**SYM400 – Preparation for Triggers assignment**

**Constraints:** PK FK Not-Null, Unique, Check (etc.) *constraints are created at the table level*

**Triggers:** we are creating rules (and apply rules) and to update automatically, we automate the rules (removes human error). *Triggers have a scope of the entire database.* Triggers consist of actions and conditions.

**DB’s** must have at least two rules, especially regarding *consistency* and *data integrity*. Both require constraints and triggers

**Improving performance:** normalization, de-normalization, joins vs. Sub-queries, Select \*.

**Triggers** generally deal with two tables ‘INSERTED’, ‘DELETED’

Updates are essentially treated as the delete table because we delete an old values

-- Create a new database called Projects

-- Create a table that will hold all employees demographic data. Employee Number is the PK

Create database projects

Create Table employees

(

emp\_no Integer Identity (1,1) NOT NULL Primary Key,

emp\_fname Char (20) Not Null,

emp\_lname Char (20) Not Null,

dept\_no Char (4) NULL

)

-- Create a table to hold the departments data. Each emaplyee belongs to only one department, and each department has many employees. one to many relationship!

CREATE Table Department

(

Dept\_No Char(4) Not Null Primary Key,

Dept\_Name Char(25) Not Null,

Location Char (30) Null

)

-- Alter the Employees table to Add an FK constraint for the Dept\_NO referring to the Department (Dept\_No)

Alter table Employees Add Constraint fk\_deptNo Foreign Key (Dept\_No) References Department (Dept\_No)

-- This data base is to keep track of projects. An employee may or may not work on a project. An Employee may work on one or many projects at the same time. Each project might be assigned

-- to one or many employees. Since this is a many to many relationship, we need to break this relationship.

-- Create a Master table that holds our Projects, with Project\_No as PK

Create Table Project

(

Project\_No Char(4) Not Null Primary Key,

Project\_Name Char (20) Not Null,

Budget Float NULL

)

-- Again, the relationship between Employees and Projects is many to many. We need to break this relationship down!

-- To restructure this relationship, create another table that will hold an entry for each PK

Create Table Works\_On

(

Emp\_No Integer Not Null,

Project\_No Char(4) Not Null,

Job Char(15) Null, -- if you are unsure about the data type, or you are willing to have a column that store various data types, use SQL\_Variant as your data type.

Enter\_Date DATE Null

)

-- Create a table to practice the differences between DEFAULT and UNIQUE. UNIQUE (Emaployees & Manager IDs)

-- In the Deparment table, my department IDs are 10,20,30,40,50,60. Ensure that data inserted are only within the same range!

ALTER Table Department Add Constraint CK\_DeptNo CHECK (Dept\_No IN (10,20,30,40,50,60))

-- Alter the Works \_ON table to add FK for the EMP\_No and another FK for the Project\_No

Alter Table Works\_ON Add Constraint fk\_EmpNO FOreign Key (Emp\_No) References EMployees (Emp\_NO)

Alter Table Works\_ON Add Constraint fk\_Project\_NO FOreign Key (Project\_No) References Project (Project\_NO)

-- Create 5 departments

Insert INTO Department

VALUES (10, 'Information Technology', 'Phoenix'),

(20, 'Accounting', 'Las Vegas'),

(30, 'Business', 'Denver'),

(40, 'Human Resources', 'Dallas'),

(50, 'Sales', 'Portland');

-- Insert 3 records to the Employees table.

INSERT INTO employees

VALUES ('Matt','Amen',20),

('Dan','Prime',10),

('Dave','Wolf',40);

-- Create 3 different projects

INSERT INTO Project

VALUES ('P1','Durango','400000'),

('P2','Apollo','950000'),

('P3','XLab','155000')

-- Insert 3 records into the Works\_On table

--this (below) will not run because there is an emp\_no of 4 there is no parent for 4

INSERT INTO Works\_On

VALUES (2, 'P1', 'System Admin', GETDATE()),

(3, 'P2', 'Data Analyst', GETDATE()),

(4, 'P3', 'Project Manager', GETDATE())

SELECT \* FROM employees;--shows that there is no emp\_no == 4

--below is the fix:

INSERT INTO Works\_On

VALUES (2, 'P1', 'System Admin', GETDATE()),

(3, 'P2', 'Data Analyst', GETDATE()),

(1, 'P3', 'Project Manager', GETDATE())

--Problems From class

--Task 1: update the project table and add a column called numEmployeesAssigned (int datatype)

ALTER TABLE Project ADD numEMployeesAssigned int NULL; --not null would force SQL to require data to be added for each of the existing columns

--Task 2: then use the update command to insert values into the new Column to correspond to the current information in the works\_on table

--my attempt: (join is not needed because the update portion calls the projet table thus there is no join neccessary)

UPDATE Project SET numEMployeesAssigned = (SELECT COUNT(DISTINCT (Emp\_No)) FROM Works\_On w JOIN Project p ON w.Project\_No = p.Project\_No WHERE w.Project\_No = p.Project\_No);

--DR. Saleh's Solution:

UPDATE Project SET numEMployeesAssigned = (SELECT COUNT(\*) FROM Works\_On w WHERE w.Project\_No = Project.Project\_No);

SELECT \* FROM Project

--WORKING WITH TRIGGERS

--whenever we assign a new employee to a project, a trigger should fire off an increment 'numEmployeesAssigned' column by 1

--questions to ask: 1) when should the trigger fire off? 2) what will the trigger do upon firing?

--1.)

INSERT INTO Works\_On VALUES (1, 'P1', 'Inspector', GETDATE());

--^ this will fire off AFTER an insert into the works\_on table

--update Project to increment 'numEmployeesAssigned' column by 1

CREATE TRIGGER incrementEmployees

ON works\_on AFTER INSERT AS BEGIN

DECLARE @projNUM char(4)

SELECT @projNum = (SELECT project\_no FROM INSERTED)

UPDATE Project SET numEMployeesAssigned = numEMployeesAssigned + 1 WHERE project\_no = @projNum

END;

--TEST

INSERT INTO Works\_On VALUES (1, 'P1', 'Inspector', GETDATE());

SELECT \* FROM Project;

--write a trigger that will fire off AFTER removing an employee from a project

CREATE TRIGGER decrementEmployees

ON works\_on AFTER DELETE AS BEGIN

DECLARE @projNum char(4)

SELECT @projNum = (SELECT project\_no FROM DELETED)

UPDATE Project SET numEMployeesAssigned = numEMployeesAssigned - 1 WHERE project\_no = @projNum

END;

--Testing the above trigger

DELETE FROM works\_on WHERE Project\_no='P1';

SELECT \* FROM Project

--what if an employee reassined from one project to another

CREATE TRIGGER reassignEmp

ON works\_on AFTER UPDATE

AS IF UPDATE (project\_no)

BEGIN

DECLARE @oldProj char(4), @newProj char(4)

SELECT @oldProj = (SELECT project\_no FROM DELETED)

SELECT @newProj = (SELECT project\_no FROM INSERTED)

UPDATE Project SET numEMployeesAssigned = numEMployeesAssigned - 1 WHERE Project\_No = @oldProj

UPDATE Project SET numEMployeesAssigned = numEMployeesAssigned + 1 WHERE Project\_No = @newProj

END;

UPDATE Works\_On SET project\_no = 'P2' WHERE emp\_no = '3' AND project\_no = 'P3';

--Problems From class

--Task 1: update the project table and add a column called numEmployeesAssigned (int datatype)

ALTER TABLE Project ADD numEMployeesAssigned int NULL; --not null would force SQL to require data to be added for each of the existing columns

--Task 2: then use the update command to insert values into the new Column to correspond to the current information in the works\_on table

--my attempt: (join is not needed because the update portion calls the projet table thus there is no join neccessary)

UPDATE Project SET numEMployeesAssigned = (SELECT COUNT(DISTINCT (Emp\_No)) FROM Works\_On w JOIN Project p ON w.Project\_No = p.Project\_No WHERE w.Project\_No = p.Project\_No);

--DR. Saleh's Solution:

UPDATE Project SET numEMployeesAssigned = (SELECT COUNT(\*) FROM Works\_On w WHERE w.Project\_No = Project.Project\_No);

SELECT \* FROM Project

--WORKING WITH TRIGGERS

--whenever we assign a new employee to a project, a trigger should fire off an increment 'numEmployeesAssigned' column by 1

--questions to ask: 1) when should the trigger fire off? 2) what will the trigger do upon firing?

--1.)

INSERT INTO Works\_On VALUES (1, 'P1', 'Inspector', GETDATE());

--^ this will fire off AFTER an insert into the works\_on table

--update Project to increment 'numEmployeesAssigned' column by 1

CREATE TRIGGER incrementEmployees

ON works\_on AFTER INSERT AS BEGIN

DECLARE @projNUM char(4)

SELECT @projNum = (SELECT project\_no FROM INSERTED)

UPDATE Project SET numEMployeesAssigned = numEMployeesAssigned + 1 WHERE project\_no = @projNum

END;

--TEST

INSERT INTO Works\_On VALUES (1, 'P1', 'Inspector', GETDATE());

SELECT \* FROM Project;

--write a trigger that will fire off AFTER removing an employee from a project

CREATE TRIGGER decrementEmployees

ON works\_on AFTER DELETE AS BEGIN

DECLARE @projNum char(4)

SELECT @projNum = (SELECT project\_no FROM DELETED)

UPDATE Project SET numEMployeesAssigned = numEMployeesAssigned - 1 WHERE project\_no = @projNum

END;

--Testing the above trigger

DELETE FROM works\_on WHERE Project\_no='P1';

SELECT \* FROM Project

--what if an employee reassined from one project to another

CREATE TRIGGER reassignEmp

ON works\_on AFTER UPDATE

AS IF UPDATE (project\_no)

BEGIN

DECLARE @oldProj char(4), @newProj char(4)

SELECT @oldProj = (SELECT project\_no FROM DELETED)

SELECT @newProj = (SELECT project\_no FROM INSERTED)

UPDATE Project SET numEMployeesAssigned = numEMployeesAssigned - 1 WHERE Project\_No = @oldProj

UPDATE Project SET numEMployeesAssigned = numEMployeesAssigned + 1 WHERE Project\_No = @newProj

END;

UPDATE Works\_On SET project\_no = 'P2' WHERE emp\_no = '3' AND project\_no = 'P3';

--TUES Class Continuation on Triggers

--write a trigger called 'audit\_trail'

CREATE TABLE audit\_budget(

projectNumber char(4),

username char(16),

date datetime,

budget\_old float,

budget\_new float

);

--write a trigger that will fire off after we update a budget for a project. The trigger shoul log the proj\_no, userName, date, old\_value, and newValue into the audit\_budget

CREATE TRIGGER audit\_trail ON Project AFTER UPDATE AS IF UPDATE (Budget)

BEGIN

DECLARE @oldBudget float, @newBudget float, @projNo char(4), @userName char(16) = 'Ryan Woodward';

SELECT @oldBudget = (SELECT Budget FROM DELETED)

SELECT @newBudget = (SELECT Budget FROM INSERTED)

SELECT @projNo = (SELECT Project\_No FROM DELETED) --DELETED is qucker than INSERTED in this scenario

INSERT INTO audit\_budget VALUES (@projNo, @username, GETDATE(), @oldBudget, @newBudget)

END;

SELECT \* FROM audit\_budget;

SELECT \* FROM Project

UPDATE Project SET budget = 117117117 WHERE Project\_No = 'P1';

DROP TRIGGER audit\_trail

--update the trigger to include column of budget difference

--had to drop the previous trigger because the update on budget conflicted

ALTER TABLE audit\_budget ADD oldNewBudgetDifference float;

CREATE TRIGGER audit\_trail2 ON Project AFTER UPDATE AS IF UPDATE (Budget)

BEGIN

DECLARE @oldBudget float, @newBudget float, @projNo char(4), @userName char(16) = 'Ryan Woodward'

SELECT @oldBudget = (SELECT Budget FROM DELETED)

SELECT @newBudget = (SELECT Budget FROM INSERTED)

DECLARE @budgetDifference float = @oldBudget - @newBudget

SELECT @projNo = (SELECT Project\_No FROM DELETED) --DELETED is qucker than INSERTED in this scenario

INSERT INTO audit\_budget VALUES (@projNo, @username, GETDATE(), @oldBudget, @newBudget, @budgetDifference)

END;

SELECT \* FROM audit\_budget;

SELECT \* FROM Project

UPDATE Project SET budget = 36363 WHERE Project\_No = 'P3';

--Salvatore's Solution

ALTER TABLE audit\_budget ADD budgetDifference float

ALTER TRIGGER audit\_trail ON Project AFTER UPDATE AS IF UPDATE (Budget)

BEGIN

DECLARE @oldBudget float,@difference float, @newBudget float, @projNo char(4), @userName char(16) = 'Ryan Woodward'

SELECT @oldBudget = (SELECT Budget FROM DELETED)

SELECT @newBudget = (SELECT Budget FROM INSERTED)

SELECT @projNo = (SELECT Project\_No FROM DELETED) --DELETED is qucker than INSERTED in this scenario

if(@oldBudget > @newBudget)

(SELECT @difference = @oldBudget - @newBudget)

else

(SELECT @difference = @newBudget - @oldBudget)

INSERT INTO audit\_budget VALUES (@projNo, @username, GETDATE(), @oldBudget, @newBudget, @budgetDifference)

END;

**ASSIGNMENT NOTES:**

**When and What for the triggers**

**TRIGGERS SQL ASSIGNMENT NEED:**

Create table Products

(

Product\_Code varchar(10),

Product\_Desc varchar(100),

Product\_QOH INT, -- Product Quantity On Hand

Product\_MIN INT, -- Minimum Quantity of the product

Product\_ListPrice FLOAT,

Product\_MinOrder INT, -- this is the minimum quantity for restocking an order

Product\_ReOrder BIT -- this indicates whether the product needs to re-ordered from the vendor (1 = Yes, 0 = No)

)

-- Insert few records

INSERT INTO Products

VALUES (1, 'Power Supplies', 8, 5, 35.00, 25, 0),

(2, 'Hard Disks', 7, 5, 120.00, 50, 0)

-- display the Products using the SELECT statement

SELECT \* from Products

**FROM TRIGGER HW assignment**

**C**REATE TRIGGER orderProduct ON Products AFTER UPDATE AS IF UPDATE (Product\_QOH)

BEGIN

DECLARE @new\_QOH int, @minm\_Qty int, @prod\_Code varchar(10)

SELECT @new\_QOH = (SELECT Product\_QOH FROM INSERTED)

SELECT @minm\_Qty = (SELECT Product\_MIN FROM INSERTED)

SELECT @prod\_Code = (SELECT Product\_Code FROM INSERTED)

IF(@new\_QOH <= @minm\_Qty)

UPDATE Products SET Product\_ReOrder = 1 WHERE @prod\_Code = Product\_Code;

END;

SELECT \* from Products ;

UPDATE Products SET Product\_QOH = Product\_QOH - 1 WHERE Product\_Code = 2;

CREATE TABLE audit\_log(

productID int,

dateOfPriceChange date,

oldPrice float,

newPrice float

);

DROP TRIGGER priceChange

CREATE TRIGGER priceChange ON Products AFTER UPDATE AS IF UPDATE (Product\_ListPrice)

BEGIN

DECLARE @productID int, @oldPrice float, @newPrice float

SELECT @productID = (SELECT Product\_Code FROM DELETED)

SELECT @oldPrice = (SELECT Product\_ListPrice FROM DELETED)

SELECT @newPrice = (SELECT Product\_ListPrice FROM INSERTED)

INSERT INTO audit\_log VALUES (@productID, GETDATE(), @oldPrice, @newPrice)

END;

SELECT \* FROM Products;

UPDATE Products SET Product\_ListPrice = 42 WHERE Product\_Code = 1;

SELECT \* FROM audit\_log;

**NOTES FROM THURS CLASS b4 Triggers DUE**

CREATE TABLE Vendor (  
 vid int,

vName varchar(20),

vCIty varChar(20),

vState varchar(2)  
)

--the trigger should fire off before inserting a new vendor, the trigger will change the case of the vState to upper case for that Vendor

CREATE TRIGGER changeCase ON Vendor AFTER INSERT AS BEGIN --could be 'for insert as' instead of after insert as begin

DECLARE @vendor\_ID int

SELECT @vendor\_ID = (SELECT vid FROM INSERTED)

UPDATE Vendor SET vState = UPPER(vState) WHERE vid = @vendor\_ID

END

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

CREATE TABLE rep(

repNumber char(2) PRIMARY KEY,

lastName char(15),

firstName char(15),

street char(15),

city char(15),

state char(2),

postalCode char(5),

commission float

)

CREATE TABLE Customer(

repNumber char(2) FOREIGN KEY REFERENCES rep(repNumber),

customerNumber char(2),

lastName char(15),

firstName char(15),

street char(15),

city char(15),

state char(2),

postalCode char(5),

balance float

)

INSERT INTO rep VALUES (1, 'Carter', 'Ron', '123 Main St', 'Phoenix', 'AZ', '85399', 0.5)

DROP TRIGGER updateCustomerBalance

DROP TABLE Customer

--Creating triggers from the above two tables

--the trigger should fire off after INSERTING a new customer the trigger will update the customer's balance to add his rep's commission to it

CREATE TRIGGER updateCustomerBalance ON Customer AFTER INSERT AS BEGIN

DECLARE @repNum char(2), @repCommission float

SELECT @repNUM = (SELECT repNumber FROM INSERTED)

SELECT @repCommission = (SELECT commission FROM rep WHERE repNumber = @repNum)

UPDATE Customer SET balance = balance + balance \* @repCommission WHERE repNumber = @repNum

END

INSERT INTO Customer VALUES(1, 17, 'Lindemann', 'Till', '180 Blumen St', 'Munich', 'BV', '98233', 1000)

SELECT \* FROM Customer