# CIS276 Lab 8 (50 points)

## Introduction

This assignment, using Transact SQL and SQL Server 2008, is similar to the work done in Lab7 with PL/SQL and Oracle. This assignment focuses on embedded SQL, stored procedures and triggers. Please note that because of differences in SQL Server, some changes are necessary in how you handle the processing. Check posted material for examples. Also, refer to the HELP files on Transact SQL available in Microsoft SQL Server Management Studio (SQL Server 2012).

This lab uses the same tables as the previous labs.  You will need to create and load them into your SQL Server account area (see pages 4-7 of this document for code you may use to do this). The names of the relations are:

**SALESPERSONS (Empid, Ename, Rank, Salary)   
CUSTOMERS (Custid, Cname, Credit)   
INVENTORY (Partid, Description, Stockqty, Reorderpnt, Price)   
ORDERS (Orderid, Empid, Custid, Salesdate)   
ORDERITEMS(Orderid, Detail, Partid, Qty)**

## Submission Notes

* There are seven stored procedures, two triggers, that you will be creating as discussed below in the "Lab Assignment". Along with testing for each object, you will be creating a testing script.
* It is suggested that you create a separate .sql file for each task as indicated using the file names as shown. This is for your benefit as it keeps the drop-create-test separate until you are finished with everything.
* Then please incorporate all of your modules into \*one\* text (SQL) file called **Lab8.sql**.
* Submit your **Lab8.sql** file into the Lab 8 dropbox by the due date.
* Your submitted file will be copied into SQL Server Management Studio and run to a .txt file which will be graded and returned.
* Testing that will be done using \*my grading\* test script will verify that the correct changes are made to the tables in the database:
  + Invalid Custid
  + Invalid Orderid
  + Invliad Custid / Orderid pairing
  + Invalid Partid
  + Qty that is less than or equal zero
  + Valid Custid, Orderid, Partid and Qty that would result in insufficient stock
  + Valid Custid, Orderid, Partid and Qty for an order with lineitems
  + Valid Custid, Orderid, Partid and Qty for an order with no previous lineitems

## Grading Notes

Turn in **Lab8.sql** that contains the following *(****grading is five points for each item****)*:

1. DROP, CREATE, and test of the procedure, **ValidateCustID**, that validates the entered Custid against the current Custids in the CUSTOMERS table.
2. DROP, CREATE, and test of the procedure, **ValidateOrderID**, that validates the entered Orderid against the current Orderids in the ORDERS table for the entered Custid.
3. DROP, CREATE, and test of the procedure, **ValidatePartID**, that validates the entered Partid against the current Partids in the INVENTORY table.
4. DROP, CREATE, and test of the procedure, **ValidateQty**, that validates the entered Qty is greater than zero.
5. DROP, CREATE, and test of the procedure, **GetNewDetail**, that determines the Detail value for new line item.
6. DROP, CREATE, and test of the UPDATE trigger on INVENTORY table, **InventoryUpdateTrg**.
7. DROP, CREATE, and test of the INSERT trigger on ORDERITEMS table, **OrderitemsInsertTrg**.
8. DROP and CREATE of the procedure that does the transaction processing, **AddLineItem**. Before the transaction does a COMMIT or ROLLBACK, print a statement that says the transaction is committed or rolled back.
9. DROP and CREATE of the procedure, **Lab8proc**, that receives the Custid, Orderid, Partid, and Qty as input parameters and essentially brings all the other parts together:
   1. Print a statement that Lab8proc begins.
   2. EXECUTE the Custid validation procedure and print a line giving the input Custid and a statement that it is valid or invalid.
   3. EXECUTE the Orderid validation procedure and print a line giving the input Orderid value and a statement that it is valid or invalid. Print another line stating the Orderid and Custid values are valid together or not.
   4. EXECUTE the Partid validation procedure and print a line giving the input Partid value and a statement that it is valid or invalid.
   5. EXECUTE the Qty validation procedure and print a line giving the input Qty value and a statement that it is valid or invalid.
   6. If all is good, print a message stating that **Lab8proc** validations were good so the transaction continues and then EXECUTE the add line item procedure; otherwise print a message stating that **Lab8proc** is ending the transaction and do not run the add line item procedure.
10. Testing of Lab8proc (similar to the tests you used for Labs 6 and 7). Please use different data for your tests.

Each of the ten items above is to be preceded by a comment box containing your name, the procedure name, the date, and a brief explanation of the procedure. Please reset the tables if your testing changes them although if your two good tests are at the end you won't need to reset your tables. The grader will always begin with a reset SalesDB so you need NOT submit your code to do that.

## Lab Assignment

For this lab you will write:

* **ValidateCustID**, a procedure that will return a value if the CustID is in the CUSTOMERS table
* **ValidateOrderID**, a procedure that will return a value if the Orderid is valid for the customer
* **ValidatePartID**, a procedure that will return a value if the Partid is in the INVENTORY table
* **ValidateQty**, a procedure that will return a value if the Qty in the new lineitem is less than zero
* **GetNewDetail**, a procedure that will determine the value of the Detail column for a new line item (SQL Server will not allow you to assign a column value to the newly inserted row inside of the trigger)
* **InventoryUpdateTrg**, a trigger on UPDATE for the INVENTORY table
* **OrderitemsInsertTrg**, a trigger on INSERT for the ORDERITEMS table
* **AddLineItem**, a procedure that does the transaction processing (adds the order item). This procedure will call GetNewDetail and perform an INSERT to the ORDERITEMS table. Being the transaction, AddLineItem is where the COMMIT / ROLLBACK occur.
* **Lab8proc**, a procedure that puts all of the above together to produce a solution for Lab8 done in SQL Server. This is a stored procedure that accepts the 4 pieces of input: Custid, Orderid, Partid, and Qty (in that order please). In this module you will validate all the data and do the transaction processing by calling the previously written and tested modules.
* Testing of Lab8proc (similar to the testing you did previously for the Oracle labs 6 and 7).

It is wise to create the above in pieces with testing as you go. Each piece is to contain the DROP, CREATE, and test(s). The final step will requires that you test the entire product. You are to use the names as shown in bold above.

This assignment is similar to but not quite the same as Lab7. You can use Lab7 as a guide and convert your code appropriately for the changes and from PL/SQL to TransactSQL.

As always, it is suggested that you do not wait until the last minute to begin this assignment. Also, please do as much as you can -- some points are always better than no points. Ask questions on the discussion board. After you have read the lab to this point please post to the discussion board that you have read the Lab8 document and are ready to begin coding. Thank you.

/\*

ResetSalesDB

CIS276 @PCC on SQL Server 2012

20080314 vjj

20110526 updated to SQL Server 2008

20150226 updated to SQL Server 2012

\*/

-- USE student account

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

-- DROP Lab8 procedures

-- triggers are not dropped here

-- as they get dropped w/tables

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

IF EXISTS (SELECT name

FROM SYSOBJECTS

WHERE name = 'ValidateCustID')

DROP PROCEDURE ValidateCustID;

GO

IF EXISTS (SELECT name

FROM SYSOBJECTS

WHERE name = 'ValidateOrderID')

DROP PROCEDURE ValidateOrderID;

GO

IF EXISTS (SELECT name

FROM SYSOBJECTS

WHERE name = 'ValidatePartID')

DROP PROCEDURE ValidatePartID;

GO

IF EXISTS (SELECT name

FROM SYSOBJECTS

WHERE name = 'ValidateQty')

DROP PROCEDURE ValidateQty;

GO

IF EXISTS (SELECT name

FROM SYSOBJECTS

WHERE name = 'GetNewDetail')

DROP PROCEDURE GetNewDetail;

GO

--IF EXISTS (SELECT name

-- FROM SYSOBJECTS

-- WHERE name = 'InventoryUpdateTrg')

-- DROP TRIGGER InventoryUpdateTrg;

--GO

--IF EXISTS (SELECT name

-- FROM SYSOBJECTS

-- WHERE name = 'OrderitemsInsertTrg')

-- DROP TRIGGER OrderitemsInsertTrg;

--GO

IF EXISTS (SELECT name

FROM SYSOBJECTS

WHERE name = 'AddLineItem')

DROP PROCEDURE AddLineItem;

GO

IF EXISTS (SELECT name

FROM SYSOBJECTS

WHERE name = 'LAB8proc')

DROP PROCEDURE LAB8proc;

GO

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

-- DROP SalesDB tables

-- triggers are dropped with tables

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

IF EXISTS (SELECT name

FROM SYSOBJECTS

WHERE name = 'ORDERS')

DROP TABLE ORDERS;

GO

IF EXISTS (SELECT name

FROM SYSOBJECTS

WHERE name = 'ORDERITEMS')

DROP TABLE ORDERITEMS;

GO

IF EXISTS (SELECT name

FROM SYSOBJECTS

WHERE name = 'CUSTOMERS')

DROP TABLE CUSTOMERS;

GO

IF EXISTS (SELECT name

FROM SYSOBJECTS

WHERE name = 'INVENTORY')

DROP TABLE INVENTORY;

GO

IF EXISTS (SELECT name

FROM SYSOBJECTS

WHERE name = 'SALESPERSONS')

DROP TABLE SALESPERSONS;

GO

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

-- CREATE SalesDB tables

-- ALTER tables for constraints

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

CREATE TABLE CUSTOMERS

(

custid SMALLINT NOT NULL ,

cname VARCHAR (25) NULL ,

credit VARCHAR (1) NULL

);

GO

CREATE TABLE INVENTORY

(

partid SMALLINT NOT NULL ,

description VARCHAR (10) NULL ,

stockqty SMALLINT NULL ,

reorderpnt SMALLINT NULL ,

price MONEY NULL

);

GO

CREATE TABLE ORDERITEMS

(

orderid SMALLINT NOT NULL ,

detail SMALLINT NOT NULL ,

partid SMALLINT NULL ,

qty SMALLINT NULL

)

GO

CREATE TABLE ORDERS

(

orderid SMALLINT NOT NULL ,

empid SMALLINT NULL ,

custid SMALLINT NULL ,

salesdate DATETIME NULL

);

GO

CREATE TABLE SALESPERSONS

(

empid INT NOT NULL ,

ename VARCHAR (15) NULL ,

rank SMALLINT NULL ,

salary MONEY NULL

);

GO

ALTER TABLE CUSTOMERS WITH NOCHECK ADD

CONSTRAINT PK\_Customers PRIMARY KEY NONCLUSTERED (custid);

GO

ALTER TABLE INVENTORY WITH NOCHECK ADD

CONSTRAINT PK\_Inventory PRIMARY KEY NONCLUSTERED (partid);

GO

ALTER TABLE ORDERITEMS WITH NOCHECK ADD

CONSTRAINT PK\_OrderItems PRIMARY KEY NONCLUSTERED (orderid, detail);

GO

ALTER TABLE ORDERS WITH NOCHECK ADD

CONSTRAINT PK\_Orders PRIMARY KEY NONCLUSTERED (orderid);

GO

ALTER TABLE SALESPERSONS WITH NOCHECK ADD

CONSTRAINT PK\_Salespersons PRIMARY KEY NONCLUSTERED (empid);

GO

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

-- Fill SalesDB tables

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

INSERT INTO CUSTOMERS

SELECT \* FROM SALESDB.DBO.CUSTOMERS;

GO

INSERT INTO SALESPERSONS

SELECT \* FROM SALESDB.DBO.SALESPERSONS;

GO

INSERT INTO INVENTORY

SELECT \* FROM SALESDB.DBO.INVENTORY;

GO

INSERT INTO ORDERS

SELECT \* FROM SALESDB.DBO.ORDERS;

GO

INSERT INTO ORDERITEMS

SELECT \* FROM SALESDB.DBO.ORDERITEMS;

GO

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

-- Make it so and advise

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

BEGIN TRANSACTION

DECLARE @v\_now DATETIME

BEGIN

SET @v\_now = GETDATE()

PRINT '----------------------------';

PRINT 'SalesDB has been initialized';

PRINT @v\_now

PRINT '----------------------------';

COMMIT

END;

-- END TRANSACTION