

Stored Procedure & Trigger

NGUYEN Hong Phuong

Email: phuongnh@soict.hust.edu.vn

Site: <https://users.soict.hust.edu.vn/phuongnh>

Contents

1. Stored Procedure

1.1. Introduction

1.2. Syntax

2. Trigger

2.1. Introduction

2.2. Syntax

1. Stored Procedure

1.1. Introduction

1.2. Syntax

1.1. Introduction to stored procedure

□ Concepts:

- A stored procedure (SP) is a set of Structured Query Language (SQL) statements with an assigned name, which are stored in a relational database management system as a group, so it can be reused and shared by multiple programs.
- Stored procedures can access or modify [data](#) in a [database](#), but it is not tied to a specific database or object, which offers a number of advantages.

1.1. Introduction to stored procedure

□ Benefits:

- provides an important layer of security between the user interface and the database.
- preserves data integrity because information is entered in a consistent manner.
- improves productivity because statements in a stored procedure only must be written once.
- offer advantages over embedding queries in a graphical user interface (GUI).

1.1. Introduction to stored procedure

- Since stored procedures are modular, it is easier to troubleshoot when a problem arises in an application.
- Stored procedures are also tunable, which eliminates the need to modify the GUI [source code](#) to improve its performance. It's easier to code stored procedures than to build a query through a GUI.
- can reduce network traffic between clients and [servers](#), because the commands are executed as a single batch of code. This means only the call to execute the procedure is sent over a [network](#), instead of every single line of code being sent individually.

1.1. Introduction to stored procedure

- Stored procedures in [SQL Server](#) can accept input parameters and return multiple values of output parameters;

1.1. Introduction to stored procedure

□ Stored procedure vs. function

- Stored procedures and functions can be used to accomplish the same task. Both can be custom-defined as part of any application, but functions are designed to send their output to a query or T-SQL statement. Stored procedures are designed to return outputs to the application, while a user-defined function returns table variables and cannot change the server environment or operating system environment.

1.1. Introduction to stored procedure

- There are 3 types:
 - System SP: provided by SQL Server, whose name starts with prefix "sp_", is used to manage SQL Server and display database and user-information.
 - SP extensions: dynamic link libraries (DLLs), written in languages like C, C ++, ..., that SQL Server can load and execute.
 - External SP: name starts with "xp_"
 - User-defined SP

1.2. Syntax

- ❑ You can use T-SQL, Enterprise Manager or wizard to create SP.
- ❑ Syntax in SQL Server:

```
CREATE PROC[EDURE] procedure_name  
{;number}  
[{@parameter data_type}[=default |  
NULL][VARYING][OUT PUT]]  
[WITH {RECOMPILE | ENCRYPTION |  
RECOMPILE, ENCRYPTION}]  
[FOR REPLICATION]  
AS sql_statement
```

Example

```
USE CompanySupplyProduct
GO
IF EXISTS(SELECT name FROM sysobjects
WHERE name='pCompany' AND type='P')
DROP PROCEDURE pCompany
GO
CREATE PROCEDURE pCompany
AS SELECT Name, NumberofEmployee
FROM Company
ORDER BY Name DESC
GO
```

- Run this procedure:
 - EXEC pCompany

-
- To see the content of a SP:
 - EXEC sp_helptext pCompany
 - Drop a SP:
 - DROP PROCEDURE procedure_name

Creating a group of SP

```
CREATE PROC group_sp;1
AS SELECT * FROM Company
GO
CREATE PROC group_sp;2
AS SELECT Name FROM Company
GO
CREATE PROC group_sp;3
AS SELECT Name, Address FROM Company
GO
```

- To get a list of name and address of the companies, use the following command:
 - EXEC group_sp;3

Parameters

- ❑ `@parameter data_type [=default | NULL]`
`[VARYING] [OUTPUT]`
- ❑ `@parameter`: name of parameter inside the procedure, can declare up to 1024 parameters inside a SP.
- ❑ `data_type`: any data type defined by the system or user-defined, except the image data type.
- ❑ `Default`: Specifies the default value for the parameter.
- ❑ `VARYING`: Applies to the returned recordset.
- ❑ `OUTPUT`: Defines this as a return parameter.

An example

- Write a stored procedure that takes 5 parameters as input, calculates the average, and outputs it:

```
CREATE PROCEDURE scores
@score1 smallint,
@score2 smallint,
@score3 smallint,
@score4 smallint,
@score5 smallint,
@myAvg smallint OUTPUT
AS SELECT @myAvg = (@score1 + @score2 +
@score3 + @score4 + @score5) / 5
```

Pass/receive values for/from parameters

❑ Transmitting in the order

```
DECLARE @AvgScore smallint
EXEC scores 10, 9, 8, 8, 10, @AvgScore OUTPUT
SELECT 'The Average Score is: ', @AvgScore
Go
```

❑ Transmitting in any order

```
DECLARE @AvgScore smallint
EXEC scores
@score1=10, @score3=9, @score2=8, @score4=8,
@score5=10, @myAvg = @AvgScore OUTPUT
SELECT 'The Average Score is: ', @AvgScore
Go
```


Pass/receive values for/from parameters

□ RETURN

```
CREATE PROC MyReturn  
@t1 smallint, @t2 smallint, @retval smallint  
AS SELECT @retval = @t1 + @t2  
RETURN @retval
```

□ Run:

```
DECLARE @myReturnValue smallint  
EXEC @myReturnValue = MyReturn 9, 9, 0  
SELECT 'The return value is: ', @myReturnValue
```

□ WITH RECOMPILE option:

- in the CREATE PROCEDURE statement: The whole procedure is recompiled every time it runs, the procedure can be optimized for new parameters.
 - in the EXEC PROCEDURE statement: Compile the stored procedure for that execution and store the new plan in the procedure buffer for later EXEC PROCEDURE commands.
- If a SP is created with the ENCRYPTION option => its contents cannot be viewed

An example

```
USE CompanySupplyProduct
GO
IF EXISTS(SELECT name FROM sysobjects
WHERE name='pCompany' AND type='P')
DROP PROCEDURE pCompany
GO
CREATE PROCEDURE pCompany WITH ENCRYPTION
AS SELECT Name, NumberofEmployee
FROM Company
ORDER BY Name DESC
GO

EXEC sp_helptext pCompany;
```

2. Trigger

2.1. Introduction

2.2. Syntax

2.1 Introduction

- A special stored procedure, which is executed automatically when there are data-changing events such as Update, Insert or Delete.
- Used to ensure data integrity or to implement certain business rules.
- When to use triggers?
 - when other data integrity measures like Constraint cannot satisfy the application's requirements

2.1 Introduction

- Constraint is a declared data integrity type: check the data before allowing it to be entered into the table
- The trigger is of procedural data integrity, so the Insert, Update, Delete happens and then activates the trigger.
- Sometimes, due to the need to change chains, triggers can be used
- Characteristics of trigger
 - A trigger can do multiple jobs, which can be triggered by multiple events

2.1 Introduction

- Triggers cannot be created on temporary or system tables
- Triggers can only be triggered automatically by events and cannot be manually run.
- A trigger can be applied to a view
- When trigger is activated
 - The newly inserted data will be contained in the "inserted" table.
 - Newly deleted data will be stored in the "deleted" table.
 - These are two temporary tables that reside in memory, and only have values inside the trigger

2.2. Syntax

- ❑ You can use T-SQL or Enterprise Manager to create triggers
- ❑ The following statements must not be used in trigger definitions: ALTER DATABASE, CREATE DATABASE, DISK INIT, DISK RESIZE, DROP DATABASE, LOAD DATABASE, LOAD LOG, RECONFIGURE, RESTORE DATABASE, RESTORE LOG

-
- Temporary tables: deleted and inserted
 - referred to as the real table but stored in internal memory, not on disk.
 - Values in this table are only accessible in triggers. Once the trigger is completed, the tables are no longer accessible.

Example

- ❑ Create the AddCompany trigger on the Company table: print a message whenever data is added to the table

```
USE CompanySupplyProduct
GO
IF EXISTS(SELECT name FROM sysobjects
WHERE name='AddCompany' AND Type='TR')
DROP TRIGGER AddCompany
GO
CREATE TRIGGER AddCompany
ON Company
FOR INSERT
AS
PRINT 'The Company table has just been inserted data'
GO
```

Create deleted trigger

- ❑ Create the table DeletedCompany to store the deleted item from the Company table
- ❑ This table should be the same to Company

```
CREATE TABLE [DeletedCompany] (  
    [CompanyID] int,  
    [Name] varchar(40),  
    [NumberofEmployee] int,  
    [Address] varchar(50),  
    [Telephone] char(15),  
    [EstablishmentDay] date,  
    PRIMARY KEY ([CompanyID])  
);
```

Create deleted trigger

- Create deleted trigger on the Company table for the delete event

```
CREATE TRIGGER tg_DeleteCompany
ON Company
FOR DELETE
AS
INSERT INTO DeletedCompany SELECT * FROM deleted
```

Create update trigger

```
CREATE TRIGGER tg_CheckPrice
ON Product
FOR UPDATE
AS
DECLARE @oldprice decimal(10,2), @newprice decimal(10,2)
SELECT @oldprice = Price FROM deleted
PRINT 'Old price ='
PRINT CONVERT(varchar(6), @oldprice)
SELECT @newprice = Price FROM inserted
PRINT 'New price ='
PRINT CONVERT(varchar(6), @newprice)
IF(@newprice > (@oldprice*1.10))
BEGIN
PRINT 'New price increased over 10%, not update'
ROLLBACK
END
ELSE
PRINT 'New price is accepted'
```

