UPROSZCZONY SYSTEM BANKOWY

AUTORKI: Magdalena Maksymiuk, Kornelia Rozpara

Założenia projektu

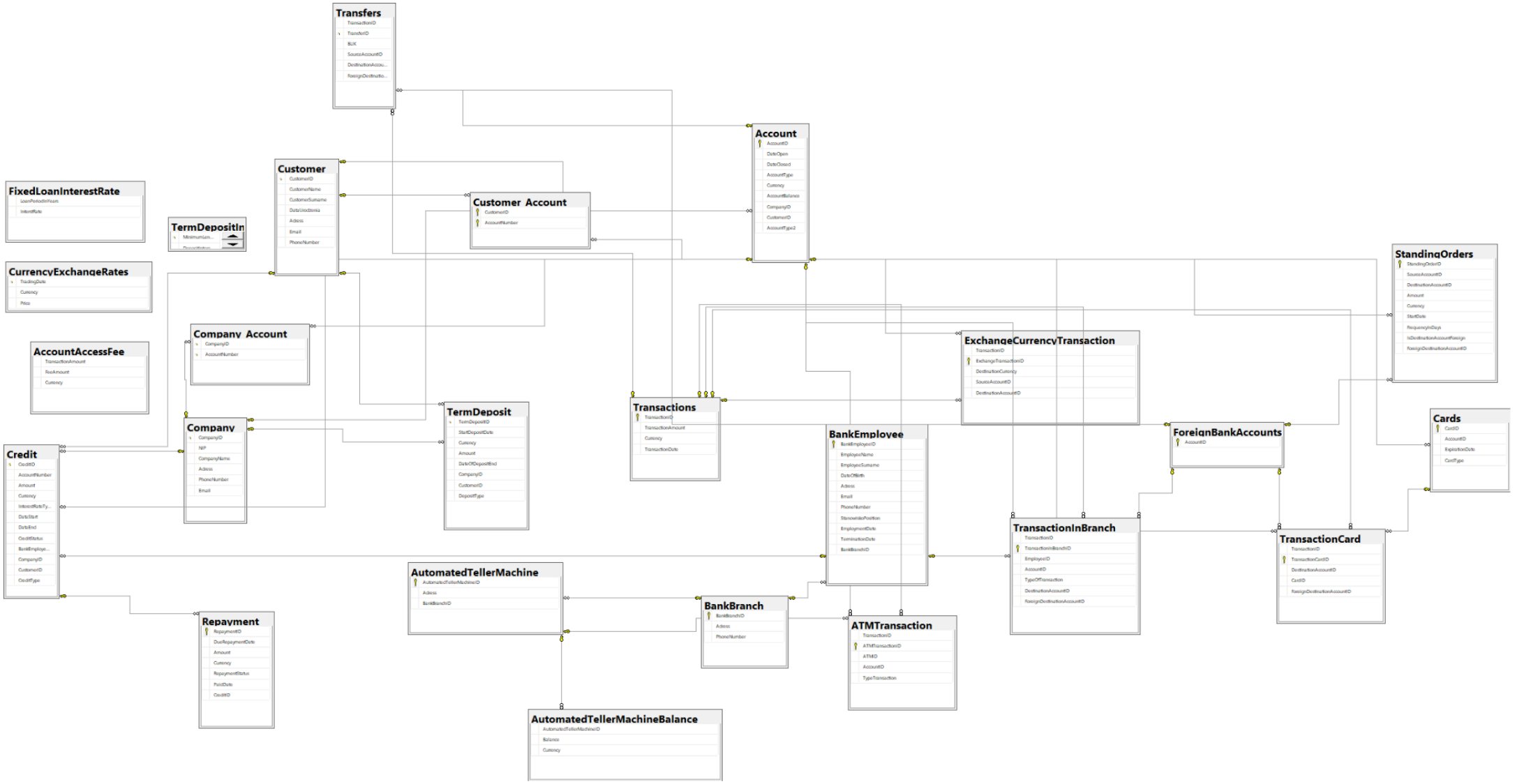
Projekt zakłada stworzenie uproszczonej bazy danych systemu bankowego, który pozwala na przetwarzanie transakcji bankowych(przelewów, wpłat/wypłat z konta, transakcji wykonanych kartą kredytową/debetową), możliwość zarządzania rachunkami bankowymi (tworzenie, modyfikacja i zamykanie kont), obsługę kredytów i rachunków oszczędnościowych oraz tworzenie raportów dotyczących finansów zarówno tych dotyczących klientów banku jak i samej placówki. Baza zakłada, że bank może udzielać jedynie kredytów o stałym oprocentowaniu, jednak w ramach rozszerzenia funkcjonalności bazy łatwo to rozwinąć.

Baza ma pozwalać na obsługę kont bankowych nie tylko w jednej walucie - co pozwala na wymianę walut w placówce oraz obsługę kart w obcych walutach.

Schemat pielęgnacji bazy danych

Bank, jako instytucja zajmująca się finansami musi mieć pewność, że dane przy awarii nie tylko zostaną utracone, ale również szybko przywrócone. Warto dlatego rozważyć tworzenie zarówno kopii różnicowych jak i pełnych. Dane w niektórych tabelach są częściej aktualizowane niż w innych dlatego przy tworzeniu kopii zapasowej trzeba uwzględnić ten fakt. Trzeba ustalić również jak długo przechowywane są historyczne dane ze względu na to, jak ważną instytucją jest bank. Zaleca się również nie tylko automatyzację procesu tworzenia kopii zapasowych, ale również testowanie procedur przywracania aby upewnić się, że dane są skutecznie przywracane w razie awarii.

Schemat bazy danych



Tabele

Tabela ForeignBankAccounts: Ta tabela przechowuje informacje kontach bankowych spoza naszego banku, jeżeli prześlemy pieniądze na takie konto znikają w naszym banku i zajmuje się nimi bank obcy.

```
CREATE TABLE ForeignBankAccounts (
   AccountID INT NOT NULL PRIMARY KEY
);
```

Tabela AccountAccessFee: Tabela AccountAccessFee definiuje opłaty za transakcje.

```
CREATE TABLE AccountAccessFee (
   TransactionAmount INT,
   FeeAmount DECIMAL(10, 2) NOT NULL,
   Currency VARCHAR(3) NOT NULL
);
```

Tabela BankBranch: Tabela BankBranch przechowuje szczegóły dotyczące oddziałów bankowych.

```
CREATE TABLE BankBranch (
   BankBranchID INT PRIMARY KEY,
   Adress VARCHAR(255) NOT NULL,
   PhoneNumber VARCHAR(20),
);
```

Tabela Customer: Tabela Klient przechowuje dane o klientach indywidualnych.

```
CREATE TABLE Customer (
   CustomerID INT PRIMARY KEY,
   CustomerName VARCHAR(50)  NOT NULL,
   CustomerSurname VARCHAR(50)  NOT NULL,
   DataUrodzenia DATE  NOT NULL,
   Adress VARCHAR(255)  NOT NULL,
   Email VARCHAR(100),
   PhoneNumber VARCHAR(20),

);
```

Tabela Company: Tabela Firma zawiera dane dotyczące podmiotów korporacyjnych, które są klientem w naszym banku.

```
CREATE TABLE Company (

   CompanyID INT PRIMARY KEY,
   NIP VARCHAR(20) NOT NULL,
   CompanyName VARCHAR(100) NOT NULL,
   Adress VARCHAR(255) NOT NULL,
   PhoneNumber VARCHAR(20),
   Email VARCHAR(100),

);
```

Tabela Customer_Account: Ta tabela ustanawia relację wiele do wielu między klientami a kontami.

```
CREATE TABLE Customer_Account (
   CustomerID INT,
   AccountNumber VARCHAR(40),
   PRIMARY KEY (CustomerID, AccountNumber),
   FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
   FOREIGN KEY (AccountNumber) REFERENCES Account(AccountID)
);
```

Tabela Account: Konto dla klientów indywidualnych i korporacyjnych w ustalonej walucie. Może być oszczędnościowe (bez oprocentowania) lub zwykłe.

```
CREATE TABLE Account (
   AccountID VARCHAR(40) PRIMARY KEY,
   DateOpen DATE  NOT NULL,
   DateClosed DATE,
   AccountType VARCHAR(20) CHECK (AccountType IN ('SavingAccount', 'StandardAccount'))
NOT NULL,
   Currency VARCHAR(3) NOT NULL,
   AccountBalance DECIMAL(15, 2) NOT NULL,
   CompanyID INT,
   CustomerID INT,
   AccountType2 VARCHAR(20) CHECK (AccountType2 IN ('CUSTOMER_ACCOUNT',
'COMPANY_ACCOUNT'))  NOT NULL,
   FOREIGN KEY(CompanyID) REFERENCES Company(CompanyID),
   FOREIGN KEY(CustomerID) REFERENCES Customer(CustomerID)

);
```

Tabela BankEmployee: Tabela BankEmployee zawiera informacje o pracownikach banku.

```
CREATE TABLE BankEmployee (
   BankEmployeeID INT PRIMARY KEY,
   EmployeeName VARCHAR(50) NOT NULL,
   EmployeeSurname VARCHAR(50) NOT NULL,
   DateOfBirth DATE NOT NULL,
   Adress VARCHAR(255) NOT NULL,
   Email VARCHAR(100),
   PhoneNumber VARCHAR(20),
   StanowiskoPosition VARCHAR(100) NOT NULL,
   EmploymentDate DATE NOT NULL,
   TerminationDate DATE DEFAULT NULL,
   BankBranchID INT NOT NULL,
   FOREIGN KEY (BankBranchID) REFERENCES BankBranch(BankBranchID)
);
```

Tabela Company_Account: Podobnie jak Customer_Account, ta tabela ułatwia relację wiele do wielu między firmami a kontami.

```
CREATE TABLE Company_Account (
   CompanyID INT,
   AccountNumber VARCHAR(40),
   PRIMARY KEY (CompanyID, AccountNumber),
   FOREIGN KEY (CompanyID) REFERENCES Company(CompanyID),
   FOREIGN KEY (AccountNumber) REFERENCES Account(AccountID)
);
```

Tabela Credit: Ta tabela rejestruje informacje o kredytach udzielanych klientom lub firmom.

```
CREATE TABLE Credit (
   CreditID INT PRIMARY KEY,
   AccountNumber VARCHAR(40) NOT NULL,
   Amount INT NOT NULL,
   Currency VARCHAR(3) NOT NULL,
   InterestRateType VARCHAR(20) CHECK (InterestRateType IN ('Fixed')) NOT NULL,
   DataStart DATE NOT NULL,
   DataEnd DATE NOT NULL,
   CreditStatus VARCHAR(20) CHECK (CreditStatus IN ('Active', 'RepaidLoan')) NOT NULL,
   BankEmployeeID INT NOT NULL,
   CompanyID INT,
   CustomerID INT,
   CreditType VARCHAR(20) CHECK (CreditType IN ('CUSTOMER_CREDIT', 'COMPANY_CREDIT'))
NOT NULL,
   FOREIGN KEY(CompanyID) REFERENCES Company(CompanyID),
   FOREIGN KEY(CustomerID) REFERENCES Customer(CustomerID),
   FOREIGN KEY (BankEmployeeID) REFERENCES BankEmployee(BankEmployeeID),
   FOREIGN KEY (AccountNumber) REFERENCES Account(AccountID),

);
```

Tabela Repayment: Tabela Repayment zawiera wszystkie raty danego kredytu oraz między innymi czy dana rata jest już spłacona.

```
CREATE TABLE Repayment (
   RepaymentID INT PRIMARY KEY IDENTITY(1,1),
   DueRepaymentDate DATE NOT NULL,
   Amount DECIMAL(15, 2) NOT NULL,
   Currency VARCHAR(3) NOT NULL,
   RepaymentStatus VARCHAR(50) CHECK (RepaymentStatus IN ('Paid', 'Unpaid')) NOT NULL,
   PaidDate DATE DEFAULT NULL,
   CreditID INT NOT NULL,
   FOREIGN KEY (CreditID) REFERENCES Credit(CreditID)
);
```

Tabela FixedLoanInterestRate: Ta tabela przechowuje ustalone stopy procentowe dla pożyczek na stałym oprocentowaniu.

```
CREATE TABLE FixedLoanInterestRate (
   LoanPeriodInYears INT NOT NULL,
   InterstRate DECIMAL(5, 2)
);
```

Tabela AutomatedTellerMachine: Tabela AutomatedTellerMachine zawiera informacje o bankomatach.

```
CREATE TABLE AutomatedTellerMachine (
   AutomatedTellerMachineID INT PRIMARY KEY,
   Adress VARCHAR(255) NOT NULL,
   BankBranchID INT NOT NULL,
   FOREIGN KEY (BankBranchID) REFERENCES BankBranch(BankBranchID)
);
```

Tabela AutomatedTellerMachineBalance:
Ta tabela rejestruje informacje o saldach bankomatów, bankomat może mieścić różne waluty w przeciwieństwie do konta.

```
CREATE TABLE AutomatedTellerMachineBalance (

   AutomatedTellerMachineID INT,
   Balance INT CHECK (Balance >= 0) NOT NULL,
   Currency VARCHAR(3) NOT NULL,
   FOREIGN KEY (AutomatedTellerMachineID) REFERENCES
AutomatedTellerMachine(AutomatedTellerMachineID)
);
```

Tabela Cards:
Tabela Cards zawiera szczegóły dotyczące kart bankowych.

```
CREATE TABLE Cards (

   CardID INT PRIMARY KEY,
   AccountID VARCHAR(40) NOT NULL,
   ExpirationDate DATE NOT NULL,
   CardType VARCHAR(50) CHECK (CardType IN ('Debit', 'Credit')) NOT NULL,
   FOREIGN KEY (AccountID) REFERENCES Account(AccountID)
);
```

Tabela Currency Exchange Rates:
Tabela CurrencyExchangeRates przechowuje kursy wymiany walut na określone daty handlowe.

```
CREATE TABLE CurrencyExchangeRates (
   TradingDate DATE PRIMARY KEY,
   Currency VARCHAR(3) NOT NULL,
   Price DECIMAL(10, 6) NOT NULL
);
```

Tabela TermDeposit: Tabela TermDeposit przechowuje informacje o lokatach terminowych.

```
CREATE TABLE TermDeposit (

   TermDepositID INT PRIMARY KEY,
   StartDepositDate DATE NOT NULL,
   Currency VARCHAR(3) NOT NULL,
   Amount DECIMAL(15, 2) NOT NULL,
   DateOfDepositEnd DATE NOT NULL,
   CompanyID INT,
   CustomerID INT,
   DepositType VARCHAR(20) CHECK (DepositType IN ('CUSTOMER', 'COMPANY'))  NOT NULL,
   FOREIGN KEY(CompanyID) REFERENCES Company(CompanyID),
   FOREIGN KEY(CustomerID) REFERENCES Customer(CustomerID)
);
```

Tabela TermDepositInterstRates: Ta tabela definiuje stopy procentowe dla lokat terminowych w zależności od okresu w latach lokaty.

```
CREATE TABLE TermDepositInterstRates (
   MinimumLengthInYears INT PRIMARY KEY,
   DepositInterestRates DECIMAL(5, 2) NOT NULL
);
```

Tabela StandingOrders: Tabela StandingOrders obsługuje zlecenia stałe dla regularnych transakcji.

```
CREATE TABLE StandingOrders (
   StandingOrderID INT PRIMARY KEY,
   SourceAccountID VARCHAR(40) NOT NULL,
   DestinationAccountID VARCHAR(40),
   Amount DECIMAL(15, 2) NOT NULL,
   Currency VARCHAR(3) NOT NULL,
   StartDate DATE NOT NULL,
   FrequencyInDays INT NOT NULL,
   IsDestinationAccountForeign VARCHAR(3) NOT NULL CHECK (IsDestinationAccountForeign IN
('Yes', 'No')),
   ForeignDestinationAccountID INT,
   FOREIGN KEY(ForeignDestinationAccountID) REFERENCES ForeignBankAccounts(AccountID) ,
   FOREIGN KEY (SourceAccountID) REFERENCES Account(AccountID) ,
   FOREIGN KEY (DestinationAccountID) REFERENCES Account(AccountID)
);
```

Tabela Transactions: Tabela Transactions rejestruje ogólne informacje o transakcjach, po tej tabeli dziedziczą różne typy transakcji które uszczegóławiają informacje o transakcjach.

```
CREATE TABLE Transactions (
   TransactionID INT PRIMARY KEY IDENTITY(1,1),
   TransactionAmount DECIMAL(15, 2) NOT NULL,
   Currency VARCHAR(3) NOT NULL,
   TransactionDate DATETIME NOT NULL
);
```

Tabela TransactionCard: Ta tabela zawiera szczegóły dotyczące transakcji kartowych. Transakcja kartą może być wykonana na konto z naszego banku lub na konto obce.

```
CREATE TABLE TransactionCard (
   TransactionID INT NOT NULL,
   TransactionCardID INT PRIMARY KEY  IDENTITY(1,1),
   DestinationAccountID VARCHAR(40),
   CardID INT NOT NULL,
   IsDestinationAccountForeign VARCHAR(3) CHECK (IsDestinationAccountForeign IN ('Yes',
'No')) NOT NULL,
   ForeignDestinationAccountID INT,
   FOREIGN KEY(ForeignDestinationAccountID) REFERENCES ForeignBankAccounts(AccountID) ,
   FOREIGN KEY (CardID) REFERENCES Cards(CardID) ,
   FOREIGN KEY (DestinationAccountID) REFERENCES Account(AccountID) ,
   FOREIGN KEY (TransactionID) REFERENCES Transactions(TransactionID)
);
```

Tabela ExchangeCurrencyTransaction: Tabela ExchangeCurrencyTransaction obsługuje transakcje wymiany walut. Do wymiany walut potrzebujemy odrębnego konta w innej docelowej walucie aby tam przesłać wymienione środki, a usunąć je z konta pierwotnego.

```
CREATE TABLE ExchangeCurrencyTransaction (
   TransactionID INT NOT NULL,
   ExchangeTransactionID INT PRIMARY KEY  IDENTITY(1,1),
   DestinationCurrency VARCHAR(3) NOT NULL,
   SourceAccountID VARCHAR(40) NOT NULL,
   DestinationAccountID VARCHAR(40) NOT NULL,
   FOREIGN KEY (SourceAccountID) REFERENCES Account(AccountID),
   FOREIGN KEY (DestinationAccountID) REFERENCES Account(AccountID),
   FOREIGN KEY (TransactionID) REFERENCES Transactions(TransactionID)
);
```

Tabela ATMTransaction:
Ta tabela zawiera informacje o transakcjach dokonywanych w bankomatach, wpłatach i wypłatach pieniędzy z konta w naszym banku. Konta z obcych banków nie mogą korzystać z naszych bankomatów.

```
CREATE TABLE ATMTransaction (
   TransactionID INT NOT NULL,
   ATMTransactionID INT PRIMARY KEY  IDENTITY(1,1),
   ATMID INT NOT NULL,
   AccountID VARCHAR(40) NOT NULL,
   TypeTransaction VARCHAR(20) CHECK (TypeTransaction IN ('MakeDeposit',
'MakeWithdrawal')),
   FOREIGN KEY (ATMID) REFERENCES AutomatedTellerMachine(AutomatedTellerMachineID),
   FOREIGN KEY (AccountID) REFERENCES Account(AccountID),
   FOREIGN KEY (TransactionID) REFERENCES Transactions(TransactionID)
);
```

Tabela TransactionInBranch: Ta tabela przechowuje transakcje przeprowadzane w oddziałach banku. Możliwa jest wpłata pieniędzy na swoje konto, wypłata, oraz zrobienie przelewu do naszego banku jak i klientów obcych banków.

```
CREATE TABLE TransactionInBranch (
   TransactionID INT NOT NULL,
   TransactionInBranchID INT PRIMARY KEY  IDENTITY(1,1),
   EmployeeID INT NOT NULL,
   AccountID VARCHAR(40) NOT NULL,
   TypeOfTransaction VARCHAR(50) CHECK (TypeOfTransaction IN ('Wpłata',
'Wypłata','Przelew','PrzelewNaKontoObce')) NOT NULL,
   DestinationAccountID VARCHAR(40),
   IsDestinationAccountForeign VARCHAR(3) CHECK (IsDestinationAccountForeign IN ('Yes',
'No','-')) NOT NULL,
   ForeignDestinationAccountID INT,
   FOREIGN KEY(ForeignDestinationAccountID) REFERENCES ForeignBankAccounts(AccountID) ,
   FOREIGN KEY (EmployeeID) REFERENCES BankEmployee(BankEmployeeID),
   FOREIGN KEY (AccountID) REFERENCES Account(AccountID) ,
   FOREIGN KEY (TransactionID) REFERENCES Transactions(TransactionID),
   FOREIGN KEY (DestinationAccountID) REFERENCES Account(AccountID)
);
```

Tabela Transfers: Tabela Transfers rejestruje transakcje przelewów zarówno na konto w naszym banku jak i na konta obce.

```
CREATE TABLE Transfers (
   TransactionID INT NOT NULL,
   TransferID INT PRIMARY KEY IDENTITY(1,1),
   BLIK VARCHAR(10),
   SourceAccountID VARCHAR(40) NOT NULL,
   DestinationAccountID VARCHAR(40),
   IsDestinationAccountForeign VARCHAR(3) CHECK (IsDestinationAccountForeign IN ('Yes',
'No')) NOT NULL,
   ForeignDestinationAccountID INT,
   FOREIGN KEY(ForeignDestinationAccountID) REFERENCES ForeignBankAccounts(AccountID) ,
   FOREIGN KEY (SourceAccountID) REFERENCES Account(AccountID) ,
   FOREIGN KEY (DestinationAccountID) REFERENCES Account(AccountID) ,
   FOREIGN KEY (TransactionID) REFERENCES Transactions(TransactionID)
);
```

Widoki/Funkcje

1) Funkcja wyświetlająca wszystkie konta danego klienta oraz salda i ich walutę.

```
go

CREATE FUNCTION Show_accounts_balances (@ClientId INT)
RETURNS TABLE
AS
    RETURN
      SELECT Account.Accountid,
             Account.Currency,
             AccountBalance
      FROM   Account
      WHERE  Account.CustomerID = @ClientID;

go
```

2) Widok wyświetlający liczbę kont danego klienta

```
CREATE VIEW number_of_accounts
AS
  SELECT Customer.CustomerID,
         CustomerName,
         CustomerSurname,
         Count(*) AS NumberOfAccounts
  FROM   Customer
         JOIN Account
           ON Account.CustomerID = Customer.CustomerID
  GROUP  BY Customer.CustomerID,
            CustomerName,
            CustomerSurname;
```

3)Funkcja, która liczy sumę w podanej walucie pieniędzy danego użytkownika na wszystkich jego kontach w danej walucie

```
GO
CREATE FUNCTION CalculateTotalAmountInCurrency1
(
   @UserID INT,
   @UserType VARCHAR(10),
   @Currency VARCHAR(3)
)
RETURNS DECIMAL(15, 2)
AS
BEGIN
   DECLARE @TotalAmount DECIMAL(15, 2);
   IF @UserType = 'Customer'
   BEGIN
       SELECT @TotalAmount = ISNULL(SUM(a.AccountBalance), 0)
       FROM Account a
       INNER JOIN Customer_Account ca ON a.AccountID = ca.AccountNumber
       WHERE a.Currency = @Currency
```

```
        AND ca.CustomerID = @UserID;
    END
    ELSE IF @UserType = 'Company'
    BEGIN
        SELECT @TotalAmount = ISNULL(SUM(a.AccountBalance), 0)
        FROM Account a
        INNER JOIN Company_Account coa ON a.AccountID = coa.AccountNumber
        WHERE a.Currency = @Currency
        AND coa.CompanyID = @UserID;
    END;
    RETURN @TotalAmount;
END;
GO

DECLARE @TotalAmount DECIMAL(15, 2);
SELECT @TotalAmount = dbo.CalculateTotalAmountInCurrency(1, 'Customer', 'PLN');
PRINT 'Total amount for CustomerID 1 in USD: ' +
CAST(dbo.CalculateTotalAmountInCurrency1(1, 'Customer', 'USD') AS VARCHAR(20));
```

4)Widok wszystkich kart w systemie wraz z numerem konta

```
GO
CREATE VIEW AllCardsForAccount AS
SELECT
    c.CardID,
    a.AccountID,
    c.ExpirationDate,
    c.CardType
FROM
    Cards c
JOIN
    Account a ON c.AccountID = a.AccountID;
GO
```

5) Widok osób z kredytami - wyświetla informacje o osobach, które kiedykolwiek wzięły kredyt.

```
CREATE VIEW  clients_with_credits AS
SELECT * FROM Customer
WHERE CustomerID IN (SELECT CustomerID FROM Credit);
```

6)Uznania (wpłaty) dla każdego konta. Jest to widok który wyświetla id konta, typ transakcji i kwota uznania

```
GO
CREATE VIEW Uznania AS
SELECT
   t.TransactionID,
   'TypTransakcji' AS TypTransakcji,
   t.TransactionAmount AS Kwota,
   a.AccountID
FROM
   Transactions t
JOIN
   TransactionCard tc ON t.TransactionID = tc.TransactionID
JOIN
   Account a ON tc.DestinationAccountID = a.AccountID
UNION ALL
SELECT
   t.TransactionID,
   'Wymiana walut' AS TypTransakcji,
   t.TransactionAmount AS Kwota,
   a.AccountID
FROM
   Transactions t
JOIN
   ExchangeCurrencyTransaction ect ON t.TransactionID = ect.TransactionID
JOIN
   Account a ON ect.DestinationAccountID = a.AccountID
UNION ALL
SELECT
   t.TransactionID,
   'Wplata w okienku' AS TypTransakcji,
   t.TransactionAmount AS Kwota,
   a.AccountID
FROM
   Transactions t
JOIN
   TransactionInBranch tib ON t.TransactionID = tib.TransactionID
JOIN
   Account a ON tib.AccountID = a.AccountID
WHERE
   tib.TypeOfTransaction = 'Wpłata'
UNION ALL
SELECT
   t.TransactionID,
   'Przelew' AS TypTransakcji,
   t.TransactionAmount AS Kwota,
```

```
    a.AccountID
FROM
    Transactions t
JOIN
    Transfers tr ON t.TransactionID = tr.TransactionID
JOIN
    Account a ON tr.DestinationAccountID = a.AccountID
UNION ALL
SELECT
    t.TransactionID,
    'Wplata w ATM' AS TypTransakcji,
    t.TransactionAmount AS Kwota,
    a.AccountID
FROM
    Transactions t
JOIN
    ATMTransaction at ON t.TransactionID = at.TransactionID
JOIN
    Account a ON at.AccountID = a.AccountID
WHERE
    at.TypeTransaction = 'MakeDeposit';
GO
```

7)Funkcja suma wydatków z danego miesiąca i roku dla danego konta

```
GO
CREATE FUNCTION dbo.GetTotalExpenditureForAccount (
    @AccountID VARCHAR(40),
    @Month INT,
    @Year INT
)
RETURNS DECIMAL(15, 2)
AS
BEGIN
    DECLARE @TotalExpenditure DECIMAL(15, 2);
    SELECT @TotalExpenditure = SUM(t.TransactionAmount)
    FROM Transactions t
    LEFT JOIN TransactionCard tc ON t.TransactionID = tc.TransactionID
    LEFT JOIN TransactionInBranch tb ON t.TransactionID = tb.TransactionID
    LEFT JOIN Transfers tr ON t.TransactionID = tr.TransactionID
    LEFT JOIN ATMTransaction atm ON t.TransactionID = atm.TransactionID
    LEFT JOIN ExchangeCurrencyTransaction ect ON t.TransactionID = ect.TransactionID
    WHERE
        (
            (tc.DestinationAccountID = @AccountID) -- Wypłata z karty
```

```
            OR (tb.AccountID = @AccountID AND tb.TypeOfTransaction = 'Wypłata')
-- Wypłata w oddziale
            OR (tr.SourceAccountID = @AccountID) -- Transfer z konta
            OR (atm.AccountID = @AccountID AND atm.TypeTransaction = 'MakeWithdrawal')
-- Wypłata w bankomacie
            OR (ect.SourceAccountID = @AccountID) -- Wymiana walut
        )
        AND MONTH(t.TransactionDate) = @Month
        AND YEAR(t.TransactionDate) = @Year

    RETURN ISNULL(@TotalExpenditure, 0);
END;
GO
DECLARE @AccountID VARCHAR(40) = 1;
DECLARE @Month INT = 1;
DECLARE @Year INT = 2020;

DECLARE @TotalExpenditure DECIMAL(15, 2);

SET @TotalExpenditure = dbo.GetTotalExpenditureForAccount(@AccountID, @Month, @Year);

SELECT @TotalExpenditure AS TotalExpenditure;
```

8) Funkcja zwracająca listę transakcji w danym bankomacie

```
CREATE FUNCTION ShowATMTransactions(@atmId INT)
RETURNS TABLE
AS
RETURN (
    SELECT
        T.TransactionID,
        T.TransactionAmount,
        T.Currency,
        A.TypeTransaction
    FROM Transactions T
    JOIN ATMTransaction A ON T.TransactionID = A.TransactionID
    WHERE A.ATMID = @atmId
);
```

9) Funkcja zwracająca wszystkie transakcje w danym oddziale banku

```
CREATE FUNCTION showBranchTransactions()
RETURNS TABLE
AS
RETURN (
    SELECT
        t.TransactionID,
        t.TransactionAmount,
        t.Currency,
        t.TransactionDate,
        t2.EmployeeID,
        t2.TypeOfTransaction,
        t2.DestinationAccountID,
        t2.ForeignDestinationAccountID
    FROM Transactions t
    JOIN TransactionInBranch t2 ON t.TransactionID = t2.TransactionID
);
```

10) Funkcja zwracająca listę lokat danego klienta

```
CREATE FUNCTION showTermDepositsForCustomer(@customerID INT)
RETURNS TABLE
AS
RETURN (
    SELECT
        t.TermDepositID,
        t.Amount,
        t.Currency,
        t.StartDepositDate,
        t.DateOfDepositEnd
    FROM TermDeposit t
    WHERE t.CustomerID = @customerID
);
```

Procedury składowane

1) **Procedura która dla danego klienta sprawdzi czy może on zaciągnąć kolejny lub pierwszy kredyt. W przypadku gdy są niezapłacone zaległe raty oraz gdy klient ma juz 3 kredyty taka możliwość będzie zablokowana.**

```
GO
CREATE PROCEDURE CheckEligibilityForLoan
   @CustomerID INT,
   @IsEligible BIT OUTPUT
AS
BEGIN
   DECLARE @UnpaidRepayments INT;
   DECLARE @ActiveLoans INT;
   DECLARE @Today DATE;
   SET @Today = GETDATE();
   SELECT @UnpaidRepayments = COUNT(*)
   FROM Repayment R
   INNER JOIN Credit C ON R.CreditID = C.CreditID
   WHERE C.CustomerID = @CustomerID
   AND R.RepaymentStatus = 'Unpaid'
   AND R.DueRepaymentDate <= @Today;
   SELECT @ActiveLoans = COUNT(*)
   FROM Credit
   WHERE CustomerID = @CustomerID AND CreditStatus = 'Active';
   IF @UnpaidRepayments = 0 AND @ActiveLoans < 3
   BEGIN
       SET @IsEligible = 1;
   END
   ELSE
   BEGIN
       SET @IsEligible = 0;
   END;
END;
GO
```

2)Procedura która spłaca zadaną ratę dla danego kredytu i nalicza odsetki 100 zł/dzień jeżeli przyjmując dzisiejszą datę rata płacona jest po terminie, dodaje nową transakcję której trigger pobierze pieniądze i wyśle na odpowiednie konto. Jeżeli wszystkie raty kredytu są spłacone to zmienia status kredytu na unactive. Wyświetla także pozostałą ilość rat do spłacenia kredytu.

```
CREATE PROCEDURE PayRepayment
   @CreditID INT,
   @RepaymentID INT
AS
BEGIN
   DECLARE @Today DATE = GETDATE();
   DECLARE @RemainingRepayments INT;
   DECLARE @LateDays INT;
   DECLARE @LateFee DECIMAL(15, 2);
   DECLARE @Amount DECIMAL(15, 2);
   DECLARE @SourceAccount VARCHAR(40);
   DECLARE @Currency VARCHAR(3);
   DECLARE @Result VARCHAR(MAX);
   SELECT @RemainingRepayments = COUNT(*)
   FROM Repayment
   WHERE CreditID = @CreditID AND RepaymentStatus = 'Unpaid';
   SELECT @LateDays = DATEDIFF(DAY, DueRepaymentDate, @Today)
   FROM Repayment
   WHERE CreditID = @CreditID AND RepaymentID = @RepaymentID;

   SET @LateFee = CASE WHEN @LateDays > 0 THEN @LateDays * 100.00 ELSE 0 END;
   SELECT @Amount = Amount
   FROM Repayment
   WHERE CreditID = @CreditID AND RepaymentID = @RepaymentID;
   SET @Amount = @Amount + @LateFee;
   SELECT @SourceAccount = AccountNumber, @Currency = Currency
   FROM Credit
   WHERE CreditID = @CreditID;
   IF (SELECT AccountBalance FROM Account WHERE AccountID = @SourceAccount) >= @Amount
   BEGIN
       INSERT INTO Transactions (TransactionAmount, Currency, TransactionDate)
       VALUES (@Amount, @Currency, @Today);
      DECLARE @id INT
      SELECT @id=TransactionID FROM Transactions WHERE TransactionDate=@Today AND
Currency=@Currency AND TransactionAmount=@Amount
       INSERT INTO Transfers VALUES(@id,NULL,@SourceAccount,NULL,'Yes',1)
       UPDATE Repayment
       SET Amount = Amount + @LateFee
       WHERE CreditID = @CreditID AND RepaymentID = @RepaymentID;
       UPDATE Repayment
```

```
        SET RepaymentStatus = 'Paid', PaidDate = @Today
        WHERE CreditID = @CreditID AND RepaymentID = @RepaymentID;
        IF @RemainingRepayments - 1 = 0
        BEGIN
            UPDATE Credit
            SET CreditStatus = 'Inactive'
            WHERE CreditID = @CreditID;
        END
        SET @Result = 'Late fee: ' + CAST(@LateFee AS VARCHAR(20)) +
                      ' Remaining repayments: ' + CAST((@RemainingRepayments - 1) AS
VARCHAR(10));
    END
    ELSE
    BEGIN
        SET @Result = 'Insufficient funds in the source account.';
    END;
    PRINT @Result;
END;

EXEC PayRepayment @CreditID = 2, @RepaymentID = 1;
```

3) Procedura wpłacając pieniądze na konto przy założeniu, że wpłacać możemy przelewem, w oddziale banku lub w bankomacie

```
CREATE PROCEDURE DepositMoney
@TransactionID INT,
    @AccountID INT,
    @Amount DECIMAL(15, 2),
    @Currency VARCHAR(3),
    @DepositType VARCHAR(20),
    @BranchID INT = NULL,
    @ATMID INT = NULL,
    @SourceAccountID INT = NULL,
  @dateTransaction
AS
BEGIN
    BEGIN TRANSACTION;
    INSERT INTO Transactions (TransactionID,TransactionAmount, Currency,
TransactionDate) VALUES (@TransactionID,@Amount, @Currency, @dateTransaction);

    SET @TransactionID = SCOPE_IDENTITY();
    IF @DepositType = 'BranchDeposit'
    BEGIN
        INSERT INTO TransactionInBranch (TransactionID, TypeOfTransaction,
AccountID, EmployeeID, DestinationAccountID)
        VALUES (@TransactionID, 'Wpłata', @AccountID, NULL, @AccountID);
    END
    ELSE IF @DepositType = 'ATMDeposit'
    BEGIN
        INSERT INTO ATMTransaction (TransactionID, ATMID, AccountID,
TypeTransaction)
        VALUES (@TransactionID, @ATMID, @AccountID, 'MakeDeposit');
    END
    ELSE IF @DepositType = 'TransferDeposit'
    BEGIN
        INSERT INTO Transfers (TransactionID, SourceAccountID,
DestinationAccountID, IsDestinationAccountForeign)
        VALUES (@TransactionID, @SourceAccountID, @AccountID, 'No');
    END
    UPDATE Account
    SET AccountBalance = AccountBalance + @Amount
    WHERE AccountID = @AccountID;
    COMMIT;
END;
```

4)Procedura wypłacająca pieniądze z konta (wypłacamy przelewem, w oddziale lub w bankomacie). Nie jest to procedura księgująca transakcje z zeszłego dnia czy później.

```
CREATE PROCEDURE WithdrawMoney
    @AccountID INT,
    @WithdrawalAmount DECIMAL(15, 2),
    @WithdrawalMethod VARCHAR(20)
AS
BEGIN
    DECLARE @CurrentBalance DECIMAL(15, 2);
    SELECT @CurrentBalance = AccountBalance
    FROM Account
    WHERE AccountID = @AccountID;
    IF @WithdrawalAmount > 0 AND @WithdrawalAmount <= @CurrentBalance
    BEGIN
        BEGIN TRANSACTION;
        IF @WithdrawalMethod = 'Transfer'
        BEGIN
            UPDATE Account
            SET AccountBalance = @CurrentBalance - @WithdrawalAmount
            WHERE AccountID = @AccountID;
            INSERT INTO Transactions (TransactionAmount, Currency,
TransactionDate)VALUES (-@WithdrawalAmount, 'CurrencyCode', GETDATE());
            INSERT INTO Transfers (SourceAccountID, DestinationAccountID,
TransactionID)VALUES (@AccountID, 'DestinationAccountID', SCOPE_IDENTITY());
        END
        ELSE IF @WithdrawalMethod = 'BranchTransaction'
        BEGIN
            UPDATE Account
            SET AccountBalance = @CurrentBalance - @WithdrawalAmount
            WHERE AccountID = @AccountID;
            INSERT INTO Transactions (TransactionAmount, Currency,
TransactionDate) VALUES (-@WithdrawalAmount, 'CurrencyCode', GETDATE());
            INSERT INTO TransactionInBranch (EmployeeID, AccountID,
TypeOfTransaction, TransactionID)
            VALUES (EmployeeID, @AccountID, 'Wypłata', SCOPE_IDENTITY());
        END
        ELSE IF @WithdrawalMethod = 'ATMTransaction'
        BEGIN
            UPDATE Account
            SET AccountBalance = @CurrentBalance - @WithdrawalAmount
            WHERE AccountID = @AccountID;
```

```
            INSERT INTO Transactions (TransactionAmount, Currency,
TransactionDate)VALUES (-@WithdrawalAmount, 'CurrencyCode', GETDATE());
            INSERT INTO ATMTransaction (ATMID, AccountID, TypeTransaction,
TransactionID) VALUES (ATMID, @AccountID, 'MakeWithdrawal', SCOPE_IDENTITY());
        END
        COMMIT;
    END
END;
```

5) Naliczanie opłaty miesięcznej za korzystanie z konta bankowego dla zadanego miesiąca i roku. Jeżeli klient nie ma pieniędzy to naliczamy ujemne wartości, tak aby gdy będzie chciał ponownie skorzystać z konta i wpłaci tam jakieś pieniądze to suma zostanie pobrana.

```
GO
CREATE PROCEDURE ApplyMonthlyFee
   @AccountID VARCHAR(40),
   @Month INT,
   @Year INT,
   @Result DECIMAL(10, 2) OUTPUT,
   @ResultCurrency VARCHAR(3) OUTPUT
AS
BEGIN
   DECLARE @Count INT
   Select @Count = Count(*) FROM Transfers as tf
   join Transactions tr ON tr.TransactionID = tf.TransactionID
   WHERE tf.SourceAccountID=@AccountID AND MONTH(tr.TransactionDate) = @Month AND
YEAR(tr.TransactionDate) = @Year;
   SELECT @ResultCurrency = Currency FROM  Account
   Where AccountID=@AccountID
   SELECT @Result=FeeAmount FROM AccountAccessFee
   WHERE Currency = @ResultCurrency AND TransactionAmount=@Count
   UPDATE Account
   SET AccountBalance = AccountBalance- @Result
   WHERE AccountID = @AccountID;
END;
GO
DECLARE @Result DECIMAL(15, 2);
DECLARE @ResultCurrency VARCHAR(3);

EXEC ApplyMonthlyFee
   @AccountID = 2,
   @Month = 1,
   @Year = 2024,
   @Result = @Result OUTPUT,
   @ResultCurrency = @ResultCurrency OUTPUT;

PRINT 'Monthly Fee: ' + CAST(@Result AS VARCHAR(20)) + ', Currency: ' + @ResultCurrency;
```

6) Procedura dotycząca depozytów o argumentach data i ID depozytu, która obliczy ile zyskasz pieniędzy gdy wybierzesz danego dnia pieniądze z depozytu

```
GO
CREATE PROCEDURE CalculateDepositProfit
   @Date DATE,
   @TermDepositID INT
AS
BEGIN
   DECLARE @StartDepositDate DATE
   DECLARE @EndDepositDate DATE
   DECLARE @DepositInterestRate DECIMAL(5, 2)
   DECLARE @Amount DECIMAL(15, 2)
   DECLARE @Profit DECIMAL(15, 2)

   SELECT @StartDepositDate = StartDepositDate, @EndDepositDate = DateOfDepositEnd
   FROM TermDeposit
   WHERE TermDepositID = @TermDepositID;

   IF @Date >= @StartDepositDate AND @Date <= @EndDepositDate
   BEGIN

       SELECT @DepositInterestRate = DepositInterestRates
       FROM TermDepositInterstRates
       WHERE MinimumLengthInYears = DATEDIFF(YEAR, @StartDepositDate, @EndDepositDate);

       SELECT @Amount = Amount
       FROM TermDeposit
       WHERE TermDepositID = @TermDepositID;

       SET @Profit = @Amount * @DepositInterestRate;


       PRINT 'Zysk dla dnia ' + CONVERT(VARCHAR, @Date) + ' wynosi ' + CONVERT(VARCHAR,
@Profit) ;
   END
   ELSE
   BEGIN
       PRINT 'Podana data nie mieści się w okresie trwania depozytu.';
   END
END;
GO

EXEC CalculateDepositProfit @Date='2022-12-12' ,@TermDepositID=1
```

Wyzwalacze

1)Wyzwalacz który gdy dodajemy nowe konto dodaje rekordy do tabel Customer_Account lub Company_Account

```
GO
CREATE TRIGGER Add_Account_Record
ON Account
AFTER INSERT
AS
BEGIN
   SET NOCOUNT ON;

   DECLARE @AccountType2 VARCHAR(20);

   SELECT @AccountType2 = AccountType2 FROM inserted;

   IF @AccountType2 = 'CUSTOMER_ACCOUNT'
   BEGIN
       INSERT INTO Customer_Account (CustomerID, AccountNumber)
       SELECT CustomerID, AccountID FROM inserted;
   END
   ELSE IF @AccountType2 = 'COMPANY_ACCOUNT'
   BEGIN
       INSERT INTO Company_Account (CompanyID, AccountNumber)
       SELECT CompanyID, AccountID FROM inserted;
   END
END;
GO
```

2)Wyzwalacz który gdy dodajemy kredyt oblicza nam jego raty pod warunkiem że klient może zaciągnąć kolejny kredyt. Liczona jest wysokość rat z uwzględnieniem oprocentowania stałego.

```
GO
CREATE TRIGGER AfterInsertCredit
ON Credit
AFTER INSERT
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @CreditID INT, @LoanPeriodInYears INT, @InterestRate DECIMAL(5, 2),@Amount INT,
@DataStart DATE, @DataEnd DATE, @Currency VARCHAR(3);
```

```
  SELECT @CreditID = CreditID, @Amount = Amount, @Currency = Currency, @DataStart =
DataStart, @DataEnd = DataEnd FROM inserted;
  DECLARE @CustomerID INT;
  SELECT @CustomerID = CustomerID
  FROM inserted;
  DECLARE @IsEligible BIT;
  EXEC CheckEligibilityForLoan @CustomerID = @CustomerID, @IsEligible = @IsEligible
OUTPUT;
IF @IsEligible = 1
BEGIN
   SET @LoanPeriodInYears = DATEDIFF(YEAR, @DataStart, @DataEnd);
   SELECT @InterestRate = InterstRate FROM FixedLoanInterestRate WHERE LoanPeriodInYears
= @LoanPeriodInYears;
   DECLARE @NumberOfPayments DECIMAL(15, 2) = DATEDIFF(MONTH, @DataStart, @DataEnd);
   DECLARE @MonthlyPayment DECIMAL(15, 2) = (@Amount * @InterestRate+@Amount) /
(@NumberOfPayments);
   DECLARE @RepaymentDate DATE = @DataStart;
   WHILE @RepaymentDate <= @DataEnd
   BEGIN
       INSERT INTO Repayment (DueRepaymentDate, Amount, Currency,
RepaymentStatus,PaidDate, CreditID)
       VALUES (@RepaymentDate, @MonthlyPayment,  @Currency, 'Unpaid', NULL,@CreditID);
       SET @RepaymentDate = DATEADD(MONTH, 1, @RepaymentDate);
   END;
   END;
END;
GO
```

3) Wyzwalacz sprawdzający czy możemy utworzyć konto o danym ID

```
CREATE TRIGGER CheckAccountIDBeforeInsert
ON Account
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @NewAccountID VARCHAR(40);
    DECLARE @ExistingAccountIDCount INT;


    SELECT @NewAccountID = AccountID
    FROM inserted;

    -- Sprawdź, czy istnieje konto o tym samym ID
    SELECT @ExistingAccountIDCount = COUNT(*)
    FROM Account
    WHERE AccountID = @NewAccountID;

    IF @ExistingAccountIDCount > 0
    BEGIN
     --wycofanie jeżeli istnieje takie id
```

```
        ROLLBACK;
    END;
END;
```

4) Wyzwalacz sprawdzający czy możemy utworzyć kartę o konkretnym ID

```
CREATE TRIGGER CheckCardIDBeforeInsert
ON Cards
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @NewCardID INT;
    DECLARE @ExistingCardIDCount INT;


    SELECT @NewCardID = CardID
    FROM inserted;


    SELECT @ExistingCardIDCount = COUNT(*)
    FROM Cards
    WHERE CardID = @NewCardID;


    IF @ExistingCardIDCount > 0
    BEGIN
        --wycofujemy jezeli istnieje takie id
        ROLLBACK;
    END;
END;
```

5) Wyzwalacz aktualizujący saldo w bankomacie po transakcji

```
CREATE TRIGGER UpdateATMBalance
ON ATMTransaction
AFTER INSERT
AS
BEGIN
    UPDATE atm
    SET Balance = Balance - t.TransactionAmount
    FROM AutomatedTellerMachineBalance atm
    JOIN INSERTED t ON atm.AutomatedTellerMachineID =
t.AutomatedTellerMachineID;
END;
```

Przykładowe dane

```
INSERT INTO ForeignBankAccounts VALUES (1)

INSERT INTO Customer VALUES(1,'XXXX','YYYYY','2000-01-23','Analizy_Matematycznej 38
Kraków','xyxy@gmail.com','990776554')
INSERT INTO Customer VALUES(2,'XXXXX','YYYYYY','2000-01-23','Grafu_Planarnego 38
Kraków','yyyxxx@gmail.com','990776554')

INSERT INTO Account
VALUES(1,'2020-01-23',NULL,'StandardAccount','PLN',0,NULL,1,'CUSTOMER_ACCOUNT')
INSERT INTO Account
VALUES(2,'2020-01-23',NULL,'StandardAccount','USD',0,NULL,1,'CUSTOMER_ACCOUNT')

INSERT INTO Cards VALUES(1,1,'2070-01-23','Debit')

INSERT INTO Customer_Account VALUES(1,1)
INSERT INTO Customer_Account VALUES(1,2)

INSERT INTO FixedLoanInterestRate VALUES(0,0.1)
INSERT INTO FixedLoanInterestRate VALUES(1,0.15)
INSERT INTO FixedLoanInterestRate VALUES(2,0.2)
INSERT INTO FixedLoanInterestRate VALUES(3,0.25)
INSERT INTO FixedLoanInterestRate VALUES(4,0.30)
INSERT INTO FixedLoanInterestRate VALUES(5,0.35)

INSERT INTO BankBranch VALUES(1,'Sql 20 Kraków','888999666')

INSERT INTO BankEmployee VALUES(1,'XXXXY','YYYYYX','2020-01-23','Ruczaj 50
Kraków','xxxyyx@gmail.com','888777000','Prezes','2020-01-23',NULL,1)



INSERT INTO AccountAccessFee VALUES(1,10,'PLN')
INSERT INTO AccountAccessFee VALUES(2,8,'PLN')
INSERT INTO AccountAccessFee VALUES(3,6,'PLN')
INSERT INTO AccountAccessFee VALUES(4,4,'PLN')
INSERT INTO AccountAccessFee VALUES(1,10,'EUR')
INSERT INTO AccountAccessFee VALUES(2,8,'EUR')
INSERT INTO AccountAccessFee VALUES(3,6,'EUR')
INSERT INTO AccountAccessFee VALUES(4,4,'USD')
INSERT INTO AccountAccessFee VALUES(1,10,'USD')
INSERT INTO AccountAccessFee VALUES(2,8,'USD')
INSERT INTO AccountAccessFee VALUES(3,6,'USD')
INSERT INTO AccountAccessFee VALUES(4,4,'USD')
INSERT INTO AccountAccessFee VALUES(0,60,'USD')
```

```
INSERT INTO AccountAccessFee VALUES(0,60,'USD')
INSERT INTO AccountAccessFee VALUES(0,60,'USD')


INSERT INTO TermDepositInterstRates VALUES(0,0)
INSERT INTO TermDepositInterstRates VALUES(1,0.1)
INSERT INTO TermDepositInterstRates VALUES(2,0.15)
INSERT INTO TermDepositInterstRates VALUES(3,0.20)
INSERT INTO TermDepositInterstRates VALUES(4,0.25)

INSERT INTO TermDeposit VALUES(1,'2020-01-23','PLN',4000,'2023-01-23',NULL,1,'CUSTOMER')

INSERT INTO AutomatedTellerMachine VALUES(1,'Sql 20 Kraków',1)
INSERT INTO AutomatedTellerMachineBalance VALUES(1,300000, 'PLN')
INSERT INTO AutomatedTellerMachineBalance VALUES(1,300000, 'USD')
INSERT INTO AutomatedTellerMachineBalance VALUES(1,300000, 'EUR')

INSERT INTO Credit
VALUES(1,1,40000,'PLN','Fixed','2020-01-23','2020-06-23','Active',1,NULL,1,'CUSTOMER_CRED
IT')

INSERT INTO Transactions VALUES(400,'PLN','2020-01-23')
INSERT INTO TransactionInBranch VALUES(1,1,1,'Wpłata',NULL,'-',NULL)

INSERT INTO Transactions VALUES(400,'PLN','2020-01-23')
INSERT INTO TransactionCard VALUES(2,2,1,'No',NULL)

INSERT INTO Transactions VALUES(4080,'PLN','2020-01-23')
INSERT INTO ExchangeCurrencyTransaction VALUES(3,'USD',1,2)

INSERT INTO Transactions VALUES(4080,'PLN','2020-01-23')
INSERT INTO Transfers VALUES(4,NULL,1,2,'No',NULL)

INSERT INTO Transactions VALUES(4080,'PLN','2020-01-23')
INSERT INTO ATMTransaction VALUES(5,1,1,'MakeDeposit')

INSERT INTO Transactions VALUES(40,'PLN','2020-01-23')
INSERT INTO ATMTransaction VALUES(6,1,1,'MakeWithdrawal')

INSERT INTO Company VALUES(1,'202390','xyz','i','99999','email')
INSERT INTO Account
VALUES(3,'2020-01-23',NULL,'StandardAccount','PLN',0,1,NULL,'COMPANY_ACCOUNT')
```