

```
/*
1.
10 pts
Write a simple trigger that only prints a message stating that a row in the
customers table has been updated and that fires only upon the execution of an
update statement on the customers table. Include a statement that will fire your
trigger and also a statement to drop your trigger.
SELECT * FROM Northwind.dbo.Customers
*/
-- write your SQL statements here:
----- Create Rows For testing -----

DROP PROCEDURE uspInsertCustomersRow
GO
CREATE PROCEDURE uspInsertCustomersRow
    @CustomerID nchar(5),
    @CompanyName nvarchar(40),
    @ContactName nvarchar(30) = NULL,
    @ContactTitle nvarchar(30) = NULL,
    @Address nvarchar(60) = NULL,
    @City nvarchar(15) = NULL,
    @Region nvarchar(15) = NULL,
    @PostalCode nvarchar(10) = NULL,
    @Country nvarchar(15) = NULL,
    @Phone nvarchar(24) = NULL,
    @Fax nvarchar(24) = NULL

AS

/*
Created By: Chris Singleton
Date: 02/26/2017
Creates a new row for Northwind.Customers database.

*/
INSERT Customers (CustomerID, CompanyName, ContactName, ContactTitle,
[Address], City
                ,Region, PostalCode, Country, Phone, Fax)
VALUES (@CustomerID, @CompanyName, @ContactName, @ContactTitle, @Address
        ,@City, @Region, @PostalCode, @Country, @Phone, @Fax)
RETURN @@Identity
-- end sproc
```

```

----- Call the Stored Procedure uspInsertOrder -----
--Note: Populates the new row. Please use a different CustomerID each time.
-- the call:

DECLARE @Ret int

--Note: Phone is deliberately not correct with a new distinct CustomerID.
EXEC @Ret = uspInsertCustomersRow 'alfkq', 'Alfreds Futterkiste', 'Maria Anders'
                                     , 'Sales Representative', 'Obere Str. 57'
                                     , 'Berlin', 'European', '12209', 'Germany'
                                     , '030-0074320', '030-0076545';

If @ret = 0
    PRINT 'error!';
else
    PRINT 'OrderId entered: ' + CAST(@ret AS varchar);
GO

----- Drop Trigger: trgCustomersUpdReminder -----

IF EXISTS (SELECT name FROM sysobjects
            WHERE name = 'trgCustomersUpdReminder' AND type = 'TR')
    DROP TRIGGER trgCustomersUpdReminder
GO

----- Customers Update Trigger -----

CREATE TRIGGER trgCustomersUpdReminder
ON Customers
FOR UPDATE
AS
--Print out the message in messages:
PRINT 'A row was just updated in the Northwind Customers Table.'
-- Note: Results pane print (below) can also do.
--SELECT 'A row was just updated in the Northwind Customers Table.' AS
'Customer's Table Updated';

GO

----- Update the Row that was created -----

UPDATE Northwind.dbo.Customers
SET Phone = '030-0074321'
,ContactName = 'Maria Anders'
WHERE CustomerID = 'alfkq'

--Checking...
SELECT * FROM Northwind.dbo.Customers
WHERE CustomerID = 'alfkq'
ORDER BY CustomerID

```

--===== Delete Unnecessary Rows =====

```
GO
DELETE TOP (1) Northwind.dbo.Customers
      WHERE CustomerID = 'alfkq'
GO
```

```
--Checking...
SELECT TOP 5 CustomerID
FROM [Northwind].[dbo].[Customers]
WHERE CustomerID LIKE 'alfk%'
GO
```

```
/*
2.
10 pts
Create a trigger that will run instead of the Update statement on the Suppliers
table. It will fire when an Update statement is executed against the Suppliers
table and will have an error message saying: "Updating information in this table
is not allowed"
```

Include a statement that will fire your trigger and also a statement to drop your trigger. (Hints: There is such a thing as an "instead of trigger". See the demo file for an example! You can use RaiseError for the error message.)

```
*/
-- write your SQL statements here:

--===== Drop Trigger: trgDisallowSuppliersUpd =====
-- Disallows anything to update in the Suppliers table.
```

```
IF EXISTS (SELECT name FROM sysobjects
      WHERE name = 'trgDisallowSuppliersUpd' AND type = 'TR')
  DROP TRIGGER trgDisallowSuppliersUpd
GO
```

```
CREATE TRIGGER trgDisallowSuppliersUpd
ON Northwind.dbo.Suppliers
FOR UPDATE
AS
      RAISERROR ('Updating information in this table is not allowed', 10,1)
ROLLBACK
```

```
GO
```

----- Testing with Update Statement -----

```
SELECT TOP 10 SupplierID, City
FROM [dbo].[Suppliers]
```

--Cause the trigger to initiate.

```
UPDATE Northwind.dbo.Suppliers
SET City = 'LUndon'
WHERE SupplierID = 1
```

/*

3.

15 pts

Northwind management is concerned that someone is updating and deleting products without following proper procedures.

1. You are asked to provide a log of all future UPDATES and DELETES performed on the products table. Unfortunately the products table does not store this information. You decide to create a log table and a trigger that will load this log table with the productid, the date and the user of the deleted or updated product.

Note: Here's how to find the user!! There is an example in the demo file in the TRIGGER trgCheckSalaryCap however it is in the part I did not present on the video since I had to end the video and thought that 4 videos would be too much for everyone. It's actually pretty easy. There is a user function that retrieves the current user. It's called "user". The user can be obtained like this:

```
Declare @User varchar(20)
Select @User = System_User
```

Include a statement that will fire your trigger and a statement to drop your trigger.

*/

-- write your SQL statements here:

--Checking table...

```
SELECT TOP 10 * FROM [Northwind].[dbo].[Products]
GO
```

```
SELECT * FROM [Northwind].[dbo].[Products] ORDER BY SupplierID
GO
```

----- Create the Log Table -----

--This is where the log will be stored.

```
CREATE TABLE Productslogtbl
(Eventtime DATETIME not null,
Eventtype VARCHAR(20) not null,
CustID varchar(5),
currUser varchar(20) );
GO
```

----- Drop Trigger: trgProductsLog -----

```
IF EXISTS (SELECT name FROM sysobjects
           WHERE name = 'trgProductsLog' AND type = 'TR')
DROP TRIGGER trgProductsLog
GO
```

----- Create Trigger trgProductsLog -----

```
CREATE TRIGGER trgProductsLog
ON Products
--on Customers
FOR update
AS
BEGIN
    SET NOCOUNT ON
    INSERT INTO Productslogtbl
        SELECT current_timestamp, 'Updated', ProductID, SYSTEM_USER
        FROM deleted;
    SET NOCOUNT OFF
END;
```

----- Insert A New Row Into Products -----

```
SELECT * FROM Products WHERE ProductID = 63;
INSERT INTO Products (ProductName, SupplierID, CategoryID, QuantityPerUnit
                    ,UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel
                    ,Discontinued)
VALUES ('Chai2', 1, 1, '10 boxes x 20 bags', 15.00, 35, 0, 10, 0 );
GO
```

----- Update Value in Products Table -----

/* now test out the trigger by executing an update on the table: */

```
UPDATE Products
SET ProductName = 'Chai Vanilla'
WHERE ProductID = 63;
GO
```

```
/*
```

```
4.
```

```
15 pts
```

In our demo file Cursors.sql there is a stored procedure called "Rebuild_All_Indexes" near the end of that file. Use this stored procedure as a model to:

```
*/
```

```
--===== Demo File Cursors.sql =====
```

```
/*
```

```
CREATE PROCEDURE Rebuild_All_Indexes
```

```
AS
```

```
    DECLARE csrTableNames cursor
```

```
    FOR
```

```
        SELECT name FROM sys.Objects WHERE type = 'U'
```

```
    OPEN csrTableNames
```

```
    -- Create a variable to capture the table names
```

```
    DECLARE @name SYSNAME
```

```
    -- Load a name into the variable
```

```
    FETCH next FROM csrTableNames INTO @name
```

```
    WHILE @@fetch_status = 0
```

```
        BEGIN
```

```
            if @name like 'Customer%'
```

```
                Begin
```

```
                    print @name
```

```
                    -- Use alter index to rebuild all of the indexes of a table
```

```
                    EXECUTE ('ALTER INDEX ALL ON ' + @name + ' REBUILD')
```

```
                End
```

```
                -- Load the next name into the variable
```

```
                FETCH next FROM csrTableNames INTO @name
```

```
        END
```

```
    CLOSE csrTableNames
```

```
    DEALLOCATE csrTableNames
```

```
-- End of Rebuild_All_Indexes
```

```
-- now call the sproc to rebuild all your indexes
```

```
Exec Rebuild_All_Indexes
```

```
*/
```

```
/*
```

1. Create a stored procedure called "uspListAllUserTables" that will print the names of

all tables in the sys.Objects table that are user tables (ie., type = 'U').

(Hint: you will not need to return a table object - just using the print statement within

the sproc will print out the tables)

2. Include a statement to execute your stored procedure and be sure to test it to ensure it works.

```
*/
```

```
-- write your SQL statements here:
```

----- Drop the Stored Procedure: uspListAllUserTables -----

```
IF EXISTS (SELECT * FROM sys.objects WHERE object_id =  
OBJECT_ID(N'[dbo].[uspListAllUserTables]') AND type in (N'P', N'PC'))  
DROP PROCEDURE [dbo].[uspListAllUserTables]  
GO
```

----- Create Stored Procedure: uspListAllUserTables -----

```
CREATE PROCEDURE uspListAllUserTables  
AS  
/*  
CREATE BY: Chris Singleton  
Date: 03/01/2017  
Prints out a list of all table names from the current user's database in SQL  
Server.  
*/  
DECLARE csrPrintAllUserTableNames cursor  
FOR  
    SELECT NAME FROM sys.Objects WHERE type = 'U'  
OPEN csrPrintAllUserTableNames  
    -- Create a variable to capture the table names  
    DECLARE @TableName SYSNAME  
    -- Load a name into the variable  
    FETCH NEXT FROM csrPrintAllUserTableNames INTO @TableName  
    WHILE @@fetch_status = 0  
    BEGIN  
        --Print all the table names using the while loop.  
        PRINT @TableName  
        -- Load the next name into the variable  
        FETCH NEXT FROM csrPrintAllUserTableNames INTO @TableName  
    END  
CLOSE csrPrintAllUserTableNames  
DEALLOCATE csrPrintAllUserTableNames  
-- End of Rebuild_All_Indexes  
  
EXEC uspListAllUserTables  
  
-- now call the sproc to rebuild all your indexes  
EXEC Rebuild_All_Indexes
```