Department of Computer Science
University of Cyprus

EPL342 –Databases

Lecture 18: Internal DB Programming I

Views/Assertions/Triggers

(Chapter 6.7-6.8 and 7.1-7.4, Elmasri-Navathe 7ED
+ TransactSQL Reference Guide
http://msdn.microsoft.com/en-us/library/bb510741.aspx)

Demetris Zeinalipour

http://www.cs.ucy.ac.cy/courses/EPL342

Περιεχόμενο Διάλεξης

Ολοκλήρωση Διάλεξης 17.

Κεφάλαιο 8.7-8.8: SQL Programming I

- Όψεις (Views) σε SQL και TSQL
- Βεβαιώσεις (Assertions) σε SQL
- Σκανδάλες (Triggers) σε SQL και TSQL

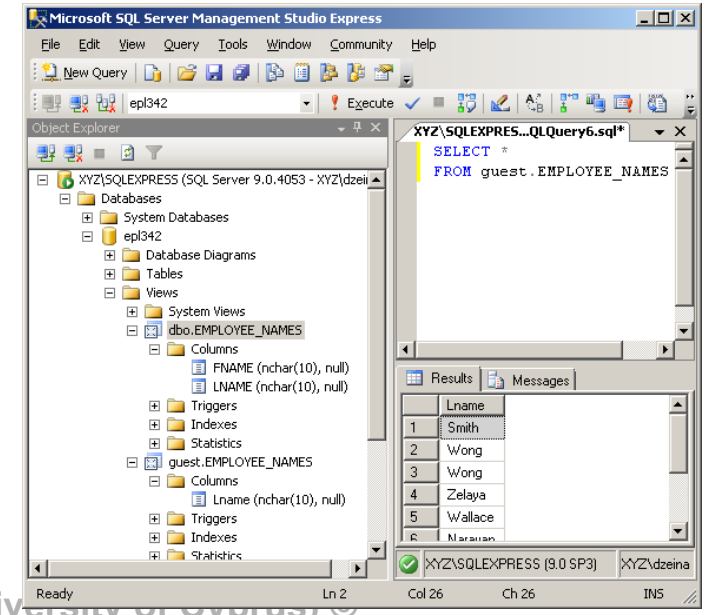
Όψεις σε SQL (Views in SQL)

- Μια όψη είναι ένας **“νοητός” πίνακας (“virtual” table)** ο οποίος παράγεται από άλλους **κανονικούς πινάκες** (**Base-Tables**)
- Στην πράξη μια όψη δεν είναι τίποτα περισσότερο από μια **αποθηκευμένη** επερώτηση **SELECT**!
- Π.χ.,

a)
```
CREATE VIEW dbo.EMPLOYEE_NAMES
AS
  SELECT FNAME, LNAME
  FROM EMPLOYEE
```

b)
```
SELECT * FROM dbo.EMPLOYEE_NAMES
```

c)
```
DROP VIEW dbo.EMPLOYEE_VIEW
```



Όψεις σε SQL (Χαρακτηριστικά)

- **Χαρακτηριστικά Όψεων**
  - Μπορούν να χρησιμοποιηθούν όπως τα υπόλοιπα tables (σε επερωτήσεις, συνενώσεις, συναθροίσεις, κτλ)
  - Περιέχουν ΠΑΝΤΑ **ενημερωμένα δεδομένα**.
  - Τα δεδομένα μιας όψης **ΔΕΝ αποθηκεύονται φυσικά** κάπου (τα δεδομένα αποθηκεύονται στα base tables)
  - Μπορούμε να **εκτελέσουμε αλλαγές** σε μια **όψη** (INSERT/UPDATE/DELETE)
    - Ενημερώσεις γίνονται μόνο εάν η όψη **ορίζεται από ένα** base-tables (όχι από περισσότερα base-tables)
    - Επίσης, όψεις με **aggregates & groupby ΔΕΝ** ενημερώνονται.
  - Εκτέλεση μιας Όψης **ΔΕΝ είναι πιο γρήγορο** από μια εκτέλεση της αντίστοιχης **SELECT επερώτησης**.
    - Αυτό επειδή το VIEW εισάγει κάποιο overhead

Όψεις σε SQL (Πλεονεκτήματα)

- **Πλεονεκτήματα Όψεων**
  - **Μειώνουν την πολυπλοκότητα ανάπτυξης:** Αυτό συμβαίνει επειδή μπορούμε να αναπαραστήσουμε περίπλοκες επερωτήσεις ως νοητούς πίνακες.
  - **Ασφάλεια**: Επιτρέπουν στον DBA να εκθέσει μόνο τις στήλες που επιθυμεί σε συγκεκριμένες ομάδες χρηστών.
- Όπως όλες οι δυνατότητες, οι όψεις πρέπει να χρησιμοποιούνται με προσοχή και όταν θεωρούνται απαραίτητες!

Όψεις σε SQL (Παράδειγμα Όψης σε TSQL)

Όψη με συνάθροιση στο SELECT.

```
CREATE VIEW Emp_Sal2
AS
    SELECT dno, SUM(salary) AS sumname
    FROM Employee
    GROUP BY dno
```

Πριν την Ενημέρωση του EMPLOYEE

dno	sumname
1	59000
2	54100
3	13000

Μετά την Ενημέρωση του EMPLOYEE

dno	sumname
1	9005
2	950100
3	13000

Επισημάνσεις:

- Σε περίπτωση ενημέρωσης του πίνακα **EMPLOYEE** ενημερώνεται αυτόματα η όψη.
- Η ίδια η όψη **ΔΕΝ** μπορεί να ενημερωθεί από τον χρήστη με **INSERT/UPDATE/DELETE** (λόγω του aggregate / group-by).

Π.χ., UPDATE Emp_Sal2 SET dno=1

Όψεις σε SQL (Σύνταξη Όψεων σε TSQL)

```
CREATE VIEW  [<schema-name>].<view.name> [(column-name-list>)]  [WITH ENCRYPTION] [[,] WITH SCHEMABINDING]
AS
        <SELECT statement>
[WITH CHECK OPTION] [;]
```

- **<schema-name>:** dbo (default), guest, κτλ.
- **<column-name-list>:** ονόματα γνωρισμάτων της νέας όψης
- **WITH ENCRYPTION:** Ο SQL κώδικας της όψης κωδικοποιείται μέσα στη βάση για να μην μπορεί να τον δει κανείς (ούτε και εσείς!).
  - Για να δείτε τον κώδικα μη-κωδικοποιημεν. όψεων: **EXEC sp_helptext view_name**;
- **WITH SCHEMABINDING:** Διασφαλίζει ότι η όψη ΔΕΝ θα μείνει **ορφανή** σε περίπτωση δομικών αλλαγών στα basetables.
  - Π.χ., εάν διαγραφεί ο πίνακας πάνω στον οποίο ορίζεται η όψη.
- **"WITH CHECK OPTION:** Κατά την τροποποίηση (insert(X)/update(X)) δεδομένων (μέσω μιας όψης) ελέγχει ότι το X είναι σύμφωνα με το WHERE του **<SELECT statement>** (έτσι ώστε να μην χαθούν τα δεδομένα από την όψη)
  - Θυμηθείτε ότι μια όψη μπορεί να μεταβληθεί (όχι για group by), άρα μου δίνεται η δυνατότητα να εισάγω ένα επιπλέον CHECK περιορισμό στην όψη

Όψεις σε SQL (Παράδειγμα Όψης σε TSQL)

```
USE AdventureWorks; -- change to specified database context
GO -- not tsql cmd. Instructs SQLStudio to execute statements.
IF OBJECT_ID ('dbo.SeattleOnly', 'V') IS NOT NULL
    DROP VIEW dbo.SeattleOnly;
GO - OBJECT_ID (int) uniquely identifies objects in DB

CREATE VIEW dbo.SeattleOnly
WITH SCHEMABINDING -- structural changes to Person.Contact
   (e.g., drop) will be prohibited.
AS
   SELECT c.LastName, c.FirstName
   FROM Person.Contact AS c
   WHERE c.City = 'Seattle' and c.Lastname='Smith'
WITH CHECK OPTION; -- any update to this view has to obey the
   WHERE condition(i.e., c.City = 'Seattle' and
   c.Lastname='Smith')
GO
```

View Identifier

Βεβαιώσεις σε SQL (CREATE ASSERTION) - **ΟΧΙ ΣΕ TSQL**

- Μια **Βεβαίωση (ASSERTION)** είναι ένας κανόνας που ορίζεται πάνω από **πολλαπλούς πίνακες**.
- Αυτός ο **κανόνας ελέγχεται** κατά οποιαδήποτε **αλλαγή της κατάστασης** των εμπλεκόμενων **Πινάκων** (INSERT, UPDATE)
  - **Αντίστοιχο** με το **CHECK** που ορίζεται ωστόσο μόνο πάνω σε **ένα πίνακα**: Π.χ. age int check (age>20);
- **Επισημάνσεις**
  - Τα **Assertions ΔΕΝ ορίζονται σε TSQL** άλλα ορίζονται σε αρκετές άλλες βάσεις όπως PostgreSQL
  - Τα **Assertions** είναι **όμοια με τις σκανδάλες** τα οποία θα μελετήσουμε σε λίγο.

Βεβαιώσεις σε SQL ASSERTION: Παράδειγμα

- **Σημασιολογικός Περιορισμός:** "Ο μισθός ενός employee ΔΕΝ πρέπει να είναι μεγαλύτερος από τον μισθό του manager του department στο οποίο δουλεύει ο employee"

Όνομα Βεβαίωσης

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS
  (SELECT *
   FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
   WHERE  E.DNO=D.NUMBER AND D.MGRSSN=M.SSN
          AND E.SALARY > M.SALARY
   )
)
```

Συνθήκη Βεβαίωσης

Σκανδάλες σε SQL (SQL Triggers)

- Μια **Σκανδάλη (Trigger)** ορίζει μια **αντίδραση** της βάσης δεδομένων σε περίπτωση **αλλαγών πλειάδων*** (INSERT, DELETE, UPDATE) σε κάποιους **προσδιορισμένους πίνακες**.
  - **ASSERTIONS:** απαγορεύουν κάποια κατάσταση
    - Π.χ., ο μισθός του υπαλλήλου ΔΕΝ μπορεί να είναι μεγαλύτερος από αυτόν του supervisor του.
  - **TRIGGERS:** δεν σημαίνει απαραίτητα ότι απαγορεύουν μια κατάσταση, άπλα ορίζουν ακολουθία εντολών που πρέπει να ενεργοποιηθεί όταν ικανοποιηθεί μια συνθήκη
    - π.χ., όταν προστεθούν/αφαιρεθούν λεφτά από τον λογαριασμό κάποιου πελάτη, στείλε email στον πελάτη για να τον ενημερώσεις
- Το SELECT, TRUNCATE ή BULK INSERT δεν ενεργοποιούν τις σκανδάλες σε TSQL.

Κατηγορίες Σκανδάλων

- Τα triggers χωρίζονται σε δυο κατηγορίες:
  - **DDL Triggers:** Ορίζουν την **αντίδραση** σε **δομικές** αλλαγές (DROP, ALTER, κτλ).
  - **DML Triggers:** Ορίζουν την **αντίδραση** σε αλλαγές πάνω σε πλειάδες μιας σχέσης ή όψης (INSERT, UPDATE, DELETE).
    - Θα επικεντρωθούμε μόνο στα DML Triggers.
- Σημειώστε ότι τα Triggers είναι ουσιαστικά «μικρά προγράμματα σε (T)SQL» *τα οποία καλούνται **ΑΥΤΟΜΑΤΑ** μόλις ενεργοποιηθεί η **ορισμένη συνθήκη** η οποία ισχύει πάνω σε **κάποιους πίνακες** ή **όψεις**.*
  - *Τα triggers ΔΕΝ μπορούμε να τα καλέσουμε (invoke) αυτόνομα (π.χ., με SELECT ή EXEC)*

Είδη Triggers στην TSQL

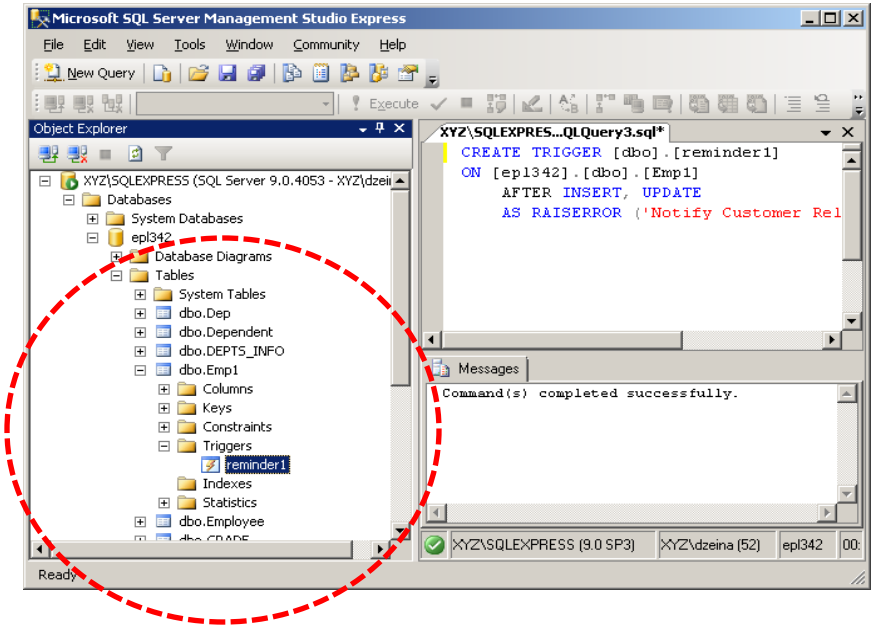
- Υπάρχουν 2 ειδών DML triggers στην TSQL:
  - AFTER|FOR
  - INSTEAD OF
- Το **AFTER|FOR** trigger εκτελείται **ΜΕΤΑ** από την πράξη της εισαγωγής, διαγραφής και ενημέρωσης.
  - ΜΕΤΑ σημαίνει πριν την ολοκλήρωση του Transaction και πρώτού η μεταβολή να είναι ορατή από άλλες δοσοληψίες.
  - Το αποτέλεσμα αποθηκεύεται σε πίνακα **INSERTED** ή **DELETED** πίνακες που παράγεται για προβολή των επηρεαζόμενων εγγραφών (και δυνατότητα για Rollback εάν επιθυμούμε).
- Το **INSTEAD OF** trigger εκτελείται **ΑΝΤΙ** της πράξη της εισαγωγής, διαγραφής και ενημέρωσης.
  - Άρα θεωρείται ένα είδος BEFORE Trigger
  - **Δεν θα μελετηθούν σε αυτό το μάθημα (1 παράδειγμα μόνο).**
- Ακολουθούν παραδείγματα …

Απλό Παράδειγμα AFTER|FOR σε TSQL

-- Παράδειγμα σκανδάλης που παράγει **μήνυμα λάθους** όποτε γίνει **εισαγωγή/διαγραφή** δεδομένων από το Emp1 Table

```
USE epl342;
GO
IF OBJECT_ID ('Reminder1', 'TR')
                                IS NOT NULL
   DROP TRIGGER Reminder1;
GO
CREATE TRIGGER reminder1
ON Emp1
    AFTER INSERT, UPDATE
    AS  -- ακολουθούν οι εντολές που πρέπει να εκτελεστούν
          RAISERROR ('Notify Customer Relations', 16, 1);
          ROLLBACK TRANSACTION; RETURN
 GO
```

trigger

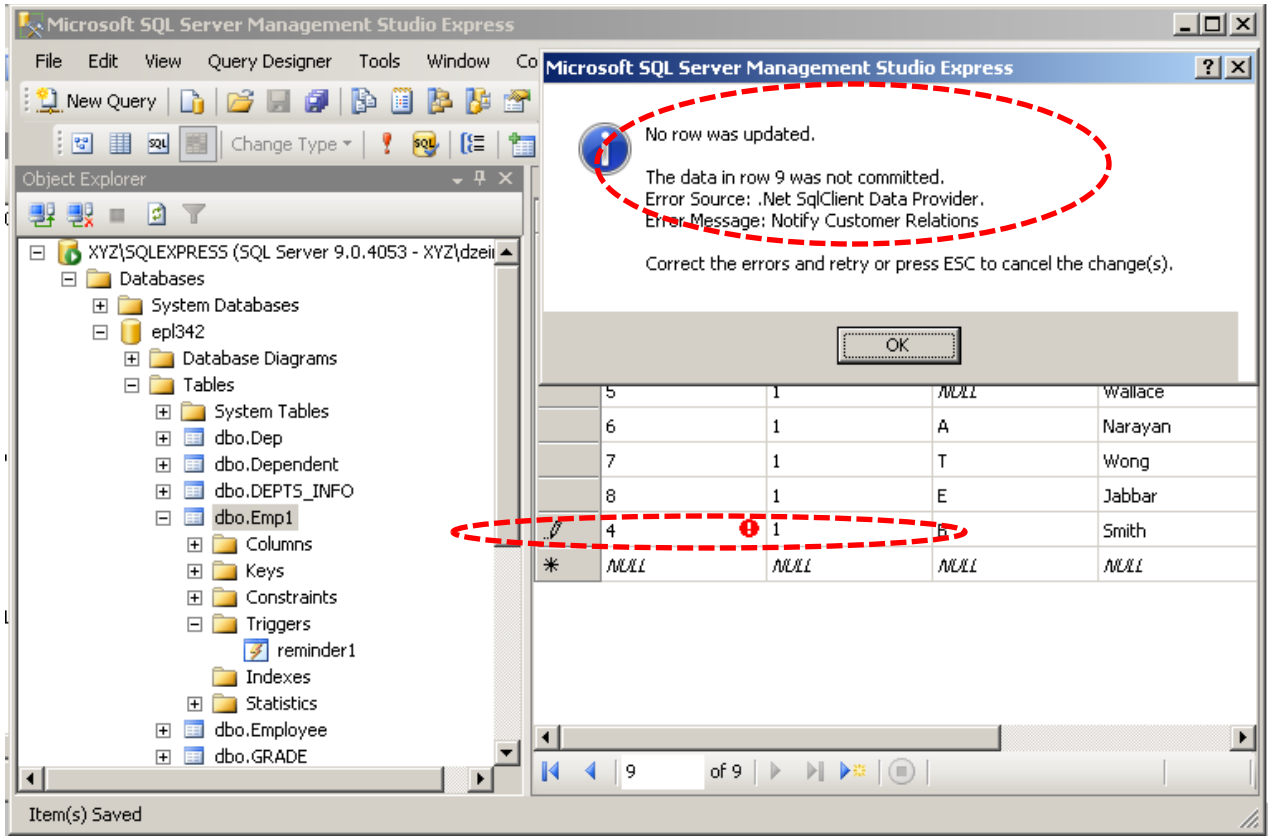


Severity:
0-18: Specified by User
20-25: FATAL Errors

State: 0-255

Απλό Παράδειγμα Trigger σε TSQL

- Κατά την εισαγωγή/ενημέρωση δεδομένων στον Πίνακα Emp1 **επιστρέφεται μήνυμα λάθους** διότι ενεργοποιείται η σκανδάλη.



Σύνταξη Σκανδάλης AFTER|FOR σε TSQL

```
CREATE TRIGGER <trigger-name>
ON [schema-name>.] <table|view-name>
    [WITH ENCRYPTION]  -- trigger code is encrypted in DB
    [EXECUTE AS <CALLER | SELF | <user>]
        -- Default: Caller (of change), SELF: Creator of Trigger, user:
{{FOR | AFTER} <[DELETE] [,] [INSERT] [,] [UPDATE]>}
AS
  <sql-statements>
```

- FOR|AFTER: Αναφέρονται στο **ίδιο πράγμα** και προστίθεται για να είναι πιο **ευανάγνωστος** ο κώδικας,
  - π.χ., AFTER DELETE AS

SQL Statements

```
IF EXISTS (SELECT ….)
        BEGIN
        …..
        END
```

Περισσότερα: http://msdn.microsoft.com/en-us/library/ms188354.aspx
Παράδειγμα σε λίγο ..

Σύνταξη Σκανδάλης σε TSQL

Άλλες Χρήσιμες Πληροφορίες για TRIGGERS

- Προσωρινή Απενεργοποίηση Σκανδάλης

  ALTER TABLE <table-name>

  <ENABLE | DISABLE> TRIGGER <ALL | <trigger-name>>

- Οι **σκανδάλες** μπορεί να καλούνται **αναδρομικά** μέχρι και **32 επίπεδα.**
- Η **εκτέλεση μιας σκανδάλης** μπορεί να προκαλέσει την **αλυσιδωτή εκτέλεση** άλλων σκανδαλών με απρόσμενα αποτέλεσμα
  - θυμηθείτε το **ON DELETE CASCADE** παράδειγμα το οποίο μπορούσε να σβήσει όλο τον πίνακα των EMPLOYEEs.

Διαδικαστικός Προγραμματισμός μέσα στην Βάση Δεδομένων!

- Οι Σκανδάλες σε μια βάση δεδομένων μπορεί να γίνουν αρκετά πιο ευφυείς με την χρήση **εντολών διαδικαστικού προγραμματισμού** που θα δούμε στην ερχόμενη διάλεξη.
- Για παράδειγμα μπορεί να ορίζονται **επαναλήψεις**, **συνθήκες έλεγχου**, **μεταβλητές**, **συναρτήσεις** και πάρα πολλά άλλα.
  - Αυτές οι δομές προγραμματισμού είναι μέρος επεκτάσεων της SQL (π.χ., TSQL, PL/SQL (Oracle)).
- Στα πλαίσια του εργαστηρίου θα δείτε και την χρήση των ενδιάμεσων πινάκων των Triggers: **Inserted** (για **Insert**), **Deleted** (για **Deletes**), **Inserted+Deleted** (για **Updates**).

Inserted+Deleted Tables σε AFTER|FOR

- Αποθηκεύουν την **ΠΡΙΝ κατάσταση** της εγγραφής σε διαγραφές και ενημερώσεις
  - Σε UPDATE περιέχει την εγγραφή **ΠΡΙΝ** (deleted) και **ΜΕΤΑ** (inserted)

Operation	*deleted* Table	*inserted* Table
INSERT	(not used)	Contains the rows being inserted
DELETE	Contains the rows being deleted	(not used)
UPDATE	Contains the rows as they were before the UPDATE statement	Contains the rows as they were after the UPDATE statement

  - Οι πίνακες αυτοί είναι ανά query, άρα δεν τίθεται πρόβλημα ταυτοχρονίας με άλλα queries.
  - Οι πίνακες αυτοί δεν μπορούν να ενημερωθούν (είναι απλές όψεις του transaction log)

```
the logical tables inserted and deleted cannot be updated
```

Deleted Tables σε AFTER|FOR

```
USE MSSQLTips;
GO

CREATE TABLE dbo.SampleTable (
  SampleTableID INT NOT NULL IDENTITY(1,1),
  SampleTableInt INT NOT NULL,
  SampleTableChar CHAR(5) NOT NULL,
  SampleTableVarChar VARCHAR(30) NOT NULL,
  CONSTRAINT PK_SampleTable PRIMARY KEY CLUSTERED (SampleTableID)
);
GO

CREATE TABLE dbo.SampleTable_Audit (
  SampleTableID INT NOT NULL,
  SampleTableInt INT NOT NULL,
  SampleTableChar CHAR(5) NOT NULL,
  SampleTableVarChar VARCHAR(30) NOT NULL,
  Operation CHAR(1) NOT NULL,
  TriggerTable CHAR(1) NOT NULL,
  AuditDateTime smalldatetime NOT NULL DEFAULT GETDATE(),
);

CREATE INDEX IDX_SampleTable_Audit_AuditDateTime ON dbo.SampleTable_Audit (AuditDateTime DESC);
GO
```

```
CREATE
TRIGGER dbo.SampleTable_DeleteTrigge
r
ON dbo.SampleTable
FOR DELETE
AS
BEGIN
   INSERT INTO dbo.SampleTable_Audit
   (SampleTableID, SampleTableInt, S
ampleTableChar, SampleTableVarChar,
Operation, TriggerTable)

   SELECT SampleTableID, SampleTable
Int, SampleTableChar, SampleTableVar
Char, 'D', 'D'
   FROM deleted;
END;
GO
```

Adding the under delete item to the dbo.SampleTable_Audit table

Inserted+Deleted Tables σε AFTER|FOR

- ```
  CREATE TRIGGER dbo.SampleTable_UpdateTrigger
  ON dbo.SampleTable
  FOR UPDATE
  AS
  BEGIN
     INSERT INTO dbo.SampleTable_Audit
     (SampleTableID, SampleTableInt, SampleTableChar, SampleTableVarChar,
  Operation, TriggerTable)
     SELECT SampleTableID, SampleTableInt, SampleTableChar, SampleTableVar
  Char, 'U', 'D'
     FROM deleted;

     INSERT INTO dbo.SampleTable_Audit
     (SampleTableID, SampleTableInt, SampleTableChar, SampleTableVarChar,
  Operation, TriggerTable)
     SELECT SampleTableID, SampleTableInt, SampleTableChar, SampleTableVar
  Char, 'U', 'I'
     FROM inserted;
  END;
  GO
  ```
  ```

Inserted Tables σε AFTER|FOR

- **Επιβολή Σημασιολογικού Κανόνα Ακεραιότητας:**
  - "Επιβεβαίωση ότι το credit rating του vendor είναι καλό εάν επιχειρήσει να γίνει εισαγωγή στον PurchaseOrderHeader πίνακα

```
CREATE TRIGGER Purchasing.LowCredit ON Purchasing.PurchaseOrderHeader
AFTER INSERT  -- ορισμός πότε να εκτελείται το trigger αυτό
AS

IF EXISTS (SELECT *
           FROM Purchasing.PurchaseOrderHeader p
              JOIN inserted AS I  ON p.PurchaseOrderID = i.PurchaseOrderID
              JOIN Purchasing.Vendor AS v ON v.BusinessEntityID = p.VendorID
           WHERE v.CreditRating = 5
           )
BEGIN
   RAISERROR ('Vendor''s credit rating is too low to accept new purchases.', 16, 1);
   ROLLBACK TRANSACTION;
RETURN
END;
```

Πίνακας με αντικείμενα υπο εισαγωγή (temp πίνακας)

Inserted Tables σε INSTEAD OF

```
CREATE TRIGGER production.trg_vw_brands
ON production.vw_brands
INSTEAD OF INSERT
AS
BEGIN
    INSERT INTO production.brand_approvals (
        brand_name
    )
    SELECT
        i.brand_name
    FROM
        inserted i
    WHERE
        i.brand_name NOT IN (
            SELECT
                brand_name
            FROM
                production.brands
        );
END
```

Προσθήκη αποτελεσμάτων σε προσωρινό πίνακα `production.brand_approvals` μέχρι εγκρίσεως.

Μπορούσε να υλοποιηθεί με φυσικό πίνακα που θα κρατεί τα προσωρινά αποτελέσματα.

Operation	*deleted* Table	*inserted* Table
INSERT	(not used)	Contains the rows being inserted
DELETE	Contains the rows being deleted	(not used)
UPDATE	Contains the rows as they were before the UPDATE statement	Contains the rows as they were after the UPDATE statement

Triggers in Other DBs

MySQL **AFTER|BEFORE** INSERT

```
CREATE TRIGGER after_members_insert
AFTER|BEFORE INSERT
ON members FOR EACH ROW
BEGIN
    IF NEW.birthDate IS NULL THEN
        INSERT INTO reminders(memberId, message)
        VALUES(new.id,CONCAT('Hi ', NEW.name, ', please update your date of birth.'));
    END IF;
END
```