Databases
Seminar 3

Stored Procedures, Global Variables, Dynamic Execution,
the Output Clause, Cursors - in SQL Server

Stored procedures

- a stored procedure contains a group of Transact-SQL statements compiled into a single execution plan;

- syntax for a simple stored procedure:

```
CREATE PROCEDURE <SPName> AS
   --sequence of SQL statements
GO
```

- a stored procedure is executed with EXEC:

```
EXEC <SPName>
```

or by specifying only the name of the procedure:

```
<SPName>
```

- simple stored procedure example – creation & execution:

```
CREATE PROCEDURE uspStudentsNames
AS
      SELECT sname
      FROM Students
GO
EXEC uspStudentsNames
```

- a simple stored procedure with one parameter – creation & execution:

```
ALTER PROCEDURE uspStudentsNames(@age INT)
AS
      SELECT sname
      FROM Students
      WHERE age = @age
GO
EXEC uspStudentsNames 20
```

- a simple stored procedure with an output parameter – creation & execution, accessing the value of the output parameter:

Stored Procedures, Global Variables, Dynamic Execution,
the Output Clause, Cursors - in SQL Server

```
CREATE PROCEDURE uspNoOfStudents(@age INT, @NoStd INT OUTPUT)
AS
      SELECT @NoStd = COUNT(*)
      FROM Students
      WHERE age = @age
GO

DECLARE @NoStd INT
SET @NoStd = 0
EXEC uspNoOfStudents 20, @NoStd = @NoStd OUTPUT
PRINT @NoStd
```

- a stored procedure with the RAISERROR statement:

```
ALTER PROCEDURE uspNoOfStudents(@age INT, @NoStd INT OUTPUT)
AS
BEGIN
      SELECT @NoStd = COUNT(*)
      FROM Students
      WHERE age = @age

      IF @NoStd = 0
        RAISERROR ('There are no students of the specified age.',10,1)
END
GO
```

- the RAISERROR statement generates an error message and initiates error processing for the session;

- syntax:

```
RAISERROR ( { msg_id | msg_str | @local_var} { , severity, state } )
```

Global variables

- special type of variables; their values are maintained by the server;
- provide information about the server, the current user session;

Stored Procedures, Global Variables, Dynamic Execution,
the Output Clause, Cursors - in SQL Server

- name prefix: '@@';
- they needn't be declared, the server is constantly maintaining them (they are system functions);
- examples:
    - @@ERROR – the error number for the last executed T-SQL statement; 0 - no error occurred;
    - @@IDENTITY – the last inserted IDENTITY value;
    - @@ROWCOUNT – the number of rows affected by the last statement;
    - @@SERVERNAME – the name of the local server on which SQL Server is running;
    - @@SPID – the session id for the current user process;
    - @@VERSION – system and build information.

Dynamic execution

- syntax:

```
EXEC (<command>)
```

- examples:

```
EXEC('SELECT sid, cid FROM Exams WHERE sid = 1')
GO

DECLARE @var VARCHAR(MAX)
SET @var = 'SELECT sid, cid FROM Exams WHERE sid = 1'
EXEC(@var)
GO
```

- drawbacks – (potential) security problems;

- alternative for EXEC – the *sp_executesql* stored procedure; reduces the risk of SQL injection;

```
EXECUTE sp_executesql N'SELECT sid, cid FROM Exams WHERE sid = @sid',
N'@sid INT', @sid = 1;
```

The OUTPUT clause

- provides access to added / modified / deleted records;

```
UPDATE Courses
SET cname = 'Database Management Systems'
OUTPUT inserted.cid, deleted.cname, inserted.cname, GETDATE(),
SUSER_SNAME()
INTO CourseChanges
WHERE cid = 'DB2'
```

Cursors

- in some cases, result sets should be processed in a row-by-row fashion; this can be achieved by opening a cursor on a result set;
- used in Transact-SQL scripts, stored procedures, triggers;
- whenever possible, set-based processing should be used instead;

Databases
Seminar 3

Stored Procedures, Global Variables, Dynamic Execution, the Output Clause, Cursors - in SQL Server

- Transact-SQL statements to declare and populate a cursor, to retrieve data:
    - DECLARE CURSOR – declares the cursor; it specifies the SELECT statement that will produce the result set;
    - OPEN – populates the cursor; the SELECT statement specified in the DECLARE CURSOR statement is executed;
    - FETCH – fetches individual rows from the result set; usually, it's executed multiple times (at least once for every row in the result set);
    - if necessary, an UPDATE or DELETE statement can be used to change the row; this step is optional;
    - CLOSE – closes the cursor, frees resources, e.g., the result set; the cursor is still declared (the OPEN statement can be used to reopen it);
    - DEALLOCATE – frees all the resources allocated to the cursor, including its name; after deallocation, the DECLARE statement must be used to rebuild the cursor.

- ISO syntax:

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

- extended 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_statement
     [ FOR UPDATE [ OF column_name [ ,...n ] ] ]
[;]
```

- FETCH
    - fetches a certain row from a cursor;
    - once the cursor is positioned on a row, various operations can be performed on the row;
    - FETCH options to obtain certain rows:
        - FETCH FIRST – returns the first row from the cursor;
        - FETCH NEXT – the row immediately following the current row;
        - FETCH PRIOR – the row before the current row;
        - FETCH LAST – the last row in the cursor;
        - FETCH ABSOLUTE n, n integer:
            - n > 0: the n^{th} row starting with the first row in the cursor;
            - n < 0: the n^{th} row before the last row in the cursor;
            - n = 0: no rows;
        - FETCH RELATIVE n, n integer:
            - n > 0: the row that is *n* rows after the current row;
            - n < 0: the row that is *n* rows before the current row;

- n = 0: the current row.

- cursor example:

```
DECLARE @PID INT, @PName VARCHAR(50)
DECLARE PlayersCursor CURSOR FOR
SELECT PID, PName
FROM Players
OPEN PlayersCursor
FETCH PlayersCursor
INTO @PID, @PName
WHILE @@FETCH_STATUS = 0
BEGIN
    --code that processes @PID, @PName
    FETCH PlayersCursor
    INTO @PID, @PName
END
CLOSE PlayersCursor
DEALLOCATE PlayersCursor
```