STORE PROCEDURE

Nguyen Minh Dao

Overview

- A stored procedure is nothing more than prepared SQL code that you save so you can reuse the code over and over again.
- In addition to running the same SQL code over and over again you also have the ability to pass parameters to the stored procedure.

Creating a stored procedure

- Syntax

```
--Transact-SQL Stored Procedure Syntax
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
        [ VARYING ] [ = default ] [ OUT | OUTPUT ] [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]

<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE AS Clause ]
```

Creating a simple stored procedure (CREATE PROCEDURE)

- Before you create a stored procedure you need to know what your end result is, whether you are selecting data, inserting data, etc..
- In this simple example we will just select all data from the Person.Address table that is stored in the AdventureWorks database.
- So the simple T-SQL code would be as follows which will return all rows from this table.
  - **SELECT * FROM AdventureWorks.Person.Address**

Creating a simple stored procedure (CREATE PROCEDURE)

- To create a stored procedure to do this the code would look like this:
- **CREATE PROCEDURE uspGetAddress**
  - **AS**
  - ***SELECT * FROM AdventureWorks.Person.Address***
- When creating a stored procedure you can either use **CREATE PROCEDURE** or **CREATE PROC**. After the stored procedure name you need to use the keyword "**AS**" and then the rest is just the regular SQL code that you would normally execute.

Creating a simple stored procedure (CREATE PROCEDURE)

- To call the procedure to return the contents from the table specified, the code would be:
  - EXEC **uspGetAddress**
- --or just simply
  - **uspGetAddress**

Results | Messages

	AddressID	AddressLine1	AddressLine2	City	StateProvinceID	PostalCode	rowguid	ModifiedDate
1	1	1970 Napa Ct.	NULL	Bothell	79	98011	9AADCB0D-36CF-483F-84D8-585C2D4EC6E9	1998-01-04 00:00:00.000
2	2	9833 Mt. Dias Blv.	NULL	Bothell	79	98011	32A54B9E-E034-4BFB-B573-A71CDE60D8C0	1999-01-01 00:00:00.000
3	3	7484 Roundtree Drive	NULL	Bothell	79	98011	4C506923-6D1B-452C-A07C-BAA6F5B142A4	2003-04-08 00:00:00.000
4	4	9539 Glenside Dr	NULL	Bothell	79	98011	E5946C78-4BCC-477F-9FA1-CC09DE16A880	1999-03-07 00:00:00.000
5	5	1226 Shoe St.	NULL	Bothell	79	98011	FBAFF937-4A97-4AF0-81FD-B849900E9BB0	1999-01-20 00:00:00.000
6	6	1399 Firestone Drive	NULL	Bothell	79	98011	FEBF8191-9804-44C8-877A-33FDE94F0075	1999-03-17 00:00:00.000
7	7	5672 Hale Dr.	NULL	Bothell	79	98011	0175A174-6C34-4D41-B3C1-4419CD6A0446	2000-01-12 00:00:00.000

Create a SQL Server stored procedure with parameters

- The real power of stored procedures is the ability to pass parameters and have the stored procedure handle the differing requests that are made.
- 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.

One Parameter

- This example assumes there will be an exact match on the City value that is passed.
- CREATE **PROCEDURE uspGetAddress**
  - **@City nvarchar(30)**
  - AS
    - SELECT *
    - FROM **AdventureWorks.Person.Address**
    - WHERE **City = @City**
- GO

One Parameter

- To call this stored procedure we would execute it as follows:
  - EXEC **uspGetAddress @City = 'New York'** (or)
  - EXEC **uspGetAddress 'New York'**

Results | Messages

	AddressID	AddressLine1	AddressLine2	City	StateProvinceID	PostalCode	rowguid	ModifiedDate
1	770	123 Union Square South	NULL	New York	54	10007	CEC2AD79-89D2-43A7-BA8C-FB2B11BDC170	2001-08-01 00:00:00.000
2	346	2596 Big Canyon Road	NULL	New York	54	10007	2D51A47B-7DBB-4EEF-95AB-62FB5FEA34B8	2002-01-24 00:00:00.000

One Parameter

- We can also do the same thing, but allow the users to give us a starting point to search the data. Here we can change the "=" to a LIKE and use the "%" wildcard.
- CREATE **PROCEDURE uspGetAddress**
  - **@City nvarchar(30)**
  - AS
    - SELECT *
    - FROM **AdventureWorks.Person.Address**
    - WHERE **City LIKE @City + '%'**
- GO

One Parameter

- In both of the proceeding examples it assumes that a parameter value will always be passed.
- If you try to execute the procedure without passing a parameter value you will get an error message such as the following:
  - Msg 201, Level 16, State 4, Procedure uspGetAddress, Line 0
  - Procedure or function 'uspGetAddress' expects parameter '@City', which was not supplied.

Default Parameter Values

- In most cases it is always a good practice to pass in all parameter values, but sometimes it is not possible.
- So in this example we use the NULL option to allow you to not pass in a parameter value.
- If we create and run this stored procedure as is it will not return any data, because it is looking for any City values that equal NULL.

Default Parameter Values

- CREATE **PROCEDURE uspGetAddress**
  - **@City nvarchar(30) = NULL**
  - AS
    - SELECT *
    - FROM **AdventureWorks.Person.Address**
    - WHERE **City = @City**
- To call this stored procedure we would execute it as follows:
  - EXEC **uspGetAddress** @City = 'New York' (or)
  - EXEC **uspGetAddress**

Default Parameter Values

- We could change this stored procedure and use the **ISNULL** function to get around this.
- So if a value is passed it will use the value to narrow the result set and if a value is not passed it will return all records.
- CREATE **PROCEDURE uspGetAddress**
  - **@City nvarchar(30) = NULL**
  - AS
    - SELECT *
    - FROM **AdventureWorks.Person.Address**
    - WHERE **City = ISNULL(@City,City)**
- GO

Default Parameter Values

- To call this stored procedure we would execute it as follows:
  - EXEC **uspGetAddress** @City = 'New York' (or)

Results | Messages

	AddressID	AddressLine1	AddressLine2	City	StateProvinceID	PostalCode	rowguid	ModifiedDate
1	770	123 Union Square South	NULL	New York	54	10007	CEC2AD79-89D2-43A7-BA8C-FB2B11BDC170	2001-08-01 00:00:00.000
2	346	2596 Big Canyon Road	NULL	New York	54	10007	2D51A47B-7DBB-4EEF-95AB-62FB5FEA34B8	2002-01-24 00:00:00.000

  - EXEC **uspGetAddress**

Results | Messages

	AddressID	AddressLine1	AddressLine2	City	StateProvinceID	PostalCode	rowguid	ModifiedDate
1	1	1970 Napa Ct.	NULL	Bothell	79	98011	9AADCB0D-36CF-483F-84D8-585C2D4EC6E9	1998-01-04 00:00:00.000
2	2	9833 Mt. Dias Blv.	NULL	Bothell	79	98011	32A54B9E-E034-4BFB-B573-A71CDE60D8C0	1999-01-01 00:00:00.000
3	3	7484 Roundtree Drive	NULL	Bothell	79	98011	4C506923-6D1B-452C-A07C-BAA6F5B142A4	2003-04-08 00:00:00.000
4	4	9539 Glenside Dr	NULL	Bothell	79	98011	E5946C78-4BCC-477F-9FA1-CC09DE16A880	1999-03-07 00:00:00.000
5	5	1226 Shoe St.	NULL	Bothell	79	98011	FBAFF937-4A97-4AF0-81FD-B849900E9BB0	1999-01-20 00:00:00.000
6	6	1399 Firestone Drive	NULL	Bothell	79	98011	FEBF8191-9804-44C8-877A-33FDE94F0075	1999-03-17 00:00:00.000
7	7	5672 Hale Dr.	NULL	Bothell	79	98011	0175A174-6C34-4D41-B3C1-4419CD6A0446	2000-01-12 00:00:00.000

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

Multiple Parameters

- To execute this you could do any of the following:
  - EXEC **uspGetAddress @City = 'Calgary'**
- --or
  - EXEC **uspGetAddress @City = 'Calgary', @AddressLine1 = 'A'**
- --or
  - EXEC **uspGetAddress @AddressLine1 = 'Acardia'**
- -- etc...

Store procedure

Returning stored procedure parameter values to a calling stored procedure
(OUTPUT)

Overview

- In a previous topic we discussed how to pass parameters into a stored procedure, but another option is to pass parameter values back out from a stored procedure.
- One option for this may be that you call another stored procedure that does not return any data, but returns parameter values to be used by the calling stored procedure.

Explanation

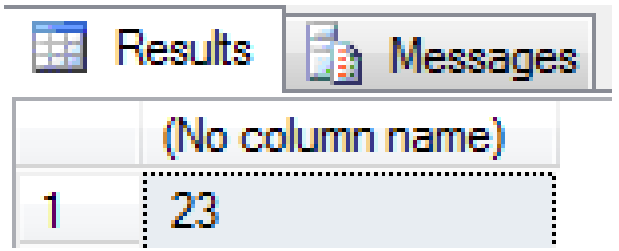
- Setting up output paramters for a stored procedure is basically the same as setting up input parameters, the only difference is that you use the OUTPUT clause after the parameter name to specify that it should return a value.
- The output clause can be specified by either using the keyword "OUTPUT" or just "OUT".

Simple Output

- CREATE **PROCEDURE 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:
  - @AddressCount int **OUT**

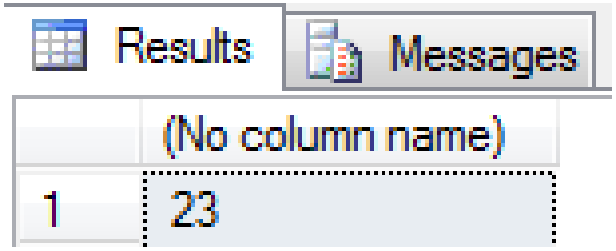
Simple Output

- 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 **uspGetAddressCount @City = 'Calgary', @AddressCount = @AddressCount OUTPUT**
  - SELECT **@AddressCount**



Simple Output

- This can also be done as follows, where the stored procedure parameter names are not passed.
  - DECLARE **@AddressCount int**
  - EXEC **uspGetAddressCount 'Calgary', @AddressCount OUTPUT**
  - SELECT **@AddressCount**



Store procedure

Using try catch in SQL Server stored procedures
(TRY...CATCH)

Overview

- A great new option that was added in SQL Server 2005 was the ability to use the Try..Catch paradigm that exists in other development languages.
- Doing error handling in SQL Server has not always been the easiest thing, so this option definitely makes it much easier to code for and handle errors.

Explanation

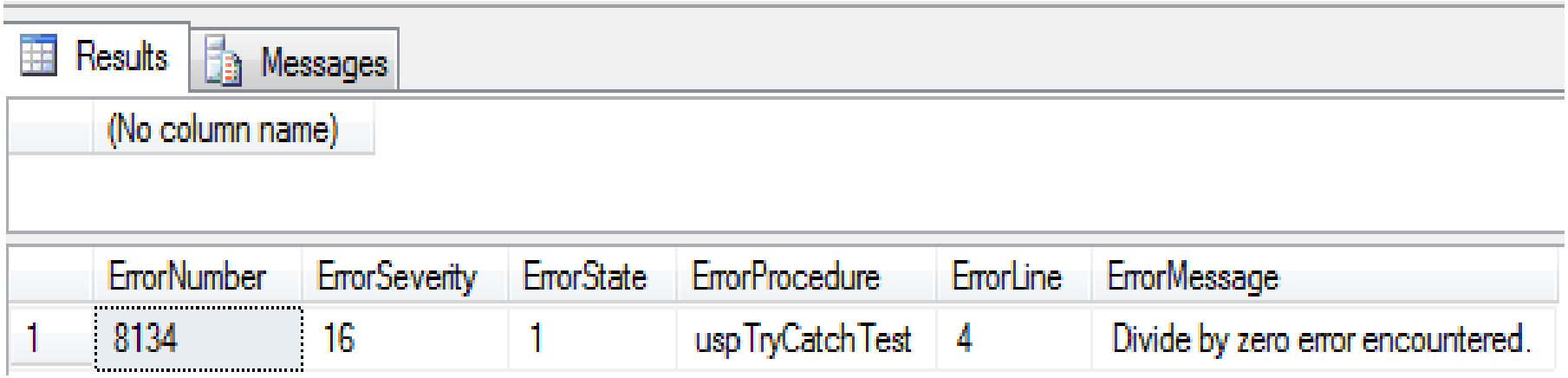
- If you are not familiar with the Try...Catch paradigm it is basically two blocks of code with your stored procedures that lets you execute some code, this is the Try section and if there are errors they are handled in the Catch section.
- Let's take a look at an example of how this can be done. As 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.

Example

- CREATE **PROCEDURE** **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**

Example

- To call the procedure to return the contents from the table specified, the code would be:
  - **uspTryCatchTest**



Store procedure

Using comments in a SQL Server stored procedure

Overview

- One very helpful thing to do with your stored procedures is to add comments to your code.
- This helps you to know what was done and why for future reference, but also helps other DBAs or developers that may need to make modifications to the code.

Explanation

- SQL Server offers two types of comments in a stored procedure; line comments and block comments.
- The following examples show you how to add comments using both techniques.
- Comments are displayed in green in a SQL Server query window.

Line Comments

- To create line comments you just use two dashes "--" in front of the code you want to comment.
- You can comment out one or multiple lines with this technique.
- In this example the entire line is commented out.
  - -- this procedure gets a list of addresses based
  - -- on the city value that is passed
  - CREATE **PROCEDURE uspGetAddress @City nvarchar(30)**
  - AS
    - SELECT *
    - FROM **AdventureWorks.Person.Address**
    - WHERE **City = @City**
  - GO

Line Comments

- This next example shows you how to put the comment on the same line.
  - -- this procedure gets a list of addresses based on the city value that is passed
  - CREATE **PROCEDURE uspGetAddress @City nvarchar(30)**
    - AS
    - SELECT *
    - FROM **AdventureWorks.Person.Address**
    - WHERE **City = @City** -- the @City parameter value will narrow the search criteria
  - GO

Block Comments

- To create block comments the block is started with "/*" and ends with "*/". Anything within that block will be a comment section.
  - /*
  - -this procedure gets a list of addresses based
  - on the city value that is passed
  - -this procedure is used by the HR system
  - */
  - CREATE **PROCEDURE uspGetAddress**
    - **@City nvarchar(30)**
    - AS
    - SELECT *
    - FROM **AdventureWorks.Person.Address**
    - WHERE **City = @City**
  - GO

Combining Line and Block Comments

- You can also use both types of comments within a stored procedure.
  - /*
  - -this procedure gets a list of addresses based
  - on the city value that is passed
  - -this procedure is used by the HR system
  - */
  - CREATE **PROCEDURE uspGetAddress**
    - **@City nvarchar(30)**
    - AS
      - SELECT *
      - FROM **AdventureWorks.Person.Address**
      - WHERE **City = @City** -- the @City parameter value will narrow the search criteria
  - GO

Naming conventions for SQL Server stored procedures

- One good thing to do for all of your SQL Server objects is to come up with a naming convention to use.
- There are not any hard and fast rules, so this is really just a guideline on what should be done.
- **Explanation**
  - SQL Server uses object names and schema names to find a particular object that it needs to work with.
  - This could be a table, stored procedure, function ,etc...
  - It is a good practice to come up with a standard naming convention for you objects including stored procedures.

Naming conventions for SQL Server stored procedures

- **Do not use sp_ as a prefix**
  - One of the things you do not want to use as a standard is "**sp_**".
  - This is a standard naming convention that is used in the master database.
  - If you do not specify the database where the object is, SQL Server will first search the master database to see if the object exists there and then it will search the user database. So avoid using this as a naming convention.

Standardize on a Prefix

- It is a good idea to come up with a standard prefix to use for your stored procedures. As mentioned above do not use "sp_", so here are some other options.
  - usp_
  - sp
  - usp
  - etc...
- To be honest it does not really matter what you use. SQL Server will figure out that it is a stored procedure, but it is helpful to differentiate the objects, so it is easier to manage.

Standardize on a Prefix

- So a few examples could be:
  - spInsertPerson
  - uspInsertPerson
  - usp_InsertPerson
  - InsertPerson
- Again this is totally up to you, but some standard is better than none.

Naming Stored Procedure Action

- Based on the actions that you may take with a stored procedure, you may use:
  - Insert ,Delete ,Update ,Select
  - Get
  - Validate
  - etc...
- So here are a few examples:
  - uspInsertPerson
  - uspGetPerson
  - spValidatePerson
  - SelectPerson
  - etc...

Store procedure

Reducing amount of network data for SQL Server stored procedures
(SET NOCOUNT ON)

Overview

- There are many tricks that can be used when you write T-SQL code.
- One of these is to reduce the amount of network data for each statement that occurs within your stored procedures.
- Every time a SQL statement is executed it returns the number of rows that were affected.
- By using "**SET NOCOUNT ON**" within your stored procedure you can shut off these messages and reduce some of the traffic.

Explanation

- As mentioned above there is not really any reason to return messages about what is occuring within SQL Server when you run a stored procedure.
- If you are running things from a query window, this may be useful, but most end users that run stored procedures through an application would never see these messages.
- You can still use **@@ROWCOUNT** to get the number of rows impacted by a SQL statement, so turning **SET NOCOUNT ON** will not change that behavior.

Not using SET NOCOUNT ON

- Here is an example without using SET NOCOUNT ON:
  - -- not using SET NOCOUNT ON
  - CREATE **PROCEDURE uspGetAddress**
    - **@City nvarchar(30)**
    - AS
      - SELECT *
      - FROM **AdventureWorks.Person.Address**
      - WHERE **City = @City**
  - GO
- The messages that are returned would be similar to this:
  - (23 row(s) affected)

Using SET NOCOUNT ON

- This example uses the SET NOCOUNT ON as shown below.
- It is a good practice to put this at the beginning of the stored procedure.
  - -- using SET NOCOUNT ON
  - CREATE **PROCEDURE uspGetAddress**
    - **@City nvarchar(30)**
    - AS
      - SET **NOCOUNT ON**
      - SELECT * FROM **AdventureWorks.Person.Address**
      - WHERE **City = @City**
  - GO
- The messages that are returned would be similar to this:
  - Command(s) completed successfully.

Using SET NOCOUNT ON and @@ROWCOUNT

- This example uses SET NOCOUNT ON, but will still return the number of rows impacted by the previous statement. This just shows that this still works.
  - -- not using SET NOCOUNT ON
  - CREATE **PROCEDURE uspGetAddress**
    - **@City nvarchar(30)**
    - AS
      - SET **NOCOUNT ON**
      - SELECT * FROM **AdventureWorks.Person.Address**
      - WHERE **City = @City**
      - PRINT **@@ROWCOUNT**
  - GO
- The messages that are returned would be similar to this:
  - 23

Deleting a SQL Server stored procedure

- **Dropping Single Stored Procedure**
  - To drop a single stored procedure you use the DROP PROCEDURE or DROP PROC command as follows.
    - DROP **PROCEDURE** **uspGetAddress**
    - GO
  - -- or
    - DROP **PROC** **uspGetAddress**
    - GO
  - -- or
    - DROP **PROC** **dbo.uspGetAddress** – **also specify the schema**

Dropping Multiple Stored Procedures

- To drop multiple stored procedures with one command you specify each procedure separated by a comma as shown below.
  - DROP **PROCEDURE uspGetAddress, uspInsertAddress, uspDeleteAddress**
  - GO
  - -- or
  - DROP **PROC uspGetAddress, uspInsertAddress, uspDeleteAddress**
  - GO

Modifying an existing SQL Server stored procedure

- When you first create your stored procedures it may work as planned, but how to do you modify an existing stored procedure. In this slide we look at the **ALTER PROCEDURE** command and it is used.
- **Modifying an Existing Stored Procedure**
  - ALTER **PROCEDURE uspGetAddress**
    - **@City nvarchar(30)**
    - AS
      - SELECT * FROM **AdventureWorks.Person.Address**
      - WHERE **City LIKE @City + '%'**
  - GO


Thank You!