EPL342 –Databases

Lab 10

SQL-DML III (Views, Triggers, Functions)

DB Programming I (Stored Procedures, Cursors)

Northwind Database Queries

Create the following views:

1. **view_EmployeeFullNames**: Displays the ID and Full Name (Last name + Firstname) of each employee
2. **view_NumberOfEmployeesByCity**: create a view that displays the city and number of employees that live in
3. **view_TotalSalesByCustomerCity**: create a view that displays the total number of sales and total number of orders for all customer's cities
4. Execute the following sql statement

   sp_helptext 'view_TotalSalesByCustomerCity'
5. To avoid displaying the sql text of view 3, enforce encryption and execute sp_helptext again to see that you have done it properly

TRIGGERS

Whenever a trigger is executed two tables are utilized:

- The **inserted** table:

  used for INSERT and UPDATE triggers

- The **deleted** table

  used for DELETE and UPDATE triggers

Both tables are valid only for the duration of the trigger

TRIGGERS (inserted, deleted tables) - Example

ID	Name
1	John
2	Anne
3	Marius
4	Steven

```
INSERT INTO table
VALUES(5,'Potter')
```

inserted

ID	Name
5	Potter

deleted

ID	Name

ID	Name
1	John
2	Anne
3	Marius
4	Steven
5	Potter

```
DELETE FROM table
WHERE ID = 1
```

inserted

ID	Name

deleted

ID	Name
1	John

ID	Name
2	Anne
3	Marius
4	Steven
5	Potter

```
UPDATE table
SET Name='Harry'
WHERE ID = 5
```

inserted

ID	Name
5	Harry

deleted

ID	Name
5	Potter

ID	Name
2	Anne
3	Marius
4	Steven
5	Harry

Northwind Database Queries

Create the following triggers:

1. **tr_AUDIT_Employees -** We need to track down **when** and by **who** a new employee is inserted to the database or a current employee is updated.
   - Create 4 new columns to the Employee table (CREATE_ID, CREATE_DATE, UPDATE_ID, UPDATE_DATE)
   - To get the current date use the GetDate() function
   - To get the current user logged in use (SELECT **USER**)
   - After you finish the trigger, test it by adding new employees and by changing employee names.
2. **tr_ORDER_TOTAL -** We need to update the total amount for each order automatically.
   - Create a new column (TOTAL type: money) to the **Orders** table
   - This column must update the total amount for each order (Lab 10-Query 9) whenever an order detail is inserted or updated

Northwind Database Queries

Create the following functions:

1. **fn_ABS -** input: int, output: positive int
2. **fn_DATE_ONLY -** input: datetime, output: string (10 chars) with the format dd/mm/yyyy
3. **fn_LEFT -** input: string A, int B, output: substring of string A, from char 0 to B
   (e.g., fn_LEFT(‘Harry Potter’, 5)=‘Harry’
4. **fn_REVERSE** – input: string A, output: reverse string A
   (e.g., fn_REVERSE(‘*Avada Kedavra*’)=‘arvadeK adavA’)

Department of Computer Science
University of Cyprus

EPL342 –Databases

Lab 10

~~SQL-DML III (Views, Triggers, Functions)~~

DB Programming I (Stored Procedures, Cursors)

Creating Stored Procedure

- In order to “save” an SQL statement in MS SQL server you should create a Stored Procedure for it
  - Accept input parameters and return multiple values in the form of output parameters to the calling procedure or batch.
  - Contain programming statements that perform operations in the database, including calling other procedures.
  - Return a status value to a calling procedure or batch to indicate success or failure (and the reason for failure).

Creating Stored Procedure

- Simple (general) syntax:

```
CREATE PROCEDURE procedure_name
    [ { @parameter data_type }  [ = default ] [ OUT | OUTPUT ] [READONLY]] [ ,...n ]
    [ WITH <procedure_option> [ ,...n ] ]
AS {
[ BEGIN ]
        sql_statement [;] [ ...n ]
[ END ]
} [;]
```

- Example

```
CREATE PROCEDURE uspGetEmployeesTest2
        @LastName nvarchar(50), @FirstName nvarchar(50)
AS
        SET NOCOUNT ON;
        SELECT FirstName, LastName, Department
        FROM vEmployeeDepartmentHistory
        WHERE FirstName = @FirstName AND LastName = @LastName;
```

- http://msdn.microsoft.com/en-us/library/ms345415.aspx

Hogwarts table

ID	Name	SID
1	Albus Dumbledore	NULL
2	Argus Filch	1
3	Filius Flitwick	1
4	Rubeus Hagrid	1
5	Madam Hooch	1
6	Gilderoy Lockhart	1
7	Minerva McGonagall	1
8	Severus Snape	1
9	Cedric Diggory	5
10	Harry Potter	7
11	Ron Weasly	7
12	Hermione Granger	7
13	Any Slytherin	8
14	Draco Malfoy	8
15	Fred Weasly	3
16	George Weasly	3

Hogwarts Cursor Example

```
DECLARE @ID int
DECLARE @Name nvarchar(100)

DECLARE c CURSOR FAST_FORWARD
FOR SELECT ID, Name FROM Hogwarts

OPEN c

FETCH NEXT FROM c INTO @ID, @Name
WHILE @@FETCH_STATUS=0
BEGIN
    --YOUR CODE HERE
    FETCH NEXT FROM c INTO @ID, @Name
END
CLOSE c
DEALLOCATE c
```

DECLARE: Variables for storing intermediate results

Specifies a FORWARD_ONLY, READ_ONLY cursor with performance optimizations enabled

OPEN: Initialize cursor and execute T-SQL statement

FETCH: Move cursor to the 1st record

WHILE: more records exist

FETCH: Move cursor to the next record

CLOSE: Release the current result set

DEALLOCATE: Removes the cursor reference and all associate data structures

Hogwarts Cursor Example

```
…
--YOUR CODE HERE
PRINT CAST(@ID as nvarchar) + ' ' + @Name
…
```

Execute the statement by pressing F5

```
1           Albus Dumbledore
2           Argus Filch
3           Filius Flitwick
4           Rubeus Hagrid
5           Madam Hooch
6           Gilderoy Lockhart
7           Minerva McGonagall
8           Severus Snape
9           Cedric Diggory
10          Harry Potter
11          Ron Weasly
12          Hermione Granger
13          Any Slytherin
14          Draco Malfoy
15          Fred Weasly
16          George Weasly
```

Recursive Procedures

How can we print the structure of the Hogwarts school? →

1. There is one Headmaster

   Headmaster:Albus Dumbledore

2. There are many teachers who are supervised by the headmaster

   Argus Filch, Filius Flitwick, Rubeus Hagrid,…

3. There are many students who are supervised by the teachers

   (e.g., Minerva McGonagall: Harry Potter, Ron Weasly, Hermione Granger)

```
Headmaster:Albus Dumbledore
  Argus Filch
  Filius Flitwick
    Fred Weasly
    George Weasly
  Rubeus Hagrid
  Madam Hooch
    Cedric Diggory
  Gilderoy Lockhart
  Minerva McGonagall
    Harry Potter
    Ron Weasly
    Hermione Granger
  Severus Snape
    Any Slytherin
    Draco Malfoy
```

Recursive Procedures

How can we find the IDs and names of persons that are supervised by a person with ID=A?

```
SELECT    ID, Name
FROM      Hogwarts
WHERE     SID=A
```

Let’s modify our cursor example to accept the SID as parameter and print the ID and names of all persons supervised by another person.

Hogwarts Tree

1. Create procedure [Hogwarts_Tree]
   - Input Parameters: @sid int
   - Output Parameters: <nothing>
   - Modify the cursor example to print the ID and name of all persors supervised by person with ID=@sid
   - Execute the procedure with @sid=1 and @sid=7

@sid=1

```
2 Argus Filch
3 Filius Flitwick
4 Rubeus Hagrid
5 Madam Hooch
6 Gilderoy Lockhart
7 Minerva McGonagall
8 Severus Snape
```

@sid=7

```
10 Harry Potter
11 Ron Weasly
12 Hermione Granger
```

Hogwarts Tree

Question: How can we extend the Hogwarts_Tree SP to print the persons that are supervised by each printed so far?

Answer: by calling the procedure with the @id of the person at the current cursor position

Include the following statement after

```
PRINT CAST(@ID as nvarchar) + ' ' + @Name
EXEC Hogwarts_Tree @ID
```

Execute the procedure with @sid=1

Hogwarts Tree

```
2 Argus Filch
Msg 16915, Level 16, State 1, Procedure hog, Line 9
A cursor with the name 'c' already exists.
Msg 16905, Level 16, State 1, Procedure hog, Line 11
The cursor is already open.
3 Filius Flitwick
…
```

Problem: Unlike common programming languages The **<c>** cursor’s scope extends to the inner calls of the stored procedure

Answer: Declare the <c> cursor as LOCAL

DECLARE c CURSOR FAST_FORWARD →

DECLARE c CURSOR **LOCAL** FAST_FORWARD

Hogwarts Tree

- Execute the procedure with @sid=1
- Notice that the order is correct (e.g., Harry Potter, Ron Weasly and Hermiony Granger are supervised by Minerva McGonnagall)
- Albus Dumbledore is not printed ☹
- We need to include some spaces to distinguish supervisors from supervisees
- One way to do that is to print spaces according to the level of recursion (e.g., Albus Dumbledore-1, Argus Filch-2, Harry Potter-3
- We can get the level of recursion easily using the @@NESTLEVEL

$\longrightarrow$ **@sid=1**

```
2 Argus Filch
3 Filius Flitwick
15 Fred Weasly
16 George Weasly
4 Rubeus Hagrid
5 Madam Hooch
9 Cedric Diggory
6 Gilderoy Lockhart
7 Minerva McGonagall
10 Harry Potter
11 Ron Weasly
12 Hermione Granger
8 Severus Snape
13 Any Slytherin
14 Draco Malfoy
```

Hogwarts Tree

- To print a number of spaces we can use the SPACE(int x) function (prints x spaces)
- Modify Hogwarts_Tree

  PRINT CAST(@ID as nvarchar) + ' ' + @Name)

  →

  PRINT SPACE(@@NESTLEVEL * 2) + CAST(@ID as nvarchar) + ' ' + @Name

Execute the procedure with @sid=1 →

@sid=1

```
2 Argus Filch
 3 Filius Flitwick
   15 Fred Weasly
   16 George Weasly
 4 Rubeus Hagrid
 5 Madam Hooch
   9 Cedric Diggory
 6 Gilderoy Lockhart
 7 Minerva McGonagall
   10 Harry Potter
   11 Ron Weasly
   12 Hermione Granger
 8 Severus Snape
   13 Any Slytherin
   14 Draco Malfoy
```

Hogwarts Tree

Implement the following tasks

1. Extend the Hogwarts_Tree SP to print also the name of the person from the first call of the procedure (e.g., @sid=1→print Albus Dumbledore.
2. Extend the Hogwarts_Tree SP to save the records in **an existing table T (e.g., Results)**
3. Extend the Hogwarts_Tree SP return the results of T **ONLY FROM THE INITIAL** call of the procedure (i.e., @sid=1)
4. Extend the Hogwarts_Tree SP to use a temporary table instead of an already designed table

Solutions

```
CREATE PROCEDURE [dbo].[Hogwarts_Tree]
@sid int
AS
SET NOCOUNT ON

DECLARE @ID int
DECLARE @Name nvarchar(100)

IF @@NESTLEVEL=1
BEGIN
        SELECT @ID=ID, @Name=Name FROM Hogwarts WHERE
        ID=@sid
        PRINT CAST(@ID as nvarchar) + ' ' + @Name

        --ALREADY DESIGNED TABLE
        DELETE FROM Hogwarts2
        INSERT INTO Hogwarts2 VALUES (@ID, @Name)

        --TEMPORARY TABLE
        CREATE TABLE #Hogwarts3 (ID INT, Name nvarchar(100));
        INSERT INTO #Hogwarts3 VALUES (@ID, @Name)
END

DECLARE c CURSOR LOCAL FAST_FORWARD
FOR SELECT ID, Name FROM Hogwarts WHERE SID=@sid
OPEN c
FETCH NEXT FROM c INTO @ID, @Name
WHILE @@FETCH_STATUS=0
BEGIN
        PRINT SPACE(@@NESTLEVEL * 2) + CAST(@ID as
        nvarchar) + ' ' + @Name

        --ALREADY DESIGNED TABLE
        INSERT INTO Hogwarts2 VALUES (@ID, @Name)

        --TEMPORARY TABLE
        INSERT INTO #Hogwarts3 VALUES (@ID, @Name)

        EXEC  Hogwarts_Tree  @ID
        FETCH NEXT FROM c INTO @ID, @Name
END
CLOSE c
DEALLOCATE c

IF @@NESTLEVEL=1
BEGIN
        --ALREADY DESIGNED TABLE
        --SELECT * FROM Hogwarts2

        --TEMPORARY TABLE
        SELECT * FROM #Hogwarts3
        DROP TABLE #Hogwarts3
END
GO
EXEC Hogwarts_Tree 1
```

Other Information

Setting multiple parameters with one SELECT statement

```
DECLARE @id int
DECLARE @name nvarchar(100)

SET @id = (SELECT ID FROM Table WHERE ID=1)
SET @name = (SELECT Name FROM Table WHERE ID=1)
```

OR

```
SELECT @id=ID, @name=Name FROM Table WHERE ID=1
```