**Stored procedures**

Create table tblDepartment

(

     ID int primary key,

     DepartmentName nvarchar(50),

     Location nvarchar(50),

     DepartmentHead nvarchar(50)

)

Go

Insert into tblDepartment values (1, 'IT', 'London', 'Rick')

Insert into tblDepartment values (2, 'Payroll', 'Delhi', 'Ron')

Insert into tblDepartment values (3, 'HR', 'New York', 'Christie')

Insert into tblDepartment values (4, 'Other Department', 'Sydney', 'Cindrella')

Go

Create table tblEmployee

(

     ID int primary key,

     Name nvarchar(50),

     Gender nvarchar(50),

     Salary int,

     DepartmentId int foreign key references tblDepartment(Id)

)

Go

Insert into tblEmployee values (1, 'Tom', 'Male', 4000, 1)

Insert into tblEmployee values (2, 'Pam', 'Female', 3000, 3)

Insert into tblEmployee values (3, 'John', 'Male', 3500, 1)

Insert into tblEmployee values (4, 'Sam', 'Male', 4500, 2)

Insert into tblEmployee values (5, 'Todd', 'Male', 2800, 2)

Insert into tblEmployee values (6, 'Ben', 'Male', 7000, 1)

Insert into tblEmployee values (7, 'Sara', 'Female', 4800, 3)

Insert into tblEmployee values (8, 'Valarie', 'Female', 5500, 1)

Insert into tblEmployee values (9, 'James', 'Male', 6500, NULL)

Insert into tblEmployee values (10, 'Russell', 'Male', 8800, NULL)

Go

A stored procedure is group of T-SQL (Transact SQL) statements. If you have a situation, where you write the same query over and over again, you can save that specific query as a stored procedure and call it just by it's name.

**Creating a simple stored procedure without any parameters**: This stored procedure, retrieves Name and Gender of all the employees. To create a stored procedure we use, **CREATE PROCEDURE** or **CREATE PROC** statement.

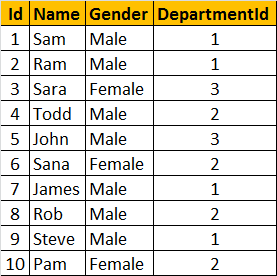
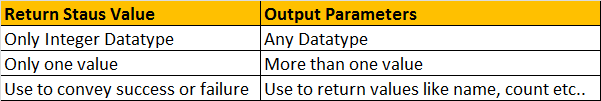
Create Procedure spGetEmployees  
as  
Begin  
  Select Name, Gender from tblEmployee  
End  
  
**Note:** When naming user defined stored procedures, Microsoft recommends not to use **"sp\_"** as a prefix. All system stored procedures, are prefixed with **"sp\_"**. This avoids any ambiguity between user defined and system stored procedures and any conflicts, with some future system procedure.  
  
**To execute the stored procedure**, you can just type the procedure name and press F5, or use EXEC or EXECUTE keywords followed by the procedure name as shown below.  
1. spGetEmployees  
2. EXEC spGetEmployees  
3. Execute spGetEmployees  
  
**Note:** You can also right click on the procedure name, in object explorer in SQL Server Management Studio and select EXECUTE STORED PROCEDURE.  
  
**Creating a stored procedure with input parameters:** This SP, accepts GENDER and DEPARTMENTID parameters. Parameters and variables have an @ prefix in their name.  
  
Create Procedure spGetEmployeesByGenderAndDepartment   
@Gender nvarchar(50),  
@DepartmentId int  
as  
Begin  
  Select Name, Gender from tblEmployee Where Gender = @Gender and DepartmentId = @DepartmentId  
End  
  
To invoke this procedure, we need to pass the value for @Gender and @DepartmentId parameters. If you don't specify the name of the parameters, you have to first pass value for @Gender parameter and then for @DepartmentId.  
EXECUTE spGetEmployeesByGenderAndDepartment 'Male', 1  
  
On the other hand, if you change the order, you will get an error stating "Error converting data type varchar to int." This is because, the value of **"Male"** is passed into @DepartmentId parameter. Since @DepartmentId is an integer, we get the type conversion error.  
**spGetEmployeesByGenderAndDepartment 1, 'Male'**  
  
When you specify the names of the parameters when executing the stored procedure the order doesn't matter.  
EXECUTE spGetEmployeesByGenderAndDepartment @DepartmentId=1, @Gender = 'Male'  
  
**To view the text, of the stored procedure**  
1. Use system stored procedure sp\_helptext 'SPName'  
OR  
2. Right Click the SP in Object explorer -> Scrip Procedure as -> Create To -> New Query Editor Window  
  
**To change the stored procedure, use ALTER PROCEDURE statement:**  
Alter Procedure spGetEmployeesByGenderAndDepartment   
@Gender nvarchar(50),  
@DepartmentId int  
as  
Begin  
  Select Name, Gender from tblEmployee Where Gender = @Gender and DepartmentId = @DepartmentId order by Name  
End  
  
**To encrypt the text of the SP**, use WITH ENCRYPTION option. Once, encrypted, you cannot view the text of the procedure, using sp\_helptext system stored procedure. There are ways to obtain the original text, which we will talk about in a later session.  
Alter Procedure spGetEmployeesByGenderAndDepartment   
@Gender nvarchar(50),  
@DepartmentId int  
WITH ENCRYPTION  
as  
Begin  
  Select Name, Gender from tblEmployee Where Gender = @Gender and DepartmentId = @DepartmentId  
End  
  
To delete the SP, use DROP PROC 'SPName' or DROP PROCEDURE 'SPName'

### Stored procedures with output parameters

**To create an SP with output parameter**, we use the keywords OUT or OUTPUT. @EmployeeCount is an OUTPUT parameter. Notice, it is specified with OUTPUT keyword.   
Create Procedure spGetEmployeeCountByGender  
@Gender nvarchar(20),  
@EmployeeCount int Output  
as  
Begin  
 Select @EmployeeCount = COUNT(Id)   
 from tblEmployee   
 where Gender = @Gender  
End

**To execute this stored procedure with OUTPUT parameter**  
  
**1.** First initialise a variable of the **same datatype** as that of the **output parameter**. We have declared @EmployeeTotal integer variable.   
**2.** Then pass the @EmployeeTotal variable to the SP. You have to specify the **OUTPUT** keyword. If you don't specify the OUTPUT keyword, the variable will be **NULL**.   
**3.** Execute  
  
Declare @EmployeeTotal int  
Execute spGetEmployeeCountByGender 'Female', @EmployeeTotal output  
Print @EmployeeTotal  
  
If you don't specify the OUTPUT keyword, when executing the stored procedure, the @EmployeeTotal variable will be NULL. Here, we have not specified OUTPUT keyword. When you execute, you will see **'@EmployeeTotal is null'** printed.  
  
Declare @EmployeeTotal int  
Execute spGetEmployeeCountByGender 'Female', @EmployeeTotal  
if(@EmployeeTotal is null)  
 Print '@EmployeeTotal is null'  
else  
 Print '@EmployeeTotal is not null'  
  
**You can pass parameters in any order, when you use the parameter names.** Here, we are first passing the OUTPUT parameter and then the input @Gender parameter.  
  
Declare @EmployeeTotal int  
Execute spGetEmployeeCountByGender @EmployeeCount = @EmployeeTotal OUT, @Gender = 'Male'  
Print @EmployeeTotal  
  
**The following system stored procedures, are extremely useful when working procedures.**  
**sp\_help** SP\_Name : View the information about the stored procedure, like parameter names, their datatypes etc. sp\_help can be used with any database object, like tables, views, SP's, triggers etc. Alternatively, you can also press ALT+F1, when the name of the object is highlighted.  
  
**sp\_helptext** SP\_Name : View the Text of the stored procedure  
  
**sp\_depends** SP\_Name : View the dependencies of the stored procedure. This system SP is very useful, especially if you want to check, if there are any stored procedures that are referencing a table that you are abput to drop. sp\_depends can also be used with other database objects like table etc.  
  
**Note:** All parameter and variable names in SQL server, need to have the @symbol.

### Stored procedure output parameters or return values

**What are stored procedure status variables?**  
Whenever, you execute a stored procedure, it returns an integer status variable. Usually, zero indicates success, and non-zero indicates failure. To see this yourself, execute any stored procedure from the object explorer, in sql server management studio.   
**1.** Right Click and select 'Execute Stored Procedure  
**2.** If the procedure, expects parameters, provide the values and click OK.  
**3.** Along with the result that you expect, the stored procedure, also returns a Return Value = 0  
  
So, from this we understood that, when a stored procedure is executed, it returns an integer status variable. With this in mind, let's understand the difference between output parameters and RETURN values. We will use the Employees table below for this purpose.  
  
  
  
  
**The following procedure returns total number of employees in the Employees table, using output parameter - @TotalCount.**  
Create Procedure spGetTotalCountOfEmployees1  
@TotalCount int output  
as  
Begin  
 Select @TotalCount = COUNT(ID) from tblEmployee  
End  
  
**Executing spGetTotalCountOfEmployees1 returns 3.**  
Declare @TotalEmployees int  
Execute spGetTotalCountOfEmployees @TotalEmployees Output  
Select @TotalEmployees  
  
**Re-written stored procedure using return variables**  
Create Procedure spGetTotalCountOfEmployees2  
as  
Begin  
 return (Select COUNT(ID) from Employees)  
End  
  
**Executing spGetTotalCountOfEmployees2 returns 3.**  
Declare @TotalEmployees int  
Execute @TotalEmployees = spGetTotalCountOfEmployees2  
Select @TotalEmployees  
  
So, we are able to achieve what we want, using output parameters as well as return values. Now, let's look at example, where return status variables cannot be used, but Output parameters can be used.  
  
**In this SP, we are retrieving the Name of the employee, based on their Id, using the output parameter @Name.**  
Create Procedure spGetNameById1  
@Id int,  
@Name nvarchar(20) Output  
as  
Begin  
 Select @Name = Name from tblEmployee Where Id = @Id  
End  
  
**Executing spGetNameById1, prints the name of the employee**  
Declare @EmployeeName nvarchar(20)  
Execute spGetNameById1 3, @EmployeeName out  
Print 'Name of the Employee = ' + @EmployeeName  
  
**Now let's try to achieve the same thing, using return status variables.**  
Create Procedure spGetNameById2  
@Id int  
as  
Begin  
 Return (Select Name from tblEmployee Where Id = @Id)  
End  
  
**Executing spGetNameById2** returns an error stating 'Conversion failed when converting the nvarchar value 'Sam' to data type int.'. The return status variable is an integer, and hence, when we select Name of an employee and try to return that we get a converion error.   
  
Declare @EmployeeName nvarchar(20)  
Execute @EmployeeName = spGetNameById2 1  
Print 'Name of the Employee = ' + @EmployeeName  
  
So, using return values, we can only return integers, and that too, only one integer. It is not possible, to return more than one value using return values, where as output parameters, can return any datatype and an sp can have more than one output parameters. I always prefer, using output parameters, over RETURN values.  
  
In general, RETURN values are used to indicate success or failure of stored procedure, especially when we are dealing with nested stored procedures.Return a value of 0, indicates success, and any nonzero value indicates failure.  
  
**Difference between return values and output parameters**  


### Advantages of using stored procedures

**The following advantages of using Stored Procedures over adhoc queries (inline SQL)**  
**1. Execution plan retention and reusability** - Stored Procedures are compiled and their execution plan is cached and used again, when the same SP is executed again. Although adhoc queries also create and reuse plan, the plan is reused only when the query is textual match and the datatypes are matching with the previous call. Any change in the datatype or you have an extra space in the query then, a new plan is created.  
  
**2. Reduces network traffic** - You only need to send, EXECUTE SP\_Name statement, over the network, instead of the entire batch of adhoc SQL code.  
  
**3. Code reusability and better maintainability** - A stored procedure can be reused with multiple applications. If the logic has to change, we only have one place to change, where as if it is inline sql, and if you have to use it in multiple applications, we end up with multiple copies of this inline sql. If the logic has to change, we have to change at all the places, which makes it harder maintaining inline sql.  
  
**4. Better Security** - A database user can be granted access to an SP and prevent them from executing direct "select" statements against a table.  This is fine grain access control which will help control what data a user has access to.  
  
**5. Avoids SQL Injection attack** - SP's prevent sql injection attack.