Project Title: Data Exploration with Azure SQL Database – Customer, Account, and Loan Feeds

Objective:

To explore and manipulate multiple related datasets using Azure SQL Database. The exploration is based on organizing the datasets, identifying data types, and exploring their relationships and contents.

Tools Required:

- Azure SQL Database
- SQL Management tools (Azure Data Studio or SQL Server Management Studio)
- Dataset - https://kaggle.com/datasets/9234c6c4d25b6eb7c3dbb15a0e33d65ae68a405d42acba8db1248defee7aff9c
- GitHub

Project steps:

1.Setting up Azure SQL Database

Step 1.1: Creating an Azure SQL Database in the Azure portal.

- Signed into the azure portal with your Azure account credentials.
- In the Azure Portal, clicked on **"Create a resource"** in the top left corner.
- In the "Search the Marketplace" box, typed **"SQL Database"** and selected **"SQL Database"** from the results.
- **Subscription**: Selected the Azure subscription you want to use.
- **Resource Group**: Choose an existing resource group or create a new one by clicking on **"Create new"**.
- **Database Name**: Entered the name of database as “CustomerAccountLoanDB”.
- **Server**:
- Did not have an existing SQL server, clicked on **"Create new"**.
- Fill in the server details:
    - **Server Name**: Entered a unique name as “sqlserver”
    - **Server Admin Login**: Chosen a username for the server admin.
    - **Password**: Entered and confirmed a password.
    - **Location**: Selected eastern Canada as the region for server.
- **Backup Storage Redundancy**: Selected local-redundant (LRS as storage redundancy.
- Clicked on **"Review + create"** and created a database for exploration.

Step 1.2: Connecting the Azure SQL Database to SSMS (SQL Server Management Studio).

Before connecting, ensure you have the following information:

- **Server Name**: This will typically be in the format "`your-server-name.database.windows.net`".
- **Database Name**: The name of Azure SQL Database i.e., "CustomerAccountLoanDB`.
- **Username**: The SQL server admin username.
- **Password**: The password associated with the SQL server admin account.
- Launch SSMS on your computer.
- In the **Connect to Server** dialog box, fill in the following details:
- **Server type**: Select **Database Engine**.
- **Server name**: Enter your Azure SQL Database server name (e.g., `your-server-name.database.windows.net`).
- **Authentication**: Choose **SQL Server Authentication**.
- **Login**: Enter your admin username.
- **Password**: Enter your admin password.
- **Connection Properties**: Here, you can specify the initial database if needed.
- **Network Protocol**: Ensure the protocol is set to **TCP/IP** (this is usually default).
- Click **Connect** to establish a connection to your Azure SQL Database.
- Now Azure SQL DB is successfully connected to SSMS.

2.Data Organization

Step 2.1: Created tables for the provided feeds.

- Customer Feed: Using the below query to create table for customers.

```
CREATE TABLE dbo.customers (
customer_id INT PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
address VARCHAR(100),
city VARCHAR(50),
state VARCHAR(50),
zip VARCHAR(20)
);
```

- Account Feed: Using the below query to create table for Accounts created by customers.

```
CREATE TABLE dbo.accounts (
        account_id INT PRIMARY KEY,
```

```
    customer_id INT,
    account_type VARCHAR(50),
    balance DECIMAL(10, 2),
    FOREIGN KEY (customer_id) REFERENCES  customers(customer_id)
);
```

- Transaction Feed: Using the below query to create table for transactions made on accounts.

```
CREATE TABLE dbo.transactions (
    transaction_id INT PRIMARY KEY,
    account_id INT,
    transaction_date DATE,
    transaction_amount DECIMAL(10, 2),
    transaction_type VARCHAR(50),
    FOREIGN KEY (account_id) REFERENCES accounts(account_id)
    );
```

- Loan Feed: Using the below query to create table for the loans taken by customers.

```
CREATE TABLE dbo.loans (
    loan_id INT PRIMARY KEY,
    customer_id INT,
    loan_amount DECIMAL (10, 2),
    interest_rate DECIMAL (5, 2),
    loan_term INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

- Loan Payment Feed: Using the below query to create table for the loan payments made by customers.

```
CREATE TABLE dbo.loan_payments (
    payment_id INT PRIMARY KEY,
    loan_id INT,
    payment_date DATE,
    payment_amount DECIMAL(10, 2),
    FOREIGN KEY (loan_id) REFERENCES loans(loan_id)
);
```

3. Data Insertion

Step 3.1: Populate tables with sample data using `INSERT INTO` statements for each table. Since the data is high in volume, I have just inserted using SSMS inbuilt features to import data. Steps I followed to import data from external flat files to tables in Azure SQL Database which is connected SSMS.

- Launch SSMS and connect to your Azure SQL Database.

- Right-click on the database (e.g., `customeraccountloanDB`).
- Select **Tasks** > **Import Data**.
- The SQL Server Import and Export Wizard will open.
- In the wizard, select **Flat File Source** as the data source. The flat files data is available as CSV format in the Kaggle datasets which is provided on the top of this document in Tools required section.
- Browse to the CSV file you saved earlier.
- Set the format options (usually default settings work fine).
- The wizard will show a mapping screen. Ensure that the columns from the CSV file map correctly to the columns in the `customers` table.
- Complete the wizard steps, review your selections, and click **Finish** to run the import process.

This way data will be imported to the associated columns into the tables which were defined in the "CustomerAccountLoanDB".

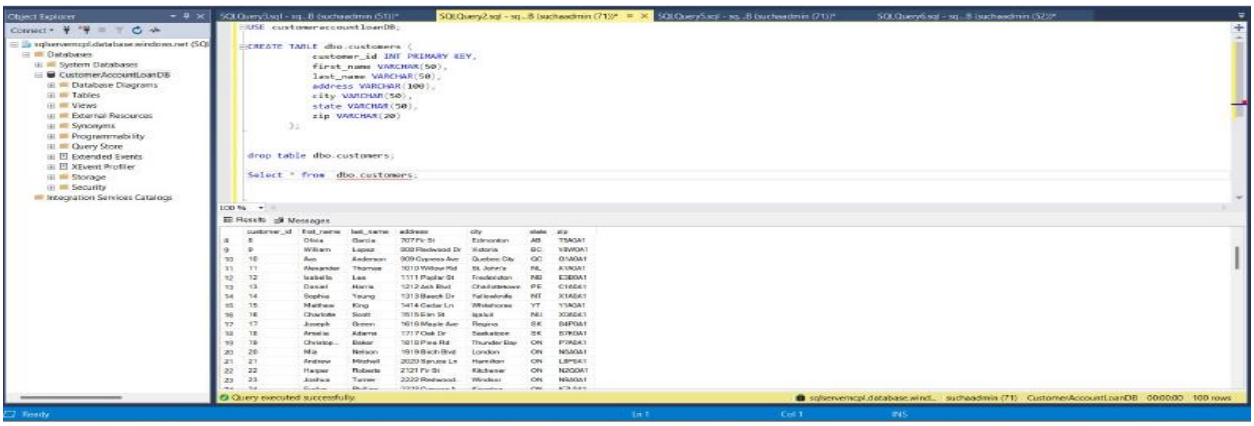
Step 3.2: Ensure data consistency and relationships, ensuring each foreign key points to valid primary keys.

- I got an error while importing data when importing the data for second table accounts stating that the tables primary key " customer_id" is depended on the foreign key of customer_id column in customers table.
- Customer_id column records in the customers table does not match with the record count of customer_id column in the accounts table.
- I resolved this error by inserting few more columns, until it matches the records with the accounts table.
- Again, inserted the records using the same import method as step 3.1.

4. Data Exploration

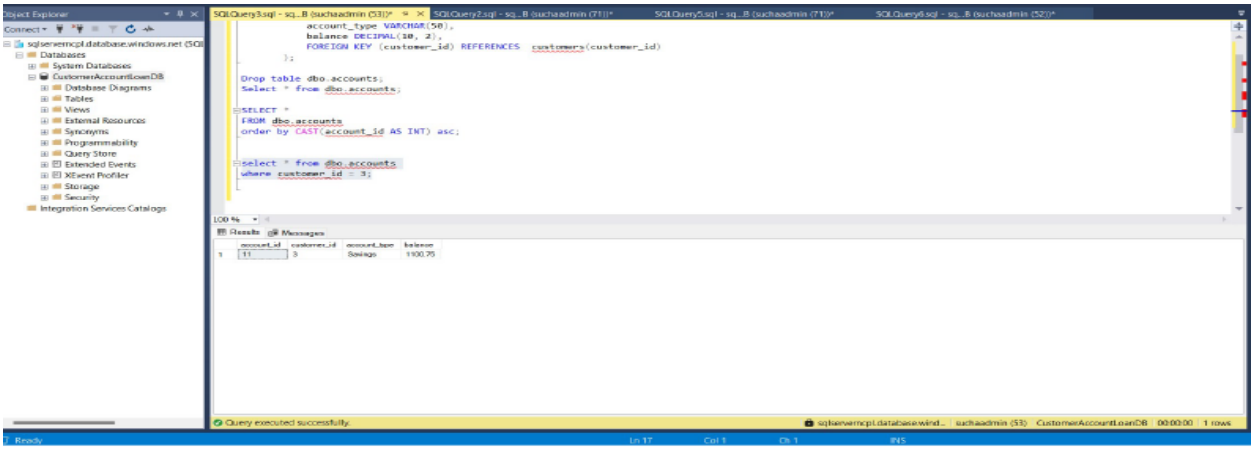
Step 4.1: Write query to retrieve all customer information.
Query:

```
Select * from  dbo.customers;
```



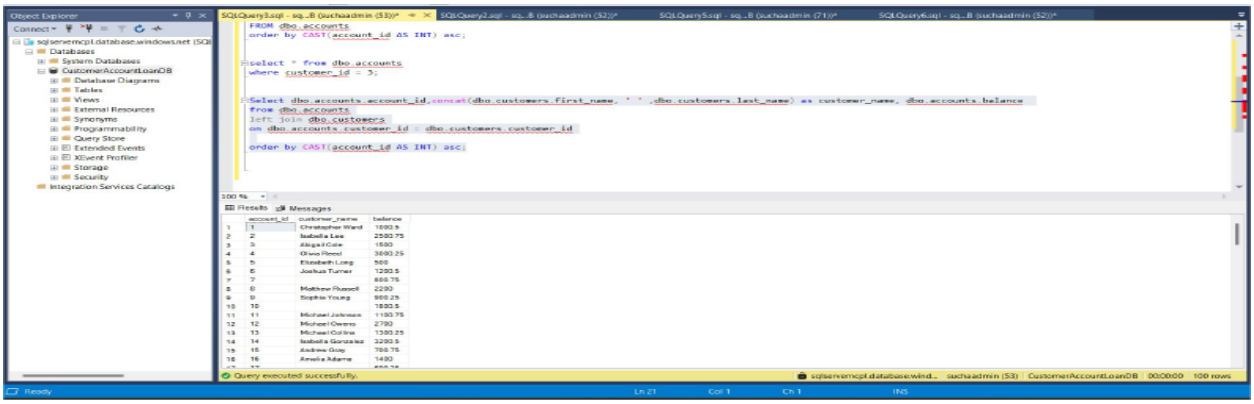
Step 4.2: Query accounts for a specific customer:
Query:

```
select * from dbo.accounts
where customer_id = 3;
```



Step 4.3: Find the customer name and account balance for each account.
Query:

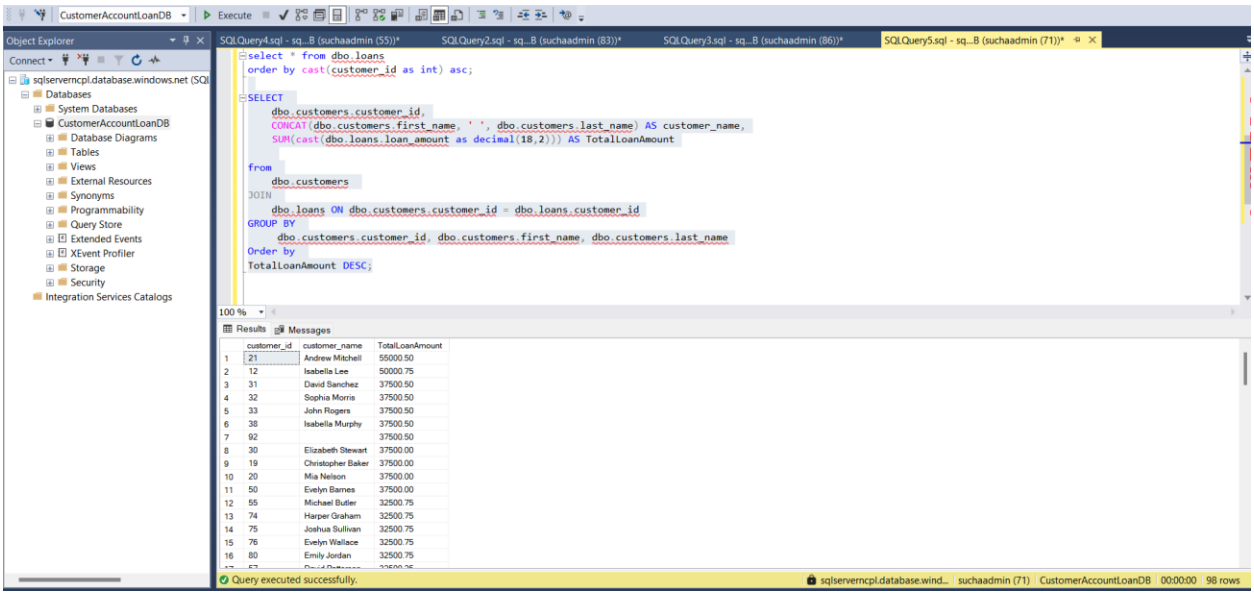
```
Select dbo.accounts.account_id,concat(dbo.customers.first_name, '
' ,dbo.customers.last_name) as customer_name, dbo.accounts.balance
from dbo.accounts
left join dbo.customers
on dbo.accounts.customer_id = dbo.customers.customer_id

order by CAST(account_id AS INT) asc;
```



Step 4.4: Analyze customer loan balances:
Query:

1) Total Loan Amounts and Outstanding Balances by Customer
   Query:

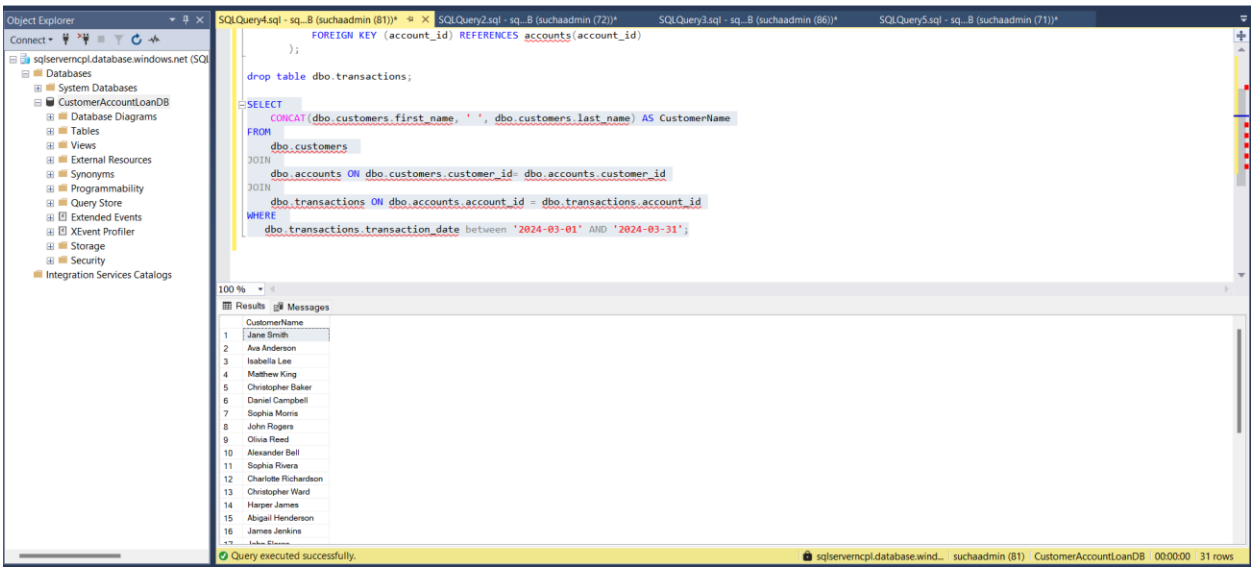
```
SELECT
    dbo.customers.customer_id,
    CONCAT(dbo.customers.first_name, ' ', dbo.customers.last_name) AS
customer_name,
SUM(cast(dbo.loans.loan_amount as decimal(18,2))) AS TotalLoanAmount

from
    dbo.customers
JOIN
    dbo.loans ON dbo.customers.customer_id = dbo.loans.customer_id
GROUP BY
     dbo.customers.customer_id, dbo.customers.first_name, dbo.customers.last_name
Order by
TotalLoanAmount DESC;
```



Step 4.5: List all customers who have made a transaction in the 2024-03.

Query:

```
SELECT
    CONCAT(dbo.customers.first_name, ' ', dbo.customers.last_name) AS CustomerName
FROM
    dbo.customers
JOIN
    dbo.accounts ON dbo.customers.customer_id= dbo.accounts.customer_id
JOIN
    dbo.transactions ON dbo.accounts.account_id = dbo.transactions.account_id
WHERE
   dbo.transactions.transaction_date between '2024-03-01' AND '2024-03-31';
```



5. Aggregation and Insights

Step 5.1: Calculate the total balance across all accounts for each customer:

Query:

select dbo.customers.customer_id, concat(dbo.customers.first_name ,' ', dbo.customers.last_name) as customer_name,

SUM(cast(dbo.accounts.balance as decimal(18,2))) as total_balance

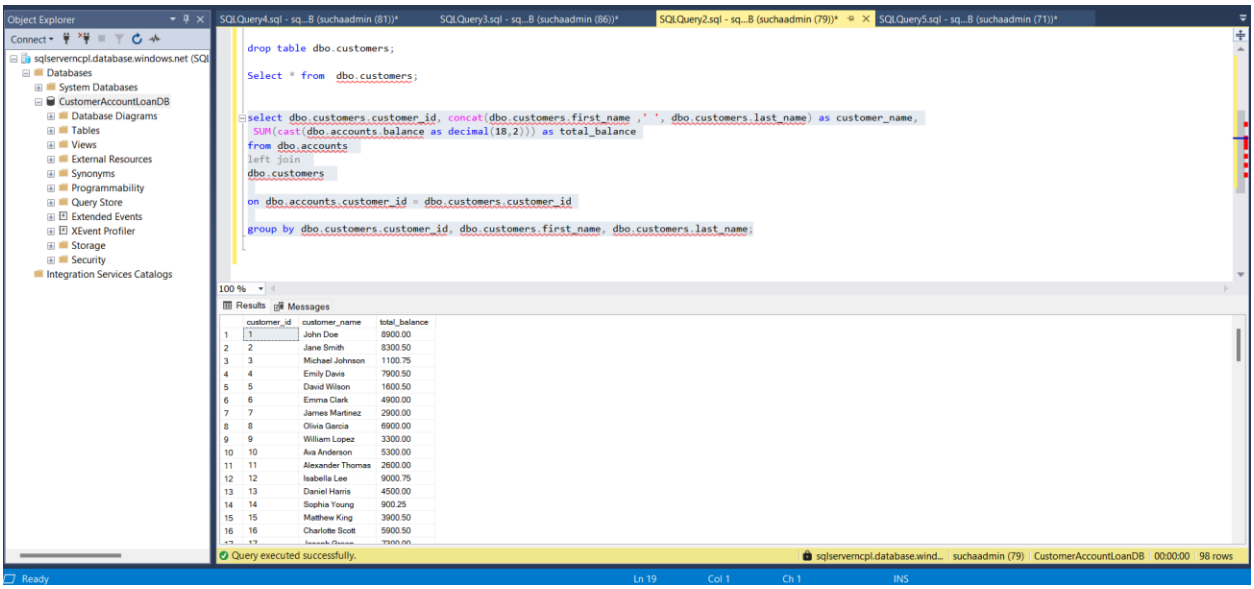
from dbo.accounts

left join

dbo.customers

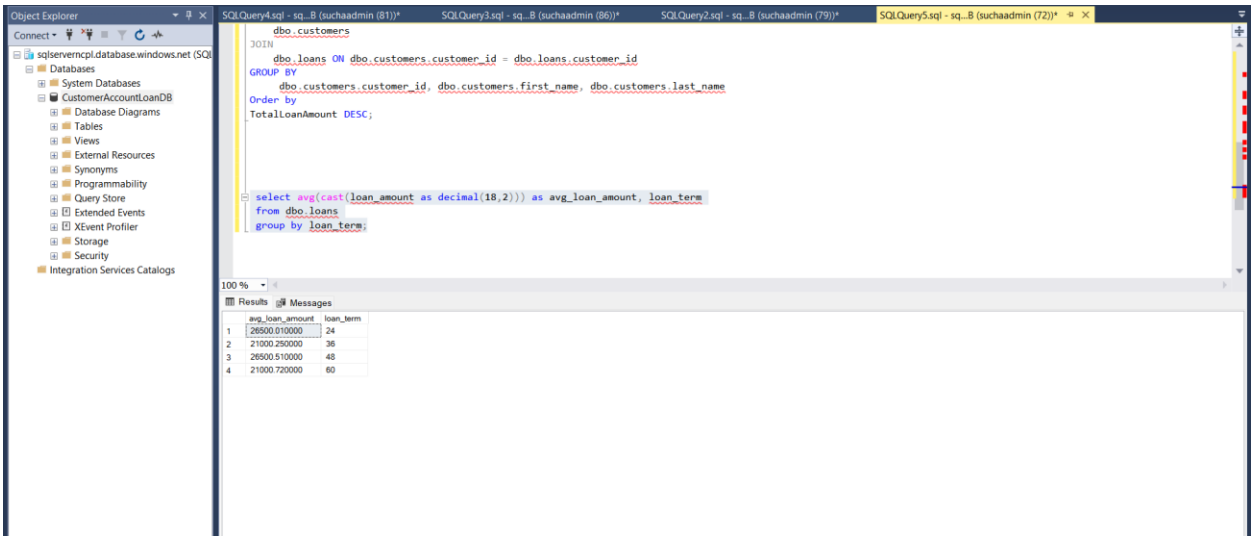
on dbo.accounts.customer_id = dbo.customers.customer_id

group by dbo.customers.customer_id, dbo.customers.first_name, dbo.customers.last_name;



5.2: Calculate the average loan amount for each loan term:

```
select avg(cast(loan_amount as decimal(18,2))) as avg_loan_amount, loan_term
from dbo.loans
group by loan_term;
```



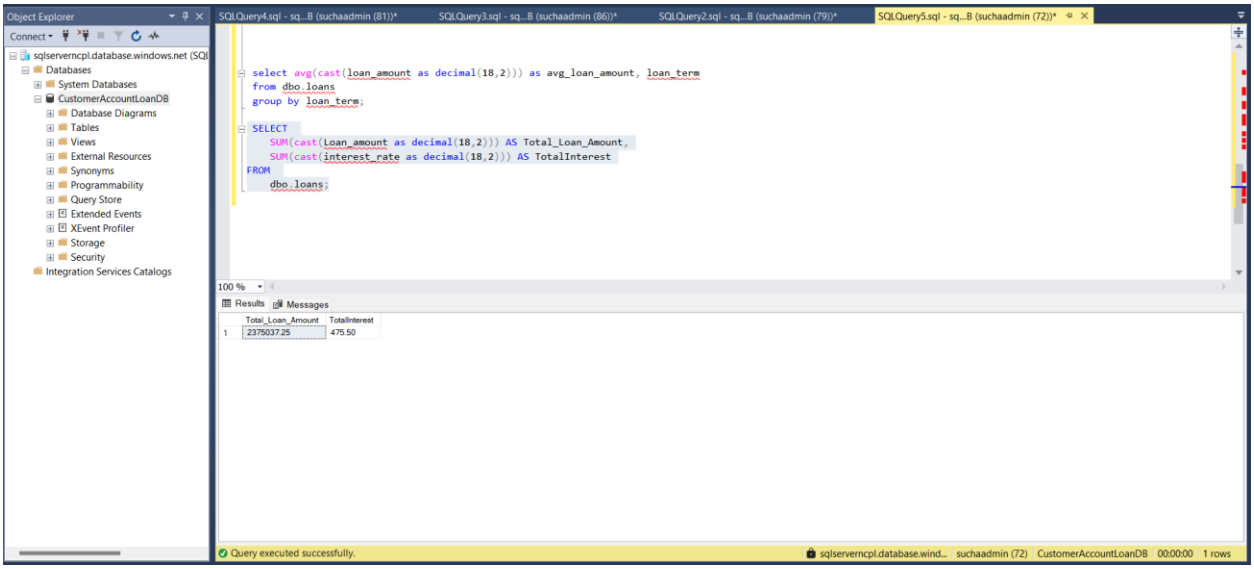
Step 5.3: Find the total loan amount and interest across all loans:

Query:

```
SELECT
    SUM(cast(Loan_amount as decimal(18,2))) AS Total_Loan_Amount,
    SUM(cast(interest_rate as decimal(18,2))) AS TotalInterest
```

FROM

dbo.loans;



Step 5.4: Find the most frequent transaction type.

Query:

SELECT

transaction_type,

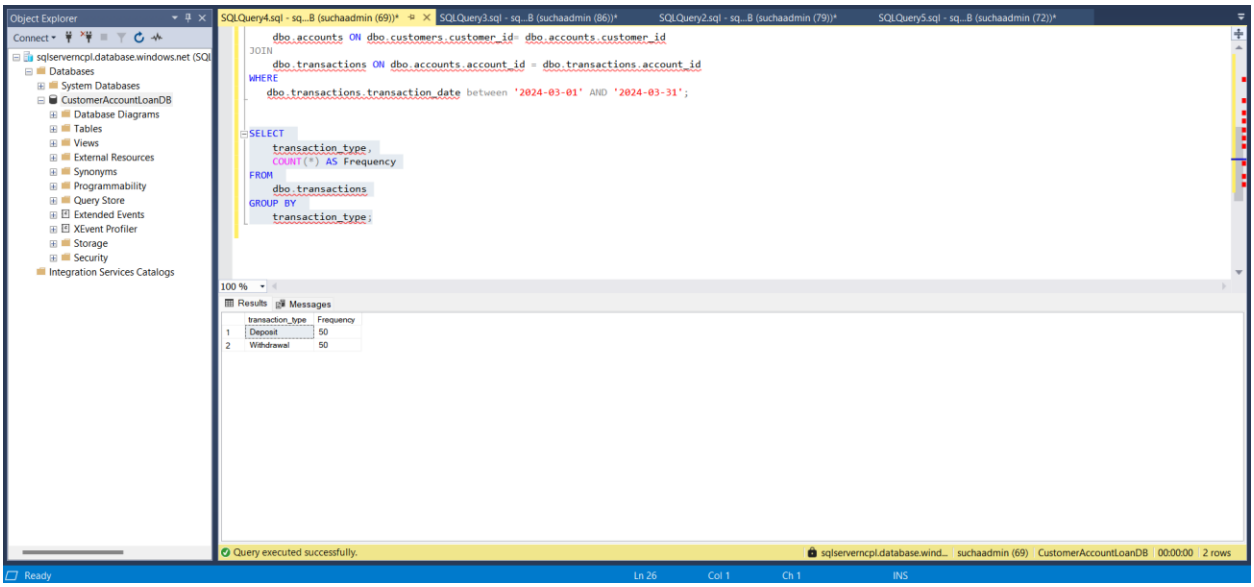
COUNT(*) AS Frequency

FROM

dbo.transactions

GROUP BY

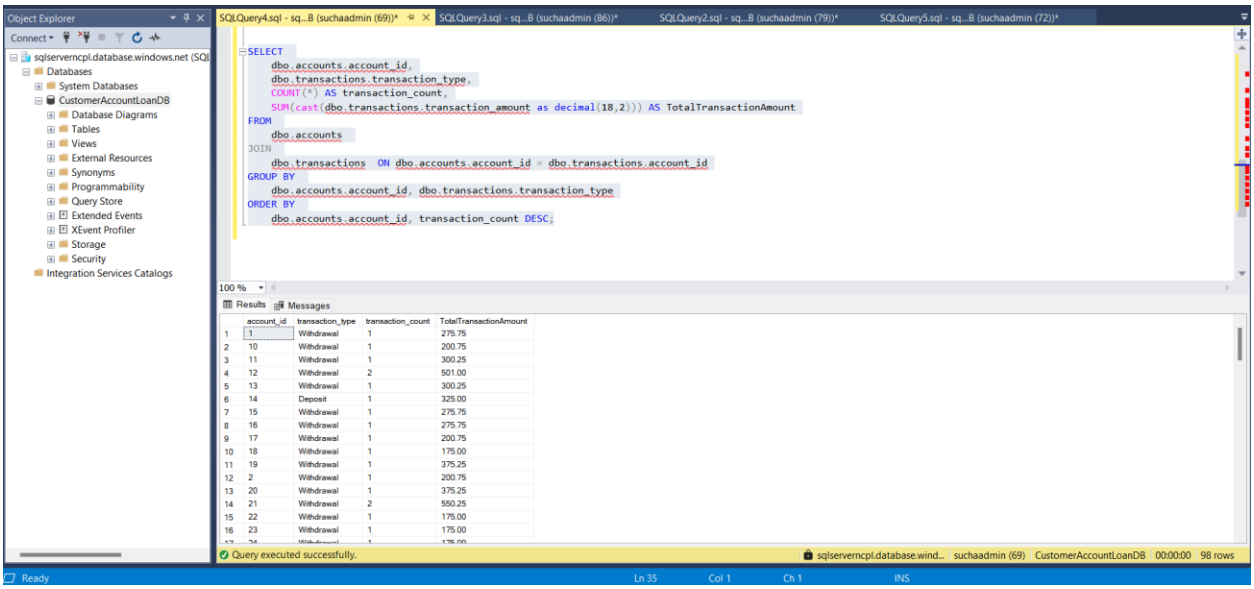
transaction_type;



Step 5.5: Analyze transactions by account and transaction type:

Query:

```
SELECT
    dbo.accounts.account_id,
    dbo.transactions.transaction_type,
    COUNT(*) AS transaction_count,
    SUM(cast(dbo.transactions.transaction_amount as decimal(18,2))) AS
TotalTransactionAmount
FROM
    dbo.accounts
JOIN
    dbo.transactions  ON dbo.accounts.account_id = dbo.transactions.account_id
GROUP BY
    dbo.accounts.account_id, dbo.transactions.transaction_type
ORDER BY
    dbo.accounts.account_id, transaction_count DESC;
```

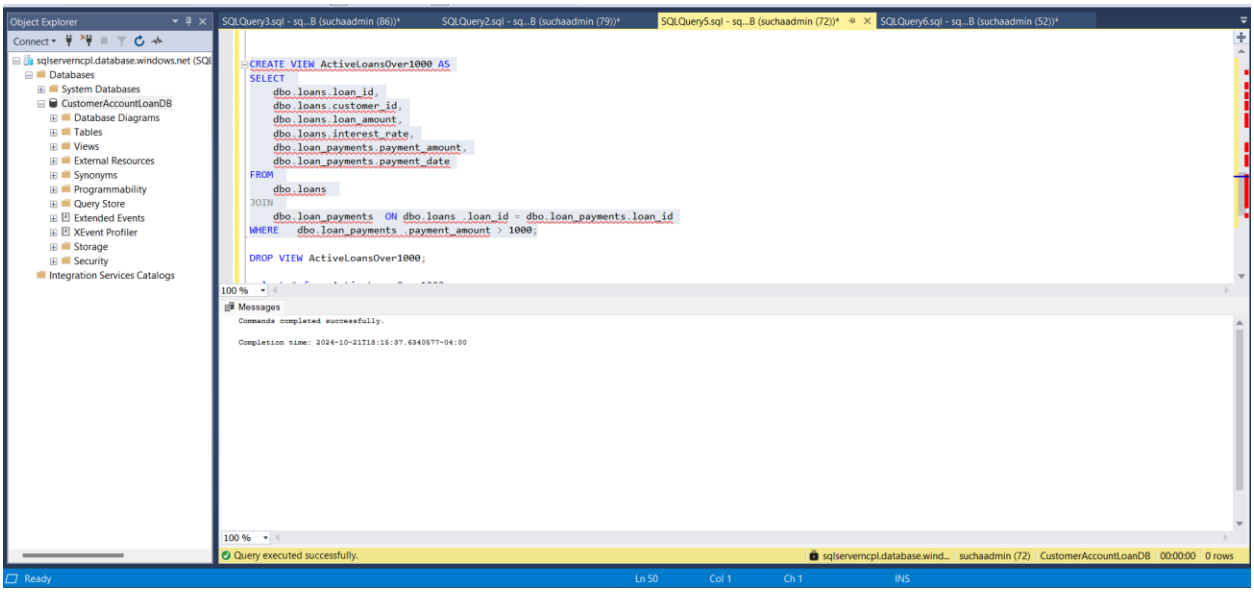


6. Advanced Analysis

Step 6.1: Create a view of active loans with payments greater than $1000

Query:

```
CREATE VIEW ActiveLoansOver1000 AS
SELECT
    dbo.loans.loan_id,
    dbo.loans.customer_id,
    dbo.loans.loan_amount,
    dbo.loans.interest_rate,
    dbo.loan_payments.payment_amount,
    dbo.loan_payments.payment_date
FROM
    dbo.loans
JOIN
    dbo.loan_payments  ON dbo.loans .loan_id = dbo.loan_payments.loan_id
WHERE   dbo.loan_payments .payment_amount > 1000;
```



Step 6.2: Create an index on `transaction_date` in the `transactions` table for performance optimization:

Query:

CREATE INDEX IX_Transaction_Date

ON transactions (transaction_date);