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 DATABASE BankSystem

GO

USE BankSystem

GO

CREATE TABLE Persons(

PersonID int IDENTITY,

FirstName nvarchar(50) NOT NULL,

LastName nvarchar(50) NOT NULL,

SSN nvarchar(10) NOT NULL

CONSTRAINT PK\_PersonID PRIMARY KEY(PersonID)

)

GO

CREATE TABLE Accounts(

AccountID int IDENTITY,

PersonID int,

Balance decimal,

CONSTRAINT PK\_AccountID PRIMARY KEY(AccountID)

)

GO

ALTER TABLE Accounts ADD CONSTRAINT FK\_PersonsAccounts FOREIGN KEY (PersonID) references Persons(PersonID)

GO

INSERT Persons (FirstName, LastName, SSN) VALUES ('Gosho', 'Ivanov', '9006291451');

INSERT Persons (FirstName, LastName, SSN) VALUES ('Gosho2', 'Ivanov2', '8006291451');

INSERT Persons (FirstName, LastName, SSN) VALUES ('Gosho3', 'Ivanov3', '7006291451');

INSERT Accounts (PersonID, Balance) VALUES (1, 100);

INSERT Accounts (PersonID, Balance) VALUES (2, 200);

INSERT Accounts (PersonID, Balance) VALUES (3, 300);

CREATE PROCEDURE PersonFullName

AS

SELECT FirstName + ' ' + LastName AS FullName

FROM Persons

GO

EXECUTE PersonFullName

GO

1. **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.**

CREATE PROCEDURE AccountBalancesInRange

@money decimal

AS

SELECT p.FirstName + ' ' + p.LastName AS FullName, a.Balance

FROM Persons p

INNER JOIN Accounts a

ON p.PersonID = a.PersonID

WHERE a.Balance > @money

GO

EXECUTE AccountBalancesInRange 100

GO

1. **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 CalculateInterests(@sum FLOAT, @interestsRate FLOAT, @numOfMonths INT)

RETURNS FLOAT

AS

BEGIN

DECLARE @newSum FLOAT

SET @newSum = @sum + @sum \* (@interestsRate / 12) \* @numOfMonths

RETURN @newSum

END

SELECT dbo.CalculateInterests(150, 0.1, 5)

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 dbo.ups\_CalculateInterest ( @accountID int, @yearInterest NUMERIC(20,2))

AS

UPDATE Accounts

SET Balance = CONVERT (money, dbo.CalculateInterests(Balance, @yearInterest, 1))

WHERE AccountID = @accountID

GO

EXEC dbo.ups\_CalculateInterest 1,0.4

GO

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

CREATE PROC dbo.usp\_WithdrawMoney (@accountID INT, @money money)

AS

BEGIN TRAN

UPDATE Accounts

SET Balance = Balance - @money

WHERE AccountID = @accountID

COMMIT TRAN

GO

CREATE PROC dbo.usp\_DepositMoney (@accountID INT, @money money)

AS

BEGIN TRAN

UPDATE Accounts

SET Balance = Balance + @money

WHERE AccountID = @accountID

COMMIT TRAN

GO

EXEC dbo.usp\_WithdrawMoney 2, 40

EXEC dbo.usp\_WithdrawMoney 1, 30

EXEC dbo.usp\_DepositMoney 1, 10

GO

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 IDENTITY,

OldSum money NOT NULL,

NewSum money NOT NULL,

AccountID INT NOT NULL,

CONSTRAINT PK\_LogID PRIMARY KEY(LogID),

CONSTRAINT FK\_Logs\_Accounts

FOREIGN KEY (AccountID)

REFERENCES Accounts(AccountID)

)

GO

CREATE TRIGGER tr\_AccountsUpdate ON Accounts FOR UPDATE

AS

INSERT INTO Logs (OldSum, NewSum, AccountID)

SELECT d.Balance,

i.Balance,

d.AccountID

FROM deleted AS d

JOIN inserted AS i

ON d.AccountID = i.AccountID

GO

EXEC dbo.usp\_DepositMoney 1, 1000

GO

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'.**

CREATE FUNCTION CheckIfHasLetters (@word nvarchar(20), @letters nvarchar(20))

RETURNS BIT

AS

BEGIN

DECLARE @lettersLen int = LEN(@letters),

@matches int = 0,

@currentChar nvarchar(1)

WHILE(@lettersLen > 0)

BEGIN

SET @currentChar = SUBSTRING(@letters, @lettersLen, 1)

IF(CHARINDEX(@currentChar, @word, 0) > 0)

BEGIN

SET @matches += 1

SET @lettersLen -= 1

END

ELSE

SET @lettersLen -= 1

END

IF(@matches >= LEN(@word) OR @matches >= LEN(@letters))

RETURN 1

RETURN 0

END

GO

CREATE FUNCTION NamesAndTowns(@letters nvarchar(20))

RETURNS @ResultTable TABLE

(

Name varchar(50) NOT NULL

)

AS

BEGIN

INSERT INTO @ResultTable

SELECT LastName FROM Employees

INSERT INTO @ResultTable

SELECT FirstName FROM Employees

INSERT INTO @ResultTable

SELECT towns.Name FROM Towns towns

DELETE FROM @ResultTable

WHERE dbo.CheckIfHasLetters(Name, @letters) = 0

RETURN

END

GO

SELECT \* FROM dbo.NamesAndTowns('oistmiahf')

1. **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.**

DECLARE empCursor CURSOR READ\_ONLY FOR

SELECT a.FirstName, a.LastName, t1.Name, b.FirstName, b.LastName

FROM Employees a

JOIN Addresses adr

ON a.AddressID = adr.AddressID

JOIN Towns t1

ON adr.TownID = t1.TownID,

Employees b

JOIN Addresses ad

ON b.AddressID = ad.AddressID

JOIN Towns t2

ON ad.TownID = t2.TownID

WHERE t1.Name = t2.Name

AND a.EmployeeID <> b.EmployeeID

ORDER BY a.FirstName, b.FirstName

OPEN empCursor

DECLARE @firstName1 NVARCHAR(50)

DECLARE @lastName1 NVARCHAR(50)

DECLARE @town NVARCHAR(50)

DECLARE @firstName2 NVARCHAR(50)

DECLARE @lastName2 NVARCHAR(50)

FETCH NEXT FROM empCursor

INTO @firstName1, @lastName1, @town, @firstName2, @lastName2

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT @firstName1 + ' ' + @lastName1 +

' ' + @town + ' ' + @firstName2 + ' ' + @lastName2

FETCH NEXT FROM empCursor

INTO @firstName1, @lastName1, @town, @firstName2, @lastName2

END

CLOSE empCursor

DEALLOCATE empCursor

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

CREATE TABLE UsersTowns (

ID INT IDENTITY,

FullName NVARCHAR(50),

TownName NVARCHAR(50)

)

INSERT INTO UsersTowns

SELECT e.FirstName + ' ' + e.LastName, t.Name

FROM Employees e

INNER JOIN Addresses a

ON a.AddressID = e.AddressID

INNER JOIN Towns t

ON t.TownID = a.TownID

GROUP BY t.Name, e.FirstName, e.LastName

DECLARE @name NVARCHAR(50)

DECLARE @town NVARCHAR(50)

DECLARE empCursor1 CURSOR READ\_ONLY FOR

SELECT DISTINCT ut.TownName

FROM UsersTowns ut

OPEN empCursor1

FETCH NEXT FROM empCursor1

INTO @town

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT @town

DECLARE empCursor2 CURSOR READ\_ONLY FOR

SELECT ut.FullName

FROM UsersTowns ut

WHERE ut.TownName = @town

OPEN empCursor2

FETCH NEXT FROM empCursor2

INTO @name

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT '-->' + @name

FETCH NEXT FROM empCursor2 INTO @name

END

CLOSE empCursor2

DEALLOCATE empCursor2

FETCH NEXT FROM empCursor1 INTO @town

END

CLOSE empCursor1

DEALLOCATE empCursor1

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:**

DECLARE @name nvarchar(MAX);

SET @name = N'';

SELECT @name+=e.FirstName+N','

FROM Employees e

SELECT LEFT(@name,LEN(@name)-1);