Department of Computer Science
University of Cyprus

EPL342 –Databases

Lecture 18: Internal DB Programming I

Views/Assertions/Triggers

(Chapter 8.7-8.8, Elmasri-Navathe 5ED +
TransactSQL Reference Guide

http://msdn.microsoft.com/en-us/library/bb510741.aspx)

Διδάσκων: Παναγιώτης Ανδρέου

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

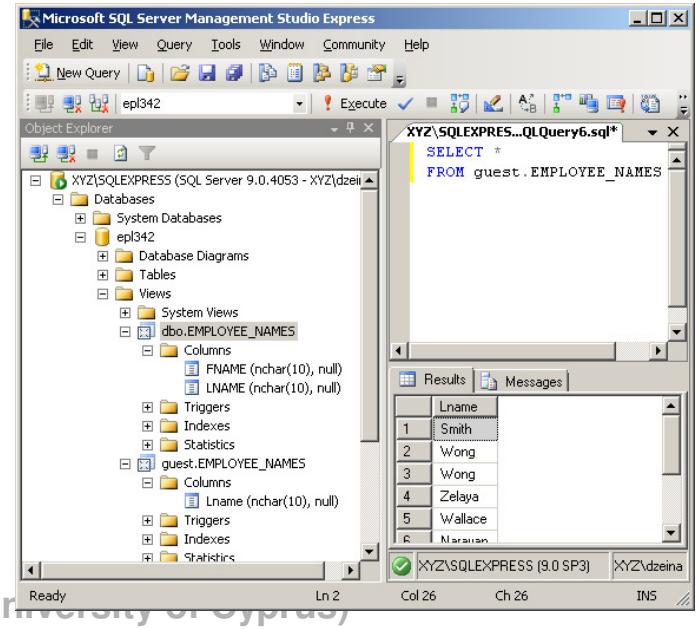
Όψεις σε SQL
(Views in SQL)

- 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).

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'
WITH CHECK OPTION; -- any update to this view has to obey the
   WHERE condition(i.e., c.City='Seattle')
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

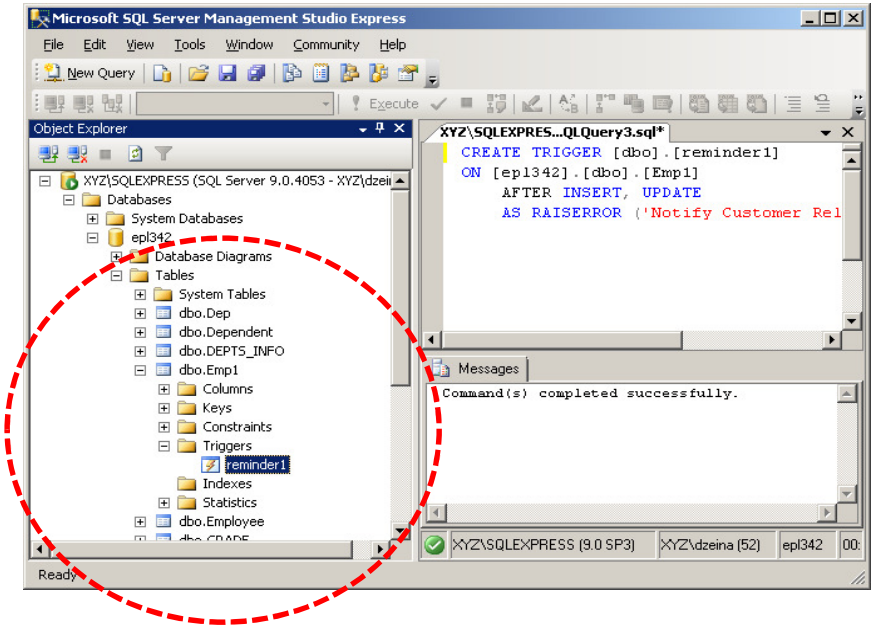
- Τα triggers χωρίζονται σε δυο κατηγορίες:
  - **DDL Triggers:** Ορίζουν την **αντίδραση** σε **δομικές** αλλαγές (DROP, ALTER, κτλ).
  - **DML Triggers:** Ορίζουν την **αντίδραση** σε αλλαγές πάνω σε πλειάδες μιας σχέσης ή όψης (INSERT, UPDATE, DELETE).
    - Θα επικεντρωθούμε μόνο στα DML Triggers.
- 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)*

Απλό Παράδειγμα Trigger 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

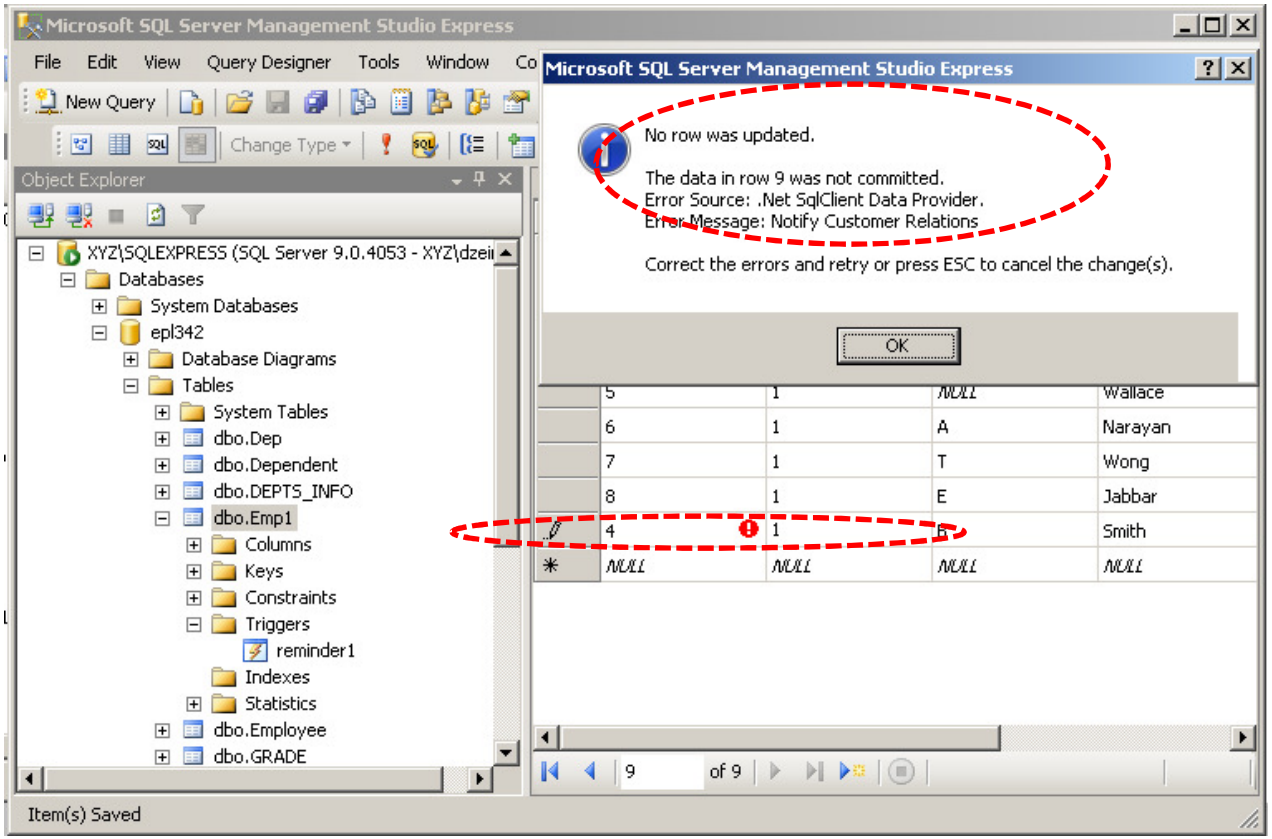


Severity:
0-18: Specified by User
20-25: FATAL Errors

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 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

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**).

Διαδικαστικός Προγραμματισμός μέσα στην Βάση Δεδομένων!

- **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)