Stored procedure and function

Presented by:
Parmonangan R. Togatoro

STORED PROCEDURE (SP) VS. USER DEFINE FUNCTION (UDF)

- **A function is a subprogram written to perform certain computations**
- **A scalar function returns only a single value (or NULL), whereas a table function returns a (relational) table comprising zero or more rows, each row with one or more columns.**
- **Functions must return a value (using the RETURN keyword), but for stored procedures this is not compulsory.**
- **Stored procedures can use RETURN keyword but without any value being passed.**

STORED PROCEDURE (SP) VS. USER DEFINE FUNCTION (UDF)

- **Functions could be used in SELECT statements, provided they don't do any data manipulation. However, procedures cannot be included in SELECT statements.**
- **A function can have only IN parameters, while stored procedures may have OUT or INOUT parameters.**
- **A stored procedure can return multiple values using the OUT parameter or return no value at all.**

USER DEFINED FUNCTION

- **UDF:**
  - a body of T-SQL statements
  - pre-compiled and pre-optimized
  - works as a single unit
  - can perform in-line to a query

- **Two types:**
  - those that return a scalar value
  - those that return a table

So..When the developers use UDF rather than SP??

UDF RETURNING A SCALAR

```
CREATE FUNCTION DayOnly (@date DATETIME)
  RETURNS varchar(10)
AS
BEGIN
  RETURN CONVERT(VARCHAR(10), @date, 101)
END
```

```
SELECT dbo.DayOnly(GETDATE()) AS Today
```

```
Results:

----------
03/15/2010
```

SCALAR UDF MUST BE DETERMINISTIC

- **Must return the same value for the same input parameters**

```
CREATE FUNCTION fnRandomInt(@max INT)
RETURNS INT
AS
BEGIN
      RETURN CEILING(@max*RAND())
END
```

```
Msg 443
Invalid use of a side-effecting
operator 'rand' within a function.
```

IMPLEMENTING 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

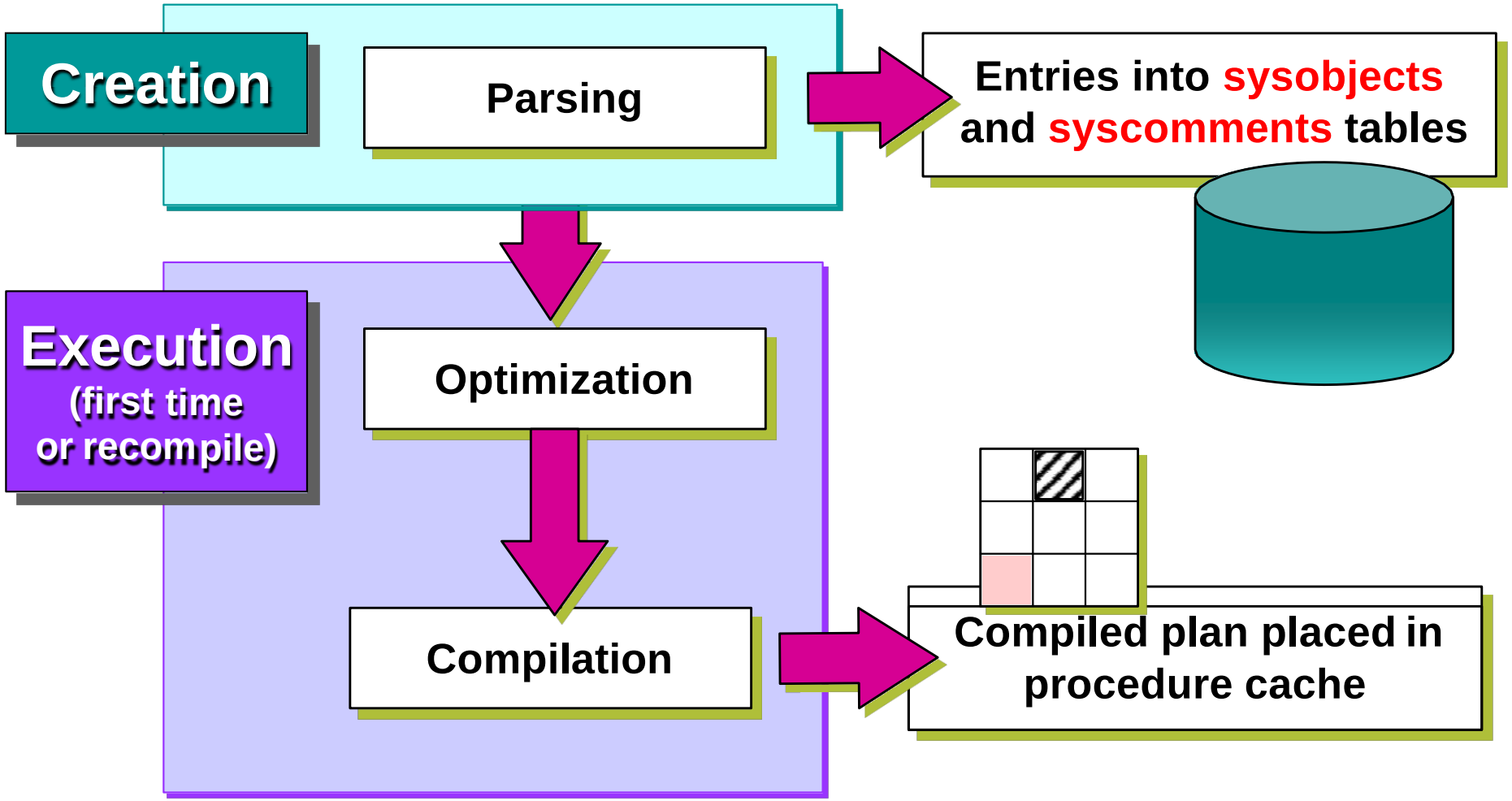
  Collection of:

  - 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



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

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*

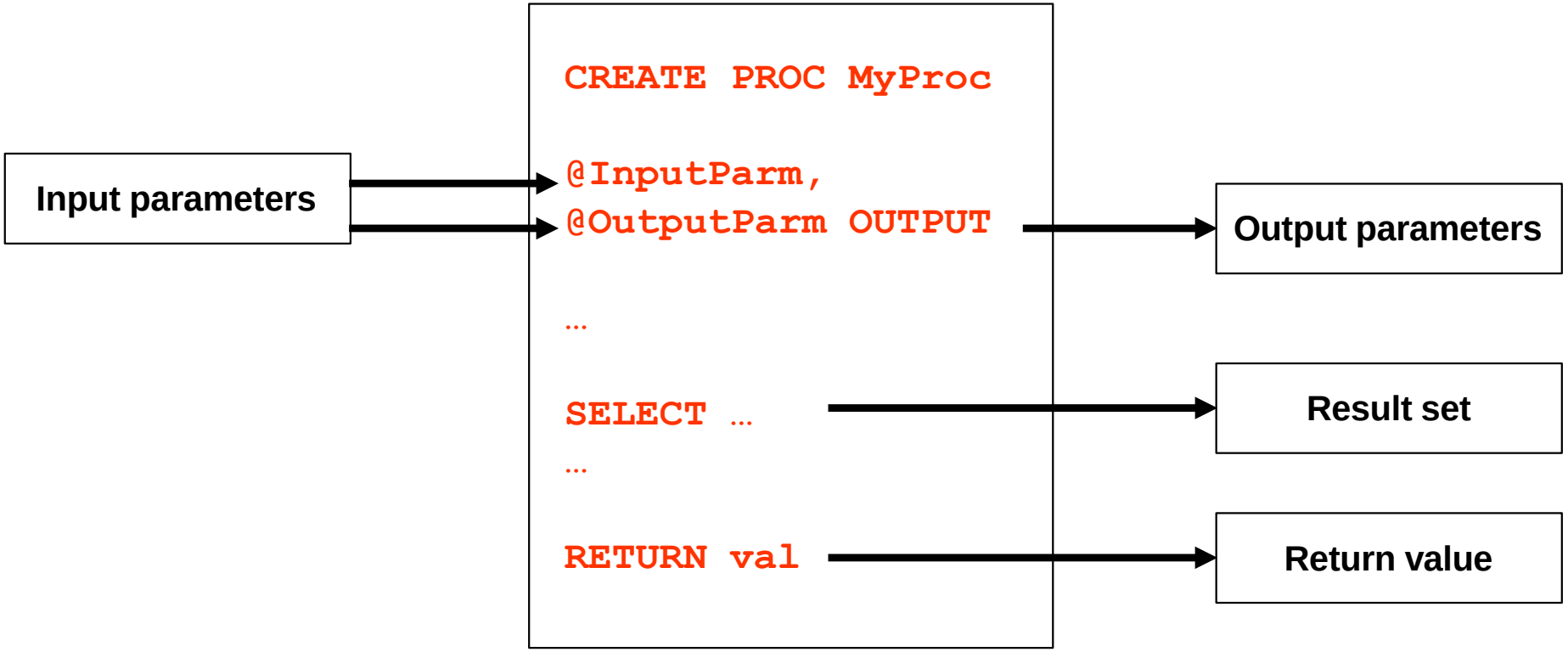
GUIDELINES FOR CREATING STORED PROCEDURES

- 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

Exec Procedure_Name [Parameter]
```

Step 1

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)
)
A
S
        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
)
A
S
        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