in Azure Synapse Analytic

```
CREATE EXTERNAL DATA SOURCE my_adls_datasource
WITH (
    TYPE = HADOOP,
    LOCATION = 'abfss://gold@sqladminx.dfs.core.windows.net/ado',
    CREDENTIAL = my_adls_credential
);

CREATE EXTERNAL FILE FORMAT parquet_file_format
WITH (
    FORMAT_TYPE = PARQUET
);
```



Creating External Tables

```sql
CREATE EXTERNAL TABLE dbo.Categories
(
    CategoryID INT,
    CategoryName NVARCHAR(255),
    Description NVARCHAR(MAX)
)
WITH (
    LOCATION = 'gold/ado/Categories/',
    DATA_SOURCE = MyExternalDataSource,
    FILE_FORMAT = ParquetFileFormat
);

CREATE EXTERNAL TABLE dbo.Customers
(
    CustomerID INT,
    CustomerName NVARCHAR(255),
    Phone NVARCHAR(50),
    Address NVARCHAR(255),
    City NVARCHAR(100),
    Country NVARCHAR(100),
    Email NVARCHAR(255)
)
WITH (
    LOCATION = 'Customers/',  -- Folder where your data is stored in ADLS
    DATA_SOURCE = my_adls_datasource,
    FILE_FORMAT = parquet_file_format
);
```

```sql
CREATE EXTERNAL TABLE dbo.Order_Details
(
    OrderDetailsID INT,
    OrderID INT,
    ProductID INT,
    Quantity INT,
    UnitPrice DECIMAL(18, 2),
    Discount DECIMAL(5, 2),
    TotalAmount DECIMAL(18, 2)
)
WITH (
    LOCATION = 'Order_Details/',  -- Folder where your data is stored in ADLS
    DATA_SOURCE = my_adls_datasource,
    FILE_FORMAT = parquet_file_format
);
CREATE EXTERNAL TABLE dbo.Orders
(
    OrderID INT,
    CustomerID INT,
    OrderDate DATE,
    Status NVARCHAR(50),
    ShippingAddress NVARCHAR(255)
)
WITH (
    LOCATION = 'Orders/',  -- Folder where your data is stored in ADLS
    DATA_SOURCE = my_adls_datasource,
    FILE_FORMAT = parquet_file_format
);
```

Creating Dimensions and Fact Table

```sql
CREATE TABLE FactOrders
(
    OrderID INT NOT NULL,
    CustomerID INT NOT NULL,
    ProductID INT NOT NULL,
    Quantity INT NOT NULL,
    UnitPrice DECIMAL(18, 2) NOT NULL,
    TotalAmount DECIMAL(18, 2) NOT NULL,
    OrderDate DATE NOT NULL,
    CONSTRAINT PK_FactOrders PRIMARY KEY NONCLUSTERED (Or
);
```

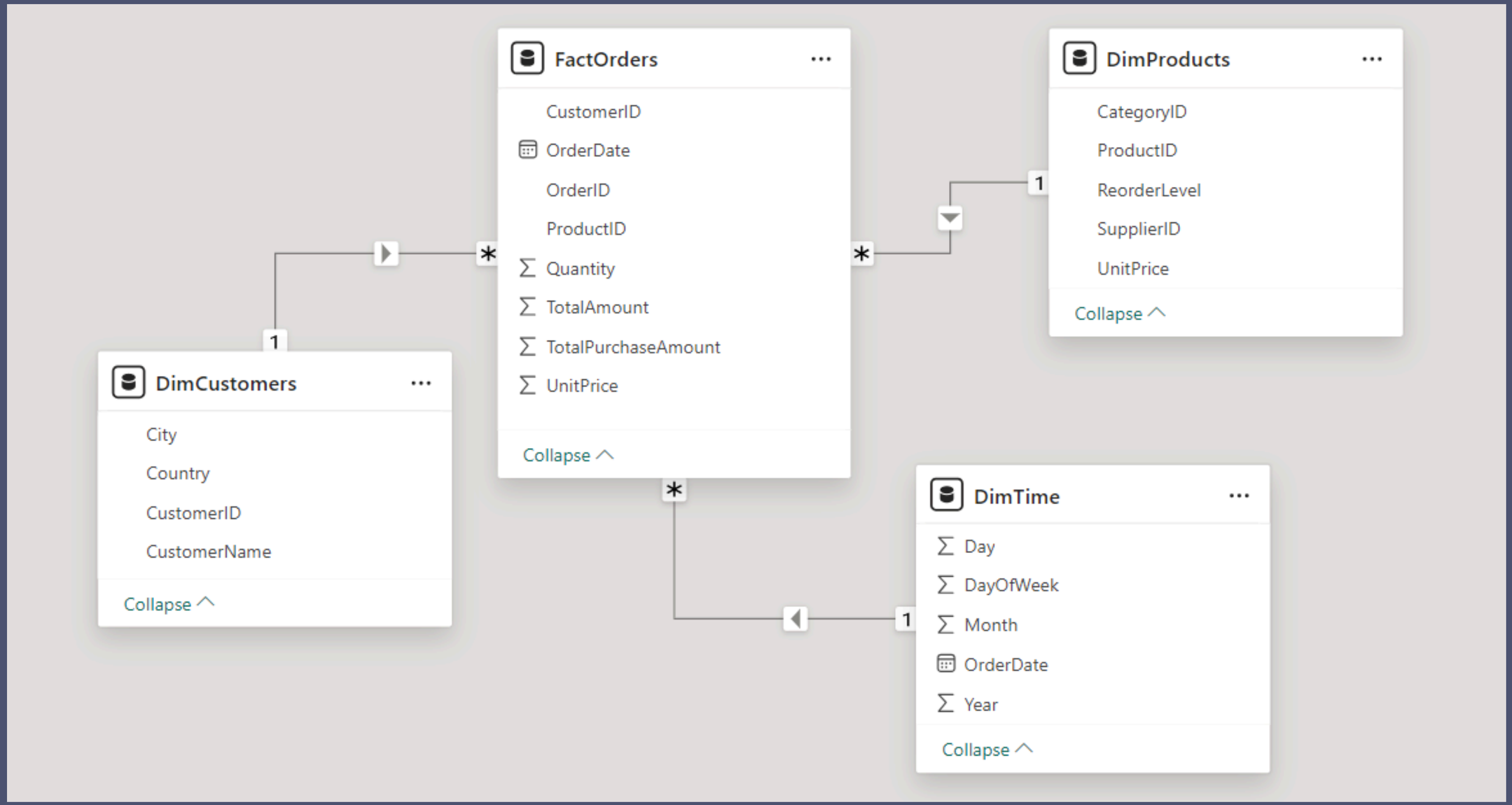
```sql
CREATE TABLE DimSuppliers
(
    SupplierID INT NOT NULL,
    ContactName NVARCHAR(255),
    Phone NVARCHAR(50),
    City NVARCHAR(100),
    Country NVARCHAR(100),
    CONSTRAINT PK_DimSuppliers PRIMARY KEY NONCLUSTERED (SupplierID) NOT ENFORCED
);

-- Dimension Table for Time
CREATE TABLE DimTime
(
    OrderDate DATE NOT NULL,
    Year INT NOT NULL,
    Month INT NOT NULL,
    Day INT NOT NULL,
    DayOfWeek INT NOT NULL,
    CONSTRAINT PK_DimTime PRIMARY KEY NONCLUSTERED (OrderDate) NOT ENFORCED  -- Une
);
```

Inserting into Dimension and Fact Tables

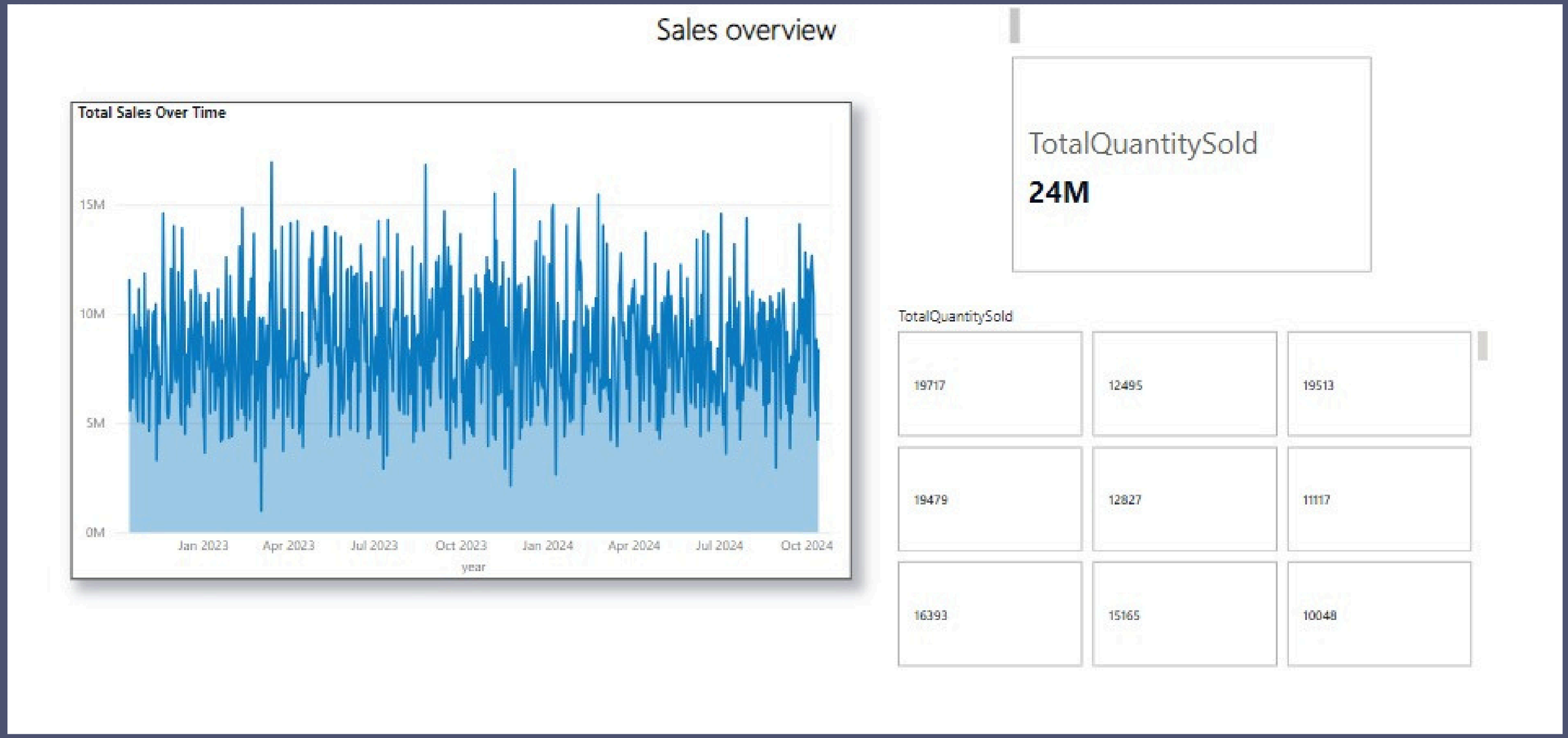
```sql
-- Insert into DimCustomers
INSERT INTO DimCustomers (CustomerID, CustomerName, Phone, Address, City, Country, Email)
SELECT DISTINCT CustomerID, CustomerName, Phone, Address, City, Country, Email
FROM dbo.Customers;

-- Insert into DimProducts
INSERT INTO DimProducts (ProductID, CategoryID, ProductName, ProductDescription, UnitPrice, SupplierID, ReorderLevel, IsActive)
SELECT DISTINCT ProductID, CategoryID, ProductName, ProductDescription, UnitPrice, SupplierID, ReorderLevel, IsActive
FROM dbo.Products;

-- Insert into DimSuppliers
INSERT INTO DimSuppliers (SupplierID, SupplierName, ContactNa
SELECT DISTINCT SupplierID, SupplierName, ContactName, Phone,
FROM dbo.Suppliers;
```

```sql
INSERT INTO DimTime (OrderDate, Year, Month, Day, DayOfWeek)
SELECT DISTINCT
    OrderDate,
    YEAR(OrderDate) AS Year,
    MONTH(OrderDate) AS Month,
    DAY(OrderDate) AS Day,
    DATEPART(WEEKDAY, OrderDate) AS DayOfWeek
FROM dbo.Orders;

INSERT INTO FactOrders (OrderID, CustomerID, ProductID, Quantity, UnitPrice, TotalAmount, OrderDate)
SELECT
    o.OrderID,
    o.CustomerID,
    od.ProductID,
    od.Quantity,
    od.UnitPrice,
    od.Quantity * od.UnitPrice * (1 - od.Discount / 100) AS TotalAmount,
    o.OrderDate
FROM
    dbo.Orders o
JOIN
    dbo.Order_Details od ON o.OrderID = od.OrderID;
```

Star Schema


FactOrders
CustomerID
OrderDate
OrderID
ProductID
Quantity
TotalAmount
TotalPurchaseAmount
UnitPrice
Collapse
DimProducts
CategoryID
ProductID
ReorderLevel
SupplierID
UnitPrice
Collapse
DimCustomers
City
Country
CustomerID
CustomerName
Collapse
DimTime
Day
DayOfWeek
Month
OrderDate
Year
Collapse
1
*
1
*
1
*
1
*

Data Reporting using Microsoft Power BI

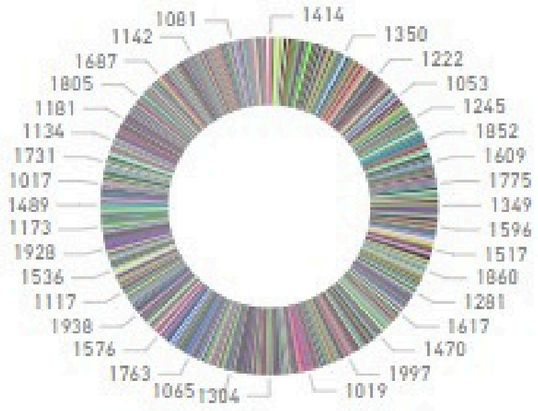


order and product overview

totalorders

480K

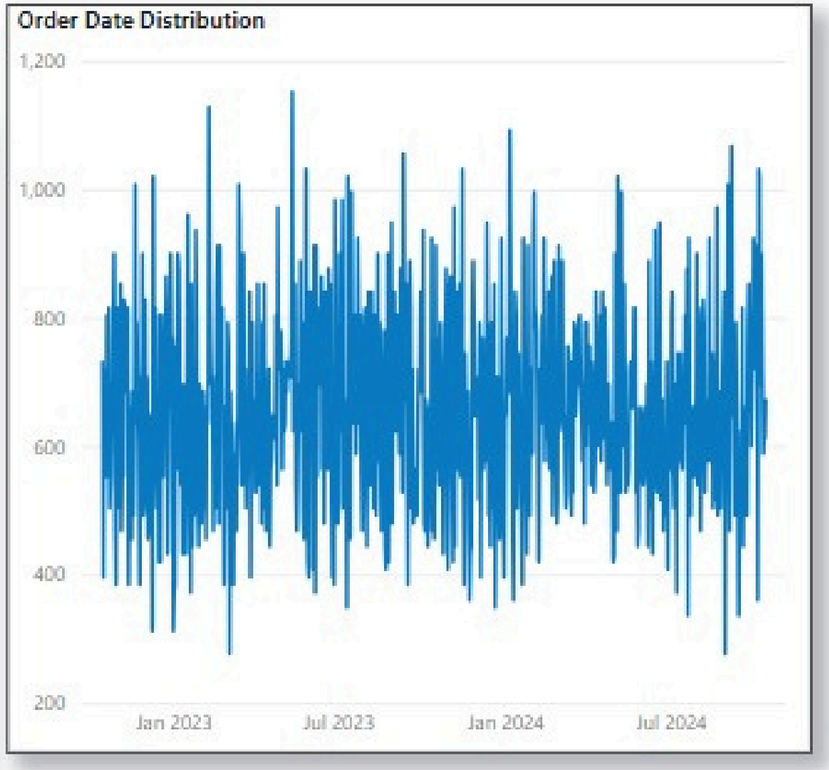
Total Quantity by category



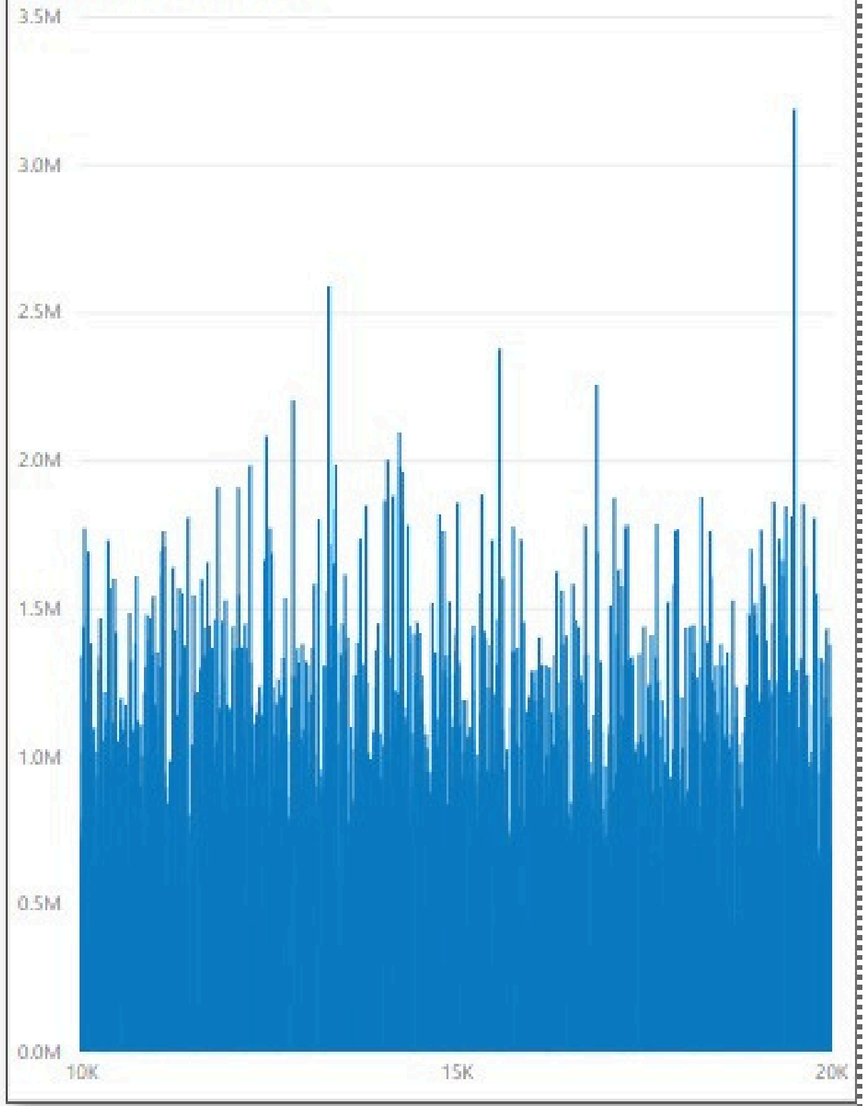
Total...

24M

Order Date Distribution



Product Contribution to Sales

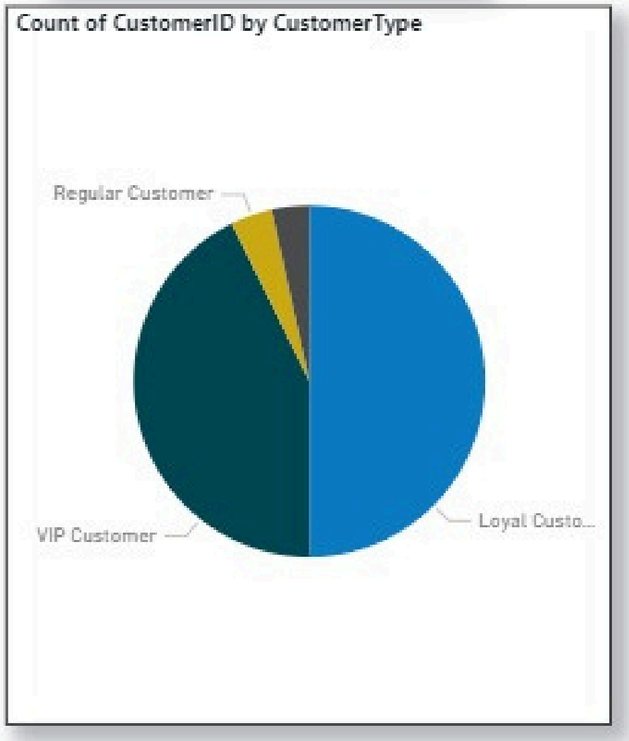


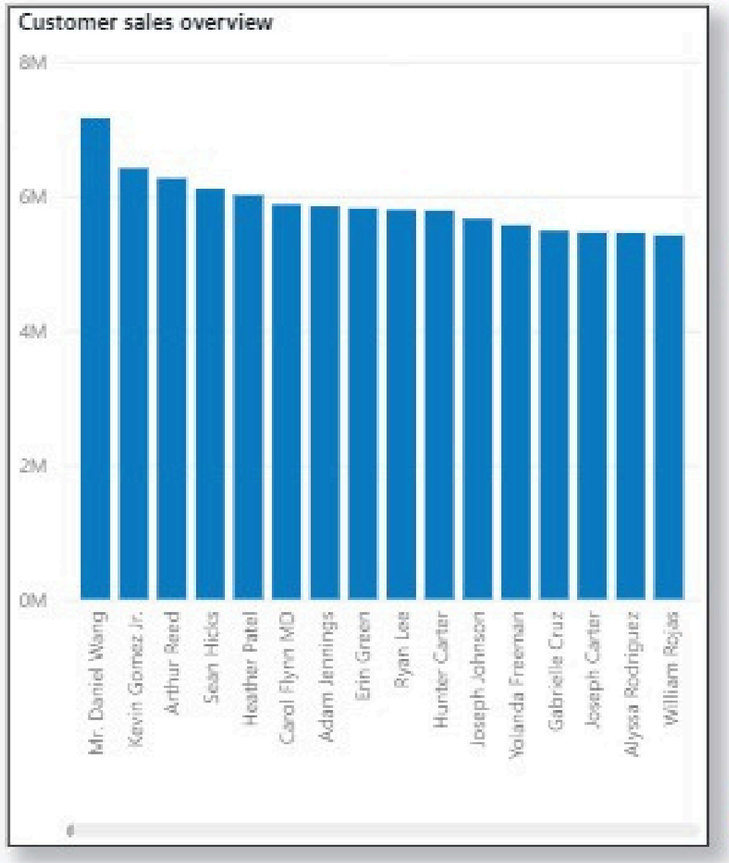
Customer Behavior and Demographics

Number of Customers

TotalCustomers

5K





Retail inventory Management Project

Supplier behavior

Totalsuppliers

1K

Supplier orders

SupplierID	OrderID	Sum of TotalAmount
1001	20454	19,568.20
1001	20700	256,181.40
1001	21483	161,821.76
1001	21978	379,720.96
1001	22170	153,531.24
1001	23365	323,442.40
1001	23835	30,873.32
1001	23860	338,732.36
1001	25502	153,722.64
1001	26415	119,089.24
1001	27834	495,200.64
1001	27903	17,088.16
1001	28379	140,050.16
1001	28429	178,238.80
1001	29288	206,076.88
1001	29669	81,315.60
1001	30118	39,858.20
1001	30724	120,920.96
1001	31365	28,892.32
1001	32061	231,833.72
1001	32364	14,484.72
1001	33123	110,694.28
Total		**5,932,174,416.52**

The distribution of suppliers in North America appears to be relatively dense, This concentration suggests a strong supplier presence in this region, potentially indicating established business relationships and opportunities for collaboration.
in contrast the supplier distribution in Africa and South America seems sparse.

Conclusion and Future Steps

- Project Conclusion:
- Successfully developed a system for efficient retail inventory management and forecasting.
- Future Enhancements:
- Integrate advanced machine learning models for improved forecast accuracy.
- Develop dashboards for real-time monitoring.
- Implement anomaly detection for stock level irregularities.