T-SQL – Homework

1. **Create a database with two tables: Persons(Id(PK), FirstName, LastName, SSN) and Accounts(Id(PK), PersonId(FK), Balance). Insert few records for testing. Write a stored procedure that selects the full names of all persons.**

```sql
USE TelerikAcademy;

-- Create tables
CREATE TABLE Persons(
      PersonID int IDENTITY,
      FirstName nvarchar(50) NOT NULL,
      LastName nvarchar(50) NOT NULL,
      SSN int NOT NULL,
      CONSTRAINT PK_Persons PRIMARY KEY(PersonID),
      CONSTRAINT CK_SSN CHECK (SSN > 99999999 AND SSN < 1000000000)
);
GO

CREATE TABLE Accounts(
      AccountID int IDENTITY,
      PersonID int NOT NULL,
      Balance money,
      CONSTRAINT PK_Accounts PRIMARY KEY(AccountID),
      CONSTRAINT FK_Accounts_Persons FOREIGN KEY(PersonID)
            REFERENCES Persons(PersonID)
);
GO

-- Insert records
INSERT INTO Persons(FirstName, LastName, SSN)
VALUES
      ('Ivancho', 'Petkov', '123456789'),
      ('Jivko', 'Jivkov', '143256789'),
      ('Ivana', 'Mancheva', '123442789'),
      ('Julia', 'Petrova', '123412121');

INSERT INTO Accounts(PersonID, Balance)
VALUES
      (1, 1020),
      (2, 30000),
      (4, 50100),
      (3, 10405);
GO
```

```
USE TelerikAcademy;
GO

-- Create procedure
CREATE PROC usp_SelectPersonsFullName
AS
        SELECT FirstName + ' ' + LastName AS [Full Name]
        FROM Persons;
GO

-- Test procedure
EXEC usp_SelectPersonsFullName;
GO
```

2. **Create a stored procedure that accepts a number as a parameter and returns all persons who have more money in their accounts than the supplied number.**

```
USE TelerikAcademy;
GO

-- Create procedure
CREATE PROC usp_SelectAllPersonsWithHigherMoney (
        @balanceLine money = 0)
AS
        SELECT p.FirstName + ' ' + p.LastName AS [Full Name], a.Balance
        FROM Persons AS p
        INNER JOIN Accounts AS a
        ON p.PersonID = a.PersonID
        WHERE a.Balance >= @balanceLine;
GO

-- Test proc
EXEC usp_SelectAllPersonsWithHigherMoney;
EXEC usp_SelectAllPersonsWithHigherMoney 30000;
GO
```

3. **Create a function that accepts as parameters – sum, yearly interest rate and number of months. It should calculate and return the new sum. Write a SELECT to test whether the function works as expected.**

```sql
USE TelerikAcademy;
GO

-- Create function
CREATE FUNCTION ufn_CalculateAnnualInterest (
      @startBalance money,
      @annualInterestRate money,
      @months int
      )
      RETURNS money
AS
BEGIN
      DECLARE @result money;
      SET @result = @startBalance * (@annualInterestRate / 100) * (@months / 12.0);
      RETURN @result;
END
GO

-- Test function
SELECT dbo.ufn_CalculateAnnualInterest(1000, 5.6, 12) AS [Annual Interest For A Year];
SELECT dbo.ufn_CalculateAnnualInterest(1000, 10, 12) AS [Annual Interest For A Year];
SELECT dbo.ufn_CalculateAnnualInterest(1000, 10, 6) AS [Annual Interest For Six Months];
SELECT dbo.ufn_CalculateAnnualInterest(1000, 10, 1) AS [Annual Interest For A Months];
GO
```

4. **Create a stored procedure that uses the function from the previous example to give an interest to a person's account for one month. It should take the AccountId and the interest rate as parameters.**

```sql
USE TelerikAcademy;
GO

-- Create procedure
CREATE PROC usp_AddAnnualInterestForOneMonth (
        @accountID int,
        @annualInterestRate money)
AS
        DECLARE @annualInterest money,
                    @personBalance money,
                    @months int = 1;

        SELECT @personBalance = a.Balance
        FROM Accounts AS a
        WHERE a.AccountID = @accountID;
        SET @annualInterest = dbo.ufn_CalculateAnnualInterest(
                                              @personBalance,
                                              @annualInterestRate,
                                              @months);

        UPDATE Accounts
        SET Balance = Balance + @annualInterest
        WHERE AccountID = @accountID;
GO

-- Test procedure
-- Before adding interest
DECLARE @accountID int = 2;
DECLARE @annualInterestRate money = 10;

SELECT *
FROM Accounts
WHERE AccountID = @accountID;

EXEC usp_AddAnnualInterestForOneMonth @accountID, @annualInterestRate;

-- After adding interest
SELECT *
FROM Accounts
WHERE AccountID = @accountID;
GO
```

5. **Add two more stored procedures WithdrawMoney(AccountId, money) and DepositMoney (AccountId, money) that operate in transactions.**

```sql
USE TelerikAcademy;
GO

-- Create procedures
CREATE PROC usp_WithdrawMoney (
        @accountID int,
        @money money)
AS
        UPDATE Accounts
        SET Balance = Balance -  @money
        WHERE AccountID = @accountID;
GO

CREATE PROC usp_DepositMoney (
        @accountID int,
        @money money)
AS
        UPDATE Accounts
        SET Balance = Balance +  @money
        WHERE AccountID = @accountID;
GO

CREATE PROC usp_SendMoney(
        @fromAccountID int,
        @toAccountID int,
        @money money)
AS
        BEGIN TRAN SendingMoney
                IF (EXISTS(SELECT * FROM Accounts WHERE AccountID = @fromAccountID) AND
                        EXISTS(SELECT * FROM Accounts WHERE AccountID = @toAccountID))
                BEGIN
                        EXEC usp_WithdrawMoney @fromAccountID, @money;
                        DECLARE @accountBalance money;
                        SELECT @accountBalance = Balance
                        FROM Accounts
                        WHERE AccountID = @fromAccountID;

                        IF (@accountBalance < 0 )
                        BEGIN
                                RAISERROR('Account has not enough money.', 16, 1);
                                ROLLBACK TRAN;
                                RETURN;
                        END
                        ELSE
                        BEGIN
                                EXEC usp_DepositMoney @toAccountID, @money;
                        END
                END
                ELSE
                BEGIN
                        RAISERROR('One of the accounts is invalid.', 16, 1);
                        ROLLBACK TRAN;
                        RETURN;
                END
        COMMIT TRAN SendingMoney;
GO
```

```sql
-- Test procedure
-- Before adding interest
DECLARE @fromAccount int = 1;
DECLARE @toAccount int = 2;
DECLARE @money money = 100;

SELECT *
FROM Accounts
WHERE AccountID = @fromAccount;

SELECT *
FROM Accounts
WHERE AccountID = @toAccount;

EXEC usp_SendMoney @fromAccount, @toAccount, @money;

-- After adding interest
SELECT *
FROM Accounts
WHERE AccountID = @fromAccount;

SELECT *
FROM Accounts
WHERE AccountID = @toAccount;
GO
```

```sql
-- This will raise an error
DECLARE @fromAccount int = 1234123;
DECLARE @toAccount int = 2;
DECLARE @money money = 100;

EXEC usp_SendMoney @fromAccount, @toAccount, @money;
GO

-- This will also raise an error
DECLARE @fromAccount int = 1;
DECLARE @toAccount int = 2;
DECLARE @money money = 10012341;

EXEC usp_SendMoney @fromAccount, @toAccount, @money;
GO
```

6. **Create another table – Logs(LogID, AccountID, OldSum, NewSum). Add a trigger to the Accounts table that enters a new entry into the Logs table every time the sum on an account changes.**

```sql
USE TelerikAcademy;
GO

CREATE TABLE Logs(
        LogID int IDENTITY,
        AccountID int NOT NULL,
        OldSum money,
        NewSum money
        CONSTRAINT PK_Logs PRIMARY KEY(LogID),
        CONSTRAINT FK_Logs_Accounts FOREIGN KEY(AccountID)
                REFERENCES Accounts(AccountID)
);
GO

-- Create Trigger
CREATE TRIGGER tr_BalanceUpdate ON Accounts FOR UPDATE
AS
        INSERT INTO Logs(AccountID, OldSum, NewSum)
        SELECT d.AccountID, d.Balance, i.Balance
        FROM Deleted AS d
        INNER JOIN Inserted as i
        ON d.AccountID = i.AccountID;
GO

-- Test the trigger
DECLARE @fromAccount int = 1;
DECLARE @toAccount int = 2;
DECLARE @money money = 100;

EXEC usp_SendMoney @fromAccount, @toAccount, @money;

-- See the result
SELECT * FROM Logs;
GO
```

7. **Define a function in the database TelerikAcademy that returns all Employee's names (first or middle or last name) and all town's names that are comprised of given set of letters. Example 'oistmiahf' will return 'Sofia', 'Smith', … but not 'Rob' and 'Guy'.**

```sql
USE TelerikAcademy;
GO

CREATE FUNCTION usp_IsComposed(
        @name nvarchar(50),
        @characters nvarchar(50)
        )
        RETURNS bit
AS
BEGIN

        DECLARE @index int = 1,
                    @foundIndex int,
                    @currentCharacter nvarchar(1),
                    @counter int,
                    @result bit;

        DECLARE @usedLetters table(LetterIndex int, Letter nvarchar(1));
        SET @characters = LOWER(@characters);

        WHILE(@index <= LEN(@name))
        BEGIN
                SET @currentCharacter = LOWER(SUBSTRING(@name, @index, 1));
                SET @foundIndex = CHARINDEX(@currentCharacter, @characters);

                IF (@foundIndex = 0)
                BEGIN
                        SET @result = 0;
                        BREAK;
                END
                ELSE
                BEGIN
                        IF(EXISTS(SELECT * FROM @usedLetters WHERE LetterIndex = @foundIndex))
                        BEGIN
                                SELECT TOP 1 @foundIndex = LetterIndex
                                FROM @usedLetters
                                WHERE Letter = @currentCharacter
                                ORDER BY Letter DESC;

                                SET @foundIndex = CHARINDEX(@currentCharacter, @characters, @foundIndex + 1);

                                IF (@foundIndex = 0)
                                BEGIN
                                        SET @result = 0;
                                        BREAK;
                                END
                        END

                        INSERT INTO @usedLetters
                        VALUES (@foundIndex, @currentCharacter);
                END
                SET @index = @index + 1;
        END

        SELECT @counter = COUNT(*) FROM @usedLetters;

        IF(@counter = LEN(@name))
        BEGIN
                SET @result = 1;
        END
        ELSE
        BEGIN
                SET @result = 0;
        END
        RETURN @result;
END
GO
```

```sql
USE TelerikAcademy;
GO

-- Create function
CREATE FUNCTION ufn_GetComposedNames (@characters nvarchar(50))
        RETURNS TABLE
AS
RETURN (
        (SELECT 'First Name: ' + e.FirstName AS Name
        FROM Employees as e
        WHERE 1 = (SELECT dbo.usp_IsComposed(e.FirstName, @characters)))
        UNION
        (SELECT 'Middle Name: ' + e.MiddleName AS Name
        FROM Employees As e
        WHERE 1 = (SELECT dbo.usp_IsComposed(e.MiddleName, @characters)))
        UNION
        (SELECT 'Last Name: ' + e.LastName AS Name
        FROM Employees AS e
        WHERE 1 = (SELECT dbo.usp_IsComposed(e.LastName, @characters)))
        UNION
        (SELECT 'Town Name: ' + t.Name AS Name
        FROM Towns AS t
        WHERE 1 = (SELECT dbo.usp_IsComposed(t.Name, @characters)))
);
GO

-- Test the functions
SELECT *
FROM dbo.ufn_GetComposedNames('oistmiahf');
GO

SELECT *
FROM dbo.ufn_GetComposedNames('RoBERto');
GO

-- Test with three equal letters i
SELECT *
FROM dbo.ufn_GetComposedNames('Kharatishvili');
GO
```

8. **Using database cursor write a T-SQL script that scans all employees and their addresses and prints all pairs of employees that live in the same town.**

```sql
USE TelerikAcademy;

DECLARE lineCursor CURSOR READ_ONLY FOR
        SELECT e1.FirstName, e1.LastName, t1.Name,
                     e2.FirstName, e2.LastName
        FROM Employees e1
        INNER JOIN Addresses a1
                     ON a1.AddressID = e1.AddressID
        INNER JOIN Towns t1
                                     ON t1.TownID = a1.TownID,
        Employees e2
        INNER JOIN Addresses a2
                     ON a2.AddressID = e2.AddressID
        INNER JOIN Towns t2
                             ON t2.TownID = a2.TownID
        WHERE t1.TownID = t2.TownID AND e1.EmployeeID <> e2.EmployeeID
        ORDER BY t1.Name, e1.FirstName, e2.FirstName;

        OPEN lineCursor
        DECLARE @firstName1 nvarchar(50),
                     @lastName1 nvarchar(50),
                @town nvarchar(50),
                @firstName2 nvarchar(50),
                @lastName2 nvarchar(50);

        DECLARE @resultTable table(
                             FirstEmployee nvarchar(100),
                             Town nvarchar(500),
                             SecondEmployee nvarchar(100)
                     );
        FETCH NEXT FROM lineCursor
                     INTO @firstName1, @lastName1, @town, @firstName2, @lastName2

        WHILE @@FETCH_STATUS = 0
        BEGIN
                INSERT INTO @resultTable
                VALUES (@firstName1 + ' ' + @lastName1, @town, @firstName2 + ' ' +
@lastName2);
                FETCH NEXT FROM lineCursor INTO @firstName1, @lastName1, @town, @firstName2,
@lastName2
        END

        CLOSE lineCursor
        DEALLOCATE lineCursor

        SELECT * FROM @resultTable;
GO
```

9. *** Write a T-SQL script that shows for each town a list of all employees that live in it. Sample output:**

```
Sofia -> Svetlin Nakov, Martin Kulov,
George Denchev
Ottawa -> Jose Saraiva
...
```

```sql
USE TelerikAcademy;

-- Result in table
DECLARE lineCursor CURSOR READ_ONLY FOR
        SELECT t.Name AS [TownName], e.FirstName + ' ' + e.LastName AS [EmployeesName]
        FROM Employees e
        INNER JOIN Addresses a
                ON a.AddressID = e.AddressID
        INNER JOIN Towns t
                ON t.TownID = a.TownID;

        OPEN lineCursor
        DECLARE @employeesName nvarchar(100),
                    @townName nvarchar(50);

        DECLARE @resultTable table(
                TownName nvarchar(50),
                EmployeesName nvarchar(4000)
        );

        FETCH NEXT FROM lineCursor INTO @townName, @employeesName;

        WHILE @@FETCH_STATUS = 0
        BEGIN
                IF (EXISTS(SELECT * FROM @resultTable WHERE TownName = @townName))
                BEGIN
                        UPDATE @resultTable
                        SET EmployeesName = EmployeesName + ', ' + @employeesName
                        WHERE TownName = @townName;
                END
                ELSE
                BEGIN
                        INSERT INTO @resultTable
                        VALUES (@townName, @employeesName);
                END

                FETCH NEXT FROM lineCursor INTO @townName, @employeesName;
        END

        CLOSE lineCursor
        DEALLOCATE lineCursor

        SELECT * FROM @resultTable
        ORDER BY TownName;

GO
```

```sql
USE TelerikAcademy;

-- Result as text
DECLARE lineCursor CURSOR READ_ONLY FOR
        SELECT t.Name AS [TownName], e.FirstName + ' ' + e.LastName AS [EmployeesName]
        FROM Employees e
        INNER JOIN Addresses a
                ON a.AddressID = e.AddressID
        INNER JOIN Towns t
                ON t.TownID = a.TownID
        ORDER BY t.Name;

        OPEN lineCursor
        DECLARE @employeeName nvarchar(100),
                    @townName nvarchar(50),
                        @employeesNameRecord nvarchar(4000),
                        @previousTownName nvarchar(50);

        FETCH NEXT FROM lineCursor INTO @townName, @employeeName;

        WHILE @@FETCH_STATUS = 0
        BEGIN
                IF (@previousTownName = @townName)
                BEGIN
                        SET @employeesNameRecord = @employeesNameRecord + ', ' + @employeeName;
                END
                ELSE
                BEGIN
                        PRINT @previousTownName + ' -> ' + @employeesNameRecord;
                        SET @previousTownName = @townName;
                        SET @employeesNameRecord = @employeeName;
                END

                FETCH NEXT FROM lineCursor INTO @townName, @employeeName;
        END
        PRINT @previousTownName + ' -> ' + @employeesNameRecord;
        CLOSE lineCursor
        DEALLOCATE lineCursor
GO
```

10. **Define a .NET aggregate function StrConcat that takes as input a sequence of strings and return a single string that consists of the input strings separated by ','. For example the following SQL statement should return a single string:**

```sql
SELECT StrConcat(FirstName + ' ' + LastName)
FROM Employees
```

```sql
-- Tutorial from http://msdn.microsoft.com/en-us/library/ms131056.aspx
-- 1) Create StrConcat.dll - The solution with the C# code are in the folder and the built dll

USE TelerikAcademy;
GO

-- 2) Enable clr to execute user code in .NET Framework
sp_configure 'clr enabled', 1
GO

-- 3) Install the changes
RECONFIGURE
GO

-- 4) Create assembly from the StrConcat.dll
CREATE ASSEMBLY StrConcat
FROM 'D:\Documents\Telerik Courses homeworks\Databases\05.T-SQL\StrConcat.dll';
GO

-- 5) Create Aggregate StrConcat function
CREATE AGGREGATE StrConcat (@input nvarchar(200)) RETURNS nvarchar(max)
EXTERNAL NAME StrConcat.Concatenate;
GO

-- 6) Now you can use it
SELECT dbo.StrConcat(FirstName + ' ' + LastName)
FROM Employees;
GO
```