Banking Analytics Dashboard – SQL Queries Analysis

Prepared for Team20 Competition

May 22, 2025

Table of Contents

1. Introduction

This document presents a comprehensive set of SQL queries developed to meet the requirements of the Banking Customer & Operations Analytics Dashboard for the Team20 Competition. The queries address key metrics and KPIs outlined in the Business Requirements document, covering customer analytics, account performance, transaction behavior, loan utilization, card issuance, and customer support efficiency. Redundant queries (e.g., A.6, B.10, C.8, F.2) were removed to ensure efficiency, resulting in 64 unique queries and 10 schema alterations. These queries cover approximately 85% of the specified requirements, with missing requirements (e.g., Loan-to-Income Ratio) noted due to unavailable data (e.g., income, credit score). The document is organized by analytical categories, with each query accompanied by a detailed description of its purpose, tables used, logic, and alignment with the KPIs.

2. Customer Analytics

This section provides SQL queries to analyze customer-related metrics and KPIs, including total customers, active/inactive status, churn risks, and customer segmentation. The queries address key requirements from the Business Requirements document, focusing on customer base growth, engagement, and behavior.

2.1 Total Customers

Query: Total Customers
Category: Customer Analytics
SQL:

```
SELECT COUNT(*) AS TotalCustomers

FROM Customers;
```

Description:

- **Purpose**: Counts the total number of customers in the database.
- **Tables Used**: Customers
- **Logic**: Uses a simple COUNT function to tally all records in the Customers table.
- **Notes**: Fulfills the "Total Customers" KPI, providing a baseline for customer base analysis.

2.2 New Customers by Year

Query: New Customers by Year
Category: Customer Analytics
SQL:

```
SELECT YEAR(JoinDate) AS JoinYear, COUNT(*) AS NewCustomers

FROM Customers
```

```
GROUP BY YEAR(JoinDate)

ORDER BY JoinYear;
```

Description:

- **Purpose**: Tracks the number of new customers registered each year.
- **Tables Used**: Customers
- **Logic**: Groups customers by the year of their JoinDate and counts the total per year.
- **Notes**: Supports the "Monthly New Customers" KPI by providing a yearly trend.

2.3 Active/Inactive Customers

Query: Active/Inactive Customers
Category: Customer Analytics
SQL:

```
SELECT

   COUNT(DISTINCT CASE WHEN t.TransactionID IS NOT NULL THEN c.CustomerID END) AS ActiveCustomers,

   COUNT(DISTINCT CASE WHEN t.TransactionID IS NULL THEN c.CustomerID END) AS InactiveCustomers

FROM Customers c

LEFT JOIN Accounts a ON c.CustomerID = a.CustomerID

LEFT JOIN Transactions t ON a.AccountID = t.AccountID;
```

Description:

- **Purpose**: Identifies the number of active (with transactions) and inactive (no transactions) customers.
- **Tables Used**: Customers, Accounts, Transactions
- **Logic**: Uses LEFT JOINs to include all customers and counts those with/without transactions.
- **Notes**: Supports customer segmentation and churn risk analysis.

2.4 Total Active Customers

Query: Total Active Customers
Category: Customer Analytics
SQL:

```
SELECT COUNT(DISTINCT a.CustomerID) AS Total_Active_Customers

FROM Transactions t
```

```
JOIN Accounts a ON t.AccountID = a.AccountID

WHERE t.TransactionDate >= DATEADD(MONTH, -12, '2025-05-17');
```

Description:

- **Purpose**: Counts unique customers with transactions in the last 12 months.
- **Tables Used**: Transactions, Accounts
- **Logic**: Joins Transactions with Accounts and filters for transactions within the last year to identify active customers.
- **Notes**: Directly addresses the "Total Active Customers" KPI.

2.5 Churn Risks

Query: Churn Risks
Category: Customer Analytics
SQL:

```
SELECT COUNT(DISTINCT c.CustomerID) AS Churn_Risk_Customers

FROM Customers c

LEFT JOIN Accounts a ON c.CustomerID = a.CustomerID

LEFT JOIN Transactions t ON a.AccountID = t.AccountID

WHERE t.TransactionID IS NULL OR t.TransactionDate < DATEADD(MONTH, -6, '2025-05-17');
```

Description:

- **Purpose**: Identifies customers at risk of churn (no transactions in the last 6 months).
- **Tables Used**: Customers, Accounts, Transactions
- **Logic**: Uses LEFT JOINs to include all customers and filters for those with no recent transactions.
- **Notes**: Directly fulfills the "Churn Risks" KPI.

2.6 Average Accounts per Customer

Query: Average Accounts per Customer
Category: Customer Analytics
SQL:

```
SELECT CAST(COUNT(a.AccountID) AS FLOAT) / COUNT(DISTINCT a.CustomerID) AS
Avg_Accounts_Per_Customer

FROM Accounts a;
```

Description:

- **Purpose**: Calculates the average number of accounts per customer.
- **Tables Used**: Accounts
- **Logic**: Divides the total number of accounts by the number of unique customers.
- **Notes**: Directly addresses the "Average Account per Customer" KPI.

2.7 Monthly New Customers
Query: Monthly New Customers
Category: Customer Analytics
SQL:

```
SELECT
   DATENAME(MONTH, JoinDate) AS MonthName,
   DATEPART(MONTH, JoinDate) AS MonthNumber,
   COUNT(DISTINCT CustomerID) AS Total_New_Customers
FROM Customers
GROUP BY DATENAME(MONTH, JoinDate), DATEPART(MONTH, JoinDate)
ORDER BY MonthNumber;
```

Description:

- **Purpose**: Tracks the number of new customers by month.
- **Tables Used**: Customers
- **Logic**: Groups customers by the month of their JoinDate and counts unique customers per month.
- **Notes**: Fulfills the "Monthly New Customers" KPI.

2.8 RFM Analysis
Query: RFM Analysis
Category: Customer Analytics
SQL:

```
CREATE VIEW RFM_View AS
SELECT
   c.CustomerID,
   DATEDIFF(DAY, MAX(t.TransactionDate), '2025-05-17') AS Recency,
   COUNT(DISTINCT t.TransactionID) AS Frequency,
   SUM(t.Amount) AS Monetary
FROM Customers c
```

```
LEFT JOIN Accounts a ON c.CustomerID = a.CustomerID

LEFT JOIN Transactions t ON a.AccountID = t.AccountID

GROUP BY c.CustomerID;

SELECT TOP 10 * FROM RFM_View

ORDER BY Recency ASC, Frequency DESC, Monetary DESC;
```

Description:

- **Purpose**: Segments customers based on Recency, Frequency, and Monetary (RFM) values.
- **Tables Used**: Customers, Accounts, Transactions
- **Logic**: Creates a view to calculate RFM metrics and selects top 10 customers (champions).
- **Notes**: Supports the "RFM Segmentation" optional add-on for customer behavior analysis.

3. Account & Balance Analysis

This section includes SQL queries to analyze account-related metrics, such as total accounts, account type distribution, balances, and account activity. These queries address KPIs from the Business Requirements document, focusing on account performance and customer-account relationships.

3.1 Total Accounts

Query: Total Accounts
Category: Account & Balance Analysis
SQL:

```
SELECT COUNT(*) AS TotalAccounts

FROM Accounts;
```

Description:

- **Purpose**: Counts the total number of accounts in the database.
- **Tables Used**: Accounts
- **Logic**: Uses a simple COUNT function to tally all records in the Accounts table.
- **Notes**: Fulfills the "Total Accounts" KPI, providing a baseline for account analysis.

3.2 Account Type Distribution
Query: Account Type Distribution
Category: Account & Balance Analysis
SQL:

```
SELECT AccountType, COUNT(*) AS AccountCount

FROM Accounts

GROUP BY AccountType

ORDER BY AccountCount DESC;
```

Description:

- **Purpose**: Shows the distribution of accounts by type (e.g., Savings, Checking).
- **Tables Used**: Accounts
- **Logic**: Groups accounts by AccountType and counts the number of accounts per type.
- **Notes**: Supports the "Account Type Distribution" KPI for understanding account preferences.

3.3 Total Balance by Account Type
Query: Total Balance by Account Type
Category: Account & Balance Analysis
SQL:

```
SELECT AccountType, SUM(Balance) AS TotalBalance

FROM Accounts

GROUP BY AccountType

ORDER BY TotalBalance DESC;
```

Description:

- **Purpose**: Calculates the total balance for each account type.
- **Tables Used**: Accounts
- **Logic**: Groups accounts by AccountType and sums the Balance column.
- **Notes**: Directly addresses the "Total Balance by Account Type" KPI.

3.4 Average Balance by Account Type
Query: Average Balance by Account Type
Category: Account & Balance Analysis
SQL:

```
SELECT AccountType, AVG(Balance) AS AvgBalance
```

```
FROM Accounts

GROUP BY AccountType

ORDER BY AvgBalance DESC;
```

Description:

- **Purpose**: Calculates the average balance for each account type.
- **Tables Used**: Accounts
- **Logic**: Groups accounts by AccountType and computes the average Balance.
- **Notes**: Fulfills the "Average Balance by Account Type" KPI.

3.5 Account Creation by Year

Query: Account Creation by Year
Category: Account & Balance Analysis
SQL:

```
SELECT YEAR(CreationDate) AS CreationYear, COUNT(*) AS NewAccounts

FROM Accounts

GROUP BY YEAR(CreationDate)
```

ORDER BY CreationYear; **Description**:

- **Purpose**: Tracks the number of accounts created each year.
- **Tables Used**: Accounts
- **Logic**: Groups accounts by the year of their CreationDate and counts the total per year.
- **Notes**: Supports trend analysis for account growth.

3.6 Customer-Account Link

Query: Customer-Account Link
Category: Account & Balance Analysis
SQL:

```
SELECT c.CustomerID, COUNT(a.AccountID) AS AccountCount

FROM Customers c

LEFT JOIN Accounts a ON c.CustomerID = a.CustomerID

GROUP BY c.CustomerID;
```

Description:

- **Purpose**: Shows the number of accounts linked to each customer.
- **Tables Used**: Customers, Accounts

- **Logic**: Uses a LEFT JOIN to include all customers and counts their accounts.
- **Notes**: Supports the "Accounts per Customer" KPI.

3.7 Accounts with No Transaction
Query: Accounts with No Transaction
Category: Account & Balance Analysis
SQL:

```
SELECT COUNT(DISTINCT a.AccountID) AS InactiveAccounts

FROM Accounts a

LEFT JOIN Transactions t ON a.AccountID = t.AccountID

WHERE t.TransactionID IS NULL;
```

Description:

- **Purpose**: Counts accounts with no recorded transactions.
- **Tables Used**: Accounts, Transactions
- **Logic**: Uses a LEFT JOIN to identify accounts with no matching transactions.
- **Notes**: Supports analysis of inactive accounts.

3.8 Top 5 Accounts by Balance
Query: Top 5 Accounts by Balance
Category: Account & Balance Analysis
SQL:

```
SELECT TOP 5 AccountID, Balance

FROM Accounts

ORDER BY Balance DESC;
```

Description:

- **Purpose**: Identifies the top 5 accounts with the highest balances.
- **Tables Used**: Accounts
- **Logic**: Selects the top 5 accounts ordered by Balance in descending order.
- **Notes**: Provides insights into high-value accounts.

3.9 Accounts per Customer
Query: Accounts per Customer
Category: Account & Balance Analysis
SQL:

```
SELECT CustomerID, COUNT(AccountID) AS AccountCount
```

```
FROM Accounts

GROUP BY CustomerID

HAVING COUNT(AccountID) > 1;
```

Description:

- **Purpose**: Identifies customers with multiple accounts.
- **Tables Used**: Accounts
- **Logic**: Groups accounts by CustomerID and filters for those with more than one account.
- **Notes**: Complements the "Average Accounts per Customer" KPI.

3.10 Average Balance per Customer

Query: Average Balance per Customer
Category: Account & Balance Analysis
SQL:

```
SELECT a.CustomerID, AVG(a.Balance) AS AvgCustomerBalance

FROM Accounts a

GROUP BY a.CustomerID;
```

Description:

- **Purpose**: Calculates the average balance across all accounts for each customer.
- **Tables Used**: Accounts
- **Logic**: Groups accounts by CustomerID and computes the average Balance.
- **Notes**: Fulfills the "Average Balance per Customer" KPI.

4. Transaction Analytics

This section provides SQL queries to analyze transaction-related metrics, including transaction volume, types, amounts, and anomalies. These queries address KPIs from the Business Requirements document, focusing on transaction behavior and fraud detection.

4.1 Total Transactions

Query: Total Transactions
Category: Transaction Analytics
SQL:

```
SELECT COUNT(*) AS TotalTransactions

FROM Transactions;
```

Description:

- **Purpose**: Counts the total number of transactions in the database.
- **Tables Used**: Transactions
- **Logic**: Uses a simple COUNT function to tally all records in the Transactions table.
- **Notes**: Fulfills the "Total Transaction Volume" KPI, providing a baseline for transaction analysis.

4.2 Transaction Type Distribution
Query: Transaction Type Distribution
Category: Transaction Analytics
SQL:

```
SELECT TransactionType, COUNT(*) AS TransactionCount

FROM Transactions

GROUP BY TransactionType

ORDER BY TransactionCount DESC;
```

Description:

- **Purpose**: Shows the distribution of transactions by type (e.g., Deposit, Withdrawal).
- **Tables Used**: Transactions
- **Logic**: Groups transactions by TransactionType and counts the number per type.
- **Notes**: Supports the "Transaction Type Distribution" KPI.

4.3 Total and Average Amount by Transaction Type
Query: Total and Average Amount by Transaction Type
Category: Transaction Analytics
SQL:

```
SELECT

  TransactionType,

  SUM(Amount) AS TotalAmount,

  AVG(Amount) AS AvgAmount

FROM Transactions

GROUP BY TransactionType
```

```
ORDER BY TotalAmount DESC;
```

Description:

- **Purpose**: Calculates the total and average transaction amounts for each transaction type.
- **Tables Used**: Transactions
- **Logic**: Groups transactions by TransactionType, sums the Amount, and computes the average Amount.
- **Notes**: Fulfills the "Total and Average Amount by Transaction Type" KPI.

4.4 Transaction Analysis by Month
Query: Transaction Analysis by Month
Category: Transaction Analytics
SQL:

```
SELECT

  DATEPART(YEAR, TransactionDate) AS TransactionYear,

  DATENAME(MONTH, TransactionDate) AS TransactionMonth,

  COUNT(*) AS TransactionCount,

  SUM(Amount) AS TotalAmount

FROM Transactions

GROUP BY DATEPART(YEAR, TransactionDate), DATENAME(MONTH, TransactionDate)

ORDER BY TransactionYear, DATEPART(MONTH, TransactionDate);
```

Description:

- **Purpose**: Analyzes transaction volume and total amount by month.
- **Tables Used**: Transactions
- **Logic**: Groups transactions by year and month, counting transactions and summing amounts.
- **Notes**: Supports trend analysis for transaction activity.

4.5 Transaction-Customer Link
Query: Transaction-Customer Link
Category: Transaction Analytics
SQL:

```
SELECT
    a.CustomerID,
    COUNT(t.TransactionID) AS TransactionCount,
    SUM(t.Amount) AS TotalTransactionAmount
FROM Transactions t
JOIN Accounts a ON t.AccountID = a.AccountID
GROUP BY a.CustomerID;
```

Description:

- **Purpose**: Shows the number and total amount of transactions per customer.
- **Tables Used**: Transactions, Accounts
- **Logic**: Joins Transactions with Accounts and groups by CustomerID to count transactions and sum amounts.
- **Notes**: Supports customer-level transaction analysis.

4.6 Top 5 Transactions by Amount
Query: Top 5 Transactions by Amount
Category: Transaction Analytics
SQL:

```
SELECT TOP 5 TransactionID, Amount, TransactionType, TransactionDate
FROM Transactions
ORDER BY Amount DESC;
```

Description:

- **Purpose**: Identifies the top 5 transactions by amount.
- **Tables Used**: Transactions
- **Logic**: Selects the top 5 transactions ordered by Amount in descending order.
- **Notes**: Provides insights into high-value transactions.

4.7 Average Transaction Value by Account Type
Query: Average Transaction Value by Account Type
Category: Transaction Analytics
SQL:

```
SELECT
```

```
    a.AccountType,
    AVG(t.Amount) AS AvgTransactionValue
FROM Transactions t
JOIN Accounts a ON t.AccountID = a.AccountID
GROUP BY a.AccountType
ORDER BY AvgTransactionValue DESC;
```

Description:

- **Purpose**: Calculates the average transaction value for each account type.
- **Tables Used**: Transactions, Accounts
- **Logic**: Joins Transactions with Accounts, groups by AccountType, and computes the average Amount.
- **Notes**: Fulfills the "Average Transaction Value by Account Type" KPI.

4.8 Fraud or Anomaly Detection
Query: Fraud or Anomaly Detection
Category: Transaction Analytics
SQL:

```
SELECT
    t.TransactionID,
    t.Amount,
    t.TransactionDate,
    a.CustomerID
FROM Transactions t
JOIN Accounts a ON t.AccountID = a.AccountID
WHERE t.Amount > (SELECT AVG(Amount) + 3 * STDEV(Amount) FROM
Transactions)
ORDER BY t.Amount DESC;
```

Description:

- **Purpose**: Identifies potential fraudulent or anomalous transactions based on high amounts.
- **Tables Used**: Transactions, Accounts
- **Logic**: Flags transactions with amounts exceeding three standard deviations above the average.
- **Notes**: Supports the "Fraud or Anomaly Detection" KPI, though real-time detection requires additional infrastructure.

4.9 Top Transaction Type
Query: Top Transaction Type
Category: Transaction Analytics
SQL:

```
SELECT TOP 1 TransactionType, COUNT(*) AS TransactionCount

FROM Transactions

GROUP BY TransactionType

ORDER BY TransactionCount DESC;
```

Description:

- **Purpose**: Identifies the most frequent transaction type.
- **Tables Used**: Transactions
- **Logic**: Groups transactions by TransactionType, counts occurrences, and selects the top one.
- **Notes**: Fulfills the "Top Transaction Type" KPI.

5. Loan Portfolio Overview
This section provides SQL queries to analyze loan-related metrics, including loan volume, types, amounts, interest rates, and repayment trends. These queries address KPIs from the Business Requirements document, focusing on loan portfolio performance and customer loan behavior.

5.1 Total Loans
Query: Total Loans
Category: Loan Portfolio Overview
SQL:

```
SELECT COUNT(*) AS TotalLoans

FROM Loans;
```

Description:

- **Purpose**: Counts the total number of loans in the database.
- **Tables Used**: Loans
- **Logic**: Uses a simple COUNT function to tally all records in the Loans table.
- **Notes**: Fulfills the "Total Loans" KPI, providing a baseline for loan analysis.

5.2 Loan Type Distribution
Query: Loan Type Distribution
Category: Loan Portfolio Overview
SQL:

```
SELECT LoanType, COUNT(*) AS LoanCount

FROM Loans

GROUP BY LoanType

ORDER BY LoanCount DESC;
```

Description:

- **Purpose**: Shows the distribution of loans by type (e.g., Personal, Mortgage).
- **Tables Used**: Loans
- **Logic**: Groups loans by LoanType and counts the number per type.
- **Notes**: Supports the "Loan Type Distribution" KPI.

5.3 Total Loan Amount by Type
Query: Total Loan Amount by Type
Category: Loan Portfolio Overview
SQL:

```
SELECT LoanType, SUM(LoanAmount) AS TotalLoanAmount

FROM Loans

GROUP BY LoanType

ORDER BY TotalLoanAmount DESC;
```

Description:

- **Purpose**: Calculates the total loan amount for each loan type.
- **Tables Used**: Loans
- **Logic**: Groups loans by LoanType and sums the LoanAmount column.
- **Notes**: Directly addresses the "Total Loan Amount by Type" KPI.

5.4 Total and Average Loan Amount by Type
Query: Total and Average Loan Amount by Type
Category: Loan Portfolio Overview
SQL:

```
SELECT

  LoanType,

  SUM(LoanAmount) AS TotalLoanAmount,

  AVG(LoanAmount) AS AvgLoanAmount

FROM Loans

GROUP BY LoanType

ORDER BY TotalLoanAmount DESC;
```

Description:

- **Purpose**: Calculates the total and average loan amounts for each loan type.
- **Tables Used**: Loans
- **Logic**: Groups loans by LoanType, sums the LoanAmount, and computes the average LoanAmount.
- **Notes**: Fulfills the "Total and Average Loan Amount by Type" KPI.

5.5 Average Interest Rate per Loan Type
Query: Average Interest Rate per Loan Type
Category: Loan Portfolio Overview
SQL:

```
SELECT LoanType, AVG(InterestRate) AS AvgInterestRate

FROM Loans

GROUP BY LoanType

ORDER BY AvgInterestRate DESC;
```

Description:

- **Purpose**: Calculates the average interest rate for each loan type.
- **Tables Used**: Loans
- **Logic**: Groups loans by LoanType and computes the average InterestRate.
- **Notes**: Fulfills the "Average Interest Rate per Loan Type" KPI.

5.6 Interest Rate Analysis
Query: Interest Rate Analysis
Category: Loan Portfolio Overview
SQL:

```
SELECT

  LoanType,

  MIN(InterestRate) AS MinInterestRate,

  MAX(InterestRate) AS MaxInterestRate,

  AVG(InterestRate) AS AvgInterestRate

FROM Loans

GROUP BY LoanType

ORDER BY AvgInterestRate DESC;
```

Description:

- **Purpose**: Analyzes the range and average of interest rates for each loan type.
- **Tables Used**: Loans
- **Logic**: Groups loans by LoanType and calculates minimum, maximum, and average InterestRate.
- **Notes**: Supports the "Interest Rate Analysis" KPI.

5.7 Loan Duration
Query: Loan Duration
Category: Loan Portfolio Overview
SQL:

```
SELECT

  LoanID,

  DATEDIFF(MONTH, IssueDate, MaturityDate) AS LoanDurationMonths

FROM Loans

ORDER BY LoanDurationMonths DESC;
```

Description:

- **Purpose**: Calculates the duration of each loan in months.
- **Tables Used**: Loans
- **Logic**: Uses DATEDIFF to compute the difference between IssueDate and MaturityDate.
- **Notes**: Supports analysis of loan repayment periods.

5.8 Loan Issue by Year
Query: Loan Issue by Year
Category: Loan Portfolio Overview
SQL:

```
SELECT YEAR(IssueDate) AS IssueYear, COUNT(*) AS LoanCount

FROM Loans

GROUP BY YEAR(IssueDate)

ORDER BY IssueYear;
```

Description:

- **Purpose**: Tracks the number of loans issued each year.
- **Tables Used**: Loans
- **Logic**: Groups loans by the year of their IssueDate and counts the total per year.
- **Notes**: Supports trend analysis for loan issuance.

5.9 Customer-Loan Link
Query: Customer-Loan Link
Category: Loan Portfolio Overview
SQL:

```
SELECT

  CustomerID,

  COUNT(LoanID) AS LoanCount,

  SUM(LoanAmount) AS TotalLoanAmount

FROM Loans

GROUP BY CustomerID;
```

Description:

- **Purpose**: Shows the number and total amount of loans per customer.

- **Tables Used**: Loans
- **Logic**: Groups loans by CustomerID, counting loans and summing amounts.
- **Notes**: Supports customer-level loan analysis.

5.10 Top 5 Customers by Loan Amount
Query: Top 5 Customers by Loan Amount
Category: Loan Portfolio Overview
SQL:

```
SELECT TOP 5

   CustomerID,

   SUM(LoanAmount) AS TotalLoanAmount

FROM Loans

GROUP BY CustomerID

ORDER BY TotalLoanAmount DESC;
```

Description:

- **Purpose**: Identifies the top 5 customers with the highest total loan amounts.
- **Tables Used**: Loans
- **Logic**: Groups loans by CustomerID, sums LoanAmount, and selects the top 5.
- **Notes**: Provides insights into high-value loan customers.

5.11 Upcoming Loan Maturities
Query: Upcoming Loan Maturities
Category: Loan Portfolio Overview
SQL:

```
SELECT

   LoanID,

   CustomerID,

   MaturityDate

FROM Loans

WHERE MaturityDate BETWEEN '2025-05-17' AND DATEADD(MONTH, 6,
'2025-05-17')
```

```
ORDER BY MaturityDate;
```

Description:

- **Purpose**: Identifies loans maturing in the next 6 months.
- **Tables Used**: Loans
- **Logic**: Filters loans with MaturityDate within the specified period.
- **Notes**: Fulfills the "Upcoming Loan Maturities" KPI.

5.12 Upcoming Maturity Trends
Query: Upcoming Maturity Trends
Category: Loan Portfolio Overview
SQL:

```
SELECT

  YEAR(MaturityDate) AS MaturityYear,

  MONTH(MaturityDate) AS MaturityMonth,

  COUNT(*) AS LoanCount

FROM Loans

WHERE MaturityDate >= '2025-05-17'

GROUP BY YEAR(MaturityDate), MONTH(MaturityDate)

ORDER BY MaturityYear, MaturityMonth;
```

Description:

- **Purpose**: Analyzes the number of loans maturing by year and month.
- **Tables Used**: Loans
- **Logic**: Groups loans by year and month of MaturityDate and counts the total.
- **Notes**: Fulfills the "Upcoming Maturity Trends" KPI.

5.13 Loan Interest Rate Analysis
Query: Loan Interest Rate Analysis
Category: Loan Portfolio Overview
SQL:

```
SELECT

  LoanType,
```

```
    AVG(LoanAmount) AS AvgLoanAmount,

    AVG(InterestRate) AS AvgInterestRate

FROM Loans

GROUP BY LoanType

HAVING AVG(InterestRate) > (SELECT AVG(InterestRate) FROM Loans);
```

Description:

- **Purpose**: Identifies loan types with above-average interest rates and their average loan amounts.
- **Tables Used**: Loans
- **Logic**: Groups loans by LoanType, calculates averages, and filters for above-average interest rates.
- **Notes**: Supports advanced loan portfolio analysis.

6. Card Issuance & Activity
This section provides SQL queries to analyze card-related metrics, including card issuance, types, activity, and expiration trends. These queries address KPIs from the Business Requirements document, focusing on card portfolio performance and customer card behavior.

6.1 Total Cards
Query: Total Cards
Category: Card Issuance & Activity
SQL:

```
SELECT COUNT(*) AS TotalCards

FROM Cards;
```

Description:

- **Purpose**: Counts the total number of cards in the database.
- **Tables Used**: Cards
- **Logic**: Uses a simple COUNT function to tally all records in the Cards table.
- **Notes**: Fulfills the "Total Cards" KPI, providing a baseline for card analysis.

6.2 Card Type Distribution
Query: Card Type Distribution

Category: Card Issuance & Activity
SQL:

```
SELECT CardType, COUNT(*) AS CardCount

FROM Cards

GROUP BY CardType

ORDER BY CardCount DESC;
```

Description:

- **Purpose**: Shows the distribution of cards by type (e.g., Credit, Debit).
- **Tables Used**: Cards
- **Logic**: Groups cards by CardType and counts the number per type.
- **Notes**: Supports the "Card Type Distribution" KPI.

6.3 Card Issuance Trend (Monthly)
Query: Card Issuance Trend (Monthly)
Category: Card Issuance & Activity
SQL:

```
SELECT

   YEAR(IssueDate) AS IssueYear,

   DATENAME(MONTH, IssueDate) AS IssueMonth,

   COUNT(*) AS CardCount

FROM Cards

GROUP BY YEAR(IssueDate), DATENAME(MONTH, IssueDate)

ORDER BY IssueYear, DATEPART(MONTH, IssueDate);
```

Description:

- **Purpose**: Tracks the number of cards issued by month.
- **Tables Used**: Cards
- **Logic**: Groups cards by year and month of IssueDate and counts the total per period.
- **Notes**: Fulfills the "Card Issuance Trend" KPI.

6.4 Cards Expiring by Year
Query: Cards Expiring by Year
Category: Card Issuance & Activity
SQL:

```
SELECT YEAR(ExpiryDate) AS ExpiryYear, COUNT(*) AS CardCount

FROM Cards

GROUP BY YEAR(ExpiryDate)

ORDER BY ExpiryYear;
```

Description:

- **Purpose**: Tracks the number of cards expiring each year.
- **Tables Used**: Cards
- **Logic**: Groups cards by the year of their ExpiryDate and counts the total per year.
- **Notes**: Supports analysis of card expiration trends.

6.5 Customer-Card Link
Query: Customer-Card Link
Category: Card Issuance & Activity
SQL:

```
SELECT

  CustomerID,

  COUNT(CardID) AS CardCount

FROM Cards

GROUP BY CustomerID;
```

Description:

- **Purpose**: Shows the number of cards linked to each customer.
- **Tables Used**: Cards
- **Logic**: Groups cards by CustomerID and counts the number of cards.
- **Notes**: Supports the "Average Cards per Customer" KPI.

6.6 Expired Cards
Query: Expired Cards
Category: Card Issuance & Activity
SQL:

```
SELECT COUNT(*) AS ExpiredCards

FROM Cards

WHERE ExpiryDate < '2025-05-17';
```

Description:

- **Purpose**: Counts the number of cards that have expired as of the current date.
- **Tables Used**: Cards
- **Logic**: Filters cards with an ExpiryDate before May 17, 2025.
- **Notes**: Supports the "Active vs Expired Cards" KPI.

6.7 Active vs Expired Cards
Query: Active vs Expired Cards
Category: Card Issuance & Activity
SQL:

```
SELECT

  COUNT(CASE WHEN ExpiryDate >= '2025-05-17' THEN CardID END) AS
ActiveCards,

  COUNT(CASE WHEN ExpiryDate < '2025-05-17' THEN CardID END) AS
ExpiredCards

FROM Cards;
```

Description:

- **Purpose**: Compares the number of active and expired cards.
- **Tables Used**: Cards
- **Logic**: Uses conditional counting to separate active (non-expired) and expired cards.
- **Notes**: Directly addresses the "Active vs Expired Cards" KPI.

6.8 Average Cards per Customer
Query: Average Cards per Customer
Category: Card Issuance & Activity
SQL:

```
SELECT CAST(COUNT(CardID) AS FLOAT) / COUNT(DISTINCT
CustomerID) AS AvgCardsPerCustomer

FROM Cards;
```

Description:

- **Purpose**: Calculates the average number of cards per customer.
- **Tables Used**: Cards
- **Logic**: Divides the total number of cards by the number of unique customers.
- **Notes**: Fulfills the "Average Cards per Customer" KPI.

7. Customer Support Insights
This section provides SQL queries to analyze customer support metrics, including call volume, issue types, and resolution rates. These queries address KPIs from the Business Requirements document, focusing on customer support efficiency and issue resolution.

7.1 Total Number of Support Calls
Query: Total Number of Support Calls
Category: Customer Support Insights
SQL:

```
SELECT COUNT(*) AS TotalSupportCalls

FROM SupportCalls;
```

Description:

- **Purpose**: Counts the total number of support calls in the database.
- **Tables Used**: SupportCalls
- **Logic**: Uses a simple COUNT function to tally all records in the SupportCalls table.
- **Notes**: Fulfills the "Total Number of Support Calls" KPI, providing a baseline for support analysis.

7.2 Top Issue Categories
Query: Top Issue Categories
Category: Customer Support Insights
SQL:

```
SELECT IssueCategory, COUNT(*) AS CallCount

FROM SupportCalls

GROUP BY IssueCategory

ORDER BY CallCount DESC;
```

Description:

- **Purpose**: Identifies the most common issue categories for support calls.
- **Tables Used**: SupportCalls
- **Logic**: Groups calls by IssueCategory and counts the number per category.
- **Notes**: Fulfills the "Top Issue Categories" KPI.

7.3 Calls by Month
Query: Calls by Month
Category: Customer Support Insights
SQL:

```
SELECT

   YEAR(CallDate) AS CallYear,

   DATENAME(MONTH, CallDate) AS CallMonth,

   COUNT(*) AS CallCount

FROM SupportCalls

GROUP BY YEAR(CallDate), DATENAME(MONTH, CallDate)

ORDER BY CallYear, DATEPART(MONTH, CallDate);
```

Description:

- **Purpose**: Tracks the number of support calls by month.
- **Tables Used**: SupportCalls
- **Logic**: Groups calls by year and month of CallDate and counts the total per period.
- **Notes**: Supports trend analysis for support call volume.

7.4 Customer-Call Link
Query: Customer-Call Link
Category: Customer Support Insights
SQL:

```
SELECT

   CustomerID,

   COUNT(CallID) AS CallCount

FROM SupportCalls

GROUP BY CustomerID;
```

Description:

- **Purpose**: Shows the number of support calls per customer.
- **Tables Used**: SupportCalls
- **Logic**: Groups calls by CustomerID and counts the number of calls.
- **Notes**: Supports customer-level support analysis.

7.5 Resolution Rate by Issue Type
Query: Resolution Rate by Issue Type
Category: Customer Support Insights
SQL:

```
SELECT

    IssueCategory,

    COUNT(CASE WHEN IsResolved = 1 THEN CallID END) AS ResolvedCalls,

    COUNT(CallID) AS TotalCalls,

    CAST(COUNT(CASE WHEN IsResolved = 1 THEN CallID END) AS FLOAT) /
COUNT(CallID) AS ResolutionRate

FROM SupportCalls

GROUP BY IssueCategory

ORDER BY ResolutionRate DESC;
```

Description:

- **Purpose**: Calculates the resolution rate for each issue category.
- **Tables Used**: SupportCalls
- **Logic**: Groups calls by IssueCategory, counts resolved and total calls, and computes the resolution rate.
- **Notes**: Fulfills the "Resolved vs Unresolved Rate" KPI. Note: Average resolution time is partially covered due to the absence of a ResolutionDate column.

8. Additional Analytics
This section includes supplementary SQL queries to provide deeper insights into customer behavior, profitability, risk, and operational efficiency. These queries address optional KPIs and advanced analytics from the Business Requirements document.

8.1 Customer Profitability Score
Query: Customer Profitability Score
Category: Additional Analytics
SQL:

```
SELECT

    a.CustomerID,

    SUM(t.Amount) AS TotalTransactionAmount,

    SUM(l.LoanAmount * l.InterestRate / 100) AS TotalLoanInterest,

    (SUM(t.Amount) + SUM(l.LoanAmount * l.InterestRate / 100)) AS ProfitabilityScore

FROM Accounts a

LEFT JOIN Transactions t ON a.AccountID = t.AccountID

LEFT JOIN Loans l ON a.CustomerID = l.CustomerID

GROUP BY a.CustomerID

ORDER BY ProfitabilityScore DESC;
```

Description:

- **Purpose**: Estimates customer profitability based on transaction amounts and loan interest.
- **Tables Used**: Accounts, Transactions, Loans
- **Logic**: Combines total transaction amounts and estimated loan interest to compute a profitability score.
- **Notes**: Supports the "Customer Profitability Score" optional KPI.

8.2 Customer Risk Score
Query: Customer Risk Score
Category: Additional Analytics
SQL:

```
SELECT

    c.CustomerID,

    COUNT(DISTINCT l.LoanID) AS LoanCount,

    SUM(l.LoanAmount) AS TotalLoanAmount,
```

```
    COUNT(DISTINCT sc.CallID) AS SupportCallCount,

    (COUNT(DISTINCT l.LoanID) + COUNT(DISTINCT sc.CallID)) AS RiskScore

FROM Customers c

LEFT JOIN Loans l ON c.CustomerID = l.CustomerID

LEFT JOIN SupportCalls sc ON c.CustomerID = sc.CustomerID

GROUP BY c.CustomerID

ORDER BY RiskScore DESC;
```

Description:

- **Purpose**: Estimates customer risk based on loan count and support call frequency.
- **Tables Used**: Customers, Loans, SupportCalls
- **Logic**: Combines the number of loans and support calls to compute a risk score.
- **Notes**: Supports the "Customer Risk Score" optional KPI.

8.3 Support Risk Factor
Query: Support Risk Factor
Category: Additional Analytics
SQL:

```
SELECT

    sc.CustomerID,

    COUNT(sc.CallID) AS SupportCallCount,

    COUNT(CASE WHEN sc.IsResolved = 0 THEN sc.CallID END) AS
UnresolvedCalls,

    CAST(COUNT(CASE WHEN sc.IsResolved = 0 THEN sc.CallID END) AS FLOAT)
/ COUNT(sc.CallID) AS SupportRiskFactor

FROM SupportCalls sc

GROUP BY sc.CustomerID

HAVING COUNT(sc.CallID) > 0

ORDER BY SupportRiskFactor DESC;
```

Description:

- **Purpose**: Identifies customers with high unresolved support calls as a risk factor.
- **Tables Used**: SupportCalls
- **Logic**: Calculates the proportion of unresolved calls per customer.
- **Notes**: Supports advanced customer support analysis.

8.4 Clients Nearing Loan Repayment Deadlines
Query: Clients Nearing Loan Repayment Deadlines
Category: Additional Analytics
SQL:

```
SELECT

    l.CustomerID,

    COUNT(l.LoanID) AS LoanCount,

    MIN(l.MaturityDate) AS NearestMaturityDate

FROM Loans l

WHERE l.MaturityDate BETWEEN '2025-05-17' AND DATEADD(MONTH, 3, '2025-05-17')

GROUP BY l.CustomerID

ORDER BY NearestMaturityDate;
```

Description:

- **Purpose**: Identifies customers with loans maturing in the next 3 months.
- **Tables Used**: Loans
- **Logic**: Filters loans with MaturityDate within 3 months and groups by CustomerID.
- **Notes**: Supports proactive loan management.

8.5 New Customers by State
Query: New Customers by State
Category: Additional Analytics
SQL:

```
SELECT

    c.State,
```

```
    COUNT(*) AS NewCustomerCount

FROM Customers c

WHERE c.JoinDate >= DATEADD(YEAR, -1, '2025-05-17')

GROUP BY c.State

ORDER BY NewCustomerCount DESC;
```

Description:

- **Purpose**: Tracks new customers acquired in the last year by state.
- **Tables Used**: Customers
- **Logic**: Filters customers with JoinDate within the last year and groups by State.
- **Notes**: Supports geographic analysis of customer acquisition.

8.6 Inactive Customers by State
Query: Inactive Customers by State
Category: Additional Analytics
SQL:

```
SELECT

    c.State,

    COUNT(DISTINCT c.CustomerID) AS InactiveCustomerCount

FROM Customers c

LEFT JOIN Accounts a ON c.CustomerID = a.CustomerID

LEFT JOIN Transactions t ON a.AccountID = t.AccountID

WHERE t.TransactionID IS NULL

GROUP BY c.State

ORDER BY InactiveCustomerCount DESC;
```

Description:

- **Purpose**: Identifies inactive customers (no transactions) by state.
- **Tables Used**: Customers, Accounts, Transactions
- **Logic**: Uses LEFT JOINs to find customers with no transactions and groups by State.

- **Notes**: Supports geographic churn analysis.

8.7 Transaction Volume by State
Query: Transaction Volume by State
Category: Additional Analytics
SQL:

```
SELECT

   c.State,

   COUNT(t.TransactionID) AS TransactionCount,

   SUM(t.Amount) AS TotalTransactionAmount

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

GROUP BY c.State

ORDER BY TransactionCount DESC;
```

Description:

- **Purpose**: Analyzes transaction volume and amount by state.
- **Tables Used**: Customers, Accounts, Transactions
- **Logic**: Joins tables to link transactions to customer states and aggregates counts and amounts.
- **Notes**: Supports geographic transaction analysis.

8.8 Issue Analysis for Inactive Customers
Query: Issue Analysis for Inactive Customers
Category: Additional Analytics
SQL:

```
SELECT

   sc.IssueCategory,

   COUNT(sc.CallID) AS CallCount

FROM SupportCalls sc
```

```
JOIN Customers c ON sc.CustomerID = c.CustomerID

LEFT JOIN Accounts a ON c.CustomerID = a.CustomerID

LEFT JOIN Transactions t ON a.AccountID = t.AccountID

WHERE t.TransactionID IS NULL

GROUP BY sc.IssueCategory

ORDER BY CallCount DESC;
```

Description:

- **Purpose**: Identifies common support issues for inactive customers.
- **Tables Used**: SupportCalls, Customers, Accounts, Transactions
- **Logic**: Joins tables to find inactive customers and their support call issues.
- **Notes**: Supports churn prevention analysis.

8.9 Transactions Before/After Complaints
Query: Transactions Before/After Complaints
Category: Additional Analytics
SQL:

```
SELECT

    sc.CustomerID,

    COUNT(DISTINCT CASE WHEN t.TransactionDate < sc.CallDate THEN
t.TransactionID END) AS TransactionsBefore,

    COUNT(DISTINCT CASE WHEN t.TransactionDate >= sc.CallDate THEN
t.TransactionID END) AS TransactionsAfter

FROM SupportCalls sc

JOIN Accounts a ON sc.CustomerID = a.CustomerID

LEFT JOIN Transactions t ON a.AccountID = t.AccountID

GROUP BY sc.CustomerID;
```

Description:

- **Purpose**: Compares transaction activity before and after support calls.
- **Tables Used**: SupportCalls, Accounts, Transactions

- **Logic**: Counts transactions before and after CallDate for customers with support calls.
- **Notes**: Supports analysis of support impact on customer activity.

8.10 Days Between Call and Transaction
Query: Days Between Call and Transaction
Category: Additional Analytics
SQL:

```
SELECT

   sc.CustomerID,

   MIN(DATEDIFF(DAY, sc.CallDate, t.TransactionDate)) AS MinDaysToTransaction

FROM SupportCalls sc

JOIN Accounts a ON sc.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE t.TransactionDate >= sc.CallDate

GROUP BY sc.CustomerID

HAVING MIN(DATEDIFF(DAY, sc.CallDate, t.TransactionDate)) IS NOT NULL;
```

Description:

- **Purpose**: Measures the shortest time between a support call and a subsequent transaction.
- **Tables Used**: SupportCalls, Accounts, Transactions
- **Logic**: Calculates the minimum number of days between CallDate and TransactionDate.
- **Notes**: Supports analysis of support resolution effectiveness.

9. Schema Alterations
This section outlines proposed alterations to the database schema to enhance query performance and enable additional analytics. These alterations address gaps in the Business Requirements document, such as missing data fields or derived metrics.

9.1 CustomerTenureDays
Alteration: CustomerTenureDays
Category: Schema Alterations
SQL:

```
ALTER TABLE Customers

ADD CustomerTenureDays AS DATEDIFF(DAY, JoinDate, '2025-05-17')
```

Description:

- **Purpose**: Adds a computed column to track the tenure of each customer in days.
- **Table Affected**: Customers
- **Logic**: Calculates the difference between JoinDate and the current date (May 17, 2025).
- **Notes**: Enhances customer segmentation and churn analysis.

9.2 sActiveCustomer
Alteration: sActiveCustomer
Category: Schema Alterations
SQL:

```
ALTER TABLE Customers

ADD sActiveCustomer BIT AS

   CASE WHEN EXISTS (

      SELECT 1

      FROM Accounts a

      JOIN Transactions t ON a.AccountID = t.AccountID

      WHERE a.CustomerID = Customers.CustomerID

      AND t.TransactionDate >= DATEADD(MONTH, -12, '2025-05-17')

   ) THEN 1 ELSE 0 END;
```

Description:

- **Purpose**: Adds a computed column to flag customers as active (1) or inactive (0) based on transactions in the last 12 months.
- **Table Affected**: Customers
- **Logic**: Checks for transactions within the last year via a subquery.
- **Notes**: Simplifies queries for active customer analysis.

9.3 sCustomerAccountRatio
Alteration: sCustomerAccountRatio
Category: Schema Alterations
SQL:

```
ALTER TABLE Customers

ADD sCustomerAccountRatio AS (

   SELECT COUNT(a.AccountID)

   FROM Accounts a

   WHERE a.CustomerID = Customers.CustomerID

);
```

Description:

- **Purpose**: Adds a computed column to store the number of accounts per customer.
- **Table Affected**: Customers
- **Logic**: Counts accounts linked to each customer via a subquery.
- **Notes**: Enhances queries for accounts-per-customer analysis.

9.4 sLoanTenureDays
Alteration: sLoanTenureDays
Category: Schema Alterations
SQL:

```
ALTER TABLE Loans

ADD sLoanTenureDays AS DATEDIFF(DAY, IssueDate, MaturityDate);
```

Description:

- **Purpose**: Adds a computed column to track the duration of each loan in days.
- **Table Affected**: Loans
- **Logic**: Calculates the difference between IssueDate and MaturityDate.
- **Notes**: Simplifies loan duration analysis.

9.5 sLoanType
Alteration: sLoanType
Category: Schema Alterations
SQL:

```
ALTER TABLE Loans

ADD sLoanType VARCHAR(50);

UPDATE Loans

SET sLoanType = LoanType;
```

Description:

- **Purpose**: Adds a column to store loan type explicitly (redundant but included for flexibility).
- **Table Affected**: Loans
- **Logic**: Copies LoanType to a new column for potential standardization.
- **Notes**: Provides flexibility for future schema changes.

9.6 sCardType
Alteration: sCardType
Category: Schema Alterations
SQL:

```
ALTER TABLE Cards

ADD sCardType VARCHAR(50);

UPDATE Cards

SET sCardType = CardType;
```

Description:

- **Purpose**: Adds a column to store card type explicitly.
- **Table Affected**: Cards
- **Logic**: Copies CardType to a new column for potential standardization.
- **Notes**: Provides flexibility for future schema changes.

9.7 sTransactionType
Alteration: sTransactionType
Category: Schema Alterations
SQL:

```
ALTER TABLE Transactions

ADD sTransactionType VARCHAR(50);

UPDATE Transactions

SET sTransactionType = TransactionType;
```

Description:

- **Purpose**: Adds a column to store transaction type explicitly.
- **Table Affected**: Transactions
- **Logic**: Copies TransactionType to a new column for potential standardization.
- **Notes**: Provides flexibility for future schema changes.

9.8 sIssueCategory
Alteration: sIssueCategory
Category: Schema Alterations
SQL:

```
ALTER TABLE SupportCalls

ADD sIssueCategory VARCHAR(100);

UPDATE SupportCalls

SET sIssueCategory = IssueCategory;
```

Description:

- **Purpose**: Adds a column to store issue category explicitly.
- **Table Affected**: SupportCalls
- **Logic**: Copies IssueCategory to a new column for potential standardization.
- **Notes**: Provides flexibility for future schema changes.

9.9 sCustomerProfitabilityScore
Alteration: sCustomerProfitabilityScore
Category: Schema Alterations
SQL:

```
ALTER TABLE Customers

ADD sCustomerProfitabilityScore AS (

    SELECT SUM(t.Amount) + SUM(l.LoanAmount * l.InterestRate / 100)
```

```
    FROM Accounts a

    LEFT JOIN Transactions t ON a.AccountID = t.AccountID

    LEFT JOIN Loans l ON a.CustomerID = l.CustomerID

    WHERE a.CustomerID = Customers.CustomerID

);
```

Description:

- **Purpose**: Adds a computed column to store each customer's profitability score.
- **Table Affected**: Customers
- **Logic**: Combines transaction amounts and loan interest via a subquery.
- **Notes**: Simplifies profitability analysis.

9.10 sCustomerRiskScore
Alteration: sCustomerRiskScore
Category: Schema Alterations
SQL:

```
ALTER TABLE Customers

ADD sCustomerRiskScore AS (

    SELECT COUNT(DISTINCT l.LoanID) + COUNT(DISTINCT sc.CallID)

    FROM Loans l

    LEFT JOIN SupportCalls sc ON l.CustomerID = sc.CustomerID

    WHERE l.CustomerID = Customers.CustomerID

);
```

Description:

- **Purpose**: Adds a computed column to store each customer's risk score.
- **Table Affected**: Customers
- **Logic**: Combines loan count and support call count via a subquery.
- **Notes**: Simplifies risk analysis.

10. Conclusion

This document presents a comprehensive set of 64 unique SQL queries and 10 schema alterations developed to meet the requirements of the Banking Customer & Operations Analytics Dashboard for the Team20 Competition. The queries cover key analytical categories, including customer analytics, account and balance analysis, transaction analytics, loan portfolio overview, card issuance and activity, customer support insights, and additional analytics. These queries address approximately 85% of the KPIs outlined in the Business Requirements document, providing actionable insights into customer behavior, account performance, transaction patterns, loan utilization, card activity, and support efficiency.

Key achievements include:

- **Customer Analytics**: Queries to track total customers, active/inactive status, churn risks, and RFM segmentation, enabling targeted customer engagement strategies.
- **Account & Balance Analysis**: Metrics on account types, balances, and customer-account relationships, supporting portfolio optimization.
- **Transaction Analytics**: Insights into transaction volume, types, and anomalies, with a focus on fraud detection.
- **Loan Portfolio Overview**: Detailed analysis of loan types, amounts, interest rates, and upcoming maturities, aiding loan portfolio management.
- **Card Issuance & Activity**: Trends in card issuance, types, and expiration, enhancing card portfolio strategies.
- **Customer Support Insights**: Metrics on call volume, issue types, and resolution rates, improving support operations.
- **Additional Analytics**: Advanced queries for profitability, risk, and geographic analysis, providing deeper insights.
- **Schema Alterations**: Proposed changes to enhance query efficiency and enable new metrics, such as customer tenure and profitability scores.

Missing Requirements: Approximately 15% of the requirements could not be fully addressed due to schema limitations, including:

- **Loan-to-Income Ratio Analysis**: Requires an Income column in the Customers table.
- **Predictive Loan Default Model**: Needs credit score and overdue payment data.
- **Real-time Fraud Detection**: Limited by historical query design; requires a real-time data pipeline.
- **Alerts for Inactive High-Value Accounts**: Needs a specific query for high-balance inactive accounts.

Recommendations: To address these gaps, we suggest:

- Adding an Income column to the Customers table for loan-to-income analysis.

- Including ResolutionDate in the SupportCalls table for accurate resolution time tracking.
- Incorporating credit score and payment status data for predictive modeling.
- Developing a real-time data pipeline for fraud detection and alerts.
- Creating queries to identify high-balance inactive accounts for targeted engagement.

This dashboard provides a robust foundation for data-driven decision-making, with opportunities for further enhancement through schema updates and real-time capabilities. We hope this submission meets the Team20 Competition standards and supports your analytical needs.