Creating SQL Server Stored Procedures

Paul Litwin
FHCRC Collaborative Data Services

CDS Brownbag Series

- This is the 11th in a series of seminars
- Materials for the series can be downloaded from www.deeptraining.com/fhcrc
- Sign up to be on our email list
- Upcoming
  - SQL Server for Administrators
  - May 2 at 12:30 in M5C813/C815

CDS

- Collaborative Data Services
  - Providing "data services" through FHCRC (and beyond)
  - Our services include...
    - Telephone interviewing of subjects
    - Data entry & scanning
    - Programming
    - Web and database hosting
- More info at http://cds.fhcrc.org

ASP.NET Master Class, Version 1.9d (01/04)
Seminar Materials

- Can be downloaded from
  - www.deeptraining.com/fhcrc
- Includes
  - This slide presentation
  - Sample scripts to create database & sprocs

Agenda

- **Basic Stored Procedure Syntax**
- Parameters and Variables
- T-SQL Control-of-Flow Statements
- Using Built-In Variables and Functions

Why Stored Procedures?

- Precompiled and stored on server (faster than ad-hoc SQL)
- Centralize business logic in one place
- Can include complex control-of-flow statements
- You can remove permissions to tables while still allowing users to execute stored procs against tables
- Help guard against SQL injection attacks
Basic Stored Procedure Syntax

CREATE PROCEDURE proc_name
AS
sql-statement

What Can a Sproc Do?

- Just about anything you can do with T-SQL, including...
  - returning rows (SELECT query)
  - updating data (UPDATE, DELETE, INSERT)
  - modifying schema (CREATE TABLE, CREATE PROCEDURE, etc.)
  - lots more...

Basic Stored Procedure Syntax Examples

- Example 1 (DML Query)
  CREATE PROCEDURE procGetCustomer1
  AS
  SELECT * FROM tblCustomer

- Example 2 (DML Update)
  CREATE PROCEDURE procMenuPricesIncrease
  AS
  UPDATE tblMenu
  SET PRICE = Price*1.1

- Example 3 (DDL Statement)
  CREATE PROC procCreateTestTable1
  AS
  CREATE TABLE tblTest
  (Id INT CONSTRAINT PrimaryKey PRIMARY KEY,
   Description VARCHAR(20))

ASP.NET Master Class, Version 1.9d (01/04)
Agenda

- Basic Stored Procedure Syntax
- **Parameters and Variables**
- T-SQL Control-of-Flow Statements
- Using Built-In Variables and Functions

Parameters

- Stored procedures without parameters are nice but not that useful...
- Parameters allow you to make generic stored procedures that can be used in a variety of situations

```sql
CREATE PROCEDURE proc_name
    @param1 datatype1 [=default]
    [Output],
    @param2 datatype2 [=default]
    [Output],
    ...
AS
sql-statement
```
**Parameter Example**

CREATE PROCEDURE procGetCustomer2
@custid INT
AS
SELECT * FROM tblCustomer
WHERE CustomerId = @custid

**Parameter Example w/ Default**

Create Procedure procGetCustomer4
@custid INT = NULL
AS
IF @custid IS NULL
SELECT * FROM tblCustomer
ORDER BY CustomerId
ELSE
SELECT * FROM tblCustomer
WHERE CustomerId = @custid

**Output Parameter Example**

CREATE PROC procInsertOrder
@orderdate DATETIME,
...
@orderid INT OUTPUT
AS
INSERT INTO tblOrder
...
SELECT @orderid = @@IDENTITY
Using Variables

- In addition to parameters, you can create local stored proc variables
- To use a variable in your stored proc, you must first declare it like so:

  DECLARE @variable datatype

Variable Example

CREATE PROC procInsertOrderTest
AS
  DECLARE @intOrderId INT

  EXECUTE procInsertOrder
  '1/1/2000', 1, 1, '1/1/2000', 'Cash', 'Test Order', @intOrderId OUTPUT

  SELECT @intOrderId AS NewOrderId

Working with Variables

- Setting a variable to a value:
  SELECT @variable = value
  (can also use SET instead of SELECT)
  For example:
  SELECT @orderid = @@IDENTITY

- Returning a variable in a recordset:
  SELECT @variable AS fieldname
  For example:
  SELECT @intOrderId AS NewOrderId
Return Value

- The SQL Server return value is a special output parameter that:
  - Is an INTEGER
  - Is returned using the RETURN statement:
    
    ```
    RETURN return_value
    ```

Return Value Example

```sql
CREATE PROC procCustomerExist AS
  IF (SELECT Count(*) FROM tblCustomer WHERE CustomerId = @custid) >= 1
    -- Success
    RETURN 0
  ELSE
    -- Failure
    RETURN 1
```

Agenda

- Basic Stored Procedure Syntax
- Parameters and Variables
- T-SQL Control-of-Flow Statements
- Using Built-In Variables and Functions
T-SQL Control-of-Flow Statements

- Stored procedures can contain two types of statements:
  - ANSI SQL statements
  - Transact-SQL Control-of-Flow statements

T-SQL Control of Flow Statements
IF...ELSE

CREATE PROC procCustomerExist
AS
  IF (SELECT Count(*) FROM tblCustomer
      WHERE CustomerId = @custid) >= 1
    -- Success
    RETURN 0
  ELSE
    -- Failure
    RETURN 1

T-SQL Control of Flow Statements
BEGIN...END

CREATE PROC procCustomerExist2
AS
  IF (SELECT Count(*) FROM tblCustomer
      WHERE CustomerId = @custid) >= 1
    BEGIN
      -- Success
      PRINT 'Success'
      RETURN 0
    END
  ELSE
    ...

ASP.NET Master Class, Version 1.9d (01/04)
T-SQL Control of Flow Statements
EXECUTE (or EXEC)

- Used to execute another stored proc
- Executing an SP with no parameters
  EXEC procGetCustomer1
- Executing an SP with parameters by position
  EXEC progGetCustomer3 "Greg", "Reddick"
- Executing an SP with parameters by name
  EXEC procGetCustomer3
  @LastName = 'Reddick', @FirstName = 'Greg'

T-SQL Control of Flow Statements
More on EXECUTE

- Executing an SP with an output parameter (from procInsertOrderTest)
  EXEC procInsertOrder
  '1/1/2000', 1, 1, '1/2/2000', 'Cash', 'Test order', @intOrderId
  OUTPUT
- Executing an SP with a return value (from procCustomerExistTest)
  EXEC @intReturn = procCustomerExist
  @custid

T-SQL Control of Flow Statements
Misc

- Comments
  -- single-line comment
  /* this is a multiple line comment */
- PRINT
  - Prints text to SQL Query Analyzer
  - Similar to Debug.Print / Trace.Write
- SET NOCOUNT ON
  - Turns off rowcount messages sent to the client
  - May be necessary if the messages confuse the client
Agenda

- Basic Stored Procedure Syntax
- Parameters and Variables
- T-SQL Control-of-Flow Statements
- **Using Built-In Variables and Functions**

Using Built-In Variables and Functions

- SQL Server has a number of built-in variables and functions that you can (and should) take advantage of

Built-In T-SQL Variables

- @@ERROR
  - error number of last executable T-SQL statement or 0 if no error
- @@IDENTITY
  - the last-assigned identity column value of the current connection
  - see procInsertOrder example
  - should use SCOPE_IDENTITY() function instead in some cases
- @@ROWCOUNT
  - number of rows affected by last SQL statement
Error Handling

- Your stored procedures can...
  - Detect errors using @@Error
  - Also errors might be detected using @@ROWCOUNT
- Roll back transactions
  - ROLLBACK TRAN
- Report errors to the calling program
  - Use RETURN with an error code (integer)
  - Use RAISERROR statement

Error Handling in Stored Procs From SpecBankSprocs.txt (1 of 3)

```sql
SELECT @Freezer = Freezer,
       @FreezerStack = FreezerStack,
       @FreezerLevel = FreezerLevel,
       @SlotRow = SlotRow,
       @SlotCol = SlotCol
FROM tblFreezerSpaces
WHERE FreezerSpaceId = @FreezerSpaceId
SET @Error = @@Error
```

Error Handling in Stored Procs From SpecBankSprocs.txt (2 of 3)

```sql
IF @Error > 0
BEGIN
  PRINT 'Error: ' + CONVERT(VarChar, @Error)
  SELECT @ErrorMsg = description
  FROM master..sysmessages
  WHERE error = @Error
  PRINT 'Message: ' + @ErrorMsg
END
```

ASPD.NET Master Class, Version 1.9d (01/04)

Page 11
Error Handling in Stored Procs
From SpecBankSprocs.txt (3 of 3)

IF (@Errors=0)
BEGIN
  COMMIT TRAN
  Return 0
END
ELSE
BEGIN
  ROLLBACK TRAN
  PRINT 'Transaction rolled back. '
  PRINT 'One or more errors encountered.'
  Return 1
END

Transactions

- SET TRANSACTION ISOLATION LEVEL
  - used to determine the level of isolation
    - READ UNCOMMITTED (lowest)
    - READ COMMITTED
    - REPEATABLE READ
    - SERIALIZABLE (highest)
- BEGIN TRAN
- COMMIT TRAN
- ROLLBACK TRAN

Built-In T-SQL Functions

- Lots of built-in functions, including...
  - DATEADD – adds an interval to a date
  - DATEDIFF – returns an interval between two dates
  - GETDATE – returns system date & time
  - RAND – returns a random number between 0 & 1
  - LTRIM – removes leading blanks
  - SUBSTRING – returns a portion of a string
  - CAST – converts from one datatype to another
    (can also use CONVERT)
  - SOUNDEX – calculates Soundex code
Built-In T-SQL Function Example

- **CAST Function Example**
  
  CAST is necessary because SQL Server doesn't automatically coerce datatypes

  ```sql
  CREATE PROC procCustomerOrderCount
  @custid INT
  AS
  ...
  PRINT 'Customer #' +
  CAST(@custid AS varchar(5)) + ' has made ' +
  CAST(@ordercount AS varchar(5)) + ' orders.'
  ```

Summary

- Stored procedures are a powerful and efficient way to execute SQL
- We've discussed:
  - Basic Stored Procedure Syntax
  - Parameters and Variables
  - T-SQL Control-of-Flow Statements
  - Using Built-In Variables and Functions

Thank You!

- Materials can be downloaded from
  - www.deeptraining.com/fhcrc
- Includes
  - This slide presentation
  - Sample scripts to create database & sprocs