



Stored Procedure

Database Programming



Stored Procedures

- A stored procedure is a saved collection of Transact-SQL statements
 - a batch that is stored in the database
 - or Precompiled script
- Basic Syntax

```
CREATE PROC          procName
    parameterList
AS
    statements
```



Stored Procedures - Advantages

- When a stored procedure is created, it is “pre-compiled”
 - When the procedure is run for the first time, all the information about it is stored in the system cache, which makes it much faster to run subsequently
- They require much less network traffic and therefore may run much more quickly than other queries passed from a client application
- They make maintenance of code much simpler because it is all managed in one place.
- Stored procedures can be used by multiple application
- Stored procedures are an excellent way of encapsulating business rules



Stored Proc - Syntax

```
CREATE { PROC | PROCEDURE }      procedure_name
      [ { @parameterdata_type } [ OUT | OUTPUT ] ] [ ,...n ]
AS
{
    [ BEGIN ]
        sql_statement [ ,...n ]
    [ END ]
}
```



CREATE PROC Statement

- CREATE PROCEDURE or CREATE PROC can be used
- The name of the stored procedure is an identifier name
- Parameters are optional
- The actual code is written after the AS keyword
- Optionally BEGIN ... END can be used for



@parameter

- One or more parameters can be declared in a CREATE PROCEDURE statement
- The value of each declared parameter must be supplied by the user when the procedure is called **unless** a default for the parameter is defined
- Declaring a parameter requires the following info:
 - ☐ The name
 - ☐ The data type
 - ☐ The default value
 - ☐ The direction



@parameter (cont.)

- The syntax is:

`@parameter_name [AS] datatype [= default|NULL] [OUTPUT|OUT]`

- The name must start with the @ sign
- The data type must be declared
- If you don't supply a default value, then
 - the parameter is assumed to be required, and
 - a value must be supplied when the stored procedure is called
- Values can be passed to stored procedures
 - by explicitly naming the parameters and assigning the appropriate value or
 - by supplying the parameter values given in the CREATE PROCEDURE statement without naming them



Data Type

- Each parameter has a data type

```
CREATE PROCEDURE      spGetEmployees
    @LastName nvarchar(50),
    @FirstName nvarchar(50)
AS
    SELECT  FirstName, LastName, JobTitle, Department
    FROM    EmployeeDepartment
    WHERE   FirstName = @FirstName AND
            LastName = @LastName;
GO
```



Using Named Parameters

- Naming the parameters when executing the stored procedure allows the parameters to be supplied in any order
- If you supply one parameter in the form @parameter = value, you must supply all subsequent parameters this way.
- If you do not supply parameters in the form @parameter = value, you must supply them in the order given in the CREATE PROCEDURE statement.



Named Parameters - Example

- my_proc expects three parameters
 - @first, @second, and @third
- My_proc can be called in one of the following ways

EXECUTE my_proc @second = 2, @first = 1, @third = 3;

EXECUTE my_proc 1, 2, 3;



Default

- If a default value is defined, the procedure can be executed without specifying a value for that parameter
- The default must be a constant or it can be NULL.
- To make a parameter optional, you have to supply a default value

```
CREATE PROCEDURE      spGetEmployees
    @DepID int = 1
AS
    SELECT      FirstName, LastName, JobTitle, Department
    FROM        Employee
    WHERE       DepID = @DepID
GO
EXECUTE  spGetEmployees
EXECUTE  spGetEmployees 2
```



OUTPUT

- OUTPUT parameters are used to return values to the caller of the procedure
 - OUTPUT indicates that the parameter is an output parameter
- The value of this option can be returned to the calling EXECUTE statement.
- The OUTPUT keyword is required
 - when the stored procedure is created, and
 - when the stored procedure is called



OUTPUT- Example

```
CREATE PROCEDURE      spGetSalary
    @EmpID int
    , @Salary money OUTPUT
AS
    SELECT  @Salary = Salary
    FROM    Employee
    WHERE   EmployeeID = @EmpID

GO

DECLARE @retVal money
EXEC spGetSalary 5, @retVal OUTPUT
select @retVal
```



ENCRYPTION

- Indicates that SQL Server will convert the original text of the CREATE PROCEDURE statement to an obscured format

```
CREATE PROCEDURE      spEncryptThis
WITH ENCRYPTION
AS
    SELECT  EmployeeID, Title, VacationHours, SickLeaveHours
    FROM    Employee;
GO
```



Executing Stored Procedures

- When a procedure is executed for the first time, it is compiled to determine an optimal access plan to retrieve the data
- Subsequent executions of the stored procedure may reuse the plan already generated
- EXEC or EXECUTE command is used
- Nested Stored Procedure
 - Nested Stored procedures refers to one stored procedure can call another



Return Values

- Return Values are used:
 - To return data, such as an identity value or the number of rows that the procedure has affected.
 - To return values to indicate the status of your stored procedures
- The recommended practice is to use return values to indicate success or failure of the stored proc
- By default, SQL Server automatically returns a value of zero when your procedure is complete.
- The RETURN value must be an integer



Return Values - Example

- To pass a return value back from our proc to the calling code, use the RETURN statement:
- Example

```
CREATE PROCEDURE  spGetNames
    @EmpID int
AS
    SELECT  FName + ' ' + LName
    FROM    Employee
    WHERE   EmployeeID = @EmpID

    Return  @@RowCount
GO

DECLARE @Return int;

EXEC      @Return = spGetNames 22;
SELECT    @Return;
```



Return Values – Example 2

```
CREATE PROCEDURE uspGetEmployeesByDep (  
    @depName varchar(50),  
    @employeeCount INT OUTPUT  
) AS  
BEGIN  
    SELECT      FName , LName  
    FROM        Employee E JOIN Department D  
               ON E.DepID = D.DepID  
    WHERE       D.DepName = @depName  
  
    SELECT @employeeCount = @@ROWCOUNT;  
END  
GO  
  
DECLARE @empCount INT  
EXEC uspGetEmployeesByDep 'Sales', @empCount OUTPUT  
SELECT @empCount AS 'Number of employees found'
```

Exercise: Update the above stored procedure so that it returns the list of employees working in ALL departmenets when the no parameter is passed to it



ALTER Stored Procedures

- ALTER PROC replaces the existing proc
- The differences between using the ALTER PROC statement and the CREATE PROC statement are:
 - ALTER PROC expects to find an existing proc
 - ALTER PROC retains any permissions that have been established for the proc
- ALTER PROC keeps the same object ID and allows the dependencies to be kept
- Dropping stored procedures
`DROP PROC|PROCEDURE sproc_name`



Calling Stored Procedures from C#

Using System.Data;

Using System.Data.SqlClient;

```
public static int InsertCity(string dbConStr, CityObj newCityObj)
{
    SqlConnection cnn = new SqlConnection();
    cnn = DatabaseManager.openAdminConnection(dbConStr);
    SqlCommand cmd = null;

    cmd = new SqlCommand();
    cmd.Connection = cnn;
    cmd.CommandType = System.Data.CommandType.StoredProcedure;
    cmd.CommandText = "[spCityInsert]";

    SqlParameter outParam = new SqlParameter();
    outParam = cmd.Parameters.Add("@CityID", SqlDbType.Int);
    outParam.Direction = ParameterDirection.Output;

    cmd.Parameters.Add("@CityName", SqlDbType.NVarChar, 50).Value = newCityObj.CityName;
    cmd.Parameters.Add("@IsActive", SqlDbType.Bit).Value = newCityObj.IsActive;

    SqlDataReader dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
    int returnValue = System.Int32.Parse(cmd.Parameters["@CityID"].Value.ToString());

    return returnValue;
}
```



Exercise A

1. Create a stored procedure that can be used to insert an employee into the Employee table
2. Create a stored procedure that can be used to change the department of an employee
 - a) Parameters (employeeID, newDepName)
 - b) Return success or failure indicator
3. Create a stored procedure that can be used to create a new department and change the departments of employees in an existing department to the new one
4. Create a stored procedure returns the list of employees working in a department when given the depID, otherwise it should return all employees



Exercise B

Create Stored Procedures for each of the following: Use The sample database “**AdvWorks**”

1. Insert into the **SalesOrder** Table a sales record given customer name
2. Insert into the **SalesOrderDetail** Table the details for all sales made by a sales order
3. Update the **TotalAmount** column of the **SalesOrder** table based on the **SalesOrderDetails** of a **SalesOrder**
4. Delete a **SalesOrder** made by a customer and all the **SalesOrderDetail** entries made for it