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.**

CREATE TABLE Persons

(

PersonID int NOT NULL PRIMARY KEY IDENTITY,

FirstName nvarchar(50) NOT NULL,

LastName nvarchar(50) NOT NULL,

SSN int NOT NULL CHECK (SSN BETWEEN 111111111 AND 999999999)

)

CREATE TABLE Accounts

(

ID INT NOT NULL PRIMARY KEY IDENTITY,

PersID INT FOREIGN KEY REFERENCES Persons(PersonID),

Balance money

)

**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 MyBank

GO

CREATE PROCEDURE usp\_GetHigherBalanceFrom (@ChoosenBalance MONEY)

AS

SELECT p.PersonID, p.FirstName + ' ' + p.LastName as [Full name], a.Balance

FROM MyBank.dbo.Accounts a INNER JOIN MyBank.dbo.Persons p

ON a.PersID = p.PersonID

WHERE a.Balance > @ChoosenBalance

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.**

CREATE FUNCTION ufn\_YearsBalance(@sum MONEY, @interestRate FLOAT, @months INT)

RETURNS MONEY

BEGIN

DECLARE @result MONEY, @monthSum MONEY, @yearsMonth INT

SET @yearsMonth = 12

SET @monthSum = (@sum \* @interestRate) / @yearsMonth

SET @result = @monthSum \* @months

RETURN @result

END

GO

--DROP FUNCTION ufn\_YearsBalance

1. **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.**

CREATE PROC usp\_GetMonthInterest(@PersonID INT, @InterestRate FLOAT)

AS

BEGIN

SELECT a.PersID, dbo.ufn\_YearsBalance(a.Balance, @InterestRate, 1)

FROM MyBank.dbo.Accounts a

WHERE a.PersID = @PersonID

END

--TEST

EXEC usp\_GetMonthInterest 5, 5

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

CREATE PROC WithdrawMoney(@AccountID INT, @money MONEY)

AS

BEGIN

UPDATE Accounts

SET Balance = Balance + @money

WHERE ID = @AccountID

END

GO

EXECUTE WithdrawMoney @AccountID = 1, @money = 999

SELECT Accounts.Balance FROM MyBank.dbo.Accounts

=DRAW MONEY=

CREATE PROC DrawMoney(@AccountID INT, @money MONEY)

AS

BEGIN

UPDATE Accounts

SET Balance = Balance - @money

WHERE ID = @AccountID

END

GO

EXEC DrawMoney 1, 200

SELECT \* FROM MyBank.dbo.Accounts

1. **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.**

CREATE TABLE Logs(

LogID INT NOT NULL PRIMARY KEY IDENTITY,

AccountID INT NOT NULL,

OldSum INT NOT NULL,

NewSum INT NOT NULL

)

1. **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'.**
2. **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.**
3. **\* 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**

**…**

1. **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:**

**SELECT StrConcat(FirstName + ' ' + LastName)**

**FROM Employees**