# Directions & Deliverable (10 pts ea) – Tim Capehart

Execute the NewEmployeeProjects.sql from the previous week in a new query window. Then, complete the following exercises in a single document. Label each question and/or letter clearly. For each exercise, provide screen shot(s) of your output. Submit your work electronically to Blackboard by the due date specified.

The following exercises utilize the ***MyGuitarShop*** database (12.5 pts ea)

1. Write a script that creates and calls a stored procedure named spInsertCategory. First, code a statement that creates a procedure that adds a new row to the Categories table. To do that, this procedure should have one parameter for the category name.

Code at least two EXEC statements that test this procedure. (Note that this table doesn’t allow duplicate category names.)

GO

CREATE PROC dbo.sp\_InsertCategory(@categoryName varchar(255))

AS

BEGIN

INSERT Categories (CategoryName)

VALUES (@categoryName)

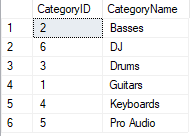
END

GO

EXEC sp\_InsertCategory 'Pro Audio'

EXEC sp\_InsertCategory 'DJ'

SELECT \* from Categories



1. Write a script that creates and calls a function 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.

CREATE FUNCTION dbo.fn\_DiscountPrice (@item\_id int)

RETURNS money

AS

BEGIN

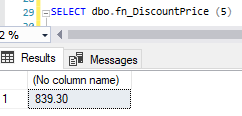
RETURN

(SELECT (ItemPrice - DiscountAmount) FROM OrderItems

WHERE ItemID = @item\_id)

END

GO



1. Write a script that creates and calls a function named fnItemTotal that 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 DiscountPrice function that you created in exercise 2, and it should return the value of the total for that item.

SELECT \* from OrderItems

GO

CREATE FUNCTION dbo.fn\_ItemTotal (@item\_id int)

RETURNS money

AS

BEGIN

DECLARE @discountPrice money

SET @discountPrice = dbo.fn\_DiscountPrice(@item\_id)

RETURN @discountPrice \*

(SELECT Quantity FROM OrderItems

WHERE ItemID = @item\_id)

END

GO

SELECT dbo.fn\_ItemTotal(5)



1. Write a script that creates and calls 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.

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.

CREATE PROC dbo.sp\_InsertProduct(

@category\_id int,

@product\_code varchar(10),

@product\_name varchar(255),

@list\_price money,

@discount\_percent money)

AS

BEGIN

IF (@list\_price < 0)

BEGIN

RAISERROR('List Price cannot be negative', 16, 1)

RETURN

END

IF (@discount\_percent < 0)

BEGIN

RAISERROR('Discount Percent cannot be negative', 16, 1)

RETURN

END

INSERT Products (CategoryID, ProductCode, ProductName, Description, ListPrice, DiscountPercent, DateAdded)

VALUES (@category\_id, @product\_code, @product\_name, '', @list\_price, @discount\_percent, GETDATE())

END

GO

EXEC dbo.sp\_InsertProduct 3, 'testCode1', 'testName1', 99.99, 20; -- good

EXEC dbo.sp\_InsertProduct 2, 'testCode2', 'testName2', 299.99, -25; -- bad

EXEC dbo.sp\_InsertProduct 1, 'testCode3', 'testName3', -299.99, 30; -- bad

1. Write a script that creates and calls a stored procedure named spUpdateProductDiscount that updates the DiscountPercent column in the Products table. This procedure should have one parameter for the product ID and another for the discount percent.

If the value for the DiscountPercent column is a negative number, the stored procedure should raise an error that indicates that the value for this column must be a positive number.

Code at least two EXEC statements that test this procedure.

CREATE PROC dbo.sp\_UpdateProductDiscount (@product\_id int, @discount\_percent money)

AS

BEGIN

IF (@discount\_percent < 0)

BEGIN

RAISERROR('Discount Percent cannot be negative', 16, 1)

RETURN

END

UPDATE Products

SET DiscountPercent = @discount\_percent

WHERE ProductID = @product\_id

END

GO

EXEC dbo.sp\_UpdateProductDiscount 1, 50; -- good

EXEC dbo.sp\_UpdateProductDiscount 2, -25; -- bad

1. 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.

CREATE TRIGGER dbo.tr\_Products\_UPDATE ON Products FOR UPDATE

AS

BEGIN

IF(EXISTS(SELECT DiscountPercent FROM inserted WHERE DiscountPercent < 0) OR

EXISTS(SELECT DiscountPercent FROM inserted WHERE DiscountPercent > 100))

BEGIN

RAISERROR('DiscountPercent must be between 0 and 100', 16, 1);

ROLLBACK TRAN

RETURN

END

IF(EXISTS(SELECT DiscountPercent FROM inserted WHERE DiscountPercent > 0) AND

EXISTS(SELECT DiscountPercent FROM inserted WHERE DiscountPercent < 1))

BEGIN

UPDATE Products

SET DiscountPercent = DiscountPercent \* 100

WHERE DiscountPercent < 1 AND DiscountPercent > 0

END

END

GO

EXEC dbo.sp\_UpdateProductDiscount 1, .1 -- good

EXEC dbo.sp\_UpdateProductDiscount 2, 300 -- bad

SELECT \* FROM Products

1. Create a trigger named Products\_INSERT that inserts the current date for the DateAdded column of the Products table if the value for that column is null.

Test this trigger with an appropriate INSERT statement.

CREATE TRIGGER dbo.tr\_Products\_INSERT ON Products FOR INSERT

AS

BEGIN

IF(EXISTS(SELECT DateAdded FROM inserted WHERE DateAdded IS NULL))

BEGIN

UPDATE Products

SET DateAdded = GETDATE()

WHERE DateAdded IS NULL

END

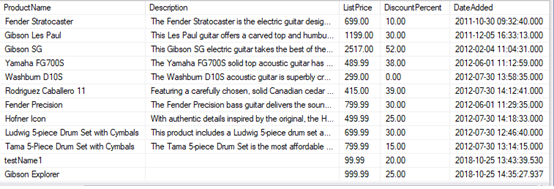
END

GO

INSERT INTO Products (CategoryID, ProductCode, ProductName, Description, ListPrice, DiscountPercent, DateAdded)

VALUES (1, 'explorer', 'Gibson Explorer', '', 999.99, 25, NULL)

SELECT \* FROM Products



1. Create a table named ProductsAudit. This table should have all columns of the Products table, except the Description column. Also, it should have an AuditID column for its primary key, and the DateAdded column should be changed to DateUpdated.

Create a trigger named Products\_UPDATE. This trigger should insert the old data about the product into the ProductsAudit table after the row is updated. Then, test this trigger with an appropriate UPDATE statement.

CREATE TABLE ProductsAudit(

AuditID int IDENTITY,

ProductID int NOT NULL,

CategoryID int NOT NULL,

ProductCode varchar(10) NOT NULL UNIQUE,

ProductName varchar(255) NOT NULL,

ListPrice money NOT NULL,

DiscountPercent money NOT NULL DEFAULT 0.00,

DateUpdated datetime NOT NULL,

CONSTRAINT PK\_ProductsAudit PRIMARY KEY (AuditID)

);

GO

CREATE TRIGGER dbo.tr\_Product\_UPDATE ON Products AFTER UPDATE

AS

BEGIN

INSERT INTO ProductsAudit(ProductID, CategoryID, ProductCode, ProductName, ListPrice, DiscountPercent, DateUpdated)

SELECT ProductID, CategoryID, ProductCode, ProductName, ListPrice, DiscountPercent, GETDATE() FROM deleted

END

UPDATE Products

SET DiscountPercent = 30

WHERE ProductID = 1

SELECT \* FROM ProductsAudit

