Store Procedure, Function, Trigger, Cursor in MSSQL Server

Chapter 6

Contents

Contents

Store procedure overview

- Stored Procedure is a group of precompiled Transact-SQL statement into a single execution plan.
  - Accept input parameters and return multiple values in the form of output parameters.
  - Contain programming statements that perform operations in the database, including calling other procedures.
  - Return a status value to indicate success or failure (and the reason for failure).

Store procedure overview (cond.)

- Types of stored procedures:
  - User-defined procedures
  - Temporary procedures
  - System procedures

- Advantage:
  - Reduced server/client network traffic
  - Stronger security
  - Reuse of code
  - Easier maintenance
  - Improved performance

Store procedure syntax

- Create a store procedure

  CREATE [OR ALTER] { PROC | PROCEDURE }

  [*schema_name*.] *procedure_name*

  [{ *@parameter* **data_type** } [VARYING] [= *default*]

  [OUT | OUTPUT | [READONLY]] [,...n]

  AS

  {*sql_statement*}

- Execute a store procedure:

  EXEC | EXECUTE [*schema_name*.] *procedure_name*

  [[@parameter =] { value | *@variable* [OUTPUT]

Store procedure example

```
CREATE OR ALTER PROCEDURE Update_Sal
                @p_emp_id CHAR (9), @p_factor NUMERIC(3,2)
AS
        DECLARE @v_count INT;
        SELECT @v_count = COUNT(*)
        FROM EMPLOYEE
        WHERE SSN = @p_emp_id;

        IF @v_count = 1
                UPDATE EMPLOYEE
                SET Salary = Salary * @p_factor
                WHERE SSN = @p_emp_id;
```

```
EXEC Update_Sal '123456789', 1.2;
```

Contents

Function overview

- SQL Server user-defined functions are routines:
  - accept parameters,
  - perform an action, such as a complex calculation,
  - and return the result of that action as a value.
- Types of functions:
  - *Scalar Function:*
    - Return a single data value of the type defined in the RETURNS clause.
    - The return type can be any data type except **text**, **ntext**, **image**, **cursor**, and **timestamp**.
  - *Table-Valued Functions:* return a **table** data type.
  - *System Functions*

Scalar function syntax

```
CREATE [ OR ALTER ] FUNCTION [ schema_name. ]
function_name
( [ { @parameter_name [ AS ]  data_type
 [ = default ] [ READONLY ] } [ ,...n ] ] )
RETURNS return_data_type
[ AS ]
   BEGIN
      function_body
      RETURN scalar_expression
   END;
```

Table-valued function syntax

CREATE [OR ALTER] FUNCTION [*schema_name.*]
function_name
([{ *@parameter_name* [**AS**] *data_type*
[*= default*] [**READONLY**] } [,...n]])
RETURNS *@return_variable* **TABLE** <*table_type_definition*>
[**AS**]
BEGIN
{function_body}
RETURN
END;

Scalar function example

- Create a scalar function:

```
CREATE OR ALTER FUNCTION Get_Sal (@p_id CHAR(9))
RETURNS DECIMAL(10,2)
AS
BEGIN
        DECLARE @v_sal DECIMAL(10,2);
        SET @v_sal = (SELECT salary
                      FROM EMPLOYEE
                      WHERE SSN = @p_id);
        RETURN @v_sal;
END;
```

- Execute:

```
SELECT dbo.Get_Sal ('333445555');
```

Table-valued function example

- Create table-valued function:

```
CREATE FUNCTION EmpAndDependent()
   RETURNS @person TABLE (
      first_name VARCHAR(15),
      last_name VARCHAR(15),
      sex CHAR,
      type VARCHAR(10)  )
AS
BEGIN
   INSERT INTO @person
   SELECT Fname, LName, Sex, 'Employee'
   FROM EMPLOYEE;
   INSERT INTO @person
   SELECT D.Dependent_name, E.LName,
            D.Sex, 'Dependent'
   FROM EMPLOYEE E, DENPENDENT D
   WHERE E.SSN = D.ESSN

   RETURN;
END;
```

- Execute:

```
SELECT * FROM    EmpAndDependent ();
```

Contents

Trigger Overview

- SQL Server triggers are special stored procedures that are **executed automatically** (by the DBMS) when an event occurs in the database server.

- SQL Server provides three type of triggers:
  - **Data manipulation language (DML) triggers:** invoked automatically in response to *INSERT, UPDATE, and DELETE* statements on a table or view.
  - **Data definition language (DDL) triggers:** fire in response to *CREATE, ALTER, and DROP* statements, and certain system stored procedures that perform DDL-like operations.
  - **Logon triggers:** fire in response to *LOGON* events.

Uses of Trigger

- Automatically generate derived column values.
- Maintain complex integrity constraints.
- Enforce complex business rules.
- Record auditing information about database changes.

Simple DML Trigger Syntax

CREATE [OR ALTER] TRIGGER [*schema*.] *trigger_name*
ON { *table_name* | *view_name* }
{ **FOR | AFTER | INSTEAD OF** }
{ [**INSERT**] [,] [**UPDATE**] [,] [**DELETE**] }
AS
{*sql_statements*}

Trigger Firing Order

1. INSTEAD OF trigger.
2. Constraints exist on the trigger table.
3. AFTER trigger runs.

- If the constraints are violated, the INSTEAD OF trigger actions are rolled back and the AFTER trigger isn't fired.

“Virtual” tables for triggers

- Two “virtual” tables that are available specifically for triggers called INSERTED and DELETED tables.
  - SQL Server uses these tables to capture the data of the modified row before and after the event occurs.

DML event	INSERTED table	DELETED table
INSERT	rows to be inserted	empty
UPDATE	new rows modified by the update	existing rows modified by the update
DELETE	empty	rows to be deleted

Trigger Example

```
CREATE  OR ALTER  TRIGGER Check_Dnumber
ON Department
FOR  INSERT, UPDATE
AS
BEGIN
        DECLARE @dnum INT;
        SELECT @dnum = DNumber from INSERTED;
        IF (@dnum > 20 OR @dnum < 0)
        BEGIN
                RAISERROR ('Invalid Dnumber !', 16, 1);
                ROLLBACK;
        END;
END;
```

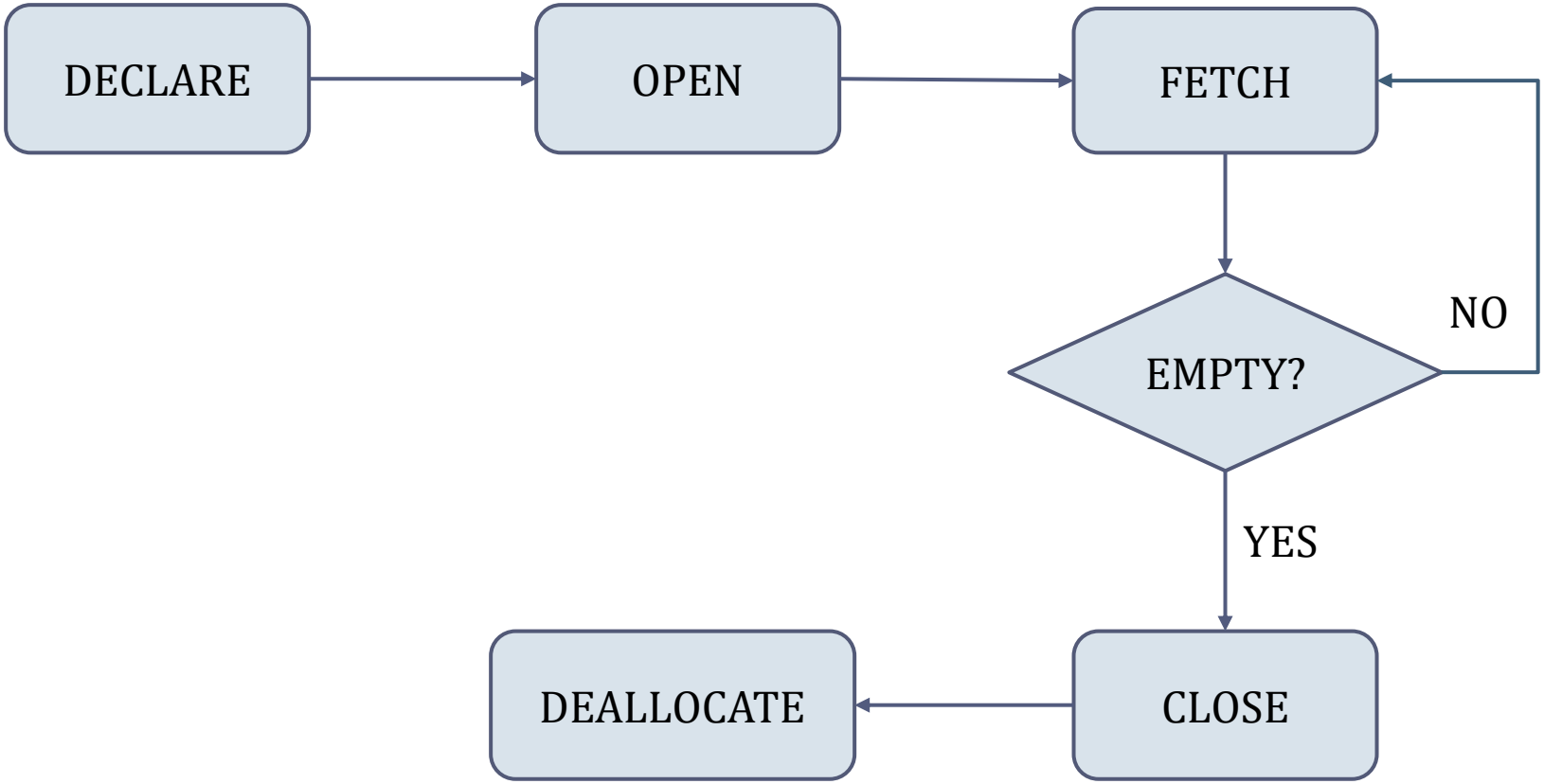
Contents

Cursor overview

- A SQL cursor is a database object that is used to retrieve data from a result set one row at a time.
- Why use a SQL Cursor?
  - In relational databases, operations are made on a set of rows. For example, a SELECT statement returns a set of rows which is called a result set.
  - Sometimes we may want to process a data set on a **row by row** basis rather than the entire result set at once.

→ Using cursors.

Cursor life cycle



Cursor syntax

- Declare a cursor:

DECLARE *cursor_name* **CURSOR**

FOR {*SELECT statements*}

[**FOR { READ ONLY | UPDATE [OF** *column_name* [,...n]] }]

- Open a cursor:

  OPEN *Cursor_name*

- Close a cursor:

  CLOSE *Cursor_name*

- Deallocate a cursor: Removes a cursor reference

  DEALLOCATE *Cursor_name*

Cursor syntax (cond.)

Statement	Description
FETCH	**FETCH** [NEXT| PRIOR | FIRST |LAST] **FROM** *Cursor_name* [**INTO** *Var_list*] • FETCH NEXT: Returns the result row immediately following the current row. • FETCH PRIOR: Returns the result row immediately preceding the current row. • FETCH FIRST: Returns the first row in the cursor. • FETCH LAST: Returns the last row in the cursor.
@@CURSOR_ROWS	Returns the number of rows currently in the opened cursor.
@@FETCH_STATUS	Returns the status of the last cursor FETCH statement
CURSOR_STATUS	Shows whether or not a cursor declaration has returned a cursor and result set.

Cursor example

```
CREATE PROCEDURE PrintEmployee_Cursor
AS BEGIN
  --declare the variables
  DECLARE @v_empID INT,
          @v_name VARCHAR(100)
 --declare and set counter.
  DECLARE @v_counter INT
  SET @v_counter = 1
 --declare the cursor for a query.
  DECLARE EmployeeCursor CURSOR
  FOR  SELECT SSN, FName + ' ' + LName
       FROM Employee
  --open cursor.
  OPEN EmployeeCursor
  --fetch the record
  FETCH NEXT FROM EmployeeCursor
  INTO @v_empID, @v_name
  --loop until records are available.
  WHILE @@FETCH_STATUS = 0
  BEGIN
    IF @v_counter = 1
     PRINT 'EmployeeSSN' + CHAR(9) + 'Name'
    --print current record.
    PRINT CAST (@v_empID AS VARCHAR(9))
    + CHAR(9) + @v_name
    --increment counter.
     SET @v_counter = @v_counter + 1
    --fetch the next record
    FETCH NEXT FROM EmployeeCursor
    INTO @v_empID, @v_name
  END
--close the cursor.
CLOSE EmployeeCursor
DEALLOCATE EmployeeCursor
END;
```


Q
A

Homework

- Error Handling in SQL Server:
  - Raiserror
  - Throw
  - Try - Catch

Exercise

1. Write a trigger for ensuring that the employee's ages must be between18 and 60.
2. Write a trigger to enforce that when an employee has a new project, his or her salary will be increased by 10% * number of hours per week working on that project.
3. Write a store procedure to read an employee's id and print the names of his/her dependents.
4. Write a function to read a project's id and return the total number of employees who work for that project.