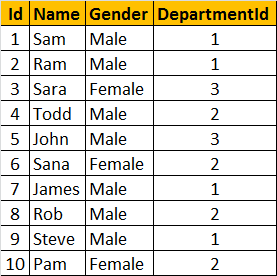
**In this video, we will**  
**1.** Understand what are stored procedure return values  
**2.** Difference between stored procedure return values and output parameters  
**3.** When to use output parameters over return values  
  
**Before watching this video, please watch**  
[Part 18 - Stored procedure basics in sql server](http://csharp-video-tutorials.blogspot.com/2012/08/stored-procedures-part-18.html)  
[Part 19 - Stored procedures with output parameters](http://csharp-video-tutorials.blogspot.com/2012/08/stored-procedures-with-output.html)  
  
   
  
   
   
  
  
**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**  
