Stored Procedures

A note by
Shah Md. Arshad Rahman Ziban

Stored Procedures

Definition

1. Subroutine
2. Precompiled
3. Save and reused over again and again
4. Avoid sql injection attack

Types

Stored Procedure is mainly two types:

System Stored Procedures: These are built-in procedures provided by the database system. An example is sp_helptext, which is used to view the definition of a stored procedure.

User-Defined Stored Procedures: These are custom procedures created by users to perform specific tasks. Users can define the logic and parameters according to their needs.

It is prohibited to use sp_ as prefix in user defined stored procedure

Syntax

Creating a Stored Procedure

```
CREATE PROC ProcedureName
AS
BEGIN
    -- Body of the procedure
    -- Your SQL code goes here
END;
```

OR

```
CREATE PROCEDURE ProcedureName
AS
BEGIN
    -- Body of the procedure
    -- Your SQL code goes here
END;
```

Executing the Stored Procedure

```
CREATE PROC ProcedureName;
```

OR

```
ProcedureName;
```

Viewing the Source Code of a Stored Procedure

```
sp_helptext 'ProcedureName';
```

Now we have table called tblEmp as follow

id	Names	Designation	Salary
1	Robert	CEO	25000
2	Michle	Manager	33000
4	Jhon	PO	40000
5	Sara	Manager	15000

Example (Without Parameters)

Suppose we want to write a stored procedure that will return name and salary from tblEmp table.

Creating the Stored Procedure:

```
CREATE PROC GetEmployee
AS
BEGIN
   SELECT Names, Salary FROM tblEmp;
END;
```

Executing the Stored Procedure:

```
EXEC GetEmployee
```

Example (With Parameters)

Now we want to create a stored procedure that will have two parameter Names and Designation.

```
CREATE PROCEDURE Proc_Emp
   @Name VARCHAR(10),
   @Designation VARCHAR(10)
AS
BEGIN
   SELECT Name, Designation
   FROM tblEmp
   WHERE Name = @Name AND Designation = @Designation;
END
```

Now to execute this procedure we need to write as follow

```
EXEC Proc_Emp 'Sara', 'Manager';
```

Modifying an Existing Stored Procedure

To change any query inside an existing procedure, you can use the ALTER keyword. This avoids having to drop and recreate the procedure from scratch.

```
ALTER PROCEDURE ProcedureName
-- add your modified code here
```

Example:

Here's the initial procedure:

```
CREATE PROCEDURE Proc_Emp
    @Name VARCHAR(10),
    @Designation VARCHAR(10)
AS
BEGIN
    SELECT Name, Designation
    FROM tblEmp
    WHERE Name = @Name AND Designation = @Designation;
END
```

Modified Version Using ALTER PROCEDURE

```
ALTER PROCEDURE Proc_Emp
    @Name VARCHAR(10),
    @Designation VARCHAR(10),
    @Salary DECIMAL(10, 2)
AS
BEGIN
    SELECT Name, Designation
    FROM tblEmp
    WHERE Name = @Name AND Designation = @Designation AND Salary = @Salary;
END
```

Dropping a Stored Procedure

If you want to delete a stored procedure, you can use the DROP PROCEDURE command. The syntax is:

```
DROP PROCEDURE ProcedureName;
```

Explanation

1. This command removes Proc_Emp from the database.
2. After executing this command, Proc_Emp will no longer exist, and any attempts to call it will result in an error until it is recreated.
3. Using DROP PROCEDURE is useful for permanently removing procedures that are no longer needed.

Advantages of encrypting stored procedure

1. **Keeps Sensitive Info Safe**: Encryption hides the stored procedure code, so personal details or important business logic stay private and can't be seen by just anyone.

2. **Protects Your Unique Ideas**: If your procedure includes special logic or calculations, encryption makes sure others can't easily copy or change it. It's like a lock for your proprietary work.

3. **Prevents Unauthorized Changes**: Once encrypted, the code can't be easily accessed or edited, so it stays secure and less prone to accidental or unwanted modifications.

4. **Limits Who Can Peek Inside**: With encryption, even users who can run the procedure can't open it in design view to see the code. They can only use it, not inspect it, which adds an extra layer of protection.

How to Encrypt a stored procedure

To encrypt a stored procedure in SQL Server, you use the WITH ENCRYPTION clause before the AS keyword. This hides the procedure's code from view, even if someone tries to look at it with tools like sp_helptext.

Here's how it works:

Syntax for Encrypting a Stored Procedure

```
CREATE PROC ProcedureName
WITH ENCRYPTION
AS
BEGIN
    -- Body of the procedure
    -- Your SQL code goes here
END;
```

Example

Suppose we want to create a procedure called Proc_Emp that retrieves employee information. Here's how to create it with encryption:

```
CREATE PROCEDURE Proc_Emp
    @Name VARCHAR(10),
    @Designation VARCHAR(10)
WITH ENCRYPTION
AS
BEGIN
    SELECT Name, Designation
    FROM tblEmp
    WHERE Name = @Name AND Designation = @Designation;
END
```

Running and Viewing the Encrypted Procedure

```
EXEC Proc_Emp 'Sara', 'Manager';
```

Stored Procedure with output parameter

Now create a stored procedure using output or out keywordwhichcountthenumber of rows where Designation is Manager:

```
CREATE PROCEDURE getRowByDesignation
    @Designation VARCHAR(15),
    @Result INT OUTPUT
AS
BEGIN
    -- Initialize the output variable
    SET @Result = 0;

    -- Count the rows with the specified designation
    SELECT @Result = COUNT(*)
    FROM tblEmp
    WHERE Designation = @Designation;
END
```

Key Points:

1. **Initialization:** SET @Result = 0; ensures that @Result has a default value in case no matching records are found.
2. **Count Assignment:** The SELECT statement directly assigns the count to @Result.
3. **Output Parameter**: The @Result parameter will return the count of employees with the specified designation.

How to execute a stored Procedure using output keyword

```
DECLARE @CountTotal INT;  -- Declare the variable to hold the count

-- Execute the stored procedure with 'Manager' as the designation
EXEC getRowByDesignation 'Manager', @CountTotal OUTPUT;

-- Print the total count
PRINT @CountTotal;
```

Key Points:

1. **Variable Declaration:** DECLARE @CountTotal INT; initializes the variable that will hold the output from the stored procedure.
2. **EXEC Statement:** The EXEC command calls your stored procedure, passing in the designation 'Manager' and the output variable @CountTotal.
3. **PRINT Statement:** PRINT @CountTotal; displays the count of employees with the designation "Manager" in the output.

What will happen if you don’t use output keyword ?

1. **No Value Update:** If you call a stored procedure without using the OUTPUT keyword for an output parameter, the value of that parameter won’t be updated outside the procedure.

2. **Original Value Stays:** The variable you passed to the procedure will keep its original value. It won’t change based on what happens inside the procedure.

Imagine you have this stored procedure:

```
CREATE PROCEDURE getRowByDesignation
    @Designation VARCHAR(15),
    @Result INT OUTPUT
AS
BEGIN
    -- Initialize the output variable
    SET @Result = 0;

    -- Count the rows with the specified designation
    SELECT @Result = COUNT(*)
    FROM tblEmp
    WHERE Designation = @Designation;
END
```

And you call it like this without OUTPUT:

```
DECLARE @CountTotal INT;

-- This will NOT update @CountTotal
EXEC getRowByDesignation 'Manager', @CountTotal ~~OUTPUT~~;

-- This will show the original value (NULL or whatever it was before)
PRINT @CountTotal;
```

Summary:

1. **With OUTPUT:** The variable will be updated with the count from the procedure.
2. **Without OUTPUT:** The variable will remain unchanged.

Some Examples of System Stored Procedure

sp_help: Provides detailed information about a stored procedure or table, including columns, data types, constraints, and more.

```
EXEC sp_help 'procedureName';
```

Example

```
-- This will provide information about the tblEmp table
EXEC sp_help 'tblEmp';
```

sp_depends: Shows dependencies for a stored procedure or table, helping you understand which objects are linked.

```
EXEC sp_depends 'procedureName';
```

Example

```
-- This will show if any stored procedures depend on tblEmp
EXEC sp_depends 'tblEmp';
```