Διάλεξη 17: Προγραμματισμός Βάσης Δεδομένων Ι

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:
Εισαγωγή στις έννοιες:

- Όψεις (Views) σε TSQL
- Αποθηκευμένες Διαδικασίες (Stored Procedures) σε TSQL
- Συναρτήσεις Χρήστη (UDFs) σε TSQL

Διδάσκων: Παναγιώτης Ανδρέου

Όψεις/Views

- 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**!
- Σύνταξη δημιουργίας όψεων:
  CREATE VIEW <view-name> **AS** <statement>
- Παραδείγματα Όψεων
  - Θεωρήστε πίνακα PERSON (όπως φαίνεται δεξιά)
  - **CREATE VIEW** GET_ALL_PERSONS **AS**
    SELECT * FROM PERSONS
  - **CREATE VIEW** GET_MALE_PERSONS **AS**
    SELECT * FROM PERSONS WHERE GENDER='M'
  - **CREATE VIEW** GET_FEMALE_PERSONS **AS**
    SELECT * FROM PERSONS WHERE GENDER='F'

PERSON (P)

PID	PNAME	GENDER
1	Andreas	M
2	Kostas	M
3	Maria	F
4	Eleni	F

Χαρακτηριστικά Όψεων

- **Τα δεδομένα μιας όψης ΔΕΝ αποθηκεύονται φυσικά** κάπου (τα δεδομένα αποθηκεύονται στους πηγαίους πίνακες)
- **Μπορούν να χρησιμοποιηθούν όπως τους υπόλοιπους πίνακες** (σε επερωτήσεις, συνενώσεις, συναθροίσεις, κτλ.)
- Περιέχουν **ΠΑΝΤΑ ενημερωμένα δεδομένα**.
- Μπορούμε να εκτελέσουμε αλλαγές σε μια όψη (INSERT/UPDATE/DELETE) κάτω από τις εξής προϋποθέσεις
  - Ενημερώσεις γίνονται μόνο εάν η όψη ορίζεται από ένα base-tables (όχι από περισσότερα base-tables)
  - Επίσης, όψεις με aggregates & group by ΔΕΝ ενημερώνονται.
- Εκτέλεση μιας Όψης είναι πιο γρήγορο από εκτέλεση της αντίστοιχης SELECT επερώτησης.
  - Αυτό επειδή το VIEW είναι ήδη μεταγλωττισμένο και έτοιμοι για εκτέλεση
  - Ο query optimizer εισάγει κάποιο overhead.

Πλεονεκτήματα Όψεων

- **Ευελιξία στην ανάπτυξη εφαρμογών**
  - μπορούμε να αναπαραστήσουμε περίπλοκες επερωτήσεις ως νοητούς πίνακες.
  - Μπορούμε να δώσουμε διαφορετική αντίληψη στον κάθε χρήστη για την βάση
- **Ασφάλεια**
  - Επιτρέπουν στον DBA να εκθέσει μόνο τις πίνακες/στήλες που επιθυμεί σε συγκεκριμένες ομάδες χρηστών.
  - Η δομή της ΒΔ μπορεί να μετατραπεί σε “μαύρο κουτί”
- **Συμβατή με προηγούμενες εκδόσεις της βάσης**
  - Ακόμα και αν έχει αλλάξει η δομή της ΒΔ, οι όψεις μπορούν να προσομοιώσουν την προηγούμενη δομή

Παράδειγμα Όψης σε SQL

Όψη με συνάθροιση στο SELECT.

```
CREATE VIEW Emp_Sal2
AS
   SELECT      dno, SUM(salary) AS SUM_SALARY
   FROM        PERSON
   GROUP BY    dno
```

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

Σύνταξη Όψεων σε T-SQL

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)

Παράδειγμα Δημιουργίας Όψης σε T-SQL

```
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
```

Stored Procedures σε TSQL

- **Stored Procedures (SPs)**, είναι διαδικασίες (T)SQL οι οποίες **αποθηκεύονται** στη **βάση δεδομένων** και οι οποίες μπορούν να **πάρουν ορίσματα** και να **επιστρέψουν τιμές**.
- **Σύνταξη:** Παρόμοια με τις όψεις, αλλά επιτρέπουν και παραμέτρους
  CREATE PROCEDURE <sp-name>[(<params)] **AS** <statement(s)>

- Παραδείγματα Stored Procedures
  - Θεωρήστε πίνακα PERSON (όπως φαίνεται δεξιά)
  - **CREATE PROCEDURE** GET_ALL_PERSONS **AS**
    SELECT * FROM PERSONS
  - **CREATE PROCEDURE** GET_PERSONS_BY_GENDER (
    @GENDER CHAR(1)
)
    AS
    SELECT * FROM PERSONS WHERE GENDER=@GENDER

PERSON (P)

PID	PNAME	GENDER
1	Andreas	M
2	Kostas	M
3	Maria	F
4	Eleni	F

Εκτέλεση/Ακύρωση Stored Procedure

- **Εκτέλεση SP (χωρίς Παραμέτρους) μέσω EXECUTE**
  - **EXECUTE** GET_ALL_PERSONS
- **Εκτέλεση SP (με Παραμέτρους) μέσω EXECUTE**
  - **EXECUTE** GET_PERSONS_BY_GENDER 'M'

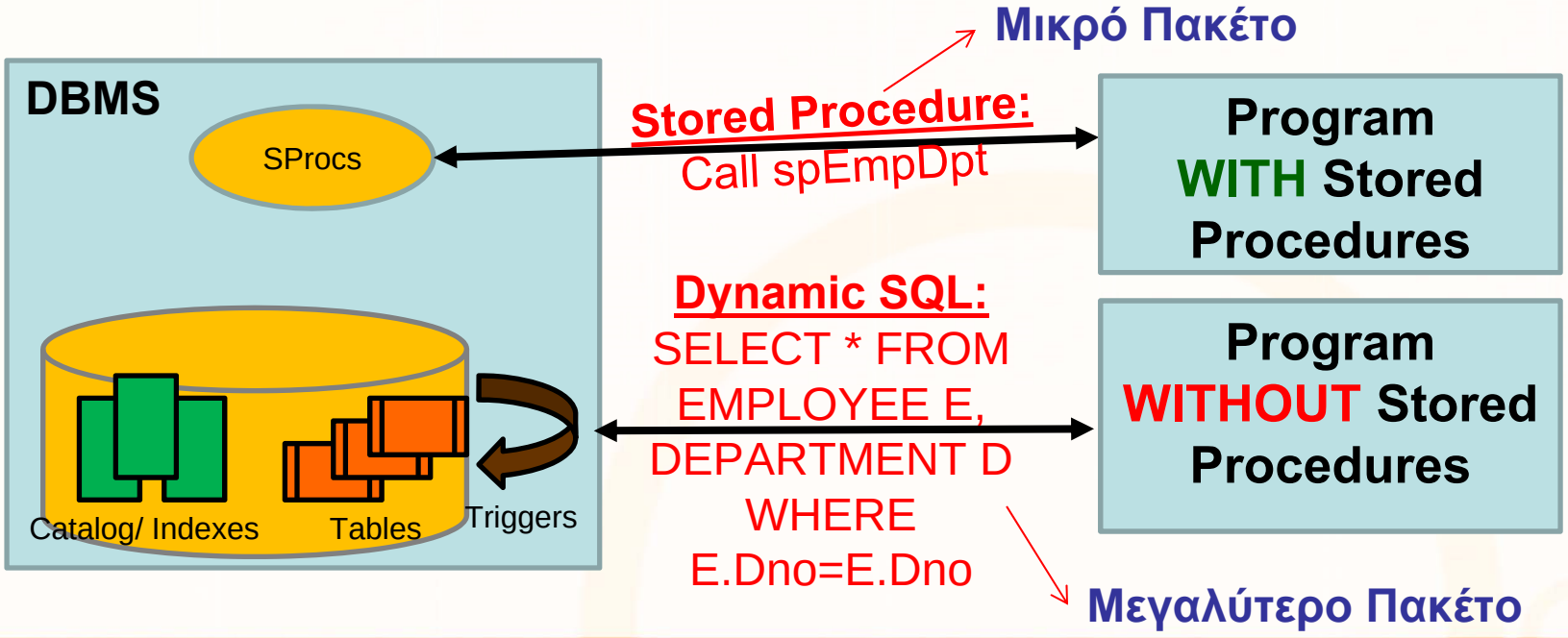
- **Ακύρωση SP**
  - **DROP PROCEDURE** spEmployee

- **Μεταβολή Sproc**
  - **ALTER PROCEDURE** GET_PERSONS_BY_GENDER (
    @GENDER CHAR(1)
    @ID INT
)
    AS
    SELECT * FROM PERSONS WHERE GENDER=@GENDER AND PID>=@PID

Χαρακτηριστικά Stored Procedures

Βασικό Πλεονέκτημα SPs:

- **Επίδοση/Μειωμένη Κίνηση Δικτύου:** Το **Sproc** είναι **Precompiled/Optimized** πριν την κλήση του. Επίσης **μειώνεται** η **κυκλοφορία δεδομένων** μεταξύ DBMS και προγράμματος εφαρμογής.
  - Υπολογισμοί με πολλά δεδομένα μπορούν να γίνουν απευθείας στην ΒΔ χωρίς να μετακινούνται μπροσ/πίσω στην εφαρμογή



Πλεονεκτήματα/Μειονεκτήματα SPs

- **Πλεονεκτήματα SPs**
  - **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 SPs**
  - **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

Stored Procedures: Παράμετροι Εισόδου

- Οι παράμετροι εισόδου/εξόδου δηλώνονται με πρόθεμα το **@** (T-SQL, PL/SQL)
- Πολλές παράμετροι εισόδου διαχωρίζονται με κόμμα (,)

- **Παράδειγμα:**

```
ALTER PROCEDURE GET_PERSONS_BY_GENDER (
        @GENDER CHAR(1),
        @ID INT
)
AS
SELECT      *
FROM        PERSONS
WHERE       GENDER=@GENDER AND PID>=@PID
```

Stored Procedures: Παράμετροι Εξόδου

- Οι παράμετροι εξόδου δηλώνονται με το επίθεμα **OUT**
- Ένα SP υποστηρίζει πολλές παραμέτρους εξόδου παρόμοια με μεταβλητές διά αναφοράς

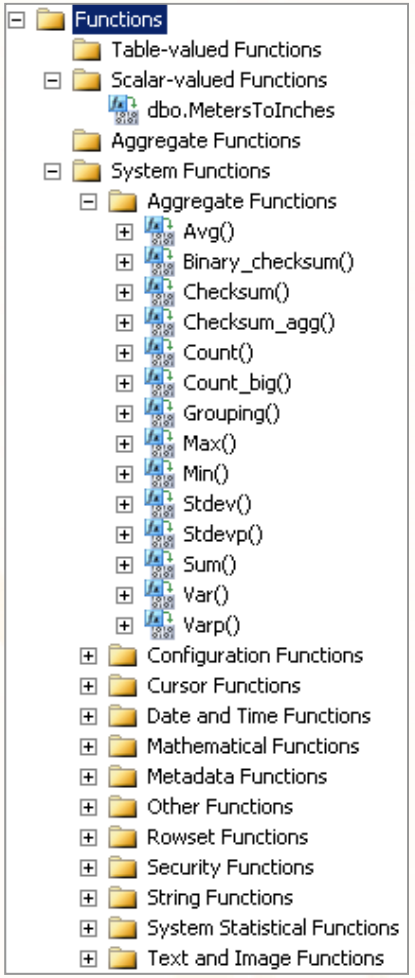
- **Παράδειγμα:**

```
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'          END
        ELSE
        BEGIN     SET @CustomerLevel = 'Standard'       END
```

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

UDF (User Defined Functions)

- **UDF (User Defined Functions): συναρτήσεις** του χρήστη (παρόμοιες με SPs) 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)
  - **Table-Valued Functions**
    Επιστρέφουν Πίνακα
  - **Aggregate Functions**
    Επιστρέφουν κάποιο συναθροιστικό αποτέλεσμα (π.χ., μια εξειδικευμένη MAX συνάρτηση)



UDF (User Defined Functions)

- **Σύνταξη (μεταβλητή εξόδου – <u>απλός τύπος</u>):**

```
CREATE FUNCTION <function-name>
(<parameter-list>)          -- Παράμετρος Εισόδου (δια τιμής)
RETURNS <data type>         -- Μεταβλητή Εξόδου (δεν μπορούμε να
                               περάσουμε τιμές δια αναφοράς )
AS
BEGIN
      RETURN <variable or value>
END
```

- **Σύνταξη (μεταβλητή εξόδου - <u>πίνακας</u>):**

```
CREATE FUNCTION <function-name> .
(<parameter-list>)          -- Παράμετρος Εισόδου (δια τιμής)
RETURNS TABLE
AS
RETURN (    <select statement>     )
```

Παραδείγματα UDFs

```
CREATE FUNCTION dbo.MetersToInches (
    @Meters DECIMAL(10,3)
)
    RETURNS DECIMAL(10,3)
AS
  BEGIN
    DECLARE @Inches DECIMAL(10,3)
    SET @Inches = (@Meters * 3.281 ) * 12
    RETURN @Inches
  END
```

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

Παραδείγματα UDFs

- **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

Παραδείγματα UDFs

- **UDF χωρίς παράμετρο**

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

- **Χρήση UDF σε Query**

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

Αποτέλεσμα Εκτέλεσης Query

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

(1 row(s) affected)

UDF (User Defined Functions)

- **Παρόμοια με Stored Procedure**

- **CREATE FUNCTION** getPERSONS(

```
        @PERSON_ID INT
    )
    RETURNS TABLE
    AS
    RETURN
    (
        SELECT *
        FROM PERSONS
        WHERE       PERSON_ID=@PERSON_ID
    )
```

Παραδείγματα UDFs

- **Για δυναμικούς πίνακες ή πίνακες που χρειάζονται περισσότερη επεξεργασία τότε χρειαζόμαστε τον ειδικό τύπο μεταβλητής TABLE** που επιστρέφει πίνακα σαν μεταβλητή

```
CREATE FUNCTION dbo.GetEmployees()
RETURNS @records TABLE (          -- Δήλωση μορφής πίνακα 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
```

Διαφορές μεταξύ SPs vs. UDFs

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

```
INSERT INTO EmpBack
EXEC spEmployee
```

Διαφορές μεταξύ SPs vs. UDFs (συν.)

- 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.