

## Project Documentation: Azure SQL Data Pipeline

---

### 1. What we needed

- An active Azure subscription with necessary permissions (Owner).
  - Azure Data Factory (ADF) resource created in my subscription.
  - Storage Account (Blob Service)
- 

### 2. Create Azure SQL Server and Database

1. Creating SQL Server team12db
  2. Create Database Team12\_DB\_Bank
- 

### 3. Create Database Tables

1. Creating Tables (**Customers, Accounts, Cards, Loans, Transactions, Support\_Calls**)
2. Adding Constraints (**Foreign Keys**)

### 4. Ingest Data Using Azure Data Factory

#### 4.1 Create Linked Services

1. In the Azure Data Studio
2. Under **Manage > Linked services**, we added:
  - **Azure SQL Database**
  - **Blob Storage** (or other source) where raw data files reside.
3. Provide authentication (Service Principal or Managed Identity).

#### 4.2 Build a Pipeline

1. Under **Author > Pipelines**, click **New pipeline**.
2. Add a **Copy data** activity.
3. Configure **Source**: selecting Blob Storage dataset (e.g., CSV files).
4. Configure **Sink**: selecting Azure SQL DB dataset targeting Team12\_DB and map columns.
5. Publish and **Debug** the pipeline to load raw data.

---

## 5. Data Cleaning & Exploratory Data Analysis in Python

### 5.1 Overview

This document describes the key steps and best practices for cleaning raw data and performing EDA using Python's scientific stack, mainly **pandas**, **NumPy**, **matplotlib** and **seaborn**.

### 5.2 Environment Setup

Importing libraries (pyodbc, Pandas, NumPy, Matplotlib, seaborn)

**5.3 Loading Data** from SQL Server and automates datetime conversion.

### 5.4 Data Cleaning

- 1- Inspecting Structure & Missing Values.
- 2- Removing Duplicates.
- 3- Data-Type Conversion & Parsing.
- 4- Outlier Detection & Handling (IQR Method)
- 5- Handling Missing Data (Categorical → forward/backward fill or mode).

### 5.5 Exploratory Data Analysis (EDA)

- 1- Descriptive Statistics.
- 2- Univariate Analysis:
  - Histograms for distributions
  - Boxplots for outliers
- 3- Bivariate Analysis:
  - Scatter plot
- 4- Time Series / Trend Analysis.
- 5- Customer Segmentation or Group-by.

### 5.6 Reporting & Next Steps

- Document key findings: unusual patterns, correlations, clusters.
- Save cleaned data.

## 5.7 Best Practices

- Keep a **data-cleaning log** (e.g., Jupyter notebook markdown cells) to record each transformation.
  - Always **version control** cleaned datasets separately.
  - Validate assumptions (e.g., distribution, normality) before choosing imputation or outlier methods.
  - Build **reusable functions** for repeated cleaning steps.
- 

## 6. Create Cleaned Database Team12\_DB\_Cleaned

- Using The Same Script From Team12\_DB
- 

## 7. Load Cleaned Data into Team12\_DB\_Cleaned Using Data Factory

- Using The Same Steps From Before
  - We Load The New .CSV Files Onto The Data Factory
- 

## 8. Creating Team Logins On Database

- Created Logins For SQL Members On Database “master”
  - Created Users From Logins On “master” & “Team12\_DB\_Bank\_Cleaned”
  - Assigned Roles (db\_owner, db\_ddladmin, db\_datareader, db\_datawriter)
- 

## 9. Giving Firewall Access To Team Member Devices

- Under Server **team12db** > **Networking**, click **Add Firewall Rule**
- Added Device IP
- Save

## KPIs Using SQL for Banking Analytics Project

### Introduction:

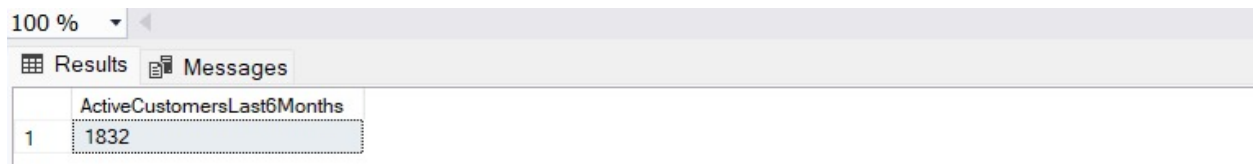
This section provides an overview of SQL queries used for data analysis in the banking context. It covers various aspects, including customer activity, account management, transaction analysis, and support call tracking.

## 1. Active Customers:

- **Purpose:** Count the number of active customers in the last six months.

```
--How many customers are currently active? Transation
Create View Customer_Active_View AS
SELECT COUNT(DISTINCT a.CustomerID) AS ActiveCustomersLast6Months
FROM Transactions t
JOIN Accounts a ON t.AccountID = a.AccountID
WHERE t.TransactionDate >= DATEADD(Month, -6, GETDATE());
```

- **Output:**



100 %

Results Messages

	ActiveCustomersLast6Months
1	1832

- This means that the number of **active customers** is **1832**, which means that our active customer ratio is **36.64%**.
- Recommendations:
  - Ensure your mobile and online banking platforms are intuitive and easy to navigate.
  - Develop products that meet specific customer segments, such as students, retirees, or small business owners.

## 2. New Customers:

- **Purpose:** Track the number of new customers joining each month for the last six months.

```
--How many new customers are joining the bank each month?
Create View Customer_New_View AS
SELECT
    FORMAT(JoinDate, 'yyyy-MM') AS month,
    COUNT(*) AS new_customers
FROM Customers
Where JoinDate >= DATEADD(MONTH, -6, GETDATE())
GROUP BY FORMAT(JoinDate, 'yyyy-MM')
ORDER BY month;
```

- **Output:**

100 %

Results Messages		
	month	new_customers
1	2024-11	10
2	2024-12	54
3	2025-01	42
4	2025-02	39
5	2025-03	32
6	2025-04	32
7	2025-05	12

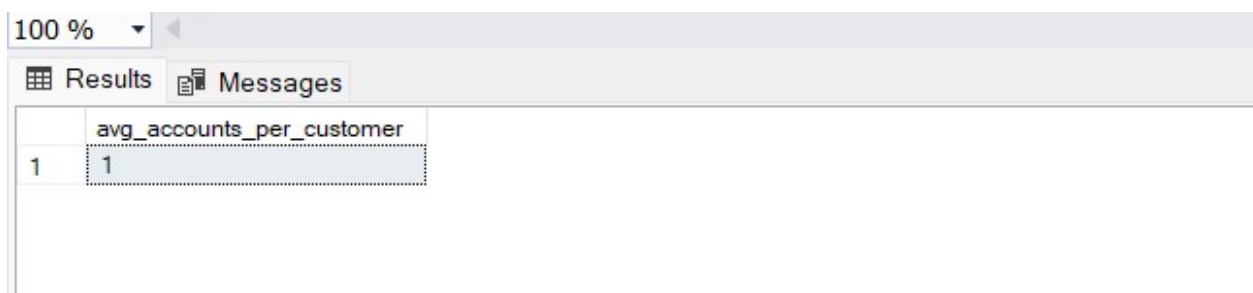
- This means that the **largest number** of new customers joined in **December** 2024, and the **smallest number** of new customers was in **November** 2024.
- Recommendations: We run more promotions in November each year to attract more new customers.

### 3. Average Accounts per Customer:

- **Purpose:** Calculate the average number of accounts held by customers.

```
--What is the average number of accounts per customer?  
Create View AVG_accountsPerCustomer_View AS  
SELECT AVG(account_count) AS avg_accounts_per_customer  
FROM (  
    SELECT distinct CustomerID, COUNT(*) AS account_count  
    FROM Accounts  
    GROUP BY CustomerID  
) sub;
```

- **Output:**



The screenshot shows a SQL query results window. At the top, there is a zoom level dropdown set to '100 %' and a back arrow. Below this are two tabs: 'Results' (active) and 'Messages'. The 'Results' tab displays a table with one column named 'avg\_accounts\_per\_customer' and one row containing the value '1'.

	avg_accounts_per_customer
1	1

- This means that each customer has an average of one account.
- Recommendations: We're adding more benefits for those with more than one account.

#### 4. Customers at Risk of Churn:

- **Purpose:** Identify customers with no activity for over six months.

```
--Which customers are at risk of churn (no activity for 6+ months)?
Create View Customers_Churn_View AS
SELECT COUNT(DISTINCT c.CustomerID) AS ChurnRiskCustomers
FROM Customers c
WHERE c.CustomerID NOT IN (
    SELECT DISTINCT a.CustomerID
    FROM Accounts a
    JOIN Transactions t ON a.AccountID = t.AccountID
    WHERE t.TransactionDate >= DATEADD(Month, -6, GETDATE())
);
```

- **Output:**

100 % ▾

Results		Messages
	ChurnRiskCustomers	
1	3168	

- This means that the number of customers who have not made any transactions for more than 6 months is: **3168**.
- Recommendations: We offer additional discounts to those who have made any transactions within a period of less than 3 months.

## 5. Total Balance by Account Type:

- **Purpose:** Summarize total balances across different account types.

```
--What is the total balance per account type (Savings, Business, etc.)?  
Create View Total_BalancePerAcoounts_View AS  
SELECT AccountType, SUM(Balance) AS total_balance  
FROM Accounts  
GROUP BY AccountType;
```

- **Output:**

100 %

Results		Messages
	AccountType	total_balance
1	Savings	79055168
2	Business	87161577
3	Checking	83154050

- Here, we find that the total bank balances for each account type are similar, as shown in the image above. However, we find that the highest bank balances are for business bank accounts.
- Recommendations: Since the total numbers are similar, we should focus our efforts on facilitating the process of opening a bank account for customers.

## 6. Transaction Volume Over Time:

- **Purpose:** Analyze transaction volume per month and year.



```
--What is the total transaction volume per month and year?--
Create View Total_TransactionVolumePerMonthsAndYear_View AS
SELECT
    FORMAT(TransactionDate, 'yyyy-MM') AS [Year-Month],
    COUNT(*) AS transaction_Num,
    SUM(Amount) AS total_volume
FROM Transactions
GROUP BY FORMAT(TransactionDate, 'yyyy-MM')
ORDER BY total_volume DESC;
```

- Output:

100 %			
Results Messages			
	Year-Month	transaction_Num	total_volume
1	2022-10	600	3165783.73
2	2024-12	612	3119058.89
3	2022-11	590	3053251.29
4	2023-08	597	2991051.97
5	2024-10	611	2971281.27
6	2023-05	592	2957270.7
7	2023-10	590	2871179.99
8	2023-04	559	2837509.75
9	2024-04	553	2831349.27
10	2025-01	549	2826429.23
11	2022-09	550	2824568.13
12	2023-01	571	2806441.61
13	2024-07	567	2805926.45
14	2024-03	568	2795765.63
15	2022-12	572	2794609.8

- As shown in the image above, there is **no relationship** between the total volume and the number of transactions carried out during the month.

## 7. Common Transaction Types:

- **Purpose:** Identify the most common transaction types (e.g., Deposit, Withdrawal).

```
--Which transaction types are most common (Deposit, Withdrawal)?
Create View Transaction_MostCommon_View AS
SELECT TransactionType, COUNT(*) AS Count_Transactions
FROM Transactions
GROUP BY TransactionType
ORDER BY Count_Transactions DESC;
```

- **Output:**

100 %		
Results Messages		
	TransactionType	Count_Transactions
1	Payment	5056
2	Transfer	5055
3	Deposit	4970
4	Withdrawal	4919

- We notice that the number of transactions is close to the types of operations, but the largest type of operations is **Payment**, with **5056** transactions.

## 8. Average Transaction Value by Account Type:

- **Purpose:** Calculate the average transaction value for each type of account.

```
--What is the average transaction value for each account type?
Create View AVG_TransactionValuePerAcoounts_View AS
SELECT a.AccountType, AVG(t.Amount) AS Avg_transaction_value
FROM Transactions t
JOIN Accounts a ON t.AccountID = a.AccountID
GROUP BY a.AccountType
Order By Avg_transaction_value DESC;
```

- **Output:**

100 %		
Results Messages		
	AccountType	Avg_transaction_value
1	Savings	5018.91191665388
2	Checking	5016.78286824068
3	Business	4981.94449189345

- This means that the highest average amount paid per transaction is in **Save accounts**.

## 9. Total Loan Amount by Loan Type:

- **Purpose:** Determine the total loan amount disbursed for each loan type (e.g., Home, Car).

```
--What is the total loan amount disbursed by loan type (Home, Car, Personal..)?  
Create View Total_LoanAmountPerLoanType_View AS  
SELECT LoanType, SUM(LoanAmount) AS total_loan_amount  
FROM Loans  
GROUP BY LoanType  
Order by total_loan_amount DESC;
```

- **output:**

.00 %

Results		Messages
	LoanType	total_loan_amount
1	Personal	154925690
2	Education	154218864
3	Car	153848761
4	Home	153665578

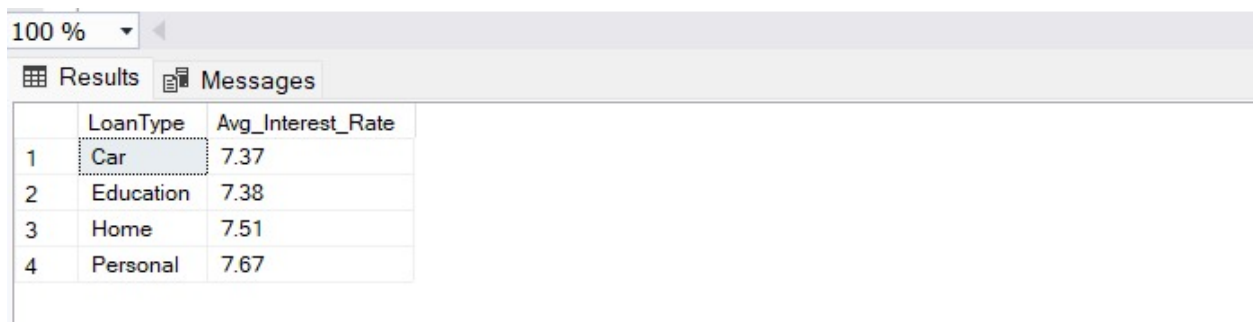
- As shown in the image above, there is a great similarity between the total loan amounts for the loan type, but the personal loan is the largest with a total amount of 154,925,690.

## 10. Average Interest Rate by Loan Type:

- **Purpose:** Calculate the average interest rate for each loan type.

```
--What is the average interest rate for each loan type?  
Create View AVG_InterestRateLoan_View AS  
SELECT LoanType, Round(AVG(InterestRate),2) AS Avg_Interest_Rate  
FROM Loans  
GROUP BY LoanType  
Order By Avg_Interest_Rate;
```

- **Output:**



	LoanType	Avg_Interest_Rate
1	Car	7.37
2	Education	7.38
3	Home	7.51
4	Personal	7.67

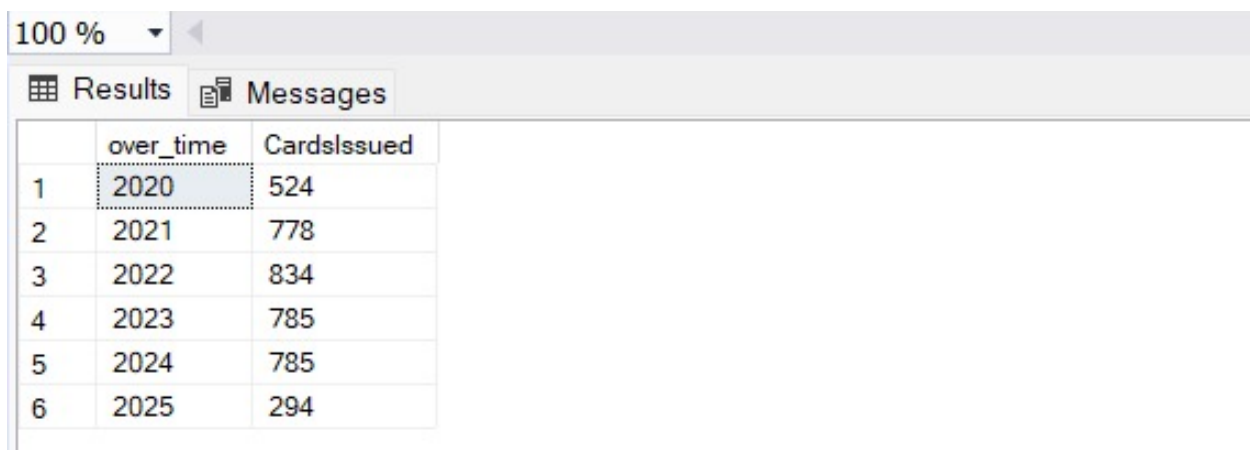
- As shown in the image above, we find that the highest interest rate for loans is the personal loan at **7.67%**.

## 11. Card Issuance Trend:

- **Purpose:** Analyze the trend of card issuance over the years.

```
--How has card issuance trended over time?  
Create View CardIssuance_View AS  
SELECT  
    FORMAT(IssuedDate, 'yyyy') AS over_time,  
    COUNT(*) AS CardsIssued  
FROM Cards  
GROUP BY FORMAT(IssuedDate, 'yyyy');
```

- **Output:**



100 %

Results Messages

	over_time	CardsIssued
1	2020	524
2	2021	778
3	2022	834
4	2023	785
5	2024	785
6	2025	294

- This means we aim to issue the largest number of cards in 2022, with the total number of cards issued reaching 834.
- Recommendations: We need to review our activities in 2022 to increase our card issuance rate.

## 12. Distribution of Active and Expired Cards:

- **Purpose:** Evaluate the distribution of active versus expired cards.

```
--What is the distribution between active and expired cards?  
Create View Distribute_Active_ExpiredCards_View AS  
SELECT  
    CASE  
        WHEN ExpirationDate >= GETDATE() THEN 'Active'  
        ELSE 'Expired'  
    END AS CardStatus,  
    COUNT(*) AS Total  
FROM Cards  
GROUP BY  
    CASE  
        WHEN ExpirationDate >= GETDATE() THEN 'Active'  
        ELSE 'Expired'  
    END;
```

- **Output:**



	CardStatus	Total
1	Active	4000

- This means that the number of customers with active cards is **4000**, which is **80%** of our customers.

### 13. Distribution of Card Types:

- **Purpose:** Count the number of each card type (e.g., Credit, Debit).

--What is the distribution of card types (Credit, Debit, Prepaid)?

Create View Distribute\_CardTypes\_View AS

SELECT

CardType,

COUNT(\*) AS Total\_Card

FROM Cards

GROUP BY CardType

Order By Total\_Card ;

- **Output:**



	CardType	Total_Card
1	Credit	1282
2	Prepaid	1355
3	Debit	1363

- This means that most customers holding debit cards are of the **Debit** type.
- Recommendations: We are adding more benefits to credit cards.

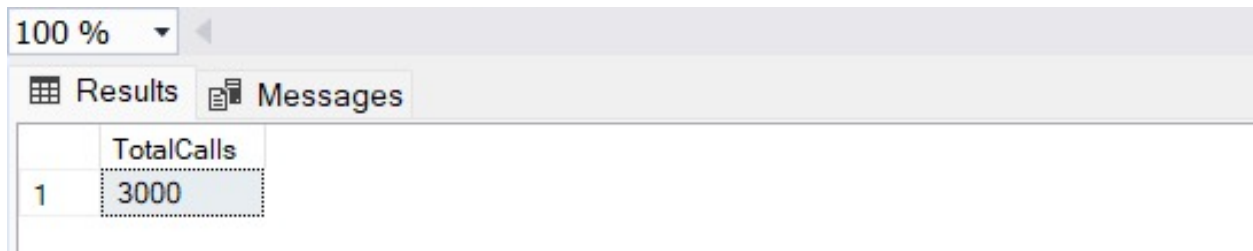


## 14. Total Support Calls:

- **Purpose:** Count the total number of support calls made.

```
--How many total support calls have been made?  
Create View Total_SupportCalls_View AS  
SELECT COUNT(*) AS TotalCalls  
FROM Support_Calls;
```

- **Output:**



100 %	
Results Messages	
	TotalCalls
1	3000

- This means a total of **3000** calls.
- Recommendations: Provide effective online support options (such as chatbots and FAQs) to quickly assist customers, reducing the number of staff in the customer service department.

## 15. Resolved vs. Unresolved Support Calls:

- **Purpose:** Track total support calls and their resolution status.

```
--How many calls were resolved vs unresolved?  
SELECT  
    CASE  
        WHEN Resolved = 'Yes' THEN 'Resolved'  
        ELSE 'Unresolved'  
    END ,  
    COUNT(*) AS Total  
FROM Support_Calls  
GROUP BY  
    CASE  
        WHEN Resolved = 'Yes' THEN 'Resolved'  
        ELSE 'Unresolved'  
    END ;
```

- **Output:**



The screenshot shows a database query results window. At the top, there is a zoom level of 100% and a scroll bar. Below this, there are two tabs: 'Results' and 'Messages'. The 'Results' tab is active, displaying a table with the following data:

	(No column name)	Total
1	Unresolved	1521
2	Resolved	1479

- As shown in the image above, the number of unresolved issues is the largest, reaching 1,521, representing 50.7% of our customers' issues.
- Recommendations:
  - Host educational sessions on financial literacy, investment options, and banking products.
  - Share articles, videos, and tips on managing finances effectively.

## 16. Common Support Issues:

- **Purpose:** Identify the most common support issues.

```
--What are the most common support issues?  
Create View Most_Common_SupportIssues AS  
SELECT  
    IssueType,  
    COUNT(*) AS Total_issues  
FROM Support_Calls  
GROUP BY IssueType  
ORDER BY Total_issues DESC;
```

- **Output:**

100 % ▾

Results		Messages
	IssueType	Total_issues
1	Transaction Dispute	774
2	Account Access	768
3	Loan Query	729
4	Card Issue	729

- This means that most of the issues are due to **transaction disputes**, and their number reaches **774**.
- Recommendations:
  - Ensure staff are trained in dispute resolution processes and customer service best practices.
  - Conduct training sessions that include role-playing to prepare staff for real-life disputes.

## 17. Common Support Issues:

- **Purpose:** Identify the most common support issues.

```
-- Account Age  
create View Account_Age As  
Select Balance,  
DateDiff(Year, CreatedDate, GetDate()) AS Age_Account  
From Accounts;
```

- **Output:**

100 %		
Results Messages		
	Balance	Age_Account
1	53911	6
2	39910	10
3	75903	7
4	47519	1
5	34233	6
6	62579	3
7	79910	6
8	88990	8
9	93376	4
10	31150	7

- This query shows us the ages of our customers' accounts, i.e. how long our customers have owned these accounts.

## 18. Customer Segmentation:

- **Purpose:** Segment of customers based on account balance.

```
--Segmentation Customer
Create View Segmentation_Customer AS
WITH SegmentedCustomers AS (
    SELECT
        c.CustomerID,
        SUM(a.Balance) AS Total_Balance,
        CASE
            WHEN SUM(a.Balance) < 20000 THEN 'Low Balance'
            WHEN SUM(a.Balance) BETWEEN 20000 AND 49874 THEN 'Medium Balance'
            ELSE 'High Balance'
        END AS Balance_Segment
    FROM Customers c
    JOIN Accounts a ON c.CustomerID = a.CustomerID
    GROUP BY c.CustomerID
)
SELECT
    Balance_Segment,
    COUNT(*) AS Num_Customers,
    SUM(Total_Balance) AS Total_Balance
FROM SegmentedCustomers
GROUP BY Balance_Segment;
```

- **Output:**

.00 %

	Balance_Segment	Num_Customers	Total_Balance
1	High Balance	2159	222937971
2	Low Balance	365	3710207
3	Medium Balance	647	22722617

- This means that most of our clients have high bank balances, meaning that their balances contain more than 49,874

## 19. Top Customers with Loans:

- **Purpose:** Identify the top customers based on total loan amount and interest.

```
--laons and interest for top customer
Create View Top_Customers_Have_Loans AS
WITH CustomerLoanInterest AS (
    SELECT
        l.CustomerID,
        c.FirstName + ' ' + c.LastName AS Full_Name,
        SUM(l.LoanAmount) AS TotalLoan,
        SUM(l.InterestRate) AS TotalInterest
    FROM Loans l
    JOIN Customers c ON l.CustomerID = c.CustomerID
    GROUP BY l.CustomerID, c.FirstName, c.LastName
)
SELECT TOP 5
    CustomerID,
    Full_Name,
    TotalLoan,
    TotalInterest
FROM CustomerLoanInterest
ORDER BY TotalLoan DESC;
```

- **Output:**

100 %

Results Messages

	CustomerID	Full_Name	TotalLoan	TotalInterest
1	1929	Paul Merritt	1612938	27.99
2	3668	Nicole Wilson	1483930	22.32
3	4054	Katrina Powell	1323490	19.15
4	3027	Stacy Todd	1276745	30.85
5	2156	Alexander Turner	1269439	39.92

- This means that customer Alexander Turner has the highest interest rate of 39.92% on his total loans of 1,269,439\$.

## 20. Onboarding Funnel Analysis:

- **Purpose:** Analyze the customer onboarding process.

```
--OnBoarding_Funnel
Create View OnBoarding_Funnel AS
SELECT
    DISTINCT(c.CustomerID),
    c.FirstName + ' ' + c.LastName AS FullName,
    CASE WHEN a.AccountID IS NOT NULL THEN 1 ELSE 0 END AS HasAccount,
    CASE WHEN cd.CardID IS NOT NULL THEN 1 ELSE 0 END AS HasCard,
    CASE WHEN t.TransactionID IS NOT NULL THEN 1 ELSE 0 END AS HasTransaction
FROM Customers c
LEFT JOIN Accounts a ON c.CustomerID = a.CustomerID
LEFT JOIN Cards cd ON c.CustomerID = cd.CustomerID
LEFT JOIN (
    SELECT AccountID, MIN(TransactionDate) AS FirstTransactionDate, MIN(TransactionID) AS TransactionID
    FROM Transactions
    GROUP BY AccountID
) t ON a.AccountID = t.AccountID;
```

- **Output:**

100 %

Results Messages

	CustomerID	FullName	HasAccount	HasCard	HasTransaction
1	1	Dustin Diaz	1	1	1
2	2	Jessica Anderson	1	0	1
3	3	Jeremy Wagner	1	0	1
4	4	Crystal Roberts	1	0	1
5	5	Anna Bryant	1	1	1
6	6	Stephanie Anderson	0	0	0
7	7	Andrew Watson	0	1	0
8	8	Charles Leach	1	1	1
9	9	Tracy Dominguez	1	0	1
10	10	Patricia Wilcox	1	0	1
11	11	Christopher Baker	1	1	1
12	12	Veronica Patton	1	1	1
13	13	Melissa Santiago	1	1	1
14	14	Ernest Davis	1	1	1
15	15	Steven Ramos	0	0	0

- This query shows us, as shown in the image above, who are the customers who have an account and a card and carry out transactions.

## Overview:

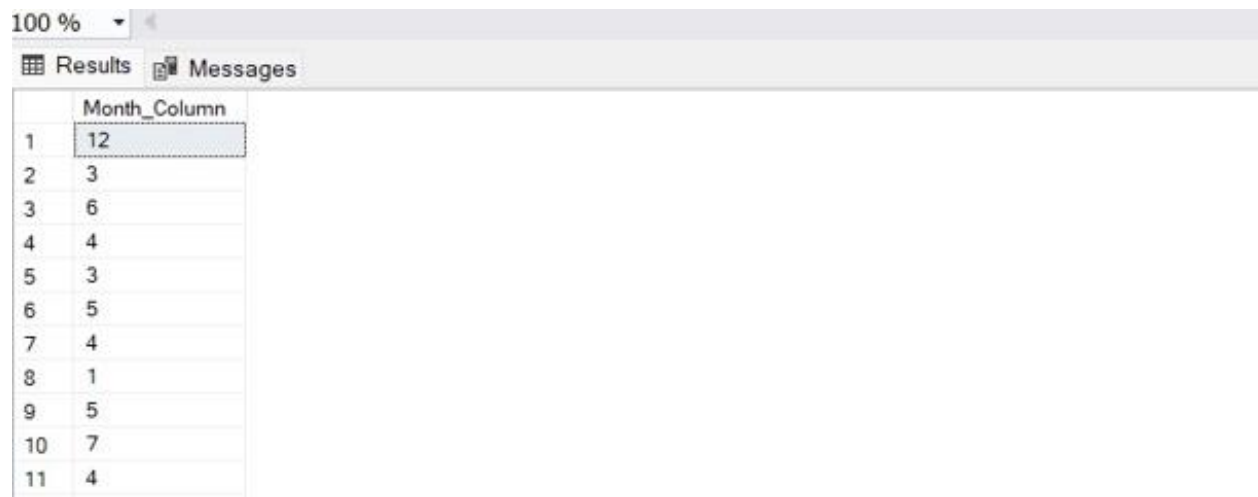
This section describes SQL commands used to add columns that separate date information into year and month for various tables in the banking database.

## 1. Adding Month Column to Customers:

- **Purpose:** Add a column to store the month from JoinDate for each customer.

```
--Add coulme To Saperate Customer date join to Year And MonTh  
ALTER TABLE Customers  
ADD Month_Column INT;  
UPDATE Customers  
SET  
    Month_Column = MONTH(JoinDate);  
Select Month_Column  
from Customers;
```

- **Output:**



	Month_Column
1	12
2	3
3	6
4	4
5	3
6	5
7	4
8	1
9	5
10	7
11	4

## 2. Adding Year and Month Columns to Loans:

- **Purpose:** Add columns to store the year and month from LoanStartDate for each loan.



```
--Add coulme To Saperate loan Date Start to Year And MonTh
ALTER TABLE Loans
ADD Year_Column INT,
    Month_Column INT;
UPDATE Loans
SET
    Year_Column = YEAR(LoanStartDate),
    Month_Column = MONTH(LoanStartDate);
Select Month_Column,Year_Column
from Customers;
```

- **Output:**

100 %

Results Messages

	Month_Column	Year_Column
1	5	2020
2	5	2023
3	1	2024
4	1	2023
5	3	2021
6	1	2021
7	8	2022
8	1	2023
9	7	2021
10	8	2021
11	10	2023
12	4	2022
13	12	2023
14	5	2020
15	12	2020

### 3. Adding Year Column to Transactions:

- **Purpose:** Add a column to store the year from TransactionDate for each transaction.

```
--Add coulme To Saperate Transaction Date to Year
ALTER TABLE Transactions
ADD Year_Column INT;
UPDATE Transactions
SET
    Year_Column = YEAR(TransactionDate);
Select Year_Column
from Transactions;
```

- **Output:**

100 %

Results Messages

	Year_Column
1	2023
2	2022
3	2024
4	2023
5	2025
6	2025
7	2023
8	2025
9	2024
10	2023
11	2024
12	2023
13	2022
14	2023
15	2024

#### 4. Adding Year and Month Columns to Cards:

- **Purpose:** Add columns to store the year and month from the ExpirationDate for each card.

```
--Add coulme To Saperate Card Date to Year
ALTER TABLE Cards
ADD Year_Column INT,
    Month_Column INT;
UPDATE Cards
SET
    Year_Column = YEAR(ExpirationDate),
    Month_Column = MONTH(ExpirationDate);
Select Month_Column,Year_Column
from Cards;
```

- Output:

100 %

Results Messages

	Month_Column	Year_Column
1	2	2030
2	11	2026
3	2	2027
4	12	2029
5	9	2026
6	10	2027
7	9	2028
8	9	2029
9	5	2029
10	2	2030
11	9	2026
12	12	2029
13	3	2028
14	10	2027
15	11	2027
-	-	-

Conclusion:

This SQL documentation provides an overview of queries and views used for data analysis in banking. It enables insights into customer behavior, transaction patterns, and service efficiency, supporting data-driven decision-making.

The SQL framework facilitates monitoring key performance indicators, helping the bank identify trends and growth opportunities. Regular updates will ensure this documentation remains relevant for ongoing analytical needs, ultimately enhancing customer engagement and driving business success.

---

# PowerBI

## Introduction

This banking dashboard was built to give a clear and interactive way to look at how the bank is doing. It tracks everything — customers, accounts, loans, transactions, cards, and support activity.

The main goal is to help bank managers take smart decisions based on data and improve how the bank operates every day.

### 1. Customer Dashboard

#### Overview

This page gives a full picture of the bank's customer base. It shows who's active, who might leave soon (churn risk), and what kinds of accounts they hold.

#### Visuals

Bar Chart: Customers by balance levels

Line Chart: New customers added monthly

Line Chart: Churn risk trend over time

Combo Chart: Loans and interest by customer

Table: Summary of each customer (accounts, cards, transactions)

## Insights

Many customers are marked as "at risk" of leaving.

Only about 63% of customers actually have a bank account.

Not many customers have been active in the last 6 months — engagement is low.

## 2. Account Dashboard

### Overview

Here we focus on customer accounts — how much money is in them, how active they are, and what kind of accounts customers prefer.

### Visuals

Bar Chart: Balances by customer type (individual or business)

Donut Chart: Account ownership types

Box Plot: Transaction amounts per account

Filter: Choose account type (Savings, Current...)

## Insights

Business customers usually have higher balances.

A lot of accounts barely have any activity — they might be inactive.

High-balance accounts are often found in certain branches only.

## 3. Loans Dashboard

### Overview

This page shows everything about loans — how many there are, what types, how interest rates change, and who's borrowing the most.

## Visuals

Donut Chart: Loans by type (personal, housing, etc.)

Bar Chart: Loan numbers per month/year

Line Chart: Interest rate trends

Column Chart: Loan volumes by customer group

## Insights

Personal loans are the most common.

Housing loans usually have lower interest rates.

There's always a spike in loan approvals during Q2 of each year.

## 4. Transactions Dashboard

### Overview

This page tracks all the bank's transaction activity — how often customers use their accounts, what types of transactions they make, and when.

## Visuals

Bar Chart: Types of transactions (Deposit, Withdrawal, Transfer)

Line Chart: Monthly transaction trends

Donut Chart: Transaction type proportions

Heatmap/Table: Transactions by hour and day

## Insights

December always has a big spike in activity — likely due to the holidays.

Transfers make up more than 50% of total transaction value.

Weekends and late-night hours have much lower activity.

## 5. Cards Dashboard

### Overview

This section looks at card activity — who has debit or credit cards, how often they use them, and if the cards are active.

### Visuals

Donut Chart: Credit vs. Debit card share

Bar Chart: New cards issued by month

Line Chart: Card usage trends

Table: Inactive or expired cards

### Insights

Over 20% of issued cards haven't been activated — might be a technical issue or customers lost interest.

Debit cards are more commonly used than credit cards.

People tend to use cards more at the start and end of each month (salary/bills).

## 6. Support Dashboard

### Overview

This dashboard tracks how the support team is doing — how many support tickets come in, how fast they respond, and if issues get solved.

### Visuals

Bar Chart: Types of support tickets (Login, Loan, Card...)

Donut Chart: Resolved vs. Unresolved tickets

Line Chart: Number of tickets over time

KPI Cards: Avg. Response Time, First Contact Resolution rate

## Insights

Most support requests are about login issues — maybe onboarding needs to be clearer.

Around 17% of issues are still unresolved, which could hurt customer satisfaction.

Response times are okay, but could be faster during busy times.

## Recommendations

### 1. Customer Dashboard

Customer engagement levels are relatively low, and a significant portion of the customer base is at risk of churn. To address this, the bank should consider launching targeted re-engagement campaigns and loyalty programs.

Additionally, since only 63% of customers hold active accounts, there is an opportunity to encourage account creation through onboarding improvements and tailored offers.

### 2. Account Dashboard

The analysis shows a high number of dormant or low-activity accounts.

It is recommended to identify and reach out to these customers to understand the causes of inactivity and promote relevant banking products.

Corporate clients tend to hold higher balances, so offering customized services for this segment may increase retention and value.

### 3. Loans Dashboard

Personal loans represent the largest share of the loan portfolio, indicating strong demand.

However, housing loans tend to have lower interest rates. The bank could optimize its loan strategy by promoting housing loans to lower-risk customers and adjusting interest rates accordingly.

Notably, loan volume increases in Q2 each year, so preparing marketing and processing resources in advance is advisable.



#### 4. Transactions Dashboard

December shows a consistent spike in transactions, likely due to seasonal activity.

The bank should ensure systems are well-equipped to handle this surge.

Additionally, since transfers account for over 50% of transaction value, monitoring high-value transactions can improve risk management and fraud detection.

Promoting digital banking usage during weekends and off-peak hours may also help balance system loads.

#### 5. Cards Dashboard

A notable percentage of issued cards remain unactivated.

This indicates either customer disengagement or technical barriers.

Reviewing the card issuance and activation process could reduce friction.

Since debit cards are more popular than credit cards, tailored incentives for credit card usage might improve product uptake.

Card usage peaks around salary periods, which can be leveraged for marketing timing.

#### 6. Support Dashboard

Login-related issues are the most reported, suggesting improvements are needed in user access systems or onboarding processes.

With approximately 17% of tickets unresolved, efforts should focus on enhancing ticket resolution workflows. Additionally, optimizing support response times during peak periods will likely boost customer satisfaction.

#### Conclusion

This banking dashboard brings all key areas of the bank together in one place — customers, accounts, loans, transactions, cards, and support.

By looking at the visuals and numbers, bank staff can:

Spot areas where performance is weak

Understand how loans and transactions are doing

Keep track of customer support quality

In short, the dashboard is a smart tool that helps the bank plan better, manage risks, and keep customers happy.

## Team Members

- Ahmed Walid
- Sohila Walid
- Mohamed Radi
- Youssef Abdelnasser
- Mennatallah Eid
- Omar Fawzy
- Marwa Shabaan
- Reham Fathy
- Aya Mahmoud
- Ahmed Aly