



EPL342 –Databases

Lecture 20: External DB Programming II

Internal Prog.: Sprocs, UDFs, Cursors,
External Prog.: Embedded SQL, JDBC,
SQL/CLI

(Chapter 9.2-9.4, Elmasri-Navathe 5ED +

TransactSQL Reference Guide

Διδάσκων: Παναγιώτης Ανδρέου



Περιεχόμενο Διάλεξης

Εσωτερικός Προγραμματισμός DB II

- Εσωτερικός Προγραμματισμός
- **Scripts/Batches** σε TSQL
- **Stored Procedures (Sprocs)** σε TSQL
- **User Defined Functions (UDFs)** σε TSQL
- **Cursors** σε TSQL

Εξωτερικός Προγραμματισμός DB

- Ενσωματωμένη SQL (Embedded SQL): SQLCA και SQLJ
- Ενσωματωμένη SQL με **Κλήσεις Συναρτήσεων (Embedded SQL with APIs and Functions Calls)**: JDBC και SQL/CLI (ex ODBC)

B) Stored Procedures σε TSQL

- **Stored Procedures (Sprocs)**, είναι εντολές (T)SQL οι οποίες αποθηκεύονται στη **βάση δεδομένων** και οι οποίες μπορούν να **πάρουν ορίσματα** και να **επιστρέψουν τιμές**.
- **Παράδειγμα:** Δημιουργία Sproc το οποίο επιστρέφει όλους τους Employee

USE ep1342

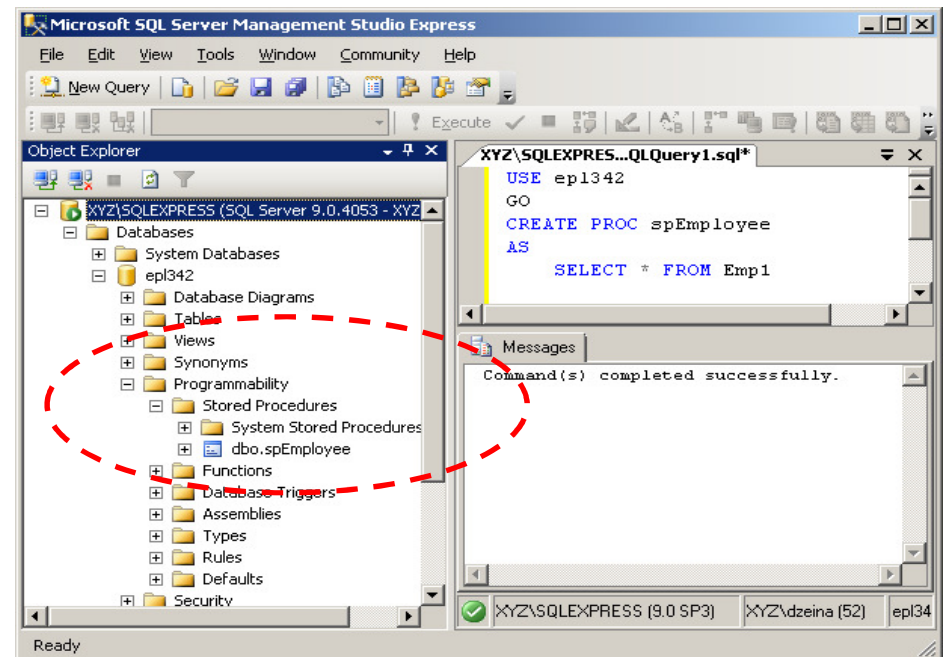
GO

CREATE PROC spEmployee

AS

SELECT * FROM Emp1

GO



B) Stored Procedures σε TSQL (Sproc χωρίς Παραμέτρους)



- Κλήση Sproc (χωρίς Παραμέτρους) μέσω EXEC

USE epl342

GO

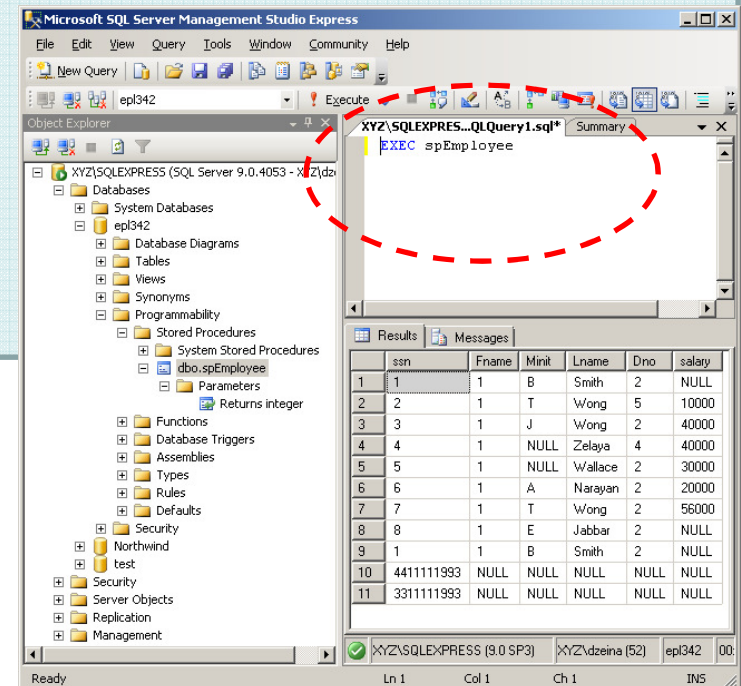
EXEC spEmployee

Άλλες Εντολές

- Ακύρωση SProc
 - DROP PROC spEmployee [;]
- Μεταβολή Sproc
 - ALTER PROC

AS SELECT * FROM Emp1

GO

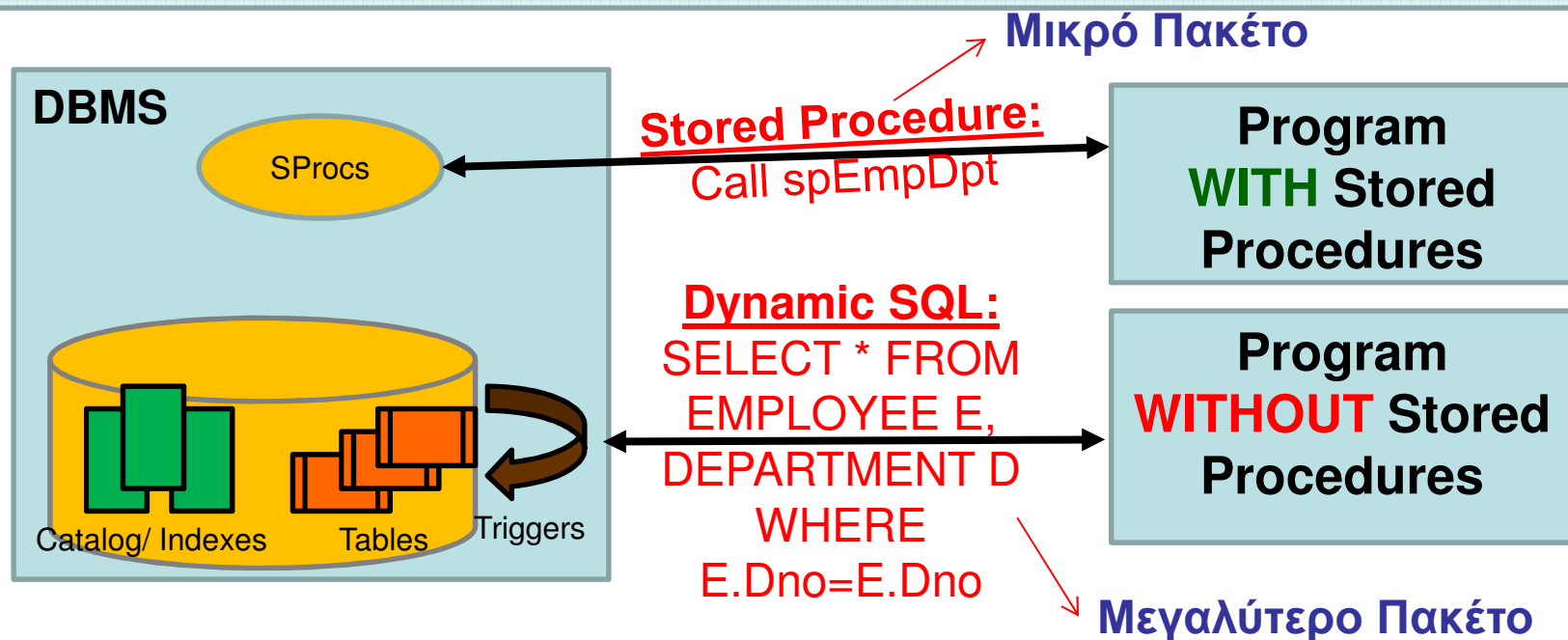


B) Stored Procedures σε TSQL (Χαρακτηριστικά Sprocs)



Βασικό Πλεονέκτημα Sprocs:

- **Επίδοση/Μειωμένη Κίνηση Δικτύου:** Το **Sproc** είναι **Precompiled/Optimized** πριν την κλήση του. Επίσης **μειώνεται η κυκλοφορία δεδομένων** μεταξύ DBMS και προγράμματος εφαρμογής.



B) Stored Procedures σε TSQL (Χαρακτηριστικά Sprocs)



- **Άλλα Πλεονεκτήματα Sprocs**

- **Ασφάλεια:** Τα Sprocs μας επιτρέπουν να δώσουμε **πρόσβαση** σε **λειτουργίες** της βάσης **χωρίς** να δίνουμε πρόσβαση στους **πίνακες**.
- **Ακρίβεια:** Οι διεπαφές (ODBC, JDBC, κτλ) στη προσπάθεια να προσφέρουν ένα κοινό υπόβαθρο λειτουργίας σε διαφορετικούς κατασκευαστές βάσεων κάνουν διάφορους συμβιβασμούς στην ακρίβεια (π.χ., μετατροπή τύπων κτλ)
- **Εύκολη μεταφορά σε άλλη Γλώσσα Προγραμματισμού:** Αυτό εφόσον όλη η λογική της βάσης υλοποιείται εσωτερικά και όχι στην γλώσσα προγραμματισμού.

- **Μειονεκτήματα Sprocs**

- **Δύσκολη Μεταφορά σε άλλο DBMS:** Αυτό λόγω του ότι οι γλώσσες που χρησιμοποιούνται σε Προγ. Γλώσσες Βάσεων Δεδομένων (π.χ., σύνταξη της TSQL) δεν είναι προτυποποιημένες

B) Stored Procedures σε TSQL (Sproc με Παράμετρο Εισόδου)



- Παράδειγμα Sproc: Βρες όλα τα name, dependent name βάσει κάποιας συμβολοσειράς αναζήτησης

```
USE ep1342
```

```
GO
```

```
CREATE PROC spEmpDep  
    @SearchName nvarchar(50)
```

```
AS
```

```
    SELECT E.fname, D.dname  
    FROM Dependent D, Emp1 E  
    WHERE D.ssn=E.ssn AND  
           D.Dname LIKE '%' + @ SearchName + '%';
```

*Παράμετρος Εισόδου
(max αριθμός = 2100)*

Κλήση Sproc: EXEC spEmpDep1 'Ach'

→ Επιστρέφει Franklin, Achilleas

EMPLOYEE

ssn	Fname	Minit	Lname	Dno
1	John	B	Smith	2
2	Franklin	T	Wong	5
3	Alicia	J	Wong	2
4	Jennifer	S	Zelaya	4
5	Ramesh	K	Wallace	2
6	Joyce	A	Narayan	2
7	Ahmad	V	English	2
8	James	E	Jabbar	2

ssn	dname
1	Natali
1	Stefanos
2	Achilleas
2	Prokopis
3	Irini

DEPENDENT

B) Stored Procedures σε TSQL

(Sproc με Παράμετρο Εισόδου/Εξόδου)



- Παράδειγμα Stored Procedure που λαμβάνει ως είσοδο ένα CustomerID και ελέγχει το ιστορικό αγορών του εν λόγω πελάτη.

```
CREATE PROCEDURE sp_CustomerLevel
@CustomerID INTEGER,
@CustomerLevel VARCHAR(20) OUT
AS
  DECLARE @PurchaseTotal DECIMAL(8,2)
  SET @PurchaseTotal = (SELECT SUM(amount) FROM transactions tr
                        WHERE tr.CustomerID = @CustomerID)
  IF @PurchaseTotal = 0
  BEGIN
    SET @CustomerLevel = 'Empty'
    RETURN
  END
  IF @PurchaseTotal > 1000
  SET @CustomerLevel = 'Standard'
  ELSE
  SET @CustomerLevel = 'Gold'
```

Μεταβλητή εξόδου

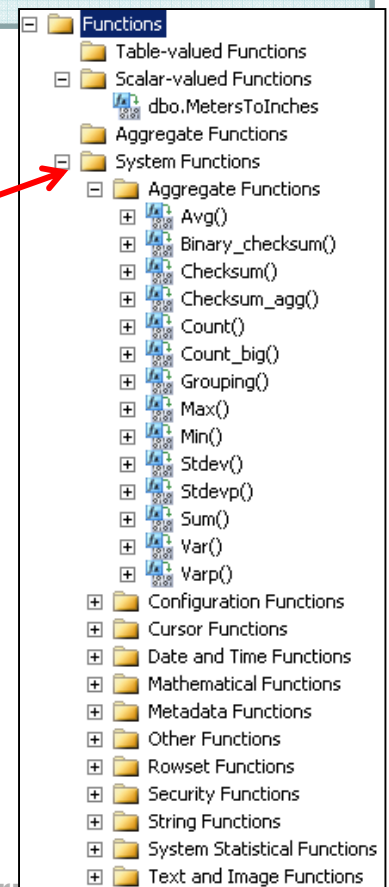
```
Κλήση Sproc sp_CustomerLevel
DECLARE @CustomerLevel VARCHAR(20)
EXEC sp_CustomerLevel 12391, @CustomerLevel OUT
PRINT @CustomerLevel
→ Επιστρέφει Empty
```


C) UDFs σε TSQL



- **UDF (User Defined Functions):** συναρτήσεις του χρήστη οι οποίες αποτελούνται από εντολές (T)SQL και οι οποίες αποθηκεύονται στη βάση δεδομένων.

- Σημειώστε ότι κάθε βάση δεδομένων παρέχει και built-in συναρτήσεις οι οποίες ονομάζονται **System Functions** (π.χ., MAX, MIN, ABS, κτλ)
- Τα UDFs (TSQL) διακρίνονται στις κατηγορίες:
 - **Scalar-valued Functions** → Επιστρέφουν Βαθμωτή Τιμή (Scalar: int, varchar, κτλ.)
 - **Table-Valued Functions** → Επιστρέφουν Πίνακα
 - **Aggregate Functions** → Επιστρέφουν κάποιο συναθροιστικό αποτέλεσμα (π.χ., μια εξειδικευμένη MAX συνάρτηση)



C) UDFs σε TSQL (Παράδειγμα I)



- **Παράδειγμα Δημιουργίας UDF (Scalar) που μετατρέπει τα Μέτρα σε Ίντσες**

```
CREATE FUNCTION dbo.MetersToInches (@Meters DECIMAL(10,3))  
-- Παράμετρος Εισόδου (δια τιμής)  
RETURNS DECIMAL(10,3) -- Τιμή Επιστροφής (δεν μπορούμε να  
περάσουμε τιμές δια αναφοράς ☹)  
AS  
BEGIN  
    DECLARE @Inches DECIMAL(10,3)  
    SET @Inches = (@Meters * 3.281 ) * 12  
    RETURN @Inches -- Επιστροφή Αποτελέσματος  
END
```

Γενικά, ισχύουν οι γνωστοί κανόνες συναρτήσεων που υπάρχουν σε άλλες γλώσσες (π.χ., εμβέλεια μεταβλητών, κτλ)

C) UDFs σε TSQL (Παράδειγμα I)



- **Κλήση UDF (Scalar):**

```
SELECT dbo.MetersToInches(123.45) AS 'Inches'
```

Επιστρέφει:

Inches

4860.473

(1 row(s) affected)

Άλλο Παράδειγμα:

```
SELECT Weight, dbo.MetersToInches(Length), Cost, ...
```

```
FROM Products
```

```
WHERE ProductID = 199232
```

C) UDFs σε TSQL (Παράδειγμα II)



- **UDF χωρίς παράμετρο**

```
CREATE FUNCTION dbo.Two ()  
RETURNS INT -- Τιμή Επιστροφής  
AS  
BEGIN  
    RETURN 2 -- Επιστροφή Αποτελέσματος  
END
```

Query

```
SELECT *  
FROM EMP1  
WHERE ssn = dbo.Two()
```

*Κλήση UDF
στο WHERE*



Αποτέλεσμα Εκτέλεσης Query

ssn	Fname	Minit	Lname	Dno	salary
2	Franklin	T	Wong	5	10000

(1 row(s) affected)

C) UDFs σε TSQL (UDF Table-Valued Function)



```
CREATE FUNCTION dbo.GetEmployees()  
RETURNS @records TABLE -- Δήλωση ότι η Συνάρτηση Επιστρέφει Πίνακα  
( -- Πεδία Σχέσης Επιστροφής  
  EmpID nchar(10) NOT NULL,  
  FirstName nchar(10) NULL  
)  
AS  
BEGIN  
  -- Εισαγωγή Όλων των αποτελεσμάτων της σχέσης Emp1 στη σχέση @records  
  INSERT INTO @records  
    SELECT ssn, fname FROM Emp1  
  -- Τερματισμός της συνάρτησης (το records επιστρέφεται έτσι και αλλιώς)  
  RETURN;  
END
```

Κλήση UDF (Table-Valued) στο FROM
Select *
FROM dbo.GetEmployees()

Σημείωση: Το αποτέλεσμα της συνάρτησης μπορεί να χρησιμοποιηθεί στο FROM

C) UDFs σε TSQL (Διαφορές με Sprocs)



- Τα **UDFs** και στα **Sprocs** έχουν περισσότερες ομοιότητες παρά διαφορές στην TSQL.
- **Βασικές διαφορές:**
 - **A) Στο πως γίνεται η κλήση:** Στα **UDFs** η κλήση συνήθως στο **SELECT** (και κάποτε στο **WHERE** ή **FROM**) ενώ στα **SPROC** η κλήση γίνεται μέσω **EXEC**.
 - **B) Στο πως επιστρέφονται τα αποτελέσματα:** Στα **UDFs** τα αποτελέσματα μπορούν να χρησιμοποιηθούν άμεσα ενώ στα **SPROC** πρέπει να τοποθετηθούν σε **ενδιάμεσους πίνακες**
Π.χ., Για χρήση αποτελεσμάτων της spEmployee σε ένα query τα εισάγω πρώτα σε ένα ενδιάμεσο πίνακα EmpBack

INSERT INTO EmpBack

EXEC spEmployee

[Ενδιάμεσοι Πίνακες σε TSQL]

(Temporary Tables in TSQL)



- Οι **Τοπικοί Ενδιάμεσοι Πίνακες** δημιουργούνται στα πλαίσια μιας σύνδεσης με την βάση (Session) και καταστρέφονται αμέσως μετά.
 - Είναι χρήσιμοι για προσωρινή αποθήκευση ενδιάμεσων αποτελεσμάτων σε προγράμματα TSQL
 - **Δημιουργία:** `CREATE TABLE #tablename (...)`
 - Σημείωση: Ο πίνακας δημιουργείται στην tempdb
 - **Εισαγωγή Δεδ.:** `INSERT INTO #TemporaryTable
EXEC sp_SomeStoredProcedure`
- **Καθολικοί Ενδιάμεσοι Πίνακες:**
 - Έχουν αντίστοιχη λειτουργία με τους Τοπικούς, με την διαφορά ότι σε αυτούς έχουν πρόσβαση όλες οι συνδέσεις με την βάση (sessions).
 - **Δημιουργία:** `CREATE TABLE ##tablename (...)`



C) UDFs σε TSQL

(Άλλες Διαφορές UDF vs. Sprocs)

- Το **UDF** είναι **υποπρόγραμμα** το οποίο γράφεται για να εκτελεί κάποιους **υπολογισμούς** και να επιστρέφει **μια μοναδική τιμή**.
- Το **Sproc** είναι **υποπρόγραμμα** το οποίο γράφεται για να εκτελεί μια **ακολουθία** από **εντολές** και να **επιστρέφει 0** ή **περισσότερες τιμές**.
- Τα **UDFs ΠΡΕΠΕΙ** να επιστρέφουν τιμή με το **RETURN** ενώ τα **SPROCs** μπορούν να χρησιμοποιούν το **RETURN** αλλά **χωρίς να επιστρέφουν τιμή**.
- UDFs μπορεί να χρησιμοποιηθούν στο **SELECT** (δεδομένου του ότι δεν κάνουν επεξεργασία πινάκων)
- Τα UDFs έχουν μόνο **IN** παραμέτρους. Τα SPROCs μπορεί να έχουν **OUT** ή **IN/OUT** παραμέτρους.

SQL/PSM - Persistent Stored Modules

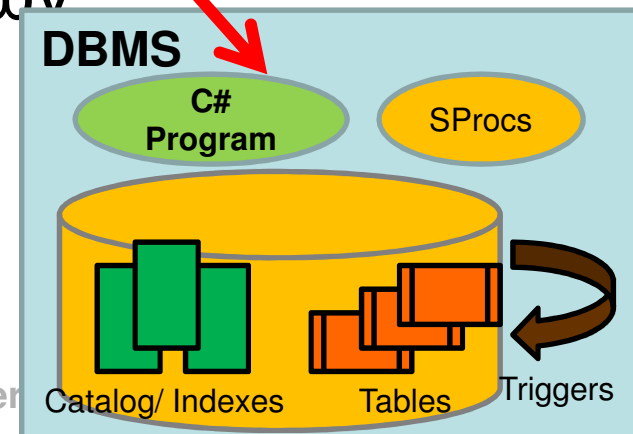


- Στο πρότυπο ANSI, τα **Sprocs** και **UDFs** ορίζονται ως μέρος της ενότητας **SQL/PSM (SQL/Persistent Stored Modules)**
 - Το **Persistent** (δηλ., μόνιμα) υποδηλώνει ότι οι λειτουργίες αυτές **αποθηκεύονται** και **εκτελούνται** από τον διαθέτη της **βάσης** παρά τον **πελάτη**.
- **Βασικό Πρόβλημα** είναι το γεγονός ότι ο κάθε **κατασκευαστής** ακολουθεί δική του σύνταξη για υλοποίηση του **PSM**
 - Η **TSQL** είναι ένα παράδειγμα μη-συμβατής γλώσσας.
 - Το ίδιο ισχύει για την **PL/SQL** (Procedural Language/SQL) της Oracle

Υλοποίηση Προγρ. Αντικειμένων Βάσης μέσω Γλωσσών Υψηλού Επίπεδου



- Τα **προγραμματιστικά αντικείμενα** μιας βάσης (UDFs, Sprocs, Triggers, User-Defined Types, κτλ) μπορούν σε πολλές βάσεις να γραφούν με **σύνταξη γλωσσών υψηλού επιπέδου** αντί με σύνταξη **SQL/PSM (TSQL, PL/SQL)** π.χ.,
 - **SQL Server:** Χρήση .NET και C# (δες SQLCLR: SQL Common Language Runtime). Ακολουθεί παράδειγμα στην επόμενη διαφάνεια.
 - **Oracle και DB2:** Υλοποίηση UDFs σε C ή JAVA
- **Λόγοι Χρήσης** αυτών των Τεχνικών:
 - Πιο πολλές **δυνατότητες** σε μια **γλώσσα υψηλού επιπέδου**
 - **Επίδοση, Ασφάλεια, κτλ.**



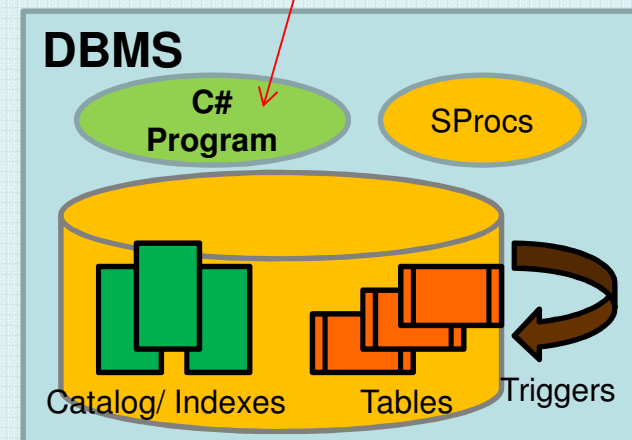
Υλοποίηση Προγρ. Αντικειμένων Βάσης μέσω Γλωσσών Υψηλού Επίπεδου



```
using System;
using System.Data;
using System.Data.Sql;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;
public partial class UserDefinedFunctions
{
    [Microsoft.SqlServer.Server.SqlFunction]
    public static SqlString RiskProfile()
    {
        // Put your code here
        return new SqlString("Hello");
    }
};
```

Σημείωση: Δεν χρειάζεται να καταλάβετε
τι αναπαριστούν οι βιβλιοθήκες στο *using*

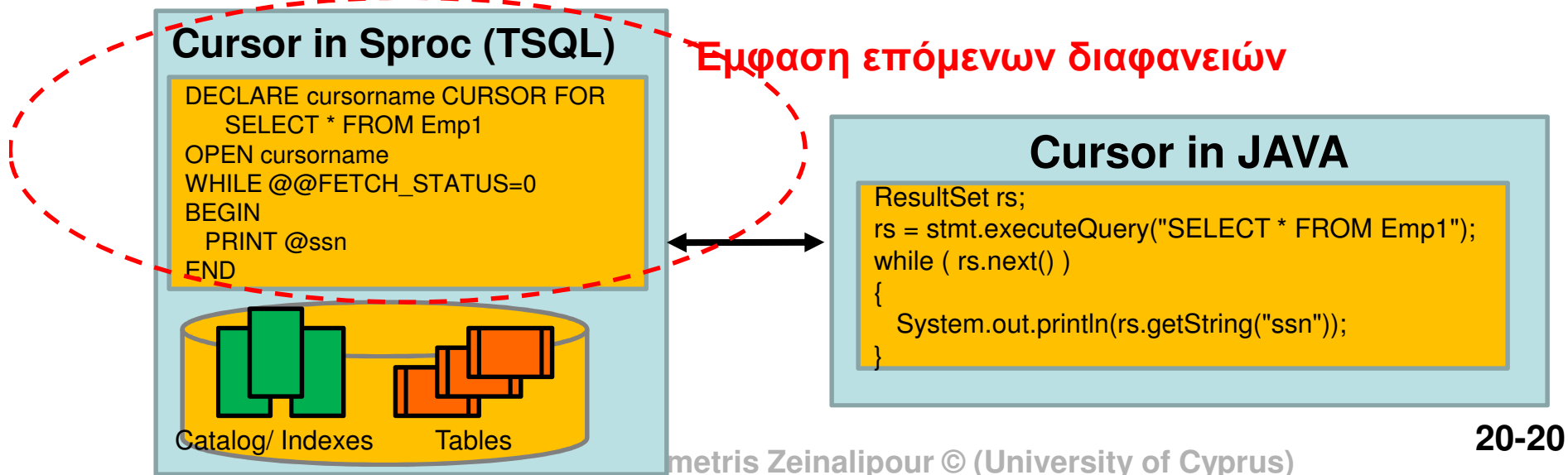
**Υλοποίηση ενός UDF
σε C#
το οποίο
ενσωματώνεται στις
λειτουργίες της βάσης**



D) Cursors (Δρομείς) σε TSQL



- Γνωρίζουμε ότι τα **αποτελέσματα** επερωτήσεων επιστρέφονται σε μια επερώτηση υπό **μορφή μιας σχέσης**
 - Π.χ., “**SELECT * FROM Emp1**” επιστρέφει ένα ολόκληρο πίνακα.
- Εάν θέλουμε να **επεξεργαστούμε** τα **αποτελέσματα** αυτά **γραμμή-γραμμή** (αντί να απλά να τυπωθούν), τότε χρησιμοποιούμε την έννοια του **Δρομέα (Cursor)**.
 - **Cursors** υπάρχουν και στην **JAVA** όπως είδατε στο φροντιστήριο.



D) Cursors (Δρομείς) σε TSQL (Παράδειγμα Χρήσης)



```
USE ep1342
DECLARE @ssn nchar(10), @fname nchar(10)
DECLARE employee_cursor CURSOR FOR -- Δήλωση μεταβλητής τύπου Cursor
    SELECT ssn,fname FROM Emp1 -- Επερώτηση που συνδέεται με τον Cursor
OPEN employee_cursor -- Άνοιγμα Cursor (Εκτέλεση Επερώτησης)
-- Ανάγνωση πρώτης γραμμής στις μεταβλητές id, fname
FETCH NEXT FROM employee_cursor
INTO @ssn, @fname
WHILE @@FETCH_STATUS=0 -- όσο δεν άδειασαν τα αποτελέσματα
BEGIN
    PRINT RTRIM(@ssn) + ', ' + RTRIM(@fname) -- RTRIM: φεύγει right spaces
    FETCH NEXT FROM employee_cursor -- Ανάγνωση επόμενης γραμμής
    INTO @ssn, @fname
END
CLOSE employee_cursor -- Κλείσιμο Cursor
DEALLOCATE employee_cursor -- Αποδέσμευση Πίνακα Ενδιάμεσων
Αποτελεσμάτων που χρησιμοποιεί ο SqlServer για το Result του Cursor .
Εναλλακτικά μένει στην μνήμη μέχρι το Κλείσιμο του Session.
```

0=OK, -1=ERROR,

ΤΥΠΩΝΕΙ
1, 1
2, Franklin
3, Alicia
4, Jennifer
5, Ramesh
6, Joyce
7, Ahmad
8, James
1, 1

D) Cursors (Δρομείς) σε TSQL (Χαρακτηριστικά Cursor)



- **Κατεύθυνση Cursor:** Η default λειτουργία του cursor είναι να κινείται μπροστά **NEXT (default)**, εγγραφή-εγγραφή μέσα σε ένα αποτέλεσμα. Υπάρχουν ορίσματα (**SCROLL**) για να κινείται διαφορετικά **FIRST, LAST, PRIOR, RELATIVE, κτλ.**
- **Ενημέρωση Αποτελεσμάτων:** Είναι δυνατό να ενημερώνεται το αποτελέσματα που προσπελάυνεται από ένα CURSOR (**READ ONLY (default) | UPDATE**)
 - π.χ., κατά την προσπέλαση αύξησε ένα γνώρισμα κατά 1
- **Εμβέλεια Cursor:** Η εμβέλεια ενός CURSOR μπορεί να περιοριστεί τοπικά μέσα στο ίδιο batch) (**LOCAL (default) | GLOBAL**)
- Στο manual υπάρχουν αρκετές άλλες εξειδικεύσεις που προσφέρονται από την TSQL

D) Cursors (Δρομείς) σε TSQL (Παράδειγμα Χρήσης)



ISO Syntax

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR  
  FOR select_statement  
  [ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]  
[;]
```

Transact-SQL Extended Syntax

```
DECLARE cursor_name CURSOR [ LOCAL | GLOBAL ]  
  [ FORWARD_ONLY | SCROLL ]  
  [ STATIC | KEYSSET | DYNAMIC | FAST_FORWARD ]  
  [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]  
  [ TYPE_WARNING ]  
  FOR select_statement  
  [ FOR UPDATE [ OF column_name [ ,...n ] ] ]  
[;]
```

D) Cursors (Δρομείς) σε TSQL

(Παράδειγμα Script για Backup Βάσεων)



```
DECLARE @name VARCHAR(50) -- database name
DECLARE @path VARCHAR(256) -- path for backup files
DECLARE @fileName VARCHAR(256) -- filename for backup
DECLARE @fileDate VARCHAR(20) -- used for file name
```

```
SET @path = 'C:\Backup\' -- κατάλογος στον οποίο θα γίνει το backup
```

```
SELECT @fileDate = CONVERT(VARCHAR(20),GETDATE(),112)
```

```
DECLARE db_cursor CURSOR FOR
SELECT name
FROM master.dbo.sysdatabases
WHERE name NOT IN ('master','model','msdb','tempdb')
```

Μορφοποίηση Ημερ. με
style 112, δηλ. σε:
20091119

```
-- Continued On Next Page
```

Πίνακας του Καταλόγου που περιέχει τα
ονόματα όλων των βάσεων

D) Cursors (Δρομείς) σε TSQL (Παράδειγμα Χρήσης)



-- Continued From Previous Page

```
OPEN db_cursor -- άνοιγμα cursor
```

```
FETCH NEXT FROM db_cursor -- ανάκτηση επόμενης τιμής σε τοπική var  
INTO @name
```

```
WHILE @@FETCH_STATUS = 0 π.χ., 'C:\Backup\epi342_20091119.BAK'  
BEGIN
```

```
-- δημιουργία filename
```

```
SET @fileName = @path + @name + '_' + @fileDate + '.BAK'
```

```
-- εντολή για backup βάσης στο directory/name @fileName
```

```
BACKUP DATABASE @name TO DISK = @fileName
```

```
-- Το RESTORE DATABASE κάνει restore την βάση
```

```
FETCH NEXT FROM db_cursor INTO @name
```

```
END
```

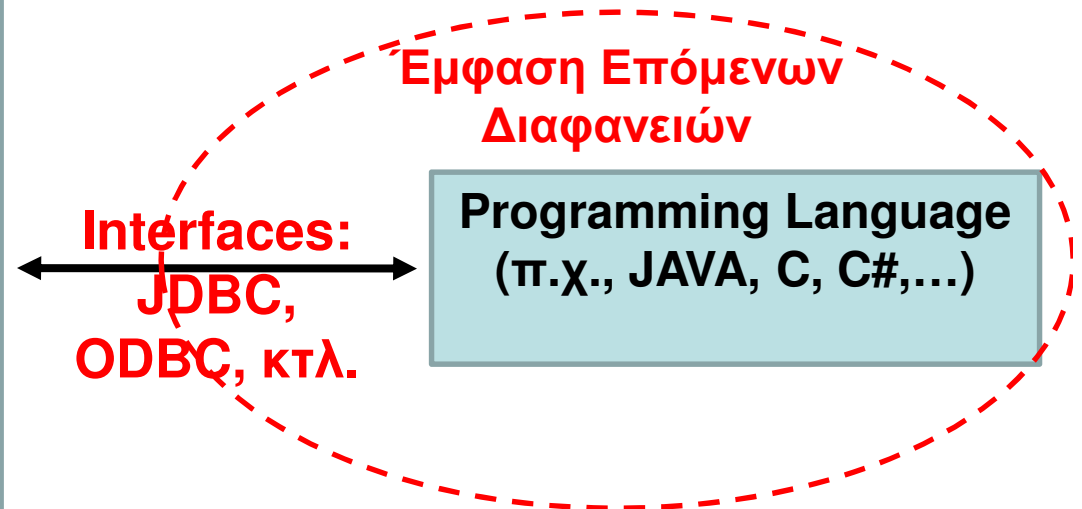
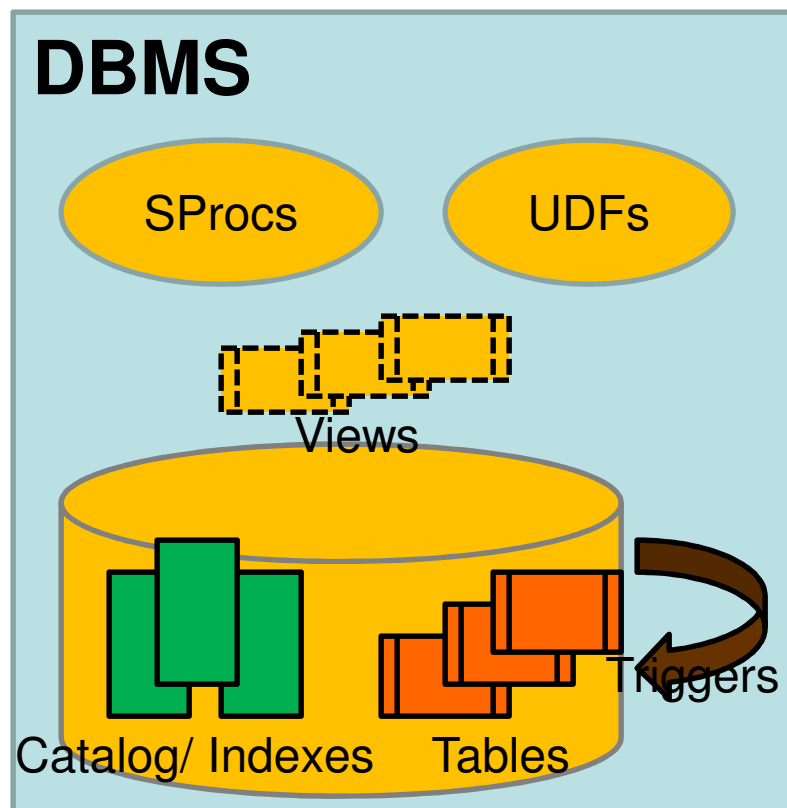
```
CLOSE db_cursor
```

```
DEALLOCATE db_cursor
```

Εξωτερικός Προγραμματισμός DB

Internal DB Programming

External DB Programming



Εξωτερικός Προγραμματισμός DB



External DB Programming

- Ενσωματωμένη SQL (Embedded SQL)
 - SQLCA και SQLJ
- Ενσωματωμένη SQL με **Κλήσεις
Συναρτήσεων (Embedded SQL with
APIs and Functions Calls)**
 - JDBC και SQL/CLI (ex ODBC)

Εμπέδωση SQL σε Γλώσσα C (SQLCA) Παράδειγμα



```
include "sqlca.h"
```

```
// Σύνδεση με την βάση δεδομένων μέσω εξειδικευμένων εντολών
```

```
int loop; // μεταβλητή C
```

```
EXEC SQL BEGIN DECLARE SECTION;
```

```
    varchar dname[16], fname[16], ...;
```

```
    char ssn[10], bdate[11], ...;
```

```
    int dno, dnumber, SQLCODE, ...;
```

```
EXEC SQL END DECLARE SECTION;
```

```
...
```

```
loop = 1;
```

```
while (loop) {
```

```
    prompt ("Enter SSN: ", ssn);
```

```
    EXEC SQL
```

```
        select FNAME, LNAME, ADDRESS, SALARY
```

```
        into :fname, :lname, :address, :salary
```

```
        from EMPLOYEE where SSN == :ssn;
```

```
        if (SQLCODE == 0) printf(fname, ...);
```

```
        else printf("SSN does not exist: ", ssn);
```

```
        prompt("More SSN? (1=yes, 0=no): ", loop);
```

```
END-EXEC
```

```
} Εάν το αποτέλεσμα ήταν σύνολο πλειάδων τότε κάπου εδώ θα μπορούσε να χρησιμοποιηθεί ένας Cursor της SQLCA για εκτύπωση αποτελεσμάτων
```

Δήλωση Μεταβλητών μεταφοράς δεδομένων από SQL σε C

Γνωρίσματα Σχέσης Βάσης

Αρχικοποίηση μεταβλητών μεταφοράς δεδομένων

Έλεγχος Λαθών

Εμπέδωση SQL σε **JAVA (SQLJ)**



- Υπάρχει η δυνατότητα **εμπέδωσης SQL** εκφράσεων και σε **JAVA**, κατ' αντίστοιχο τρόπο με το **SQLCA** που είδαμε στην προηγούμενη διαφάνεια.
- Αυτή η προσπάθεια, που **ξεκίνησε** από την **Oracle** ονομάζεται **SQLJ** και έχει στόχο να κάνει τον προγραμματισμό **JAVA+Oracle** ευκολότερο.
 - Στο **SQLJ**, οι εμπεδωμένες **SQL** εκφράσεις μετατρέπονται από τον μεταφραστή **sqlj** σε **JDBC** κλήσεις, οπότεν αυτή η τεχνολογία είναι απλά ένα επίπεδο αφαιρετικότητας πάνω από το **JDBC**.

Εμπέδωση SQL σε JAVA (SQLJ) Παράδειγμα



```
ssn = readEntry("Enter a SSN: ");  
try {  
    #sql{select FNAME, LNAME, ADDRESS, SALARY  
    into :fname, :lname, :address, :salary  
    from EMPLOYEE where SSN = :ssn};  
}  
catch (SQLException se) {  
    System.out.println("SSN does not exist:  
    ", +ssn);  
    return;  
}
```

```
System.out.println(fname + " " + lname + ... ) ;
```

Ενσωματωμένη SQL μέσω Κλήσεων Συναρτήσεων



- Ο πιο παραδοσιακός τρόπος σύνδεσης μια εφαρμογής με μια DBMS είναι μέσω **κλήσεων συναρτήσεων**.
 - Εάν η εφαρμογή είναι γραμμένη σε **JAVA** τότε χρησιμοποιείται το **JDBC**, το οποίο είδατε στο φροντιστήριο.
- Εάν η εφαρμογή είναι γραμμένη σε άλλη γλώσσα τότε χρησιμοποιούνται άλλα API (Application Protocol Interfaces και βιβλιοθήκες).
 - **SQL/CLI** (ο διάδοχος του ODBC) ή **OLEDB** (σε Win)
 - Microsoft's ActiveX Data Objects **ADO** ('96), **ADO.NET**
 - Σύνοψη τεχνολογιών Πρόσβασης σε Δεδομένα από την Microsoft:
<http://support.microsoft.com/kb/190463>

Ενσωματωμένη SQL μέσω Κλήσεων Συναρτήσεων



Για σύνδεση μέσω ODBC σε Windows απαιτείται η δήλωση μιας βάσης στον Data Source Administrator των Windows

