Implementing Stored Procedures

Presented By:

Verawaty Situmorang

Overview

- Introduction to Stored Procedures
- Creating, Executing, and Modifying Stored Procedures
- Using Parameters in Stored Procedures

Introduction of Stored Procedures

A stored procedure is a named collection of Transact-SQL statements that is stored on the server. Stored procedures are a method of encapsulating repetitive tasks that executes efficiently.

A precompiled collection of Transact-SQL statements stored under a name and processed as a unit. SQL Server-supplied stored procedures are called system stored procedures.

- Named Collections of Transact-SQL Statements
- Encapsulate Repetitive Tasks
- Five Types (System, Temporary, Local, Extended and Remote)
- Accept Input Parameters and Return Values
- Return Status Value to Indicate Success or Failure

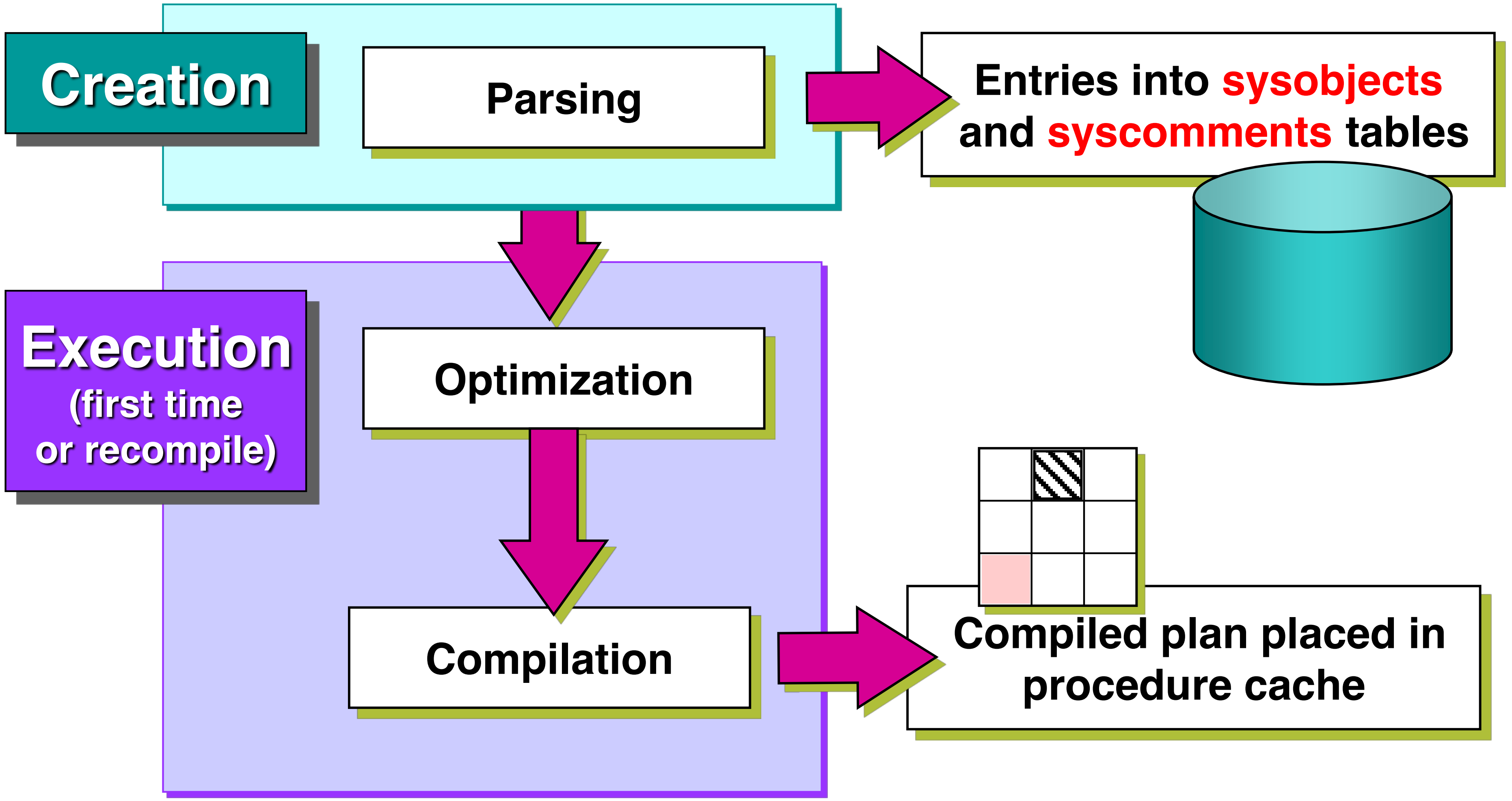
Introduction of Stored Procedures

- Named program compiled and stored IN the server as an independent database object

  Collection of:
  - SQL-statements and/or
  - procedural logic (if-statements, while-statements, etc.) and/or
  - contain programming statements that perform operations in the database. These include calling other procedures.
  - calls of built-in functions (getdate(), etc.)
  - Return a status value to a calling program to indicate success or failure (and the reason for failure)

- Can be called from a client
  - or from another stored procedure
  - parameters may be passed and returned
  - returned error codes may be checked

Initial Processing of Stored Procedures


Creation
Parsing
Entries into sysobjects and syscomments tables
Execution (first time or recompile)
Optimization
Compilation
Compiled plan placed in procedure cache

Benefits and Drawbacks Stored Proc

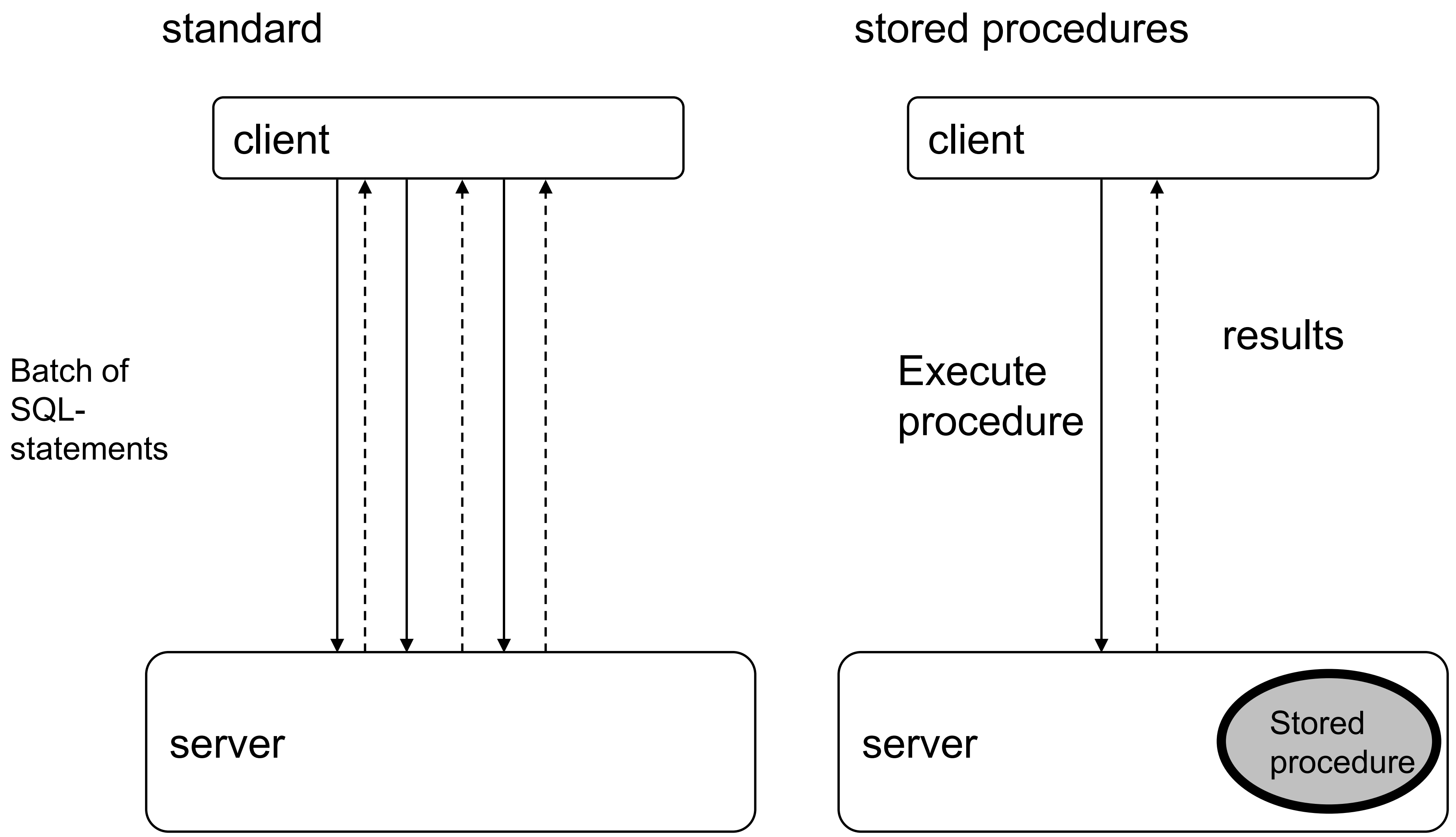
Benefits

- Faster execution (Improve Performance)
  - Precompiled and optimized
- Reduced server/client network traffic
- Restricted, function-based access to tables (Provide Security Mechanisms)
- Reuse of Code
- Easier maintenance
- Automation of complex transactions
- Share Application Logic
- Shield Database Schema Details

Drawbacks

- Non-standard
  - not portable across platforms
  - no standard way to pass or describe the parameters
  - no good support by tools
- Complex coding
- Performance may be poor if the execution plan is not refreshed

Stored Procedures vs SQL



Creating, executing and ModifyingStored Procedures

- **Create** :

  CREATE PROC[EDURE] *procedure_name* [**;** *number*]

  [*@parameter data_type* [,*@parameter data_type*] [**=** *default* [OUTPUT]]

  [WITH RECOMPILE] | ENCRYPTION] | RECOMPILE **,** ENCRYPTION]
- **Execute** :Execute procedure_name[parameter 1,…….]
- **Modifying** : Alter procedure ……
- Use sp_help or sp_helptext to Display Information
- Example Create:

  Create procedure contoh_sp

  As

  *Select * from Product*
- Example Drop : DROP PROCEDURE *procedure name*

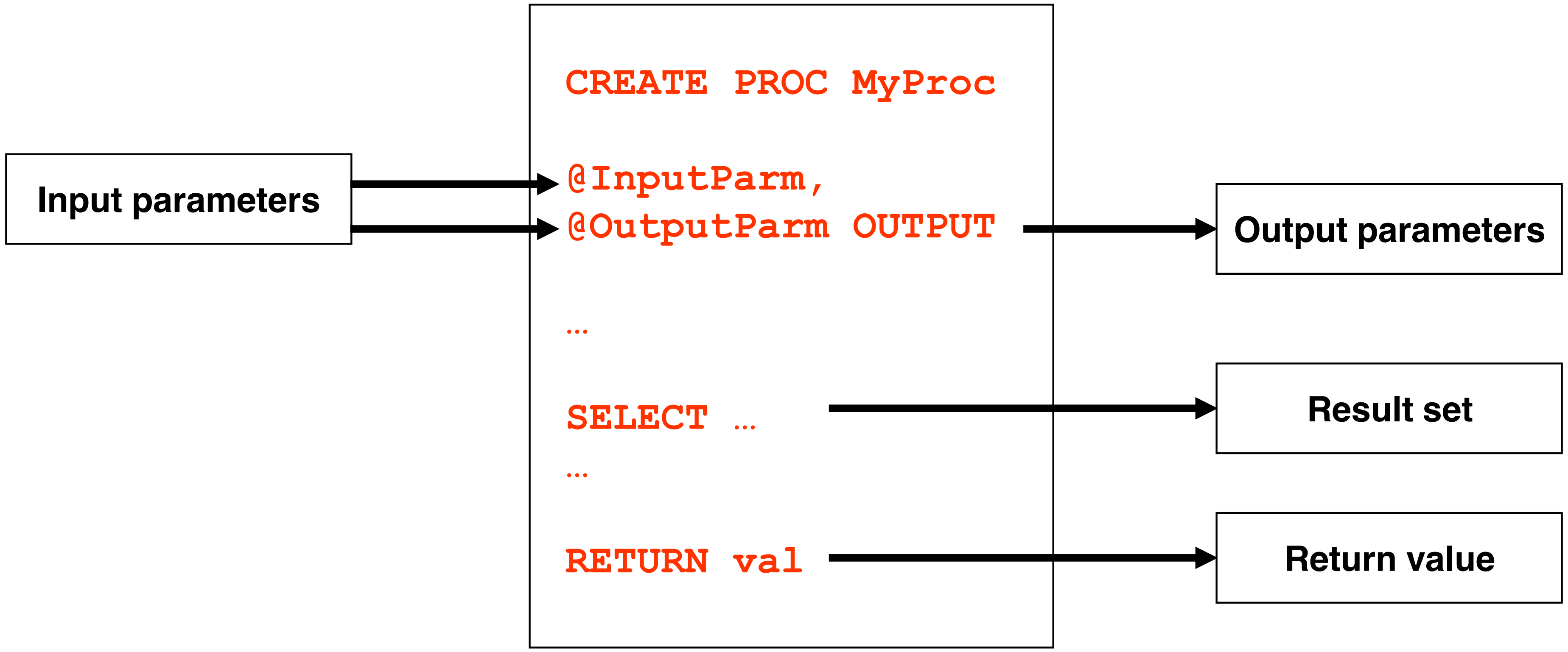
Guidelines for Creating Stored Procedures

- dbo User Should Own All Stored Procedures
- Create, Test, and Debug on Server
- Avoid sp_ Prefix in Stored Procedure Names
- Minimize Use of Temporary Stored Procedures
- Input parameters allow information to be passed into a stored procedure. To define a stored procedure that accepts input parameters, you declare one or more variables as parameters in the CREATE PROCEDURE statement.

Guidelines for Creating Stored Procedures

- The maximum number of parameters in a stored procedure is 1024.
- Parameters are local to a stored procedure. The same parameter names can be used in other stored procedures.

Input Parameters and Information returned



Using Input Parameters

```
Create Proc Pname
@myname varchar(20)  = Alice
as
print 'My Name is' + ' ' + @myname
```

Step 1

```
Exec Procedure_Name [Parameter]
```

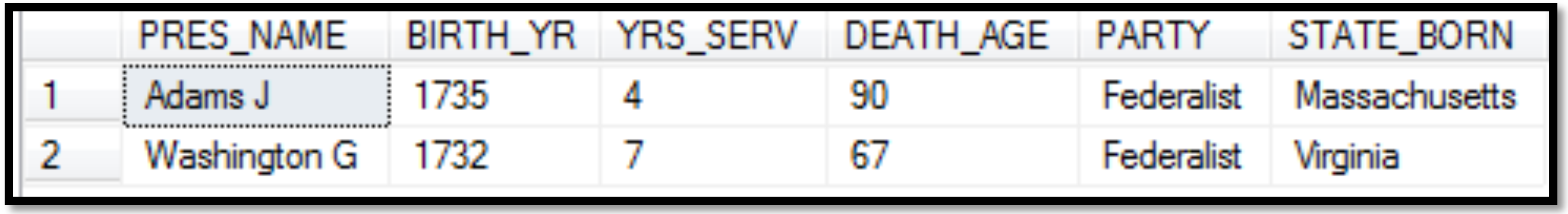
Step 2

Exec pname Alice	My Name is **Alice**
Exec pname ‘Alice O Leary’	My Name is **Alice O Leary**
Exec pname	My Name is **Alice**
Pname	My Name is **Alice**

Example

```
create proc pres_proc
@party as varchar(15)
as
select * from PRESIDENT
where PARTY=@party

exec pres_proc 'Federalist'
```



	PRES_NAME	BIRTH_YR	YRS_SERV	DEATH_AGE	PARTY	STATE_BORN
1	Adams J	1735	4	90	Federalist	Massachusetts
2	Washington G	1732	7	67	Federalist	Virginia

Example Input Parameters, with Default

```
CREATE PROC spEmployee
    @LastName nvarchar(50) = NULL     -- Default NULL
AS
BEGIN
  IF @LastName IS NULL                -- EXEC spEmployee
    SELECT * FROM HumanResources.Employee
  ELSE                                -- EXEC spEmployee ‘A’
    SELECT c.LastName, c.FirstName, e.*
    FROM Person.Contact c
         INNER JOIN HumanResources.Employee e
           ON c.ContactID = e.ContactID
    WHERE c.LastName LIKE @LastName + '%'
END
```

Executing Stored Procedures with Input Parameters

- Passing Values by Reference

```
EXEC addadult
  @firstname = 'Linda',
  @lastname = 'LaBrie',
  @street = 'Dogwood Drive',
  @city = 'Sacramento',
  @state = 'CA',
  @zip = '94203'
```

- Passing Values by Position

```
EXEC addadult 'LaBrie', 'Linda', null,
'Dogwood Drive', 'Sacramento', 'CA','94203', null
```

Updating Data

- UPDATE statement
- NOCOUNT option : When SET NOCOUNT is ON, the count is not returned.

```
CREATE PROCEDURE p_UpdateCategory
(
      @CategoryID int = null,
      @CategoryName varchar(50)
)
AS
      SET NOCOUNT ON
      UPDATE Categories
      SET Category = @CategoryName
      WHERE CategoryID = @CategoryID
```

Inserting Data

- INSERT Statement

```
CREATE PROCEDURE p_InsertCustomer
(
        @FName varchar(50),
        @LName varchar(50)
)
AS
        SET NOCOUNT ON
        INSERT INTO Customers (FirstName, LastName)
        VALUES (@FName, @LName)
```

Deleting Data

- DELETE Statement

```
CREATE PROCEDURE p_DeleteCategory
(
	@CategoryID int = null
)
AS
	SET NOCOUNT ON
	DELETE FROM Categories
	WHERE CategoryID = @CategoryID
```

Returning Values with Output Parameters

Creating Stored Procedure

```
CREATE PROCEDURE mathtutor
      @m1 smallint,
      @m2 smallint,
      @result smallint OUTPUT
AS
      SET @result = @m1 * @m2
```

Executing Stored Procedure

```
DECLARE @answer smallint
EXECUTE mathtutor 5, 6, @answer OUTPUT
SELECT 'The result is: ' , @answer
```

Results of Stored Procedure

```
The result is:  30
```

OUTPUT Parameter

Stored procedures can return information to the calling stored procedure or client with output parameters (variables designated with the OUTPUT keyword).

By using output parameters, any changes to the parameter that result from the execution of the stored procedure can be retained, even after the stored procedure completes execution.

To use an output parameter, the OUTPUT keyword must be specified in both the CREATE PROCEDURE and EXECUTE statements.

If the keyword OUTPUT is omitted when the stored procedure is executed, the stored procedure still executes, but it does not return a value. i.e. Shows NULL .

Use TRY/CATCH Blocks for error handling

```
BEGIN TRY
  CREATE TABLE OurIfTest (Col1 int PRIMARY KEY)
END TRY
BEGIN CATCH
  DECLARE      @ErrorNo       int,
               @Message       nvarchar(4000)
  SELECT
      @ErrorNo        = ERROR_NUMBER(),
      @Message        = ERROR_MESSAGE()
  IF @ErrorNo = 2714
      PRINT 'WARNING: Skipping CREATE as table already exists.'
  ELSE
      RAISERROR (@Message, 16, 1)
END CATCH
```

Debugging Stored Procedure

- Print statements
- Using temporary tables
- Execute parts of SQL separately
- Debugger SQL Server

Thank You