# Stored Procedures, Global Variables, Dynamic Execution, the OUTPUT clause, Cursors – in SQL Server

## Stored Procedures (SP)

* Contains more Transact-SQL statements

CREATE PROCEDURE <SPName>

AS

…..

….

GO

CREATE PROCEDURE <SPName>

AS

BEGIN

…..

….

END

GO

Execution of procedures:

EXEC <SPName>

Or:

<SPName>

Example:

CREATE PROCEDURE uspActorsNames

AS

SELECT name

FROM Actors A

GO

EXEC uspActorsNames

uspActorsNames

CREATE PROCEDURE uspActorsNames(@age INT = 18)

AS

SELECT name

FROM Actors A

WHERE age > @age

GO

EXEC uspActorsNames 20

EXEC uspActorsNames @age = 20

uspActorsNames 20

CREATE PROCEDURE uspActorsNames(@age INT, @NoActors INT OUTPUT)

AS

SELECT @NoActors = count(\*)

FROM Actors A

WHERE age > @age

GO

DECLARE @NoAct INT

SET @NoAct = 0

uspActorsNames @age = 20, @NoActors = @NoAct OUTPUT

uspActorsNames 20, @NoAct OUTPUT

PRINT @NoAct

CREATE PROCEDURE uspActorsNames(@age INT, @NoActors INT OUTPUT)

AS

SELECT @NoActors = count(\*)

FROM Actors A

WHERE age > @age

IF @NoActors = 0

RAISERROR(‘There are no actors of the specified age’, 10,1)

GO

* RAISERROR(msg\_str | localvariable, severity, state)
* Severity – 0..25, 20-25 fatal errors
* State – 0-255, default =1 (same error raised in different locations)
* BEGIN TRY
  + ….. code with the RAISERROR func
  + ….
* END TRY
* BEGIN CATCH
  + ERROR\_MESSAGE()
  + ERROR\_SEVERITY()
  + ERROR\_STATE()
* END CATCH

## Global variables

* Special type of variables, that are maintained by the db server
* No need to be declared
* Provide info about the server, the current user …
* Prefixed with @@
* Also called System functions
* Examples
  + @@ERROR – the error number for the last executed statement, 0-no error has occured
  + @@IDENTITY – the last identity inserted value
  + @@ROWCOUNT – number of rows affected by the last executed SQL statement
  + @@SERVERNAME – local server name on which SQL Server is running
  + @@SPID – session id for the current user
  + @@VERSION – system and build info
* Print @@ERROR

CREATE TABLE Name(

Id INT IDENTITY(1,1),

cname VARCHAR(10)

)

Insert Name (cname) VALUES(‘Name’)

1, ‘Name’

## Dynamic Execution

EXEC (<command>)

EXECUTE(<command>)

Examples:

EXEC(‘SELECT aid, name FROM Actors WHERE aid = 1’)

DECLARE @select VARCHAR(MAX)

SET @select = ‘SELECT aid, name FROM Actors WHERE aid = 1’

EXEC(@select)

* Potential security problems

SQL injection

* ‘SELECT \* From admins where password =”’ + ‘” OR 1=1 ----’
* ‘SELECT \* From admins where password =”’ + ‘”; DROP TABLE admins’

Alternative for EXEC – reducing the risk of sql injection

* **sp\_executesql** stored procedure
* EXEC sp\_executesql N’SELECT aid, name FROM Actors WHERE aid=@aid’, N’@aid INT, @aid =1;
* EXEC sp\_executesql N’SELECT aid, name FROM Actors WHERE aid=@aid’, N’@aid INT, @aid =’” OR 1=1’; -> error

## The OUTPUT clause

* Provide info about the deleted, updated or inserted values
* User has access to the deleted, inserted tables during INSERT, UPDATE or DELETE SQL statements

UPDATE Exams

SET grade = 10

OUTPUT inserted.eid, inserted.sid, inserted.grade, deleted.eid, deleted.sid, deleted.grade, GETDATE(), SUSER\_NAME()

INTO ExamLogs

WHERE ….

INSERT into TableName (col1, col2,…)

OUTPUT inserted.col1, inserted.col2

INTO @inserted\_log

VALUES (‘col1 value’, ‘col2 value’)

DELETE FROM TableName

OUTPUT deleted.col1, deleted.col2

INTO ….

WHERE col1 = 34

## Cursors

* When processing a result set (the SELECT result) row by row
* Not very efficient,
* It’s recommended to use set-based processing, whenever possible
* SQL statements used when working with cursors:
  + DECLARE CURSOR – declare the cursor and specifies the SELECT based on which result set the cursor is going to work
  + OPEN – populate with values the cursor, the select statement is going to be executed
  + FETCH – executed multiple times + fetch individual rows
    - Fetch a row from the cursor
    - Options:
      * FETCH FIRST – the cursor points to the first row of the result set + return the first row
      * FETCH NEXT – the next row after the current one
      * FETCH PRIOR – the previous row
      * FETCH LAST – the last row of the result set
      * FETCH ABSOLUTE n, n integer (e.g. FETCH ABSOLUTE 10)
        + n> 0 -> the cursor will point to the n-th row starting from the **first in the cursor**
        + n = 0 -> no rows
        + n<0 -> the n-th row before the last row of the cursor
      * FETCH RELATIVE n, n integer
        + n>0 -> the cursor will point to the n-th row starting from the **current row**
        + n < 0 - the n-th row before the current row of the cursor
        + n=0 -> the current row
  + (optional) UPDATE, DELETE
  + CLOSE – free the resources allocated for the cursor (e.g. result set)
  + DEALLOCATE – free all resources allocated with the cursor eg its name, …

E.g.

DECLARE @Aid INT, @AName VARCHAR(100)

DECLARE ActorsCursor CURSOR FOR

SELECT aid, aname

FROM Actors

OPEN ActorsCursor

FETCH ActorsCursor

INTO @Aid, @AName

WHILE @@FETCH\_STATUS = 0

BEGIN

--- sql code processing the current row, @Aid, @AName

FETCH ActorsCursor

INTO @Aid, @AName

END

CLOSE ActorsCursor

DEALLOCATE ActorsCursor

ISO syntax:

DECLARE CursorName [INSENSITIVE] [SCROLL] CURSOR

FOR select\_state

[FOR {READ-ONLY | UPDATE [OF col\_name [, … n]]} ]

Transact-SQL syntax:

DECLARE cursor\_name CURSOR [LOCAL | GLOBAL]

[FORWARD ONLY | SCROLL]

[STATIC | KEYSET | DYNAMIC | FAST\_FORWARD]

[READ\_ONLY | SCROLL\_LOCKS | OPTIMISTIC]

[TYPE\_WARNING]

FOR select\_state

[FOR UPDATE [OF col\_name [,…n]] ]

More details: <https://docs.microsoft.com/en-us/sql/t-sql/language-elements/declare-cursor-transact-sql?view=sql-server-ver15>