# Directions & Deliverable – Tim Capehart

Complete the following exercises in a single document. Label each question and/or letter clearly. For each excercise, provide screen shot(s) that demonstrate your code is correct and working. Submit your work electronically to Blackboard by the due date specified.

Determine the appropriate data types to use for all attributes and parameters, unless otherwise specified.

(10 pts ea)

1. Create the following database and tables based on the following specifications.

Database: Bank

Customer (CustomerId, FirstName, LastName, E-mail, SSN)

Account (AccountId, CustomerId, Balance)

1. Primary keys should use the self-incrementing IDENTITY property.
2. Strings should use the NVARCHAR(50) type.
3. CustomerId is a FK in Account.
4. All properties should be required except Balance, which uses the MONEY type.
5. SSN should be numerical.

USE master

CREATE DATABASE Bank;

GO

USE Bank

CREATE TABLE Customer(

CustomerID int NOT NULL IDENTITY,

FirstName nvarchar(50) NOT NULL,

LastName nvarchar(50) NOT NULL,

Email nvarchar(50) NOT NULL,

SSN int NOT NULL UNIQUE,

CONSTRAINT PK\_Customer PRIMARY KEY (CustomerID)

);

CREATE TABLE Account(

AccountID int NOT NULL IDENTITY,

CustomerID int NOT NULL,

Balance money NULL

CONSTRAINT PK\_Account PRIMARY KEY (AccountID),

CONSTRAINT FK\_Account\_Customer FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)

);

1. Insert 50 customers into the Customers table using a loop. For SSN, use a 9-digit random number. Research online how to generate the random number. Ensure that SSN does not belong to another customer.

DECLARE @i int = 1

DECLARE @random int

WHILE (@i <= 50)

BEGIN

SET @random = FLOOR(RAND()\*(999999999 - 100000000 + 1) + 100000000)

INSERT INTO Customer(FirstName, LastName, Email, SSN) VAlUES

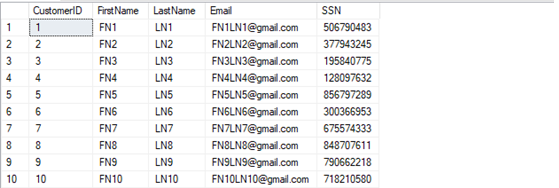
('FN' + CAST(@i AS varchar(2)), 'LN' + CAST(@i AS varchar(2)),

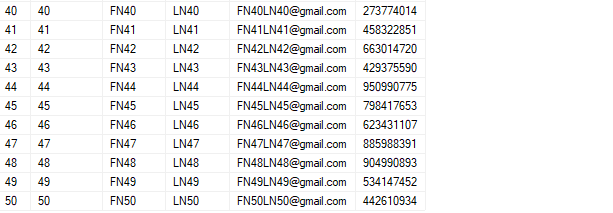
'FN' + CAST(@i AS varchar(2)) + 'LN' + CAST(@i AS varchar(2)) + '@gmail.com', @random)

SET @i = @i + 1

END

SELECT \* FrOM Customer





1. Insert 20 accounts. For the Balance, generate a random decimal.

DECLARE @j int = 1

DECLARE @randCustNum int = 0

DECLARE @randBalance money = 0

WHILE (@j <= 20)

BEGIN

SET @randCustNum = FLOOR(RAND()\*(50) + 1)

SET @randBalance = RAND()\*(20000) + 1

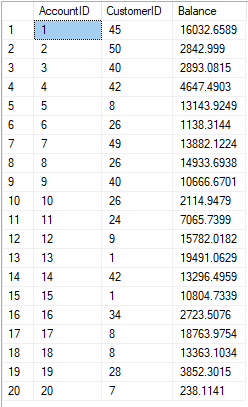
INSERT INTO Account(CustomerID, Balance) VALUES

(@randCustNum, @randBalance)

SET @j = @j + 1

END

SELECT \* FROM Account



1. Create a user stored procedure (usp\_SelectCustomerFullNames) that prints out the customers’ full names in this format (LastName, FirstName).

Demonstrate execution of the stored procedure.

CREATE PROC dbo.usp\_SelectCustomerFullNames

AS

SELECT LastName + ', ' + FirstName AS [Full Name]

FROM Customer

GO

EXEC dbo.usp\_SelectCustomerFullNames



1. Create a user stored procedure (usp\_EmailPromotion) that accepts as a parameter an amount that is used by the stored procedure to produce a list of full names of customers and their e-mail addresses that a promotional email can be sent to.

CREATE PROC dbo.usp\_EmailPromotion(@amount money = 0)

AS

SELECT FirstName + ' ' + LastName AS [Full Name], Email

FROM Customer c JOIN Account a ON c.CustomerID = a.CustomerID

WHERE Balance >= @amount

GO

1. Create a function (fn\_CalculateInterest) that accepts the following parameters:
2. Balance
3. Yearly interest rate
4. Number of months

The function should calculate and return the amount of interest that would result by letting the balance accumulate interest for X amount of months.

CREATE FUNCTION fn\_CalculateInterest(@balance money, @APR float, @months int)

RETURNS money

AS

BEGIN

RETURN (@balance \* @APR / 100 / 12 \* @months)

END

GO

1. Create a stored procedure (usp\_PostSixMonthsInterest) that will add the amount of interest an account has earned after a six month period to the account. The stored procedure should use fn\_CalculateInterest to calculate the interest and should take the following parameters as input.
2. Account Id
3. Yearly interest rate

CREATE PROC dbo.usp\_PostSixMonthsInterest(@account\_id int, @apr float)

AS

UPDATE Account

SET Balance = Balance + dbo.fn\_CalculateInterest (Balance, @apr, 6)

WHERE AccountID = @account\_id

Go

1. Add a stored procedure (usp\_AccountDeposit) that operates in transactions (e.g. BEGIN TRAN, COMMIT TRAN). The stored procedure should add the deposit amount to the account’s balance. The following parameters are expected as input:
2. Account Id
3. Deposit amount

CREATE PROC dbo.usp\_AccountDeposit (@account\_id int, @deposit\_amount money)

AS

BEGIN TRAN

UPDATE Account

SET Balance = Balance + @deposit\_amount

WHERE AccountID = @account\_id

COMMIT TRAN

GO

1. Add a stored procedure (usp\_AccountWithdrawal) that operates in transactions (e.g. BEGIN TRAN, COMMIT TRAN). The stored procedure should add the withdrawal amount to the account’s balance. The following parameters are expected as input:
2. Account Id
3. Withdrawal amount

CREATE PROC dbo.usp\_AccountWithdrawal (@account\_id int, @withdraw\_amount money)

AS

BEGIN TRAN

UPDATE Account

SET Balance = Balance - @withdraw\_amount

WHERE AccountID = @account\_id

COMMIT TRAN

GO

1. Create the following table ClosedAccounts Account (AccountId, CustomerId, Balance). Then, create a stored procedure or function that accepts a customer Id as a parameter and completes the following operations:
2. Moves any open accounts to the ClosedAccounts archive table.
3. Returns the amount that should be returned to the customer for all of their accounts. It may also return a negative amount, which indicates the customer owes the bank money.

CREATE PROC dbo.usp\_MoveClosingAccounts (@customer\_id int, @total money = 0 OUTPUT)

AS

BEGIN

INSERT ClosedAccounts

SELECT \* FROM Account

WHERE CustomerID = @customer\_id

SET @total = (SELECT SUM(Balance) FROM Account

WHERE CustomerID = @customer\_id)

END

GO

DECLARE @totalBalance money

EXEC dbo.usp\_MoveClosingAccounts 45, @totalBalance OUTPUT

SELECT @totalBalance

SELECT \* from ClosedAccounts

