Lecture 9:Triggers

**Triggers**

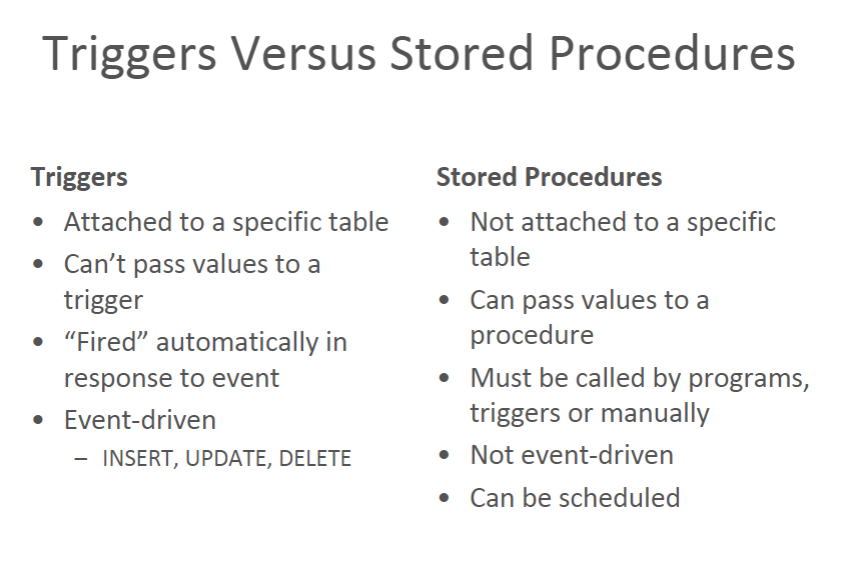
* Examples uses
* CREATE TRIGGER syntax

**Examples of trigger code**

* Deleting a trigger
* Metadata about triggers

**Triggers**

* Supported by most RDBMS
* A trigger is a special stored procedure attached to a specific table
* Managed by DBMS



**Trigger Examples**

* Reduce inventory when an item is sold
* Permit changes to employee records during business hours only
* Automatically ensure the salary is within range for the job when an employee receives a job change
* Audit trail / activity logging
  + Record changes to bank account
* Withdrawal, deposit, modifying customer profile

**Why Use Triggers?**

* Enforce business rules too complex for CHECK constraints or Referential Integrity
* Automate activity
* Access or modify other tables
* Downside: will slow down triggering operation

(INSERT, UPDATE or DELETE)

**Scenario**

* A part can be supplied by many suppliers
* A supplier can supply many parts
* Sounds like M:N
* Introduce a Quote table, where Quote represents the price quoted on one Part by one Supplier
* What does the ERD look like now?

**Use A Trigger To Enforce A Business Rule**

* Business rule #1:
* No more than three suppliers are permitted to supply any single part
* A trigger can check how many rows already exist for a specific part and prevent an INSERT

**Use A Trigger To Enforce A Business Rule**

* Business rule #2:
  + The lowest cost supplier will always be used
* A trigger can disallow any order that does not use the lowest quote for a part

**Use A Trigger To Perform Calculations**

* Use a trigger to calculate the order amount for the part based on the supplier chosen to provide the parts

**Trigger Syntax**

CREATE TRIGGER Trigger\_Name

ON Table\_Name

AFTER [INSERT] [,] [UPDATE] [,] [DELETE]

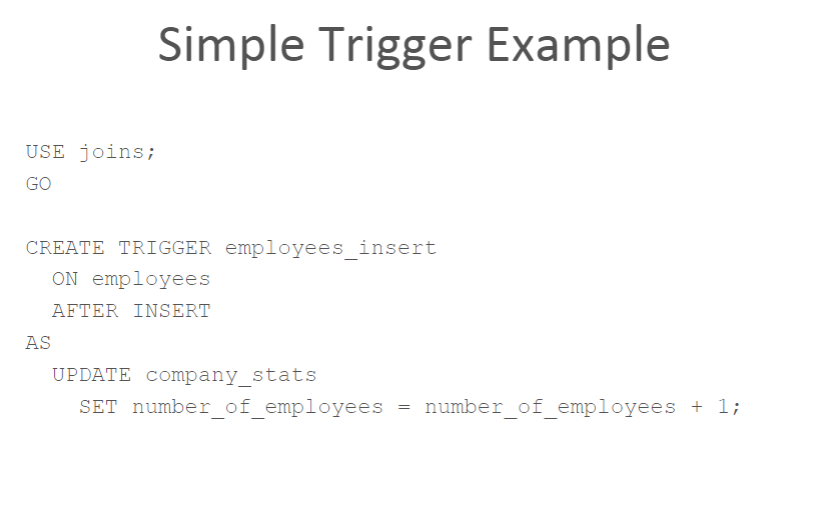
AS

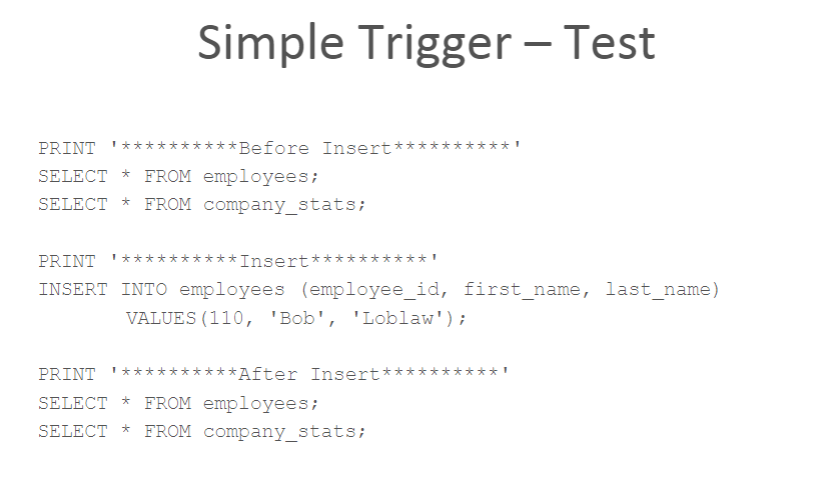
-- Optionally declare variables here

BEGIN

-- Insert statements for trigger here

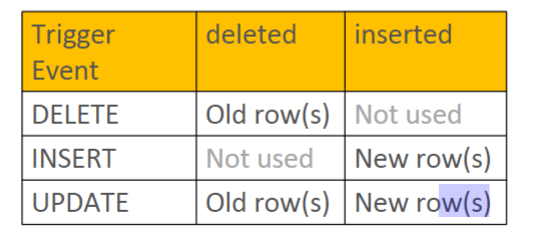
END

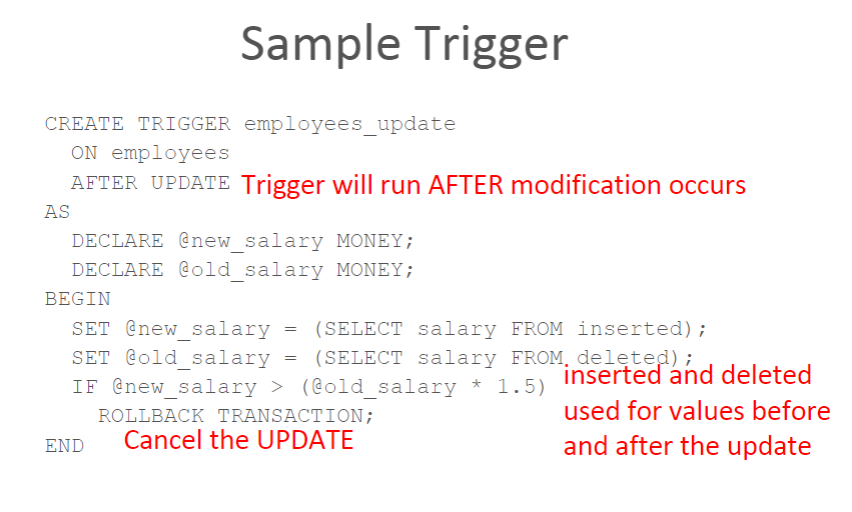


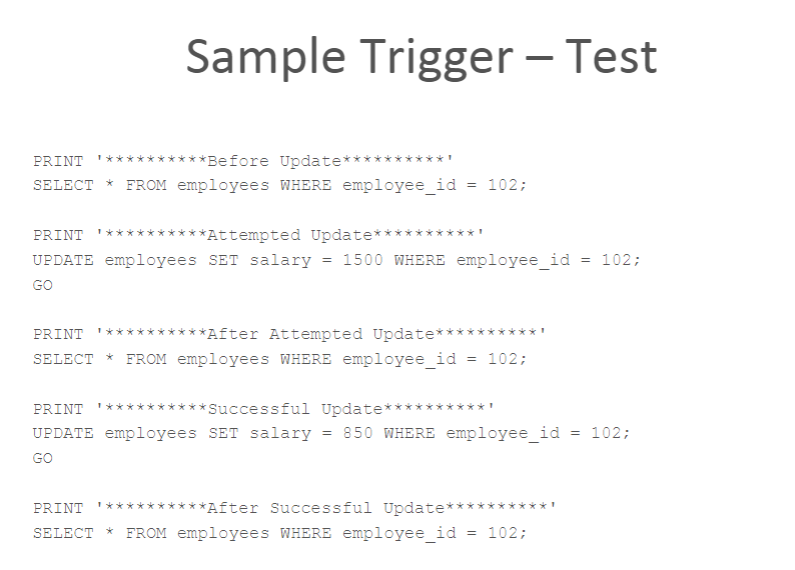


**Deleted And Inserted**

* During trigger execution, two special tables are used: deleted and inserted
* SQL Server automatically creates and manages these tables
* They have identical schema to the table being modified



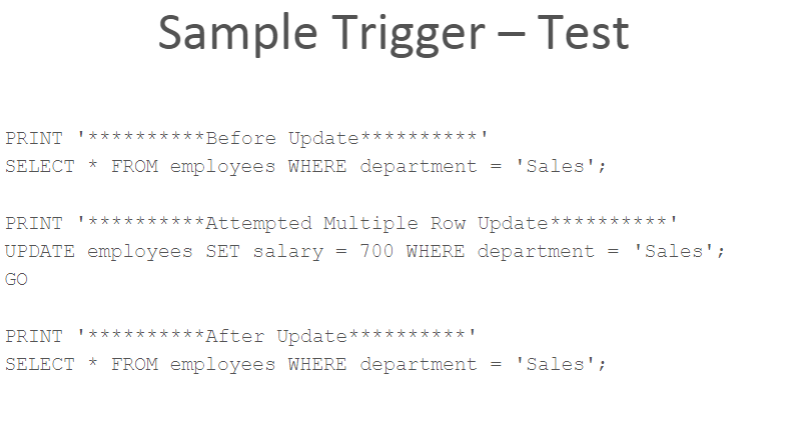




**employees\_update Trigger**

* If multiple rows are updated employees\_update will fail

SET @new\_salary = (SELECT salary FROM inserted);



**Sample Trigger 2.0**

ALTER TRIGGER employees\_update

ON employees

AFTER UPDATE

AS

BEGIN

IF EXISTS (SELECT i.salary FROM inserted i

JOIN deleted d

ON i.employee\_id = d.employee\_id

WHERE i.salary > (d.salary \* 1.5))

ROLLBACK TRANSACTION;

END

**Sample Trigger 2.0 – Test**

PRINT '\*\*\*\*\*\*\*\*\*\*Before Update\*\*\*\*\*\*\*\*\*\*'

SELECT \* FROM employees WHERE department ='Sales';

PRINT '\*\*\*\*\*\*\*\*\*\*Successful Multiple Row Update\*\*\*\*\*\*\*\*\*\*

UPDATE employees SET salary = 600 WHERE department ='Sales';

GO

PRINT '\*\*\*\*\*\*\*\*\*\*Attempted Single Row Update of 102\*\*\*\*\*\*\*\*\*\*'

UPDATE employees SET salary = 1500 WHERE employee\_id = 102;

GO

PRINT '\*\*\*\*\*\*\*\*\*\*After Update\*\*\*\*\*\*\*\*\*\*'

SELECT \* FROM employees WHEREdepartment ='Sales';

**Sample Trigger 2.1 (With Cursor)**

ALTER TRIGGER employees\_update

ON employees

AFTER UPDATE

AS

DECLARE employees\_cursor CURSOR

FOR SELECT i.salary, d.salary

FROM inserted i

JOIN deleted d

ON i.employee\_id = d.employee\_id;

DECLARE @new\_salary MONEY;

DECLARE @old\_salary MONEY;

**Sample Trigger 2.1 (With Cursor)**

BEGIN

OPEN employees\_cursor;

FETCH NEXT FROM employees\_cursor

INTO @new\_salary, @old\_salary;

WHILE @@FETCH\_STATUS = 0

BEGIN

IF @new\_salary > @old\_salary \* 1.5

BEGIN

ROLLBACK TRANSACTION;

BREAK;

END

FETCH NEXT FROM employees\_cursor

INTO @new\_salary, @old\_salary;

END

CLOSE employees\_cursor;

END`

**Sample Trigger 2.1 – Test**

PRINT '\*\*\*\*\*\*\*\*\*\*Before Update\*\*\*\*\*\*\*\*\*\*'

SELECT \* FROM employees WHEREdepartment ='Sales';

PRINT '\*\*\*\*\*\*\*\*\*\*Successful Multiple Row Update\*\*\*\*\*\*\*\*\*\*'

UPDATE employees SET salary = 750 WHERE department ='Sales';

GO

PRINT '\*\*\*\*\*\*\*\*\*\*Attempted Single Row Update of 102\*\*\*\*\*\*\*\*\*\*'

UPDATE employees SET salary = 1500 WHERE employee\_id = 102;

GO

PRINT '\*\*\*\*\*\*\*\*\*\*After Update\*\*\*\*\*\*\*\*\*\*'

SELECT \* FROM employees WHEREdepartment ='Sales';

**Firing Triggers**

* Timing choices
  + AFTER: after firing activity occurs
  + INSTEAD OF: allows DBA to take complete control of modification
  + Oracle, MySQL and DB2 also support BEFORE triggers
* AFTER example
  + Insert, update or delete occurs first
  + Trigger logic is executed
  + Trigger can “roll back” data modification

**INSERT without Trigger**

USE CHDB;

PRINT '

\*\*\*\*\*\*\*\*\*\*

Before Insert

\*\*\*\*\*\*\*\*\*\*

'

SELECT po.purchase\_order\_id, total\_amount, SUM(quantity \* unit\_cost) AS Actual

FROM purchase\_orders po

JOIN purchase\_order\_lines pol

ON po.purchase\_order\_id = pol.purchase\_order\_id

WHERE po.purchase\_order\_id = 50

GROUP BY po.purchase\_order\_id, total\_amount;

PRINT '

\*\*\*\*\*\*\*\*\*\*

Insert

\*\*\*\*\*\*\*\*\*\*

'

INSERT INTO purchase\_order\_lines

VALUES(50, 2, 20, 10, 24.89, 0, 0, NULL);

PRINT '\*\*\*\*\*\*\*\*\*\*After Insert\*\*\*\*\*\*\*\*\*\*'

SELECT po.purchase\_order\_id, total\_amount, SUM(quantity \* unit\_cost) AS Actual

FROM purchase\_orders po

JOIN purchase\_order\_lines pol

ON po.purchase\_order\_id = pol.purchase\_order\_id

WHERE po.purchase\_order\_id = 50

GROUP BY po.purchase\_order\_id, total\_amount;

PRINT '\*\*\*\*\*\*\*\*\*\*Delete to Reset\*\*\*\*\*\*\*\*\*\*'

DELETE FROM purchase\_order\_lines

WHERE purchase\_order\_id = 50 AND line\_num = 2;

**Sample Trigger:**

purchase\_order\_line\_insert

CREATE TRIGGER purchase\_order\_lines\_insert

ON purchase\_order\_lines

AFTER INSERT

AS

UPDATE purchase\_orders

SET total\_amount = total\_amount +

(SELECT SUM(quantity \* unit\_cost)

FROM inserted

WHERE purchase\_orders.purchase\_order\_id = inserted.purchase\_order\_id)

WHERE purchase\_orders.purchase\_order\_id IN

(SELECT purchase\_order\_id FROM inserted);

**Test Trigger:**

purchase\_order\_lines\_insert

PRINT '\*\*\*\*\*\*\*\*\*\*Before Insert\*\*\*\*\*\*\*\*\*\*'

SELECT po.purchase\_order\_id, total\_amount, SUM(quantity \* unit\_cost) AS Actual

FROM purchase\_orders po

JOIN purchase\_order\_lines pol

ON po.purchase\_order\_id = pol.purchase\_order\_id

WHERE po.purchase\_order\_id = 50

GROUP BY po.purchase\_order\_id, total\_amount;

PRINT '\*\*\*\*\*\*\*\*\*\*Insert\*\*\*\*\*\*\*\*\*\*'

INSERT INTO purchase\_order\_lines

VALUES(50, 2, 20, 10, 24.89, 0, 0, NULL);

PRINT '\*\*\*\*\*\*\*\*\*\*After Insert\*\*\*\*\*\*\*\*\*\*'

SELECT po.purchase\_order\_id, total\_amount, SUM(quantity \* unit\_cost) AS Actual

FROM purchase\_orders po

JOIN purchase\_order\_lines pol

ON po.purchase\_order\_id = pol.purchase\_order\_id

WHERE po.purchase\_order\_id = 50

GROUP BY po.purchase\_order\_id, total\_amount;

**One Trigger Causes Need For Others**

* If an INSERT trigger is keeping the master table updated when new records are inserted into a child table
* What happens on UPDATEs or DELETES to the child table?
* Additional triggers are required to keep the tables in sync

**Sample Trigger:**

purchase\_order\_lines\_update

CREATE TRIGGER

purchase\_order\_lines\_update

ON purchase\_order\_lines

AFTER UPDATE

AS

UPDATE purchase\_orders

SET total\_amount = total\_amount +

(SELECT SUM(quantity \* unit\_cost)

FROM inserted

WHERE purchase\_orders.purchase\_order\_id = inserted.purchase\_order\_id) -

(SELECT SUM(quantity \* unit\_cost)

FROM deleted

WHERE purchase\_orders.purchase\_order\_id = deleted.purchase\_order\_id)

WHERE purchase\_orders.purchase\_order\_id IN (SELECT purchase\_order\_id FROM inserted);

Test Trigger: purchase\_order\_line\_update

PRINT '\*\*\*\*\*\*\*\*\*\*Before Update\*\*\*\*\*\*\*\*\*\*'

SELECT po.purchase\_order\_id, total\_amount, SUM(quantity \* unit\_cost) AS Actual

FROM purchase\_orders po

JOIN purchase\_order\_lines pol

ON po.purchase\_order\_id = pol.purchase\_order\_id

WHERE po.purchase\_order\_id = 50

GROUP BY po.purchase\_order\_id, total\_amount;

UPDATE purchase\_order\_lines

SET quantity = 15

WHERE purchase\_order\_id = 50 AND line\_num = 2;

PRINT '\*\*\*\*\*\*\*\*\*\*After Update\*\*\*\*\*\*\*\*\*\*'

SELECT po.purchase\_order\_id, total\_amount, SUM(quantity \* unit\_cost) AS Actual

FROM purchase\_orders po

JOIN purchase\_order\_lines pol

ON po.purchase\_order\_id = pol.purchase\_order\_id

WHERE po.purchase\_order\_id = 50

GROUP BY po.purchase\_order\_id, total\_amount;

**Sample Trigger:**

purchase\_order\_lines\_delete

CREATE TRIGGER purchase\_order\_lines\_delete

ON purchase\_order\_lines

AFTER DELETE

AS

UPDATE purchase\_orders

SET total\_amount = total\_amount -

(SELECT SUM(quantity \* unit\_cost)

FROM deleted

WHERE purchase\_orders.purchase\_order\_id = deleted.purchase\_order\_id)

WHERE purchase\_orders.purchase\_order\_id IN (SELECT purchase\_order\_id FROM deleted);

**Test Trigger:**

purchase\_order\_lines\_delete

PRINT '\*\*\*\*\*\*\*\*\*\*Before Delete\*\*\*\*\*\*\*\*\*\*'

SELECT po.purchase\_order\_id, total\_amount, SUM(quantity \* unit\_cost) AS Actual

FROM purchase\_orders po

JOIN purchase\_order\_lines pol

ON po.purchase\_order\_id = pol.purchase\_order\_id

WHERE po.purchase\_order\_id = 50

GROUP BY po.purchase\_order\_id, total\_amount;

PRINT '\*\*\*\*\*\*\*\*\*\*Delete\*\*\*\*\*\*\*\*\*\*'

DELETE FROM purchase\_order\_lines

WHERE purchase\_order\_id = 50 AND line\_num = 1;

PRINT '\*\*\*\*\*\*\*\*\*\*After Delete\*\*\*\*\*\*\*\*\*\*'

SELECT po.purchase\_order\_id, total\_amount, SUM(quantity \* unit\_cost) AS Actual

FROM purchase\_orders po

JOIN purchase\_order\_lines pol

ON po.purchase\_order\_id = pol.purchase\_order\_id

WHERE po.purchase\_order\_id = 50

GROUP BY po.purchase\_order\_id, total\_amount;

**Deleting A Trigger**

* To delete an existing trigger use:

DROP TRIGGER Trigger\_Name

**Data Dictionary Tables For Triggers**

* Every CREATE TRIGGER statement generates entries in

–

sys.triggers

–

sys.trigger\_events

–sys.events

–sys.sql\_modules