Additional Notes ON - Stored Procedure

Overview

A stored procedure is nothing more than prepared SQL code that you save so you can reuse the code over and over again. So if you think about a query that you write over and over again, instead of having to write that query each time you would save it as a stored procedure and then just call the stored procedure to execute the SQL code that you saved as part of the stored procedure.

In addition to running the same SQL code over and over again you also have the ability to pass parameters to the stored procedure, so depending on what the need is the stored procedure can act accordingly based on the parameter values that were passed.

Take a look through each of these topics to learn how to get started with stored procedure development for SQL Server.

You can either use the outline on the left or click on the arrows to the right or below to scroll through each of these topics.

Input Parameter:

One Parameter

In this example we will query the Person.Address table from the AdventureWorks database, but instead of getting back all records we will limit it to just a particular city. This example assumes there will be an exact match on the City value that is passed.

```
USE AdventureWorks
GO

CREATE PROCEDURE dbo.uspGetAddress @City nvarchar(30)
AS
SELECT *
FROM Person.Address
WHERE City = @City
GO
```

To call this stored procedure we would execute it as follows:

```
EXEC dbo.uspGetAddress @City = 'New York'
```

Multiple Parameters

Setting up multiple parameters is very easy to do. You just need to list each parameter and the data type separated by a comma as shown below.

```
USE AdventureWorks
GO

CREATE PROCEDURE dbo.uspGetAddress @City nvarchar(30) = NULL, @AddressLine1
nvarchar(60) = NULL
AS
SELECT *
FROM Person.Address
WHERE City = ISNULL(@City,City)
AND AddressLine1 LIKE '%' + ISNULL(@AddressLine1 ,AddressLine1) + '%'
GO
```

To execute this you could do any of the following:

```
EXEC dbo.uspGetAddress @City = 'Calgary'
--or
EXEC dbo.uspGetAddress @City = 'Calgary', @AddressLine1 = 'A'
--or
EXEC dbo.uspGetAddress @AddressLine1 = 'Acardia'
```

Simple Output

```
CREATE PROCEDURE dbo.uspGetAddressCount @City nvarchar(30), @AddressCount
int OUTPUT
AS
SELECT @AddressCount = count(*)
FROM AdventureWorks.Person.Address
WHERE City = @City
```

Or it can be done this way:

```
CREATE PROCEDURE dbo.uspGetAddressCount @City nvarchar(30), @AddressCount
int OUT
AS
SELECT @AddressCount = count(*)
FROM AdventureWorks.Person.Address
WHERE City = @City
```

To call this stored procedure we would execute it as follows. First we are going to declare a variable, execute the stored procedure and then select the returned valued.

```
DECLARE @AddressCount int
EXEC dbo.uspGetAddressCount @City = 'Calgary', @AddressCount =
@AddressCount OUTPUT
SELECT @AddressCount
```

This can also be done as follows, where the stored procedure parameter names are not passed.

```
DECLARE @AddressCount int
EXEC dbo.uspGetAddressCount 'Calgary', @AddressCount OUTPUT
SELECT @AddressCount
```

Exceptional Handling:

Try ... Catch

If you are not familiar with the Try...Catch paradigm it is basically two blocks of code with your stored procedures that lets you execute some code, this is the Try section and if there are errors they are handled in the Catch section.

Let's take a look at an example of how this can be done. As you can see we are using a basic SELECT statement that is contained within the TRY section, but for some reason if this fails it will run the code in the CATCH section and return the error information.

```
CREATE PROCEDURE dbo.uspTryCatchTest
AS
BEGIN TRY
    SELECT 1/0
END TRY
BEGIN CATCH
    SELECT ERROR_NUMBER() AS ErrorNumber
     ,ERROR_SEVERITY() AS ErrorSeverity
     ,ERROR_STATE() AS ErrorState
     ,ERROR_PROCEDURE() AS ErrorProcedure
     ,ERROR_LINE() AS ErrorLine
     ,ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

For N