**Instructions**

In this lab you will use MyGuitarShop database to create a stored procedure, two functions, and a trigger. You will prepare a Word document that contains your Transact-SQL statements and screenshots that prove that the programming modules work as intended.

1. Create a stored procedure named spInsertProduct that inserts a row into the Products table. This stored procedure should accept five parameters. One parameter for each of these columns: CategoryID, ProductCode, ProductName, ListPrice, and DiscountPercent. (Note that ProductID is an identity column.) This stored procedure should set the Description column to an empty string, and it should set the DateAdded column to the current date. If the value for the ListPrice column is a negative number, the stored procedure should raise an error that indicates that this column doesn’t accept negative numbers. Similarly, the procedure should raise an error if the value for the DiscountPercent column is a negative number. Code at least two EXEC statements that test this procedure, one with correct parameters, and one that causes an error. Run SELECT query that shows the inserted row(s). In the Word document include all your statements and screenshots from the execution.

IF OBJECT\_ID('spInsertProduct') IS NOT NULL

DROP PROC spInsertProduct;

GO

CREATE PROC spInsertProduct

@CategoryID int, @ProductCode varchar(10),

@ProductName varchar(255), @ListPrice money,

@DiscountPercent money

AS

IF @ListPrice < 0

THROW 50001, 'ListPrice cannot be a negative number!.', 1;

IF @DiscountPercent < 0

THROW 50001, 'DiscountPercent cannot be a negative number!.', 1;

IF EXISTS(SELECT \* FROM Categories WHERE CategoryID = @CategoryID)

INSERT Products

VALUES (@CategoryID, @ProductCode,

@ProductName, '', @ListPrice,

@DiscountPercent, GETDATE());

ELSE

THROW 50001, 'Not a valid CategoryID!', 1; -- creates a new exception to throw the error to

-- run with correct input: 1

EXEC spInsertProduct

1,'fender','Fender Telecaster', 599.99, 20.00;

SELECT \* FROM Products -- added a row of data

-- run with incorrect ListPrice (negative number)

EXEC spInsertProduct

1,'fender','Fender Telecaster', -599.99, 20.00;

-- run with incorrect DiscountPercent (negative number)

EXEC spInsertProduct

1,'fender','Fender Telecaster', 599.99, -20.00;

2. Create two functions:

a. The first function is named fnDiscountPrice that calculates the discount price of an item in the OrderItems table (discount amount subtracted from item price). To do that, this function should accept one parameter for the item ID, and it should return the value of the discount price for that item.

b. The second function is named fnItemTotal and calculates the total amount of an item in the OrderItems table (discount price multiplied by quantity). To do that, this function should accept one parameter for the item ID, it should use the fnDiscountPrice function that you just created, and it should return the value of the total for that item.

Call both functions, and include in the Word document all Transact-SQL statements and execution screenshots.

-- create a function that calculates discount price of an item

-- in the OrderItems table with a given ItemID.

create function fnDiscountPrice

(@ItemID int)

returns money

begin

return(select ItemPrice - DiscountAmount

from OrderItems

where ItemID = @ItemID);

end

--Test it

print dbo.fnDiscountPrice(5) -- 839.30

--make it pretty

print format(dbo.fnDiscountPrice(5), 'c') -- $839.30

-- check it

select (ItemPrice - DiscountAmount)

from OrderItems

where ItemID = 5 -- 839.30

-- create function that calculates the total amount of an item in the OrderItems table with a given itemID.

-- idea: Call the previous function to calculate discount price and multiply by quantity to get total item amount

create function fnItemTotal

(@ItemID int)

returns money

begin

return (select (dbo.fnDiscountPrice(ItemID) \* Quantity)

from OrderItems

where ItemID = @ItemID);

end

--Test it

print dbo.fnItemTotal(5) -- 1678.60

--make it pretty

print format(dbo.fnItemTotal(5), 'c') -- $1,678.60

-- check it

select ((ItemPrice - DiscountAmount) \* Quantity)

from OrderItems

where ItemID = 5 -- 1678.60

3. Create a trigger named Products\_UPDATE that checks the new value for the DiscountPercent column of the Products table. This trigger should raise an appropriate error if the discount percent is greater than 100 or less than 0. If the new discount percent is between 0 and 1, this trigger should modify the new discount percent by multiplying it by 100. That way, a discount percent of .2 becomes 20. Test this trigger with an appropriate UPDATE statement.

IF OBJECT\_ID('Products\_UPDATE') IS NOT NULL

DROP TRIGGER Products\_UPDATE;

GO

CREATE TRIGGER Products\_UPDATE

ON Products

AFTER UPDATE

AS

IF EXISTS (SELECT \* FROM Products WHERE DiscountPercent < 0)

BEGIN;

THROW 50001, 'Error - DiscountPercent cannot be < 0.', 1;

END;

IF EXISTS (SELECT \* FROM Products WHERE DiscountPercent > 100)

BEGIN;

THROW 50001, 'Error - DiscountPercent cannot be > 100.', 1;

END;

IF EXISTS (SELECT \* FROM Products WHERE DiscountPercent < 1)

BEGIN;

UPDATE Products

SET DiscountPercent = (SELECT DiscountPercent FROM inserted) \* 100

WHERE ProductID IN (SELECT ProductID FROM inserted)

END;

-- Testing the Trigger by UPDATE statement:

UPDATE Products

SET DiscountPercent = -30 -- DiscountPercent < 0

WHERE ProductID = 1;

UPDATE Products

SET DiscountPercent = 101 -- DiscountPercent > 100

WHERE ProductID = 1;

UPDATE Products

SET DiscountPercent = 0.42 – DiscountPercent < 1

WHERE ProductID = 1;

-- Check it

select \* from Products