USE [master]

GO

CREATE DATABASE [PersonsAccounts]

CONTAINMENT = NONE

ON PRIMARY

( NAME = N'PersonsAccounts', FILENAME = N'E:\Microsoft SQL Server\MSSQL12.SQLEXPRESS\MSSQL\DATA\PersonsAccounts.mdf' , SIZE = 5120KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )

LOG ON

( NAME = N'PersonsAccounts\_log', FILENAME = N'E:\Microsoft SQL Server\MSSQL12.SQLEXPRESS\MSSQL\DATA\PersonsAccounts\_log.ldf' , SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)

GO

ALTER DATABASE [PersonsAccounts] SET COMPATIBILITY\_LEVEL = 120

GO

IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))

begin

EXEC [PersonsAccounts].[dbo].[sp\_fulltext\_database] @action = 'enable'

end

GO

ALTER DATABASE [PersonsAccounts] SET ANSI\_NULL\_DEFAULT OFF

GO

ALTER DATABASE [PersonsAccounts] SET ANSI\_NULLS OFF

GO

ALTER DATABASE [PersonsAccounts] SET ANSI\_PADDING OFF

GO

ALTER DATABASE [PersonsAccounts] SET ANSI\_WARNINGS OFF

GO

ALTER DATABASE [PersonsAccounts] SET ARITHABORT OFF

GO

ALTER DATABASE [PersonsAccounts] SET AUTO\_CLOSE OFF

GO

ALTER DATABASE [PersonsAccounts] SET AUTO\_SHRINK OFF

GO

ALTER DATABASE [PersonsAccounts] SET AUTO\_UPDATE\_STATISTICS ON

GO

ALTER DATABASE [PersonsAccounts] SET CURSOR\_CLOSE\_ON\_COMMIT OFF

GO

ALTER DATABASE [PersonsAccounts] SET CURSOR\_DEFAULT GLOBAL

GO

ALTER DATABASE [PersonsAccounts] SET CONCAT\_NULL\_YIELDS\_NULL OFF

GO

ALTER DATABASE [PersonsAccounts] SET NUMERIC\_ROUNDABORT OFF

GO

ALTER DATABASE [PersonsAccounts] SET QUOTED\_IDENTIFIER OFF

GO

ALTER DATABASE [PersonsAccounts] SET RECURSIVE\_TRIGGERS OFF

GO

ALTER DATABASE [PersonsAccounts] SET DISABLE\_BROKER

GO

ALTER DATABASE [PersonsAccounts] SET AUTO\_UPDATE\_STATISTICS\_ASYNC OFF

GO

ALTER DATABASE [PersonsAccounts] SET DATE\_CORRELATION\_OPTIMIZATION OFF

GO

ALTER DATABASE [PersonsAccounts] SET TRUSTWORTHY OFF

GO

ALTER DATABASE [PersonsAccounts] SET ALLOW\_SNAPSHOT\_ISOLATION OFF

GO

ALTER DATABASE [PersonsAccounts] SET PARAMETERIZATION SIMPLE

GO

ALTER DATABASE [PersonsAccounts] SET READ\_COMMITTED\_SNAPSHOT OFF

GO

ALTER DATABASE [PersonsAccounts] SET HONOR\_BROKER\_PRIORITY OFF

GO

ALTER DATABASE [PersonsAccounts] SET RECOVERY SIMPLE

GO

ALTER DATABASE [PersonsAccounts] SET MULTI\_USER

GO

ALTER DATABASE [PersonsAccounts] SET PAGE\_VERIFY CHECKSUM

GO

ALTER DATABASE [PersonsAccounts] SET DB\_CHAINING OFF

GO

ALTER DATABASE [PersonsAccounts] SET FILESTREAM( NON\_TRANSACTED\_ACCESS = OFF )

GO

ALTER DATABASE [PersonsAccounts] SET TARGET\_RECOVERY\_TIME = 0 SECONDS

GO

ALTER DATABASE [PersonsAccounts] SET DELAYED\_DURABILITY = DISABLED

GO

USE [PersonsAccounts]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Accounts](

[AccountID] [int] IDENTITY(1,1) NOT NULL,

[PersonID] [int] NOT NULL,

[Balance] [money] NOT NULL,

CONSTRAINT [PK\_Accounts] PRIMARY KEY CLUSTERED

(

[AccountID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Logs](

[LogID] [int] IDENTITY(1,1) NOT NULL,

[AccountID] [int] NOT NULL,

[OldSum] [money] NOT NULL,

[NewSum] [money] NOT NULL,

CONSTRAINT [PK\_Logs] PRIMARY KEY CLUSTERED

(

[LogID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Persons](

[PersonID] [int] IDENTITY(1,1) NOT NULL,

[FirstName] [nvarchar](50) NOT NULL,

[LastName] [nvarchar](50) NOT NULL,

[SSN] [nvarchar](50) NOT NULL,

CONSTRAINT [PK\_Persons] PRIMARY KEY CLUSTERED

(

[PersonID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

SET IDENTITY\_INSERT [dbo].[Accounts] ON

INSERT [dbo].[Accounts] ([AccountID], [PersonID], [Balance]) VALUES (1, 1, 58.0000)

INSERT [dbo].[Accounts] ([AccountID], [PersonID], [Balance]) VALUES (2, 2, 168.0000)

INSERT [dbo].[Accounts] ([AccountID], [PersonID], [Balance]) VALUES (3, 3, 260.0000)

SET IDENTITY\_INSERT [dbo].[Accounts] OFF

SET IDENTITY\_INSERT [dbo].[Persons] ON

INSERT [dbo].[Persons] ([PersonID], [FirstName], [LastName], [SSN]) VALUES (1, N'Tedi', N'Velikova', N'3412456')

INSERT [dbo].[Persons] ([PersonID], [FirstName], [LastName], [SSN]) VALUES (2, N'Ivan', N'Petrov', N'5124616')

INSERT [dbo].[Persons] ([PersonID], [FirstName], [LastName], [SSN]) VALUES (3, N'Kimbo', N'Ananiev', N'6234566')

SET IDENTITY\_INSERT [dbo].[Persons] OFF

ALTER TABLE [dbo].[Accounts] WITH CHECK ADD CONSTRAINT [FK\_Accounts\_Persons] FOREIGN KEY([PersonID])

REFERENCES [dbo].[Persons] ([PersonID])

GO

ALTER TABLE [dbo].[Accounts] CHECK CONSTRAINT [FK\_Accounts\_Persons]

GO

ALTER TABLE [dbo].[Logs] WITH CHECK ADD CONSTRAINT [FK\_Logs\_Accounts] FOREIGN KEY([AccountID])

REFERENCES [dbo].[Accounts] ([AccountID])

GO

ALTER TABLE [dbo].[Logs] CHECK CONSTRAINT [FK\_Logs\_Accounts]

GO

USE [master]

GO

ALTER DATABASE [PersonsAccounts] SET READ\_WRITE

GO

use PersonsAccounts

go

--task 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 proc usp\_PersonsFullnames

as

select FirstName + ' ' + LastName as Fullname

from Persons

go

exec usp\_PersonsFullnames

go

--task 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.

---------------------------------------------------------------------------------------------------

create proc usp\_PersonsWithMoreMoneyThan(@number money)

as

select p.FirstName + ' ' + p.LastName as Name

from Persons p

inner join Accounts a

on p.PersonID = a.PersonID

where a.Balance > @number

go

exec usp\_PersonsWithMoreMoneyThan 50

go

--task 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 dbo.ufn\_CalculateSum(@sum money, @interestRate decimal, @months decimal)

returns money

as

begin

return @sum + @sum \* ((@interestRate \* @months/12)/100)

end

go

select dbo.ufn\_CalculateSum(150, 40, 5) as [New Sum]

go

--task 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.

---------------------------------------------------------------------------------------------------

create proc usp\_GiveInterestRateForOneMonth(@accountId int, @interestRate decimal)

as

update Accounts

set Balance = dbo.ufn\_CalculateSum(Balance, @interestRate, 1)

where AccountID = @accountId

go

exec usp\_GiveInterestRateForOneMonth 3, 30

go

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

-------------------------------------------------------------------------------------------------

create proc usp\_WithdrawMoney(@accountId int, @money money)

as

begin tran

declare @newBalance money

if not exists(select 1 from Accounts where AccountID = @accountId)

begin

raiserror('Invalid id', 16, 1)

rollback tran

return

end

else

begin

select @newBalance = Balance - @money

from Accounts

where AccountID = @accountId

if(@newBalance < 0)

begin

raiserror('No money.', 16, 1)

rollback tran

return

end

else

begin

update Accounts

set Balance = @newBalance

where AccountID = @accountId

commit

end

end

go

create proc usp\_DepositMoney(@accountId int, @money money)

as

begin tran

declare @newBalance money

if not exists(select 1 from Accounts where AccountID = @accountId)

begin

raiserror('Invalid id', 16, 1)

rollback tran

return

end

else

begin

select @newBalance = Balance + @money

from Accounts

where AccountID = @accountId

update Accounts

set Balance = @newBalance

where AccountID = @accountId

commit

end

go

--task 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.

---------------------------------------------------------------------------------------------------

create trigger tr\_AccountsUpdate on Accounts for update

as

insert into Logs(AccountID, OldSum, NewSum)

select d.AccountID, d.Balance, i.Balance

from deleted d

inner join inserted i

on d.AccountID = i.AccountID

go

create trigger tr\_AccountsInsert on Accounts for insert

as

insert into Logs(AccountID, OldSum, NewSum)

select i.AccountID, null, i.Balance

from inserted i

go

create trigger tr\_AccountsDelete on Accounts for delete

as

insert into Logs(AccountID, OldSum, NewSum)

select d.AccountID, d.Balance, null

from deleted d

go

exec usp\_DepositMoney 3, 100

go

--task 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.

---------------------------------------------------------------------------------------------------

use TelerikAcademy

go

select e.EmployeeID ,e.FirstName + e.LastName as EmployeeName, t.TownID, t.Name as TownName

into #TempEmployeesWithTowns

from Employees e

inner join Addresses a

on e.AddressID = a.AddressID

inner join Towns t

on a.TownID = t.TownID

create unique clustered index Idx\_TemEmp on #TempEmployeesWithTowns(EmployeeID)

declare empCursor cursor read\_only for

select EmployeeID, EmployeeName, TownID, TownName

from #TempEmployeesWithTowns

open empCursor

declare @employeeID int, @employeeName varchar(100), @townID int, @townName varchar(50)

fetch next from empCursor into @employeeID, @employeeName, @townID, @townName

create table #TempEmployeeFromSameTownPairs (FirstEmployeeName varchar(100), SecondEmployeeName varchar(100), TownName varchar(50))

while @@FETCH\_STATUS = 0

begin

insert into #TempEmployeeFromSameTownPairs (FirstEmployeeName, SecondEmployeeName, TownName)

select @employeeName, EmployeeName, @townName

from #TempEmployeesWithTowns e

where e.TownID = @townID and e.EmployeeID <> @employeeID

fetch next from empCursor into @employeeID, @employeeName, @townID, @townName

end

close empCursor

deallocate empCursor

select TownName, FirstEmployeeName, SecondEmployeeName

from #TempEmployeeFromSameTownPairs

order by TownName, FirstEmployeeName, SecondEmployeeName

drop table #TempEmployeeFromSameTownPairs

drop table #TempEmployeesWithTowns

go

--task 9.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.

---------------------------------------------------------------------------------------------------

use [TelerikAcademy]

go

select e.FirstName + ' ' + e.LastName as EmployeeName, t.TownID

into #TempEmployeesWithTowns

from Employees e

inner join Addresses a

on e.AddressID = a.AddressID

inner join Towns t

on a.TownID = t.TownID

create index Idx\_TemTown on #TempEmployeesWithTowns(TownID)

declare townCursor cursor read\_only for

select TownID, Name

from Towns

open townCursor

declare @townID int, @townName varchar(50)

fetch next from townCursor into @townID, @townName

while @@FETCH\_STATUS = 0

begin

declare empCursor cursor read\_only for

select EmployeeName from #TempEmployeesWithTowns

where TownID = @townID

open empCursor

declare @employeeName varchar(150), @employeesList varchar(MAX)

set @employeesList = ''

fetch next from empCursor into @employeeName

while @@FETCH\_STATUS = 0

begin

set @employeesList = CONCAT(@employeesList, @employeeName, ', ')

fetch next from empCursor into @employeeName

end

close empCursor

deallocate empCursor

set @employeesList = LEFT(@employeesList, LEN(@employeesList) - 1)

print @townName + ' -> ' + @employeesList

fetch next from townCursor into @townID, @townName

end

close townCursor

deallocate townCursor

drop table #TempEmployeesWithTowns

go