SQL Server 2012 – Database Development

Lesson 3: Procedures and Functions

Capgemini

Lesson Objectives

- Database programming
- Creating, Executing, Modifying, and Dropping Stored Procedures and Functions
- Implementing Exception Handling

Overview

- Introduction to Stored Procedures
- Creating, Executing, Modifying, and Dropping Stored Procedures
- Using Parameters in Stored Procedures
- Executing Extended Stored Procedures
- Handling Error Messages

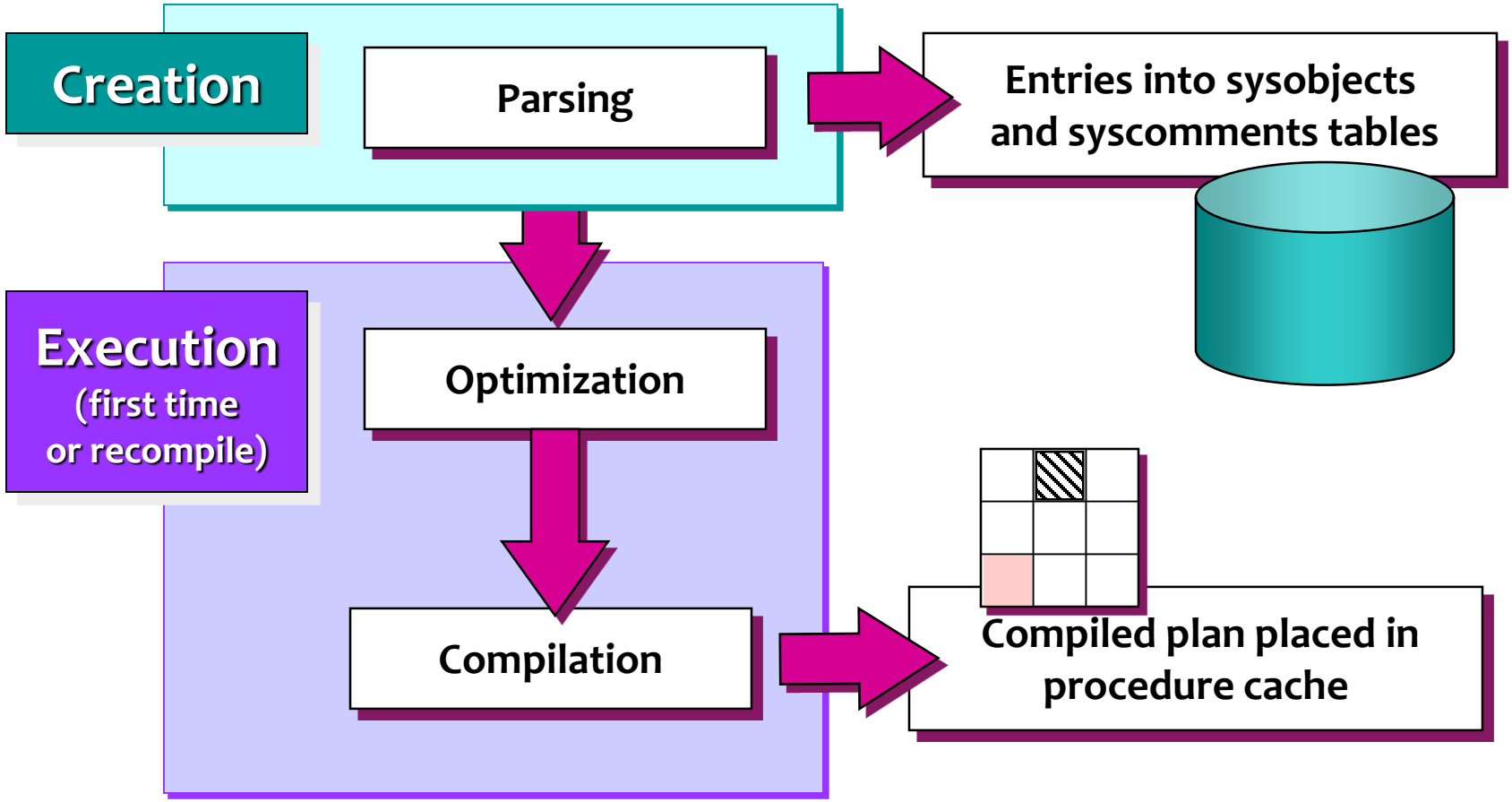
Definition

- Named Collections of pre compiled Transact-SQL Statements
- Stored procedures can be used by multiple users and client programs leading to reuse of code
- Abstraction of code and better security control
- Reduces network work and better performance
- Can accept parameters and return value or result set

Types

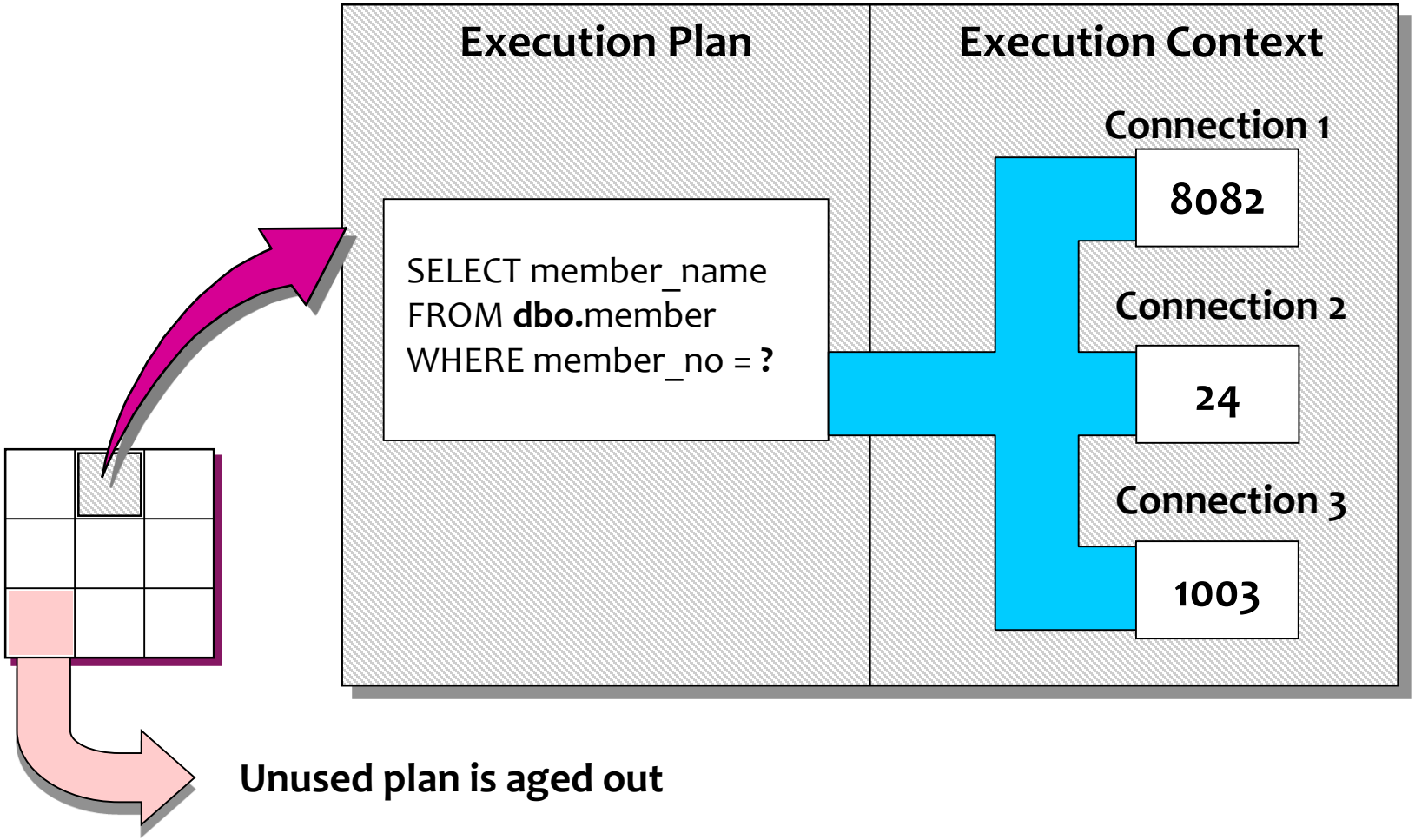
- T-SQL supports the following types of procedure
  - System -
    - Procedures pre-built in SQL Server itself
    - Available in master database
    - Name starts with sp_
- Temporary
  - – name starts with #(Local) or ## (Global) and stored in tempdb
  - Available only for that session
- Extended
  - execute routines written in programming languages like C,C++,C# or VB.NET
  - May have names starting with xp_

Initial Processing of Stored Procedures



Subsequent Processing of Stored Procedures

Execution Plan Retrieved



Advantages

- Share Application Logic across multiple clients
- Shield Database Schema Details (Abstraction)
- Provide Security Mechanisms
- Reduce Network Traffic
- Improve Performance

CREATE PROCEDURE Statement

- Syntax

```
CREATE { PROC | PROCEDURE } [schema_name.]
procedure_name     [ { @parameter data_type }        [
VARYING ] [ = default ] [ [ OUT [ PUT ]     ] [ ,...n ]
 AS
{ <sql_statement> [;][ ...n ] |
<method_specifier> } [;]
Return  <value>
```

Example

- Code Snippet

```
USE AdventureWorks2012;
GO
 CREATE PROCEDURE HumanResources.uspGetEmployeesTest2
@LastName nvarchar(50),
 @FirstName nvarchar(50) AS
 SET NOCOUNT ON;
SELECT FirstName, LastName, Department FROM
HumanResources.vEmployeeDepartmentHistory WHERE FirstName =
@FirstName AND LastName = @LastName AND EndDate IS NULL;
 GO
```

Executing Stored Procedures

- Code Snippet

```
EXECUTE HumanResources.uspGetEmployeesTest2 N'Ackerman', N'Pilar';
-- Or
EXEC HumanResources.uspGetEmployeesTest2 @LastName = N'Ackerman',
@FirstName = N'Pilar';
GO
-- Or
EXECUTE HumanResources.uspGetEmployeesTest2 @FirstName = N'Pilar',
@LastName = N'Ackerman';
GO
```

Note : You need to have execute permission for the procedure to execute it

Altering and Dropping Procedures

- Altering Stored Procedures
  - Include any options in ALTER PROCEDURE
  - Does not affect nested stored procedures

```
ALTER { 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 ]
```

DROP PROCEDURE <stored procedure name>;

Demo

- Creating Stored Procedures

Stored Procedures Using Parameters

- Stored procedures can take parameters OR arguments and return value
- Parameters can of the following type
- INPUT
  - Default Type
  - IN or INPUT keyword is used to define variables of IN type
  - Used to pass a data value to the stored procedure
- OUTPUT
  - Allow the stored procedure to pass a data value or a back to the caller.
  - OUT keyword is used to identify output paramter

Stored Procedures Using Parameters

```
CREATE PROCEDURE usp_ProductCountByCategory (
     @i_catid  INT ,
     @o_Prodcount INT OUT
  )
AS
BEGIN
    IF @i_catid is NULL OR @i_catid < 0
       return -1
    SELECT  @o_Prodcount=count(ProductID) from Products
     WHERE CategoryID=@i_catid
END
```

- To Execute

```
DECLARE  @prodcount  INT
EXEC  usp_ProductCountByCategory  1234, @prodcount  OUT
```

Returning a Value from Stored Procedures

- Values can be returned from stored procedure using the following options
  - OUTPUT parameter
    - More than 1 parameter can be of type OUTPUT
  - Return statement
    - Used to provide the execution status of the procedure to the calling program
    - Only one value can be returned
    - to -99 are reserved for internal usage , one can return customized values also
- Return value can be processed by the calling program as

  exec @return_value = <storedprocname>

WITH RESULT SETS

- In earlier versions of SQL server when we wished to change a column name or datatype in the resultset of a stored procedure, all the references needed to be changed. There was no simple way to dump the output of a stored procedure without worrying about the column names and data types.
- The EXECUTE statement has been extended in SQL Server 2012 to include the WITH RESULT SETS option. This allows you to change the column names and data types of the result set returned in the execution of a stored procedure

Recompiling Stored Procedures

- Stored Procedures are recompiled to optimize the queries which makes up that Stored Procedure
- Stored Procedure needs recompilation when
  - Data in underlying tables are changed
  - Indexes are added /removed in tables
- Recompilation can be done by Using
  - CREATE PROCEDURE [WITH RECOMPILE]
  - EXECUTE [procedure]WITH RECOMPILE]
  - sp_recompile [procedure]

To View the Definition of Stored Procedure

- To view the definition of a procedure in Query Editor
  - EXEC sp_helptext N'AdventureWorks2012.dbo.uspLogError';
- To view the definition of a procedure with System Function: OBJECT_DEFINITION
  - SELECT OBJECT_DEFINITION (OBJECT_ID(N'AdventureWorks2012.dbo.uspLogError'));
  - Change the database name and stored procedure name to reference the database and stored procedure that you want.

Guidelines

- One Stored Procedure for One Task
- Create, Test, and Troubleshoot
- Avoid sp_ Prefix in Stored Procedure Names
- Use Same Connection Settings for All Stored Procedures
- Minimize Use of Temporary Stored Procedures

Error Handling in Procedures

- SQL Server 2005 onwards error handling can be done with
  - TRY .. CATCH blocks
  - @@ERROR global variable
- If a statements inside a TRY block raises an exception then processing of TRY blocks stops and is then picked up in the CATCH block
- The syntax of the TRY CATCH is

```
BEGIN TRY
        --- statements
END TRY
BEGIN CATCH
  --- statements
 END CATCH
```

Error Handling

Using @@Error

```
DECLARE  @v_deptcode int
DECLARE @v_deptname
varchar(10)
DECLARE @errorcode int

set @v_deptcode=10
set @v_deptname='Pre sales'

insert into dept
values(@v_deptcode,'Pre sales')

set  @errorcode = @@ERROR
if @errorcode > 0
begin
        print 'error'
        print @errorcode
end
else
    print 'added successfully'
```

Using TRY ..CATCH

```
DECLARE  @v_deptcode int
DECLARE @v_deptname
varchar(10)
DECLARE @errorcode int

set @v_deptcode=10
set @v_deptname='Pre sales'

BEGIN TRY
    insert into dept
values(@v_deptcode,'Pre sales')
END TRY

BEGIN CATCH
   PRINT 'An error occurred while
inserting
   PRINT ERROR_NUMBER()

END CATCH
```

Error Handling using RAISEERROR

- RAISERROR can be used to
  - Return user defined or system messages back to the application
  - Assign a specific error number , severity and state to a message
- Can be associated to a Query or a Procedure
- Has the following syntax
- RAISERROR (message ID | message str),severity, state
- Message ID has to be a number greate than 50,000
- Can be used along with TRY ..CATCH /other error handling mechanisms

Example of Raisererror with TRY ..CATCH

```
CREATE  Procedure usp_updateprodprice
      @i_vcategory  int,
       @i_vpriceinc   money
As
BEGIN
         if @i_vcategory  is NULL or @i_vcategory <=0
          begin
             raiserror (50001, 1,1)
            return
          end
        if  @i_vpriceinc  <= 0
          begin
             raiserror  (50002, 1,1)
            return
          end
```

Example of Raisererror with TRY ..CATCH

```
if  not exists( SELECT 'a' FROM Categories
            WHERE  CategoryID = @i_vcategory)
          begin
             raiserror  (50003,1,1)
               return
          end
        BEGIN TRY
              insert into revised_product
             select ProductID,ProductName,
unitPrice,@i_vcategory,unitPrice+unitPrice*@i_vpriceinc
                 FROM Products where CategoryID=@i_vcategory

           return
       END TRY

       BEGIN CATCH
          raiserror  (50004,1,1)
          rollback tran
         -- return -1
      END CATCH
END
```

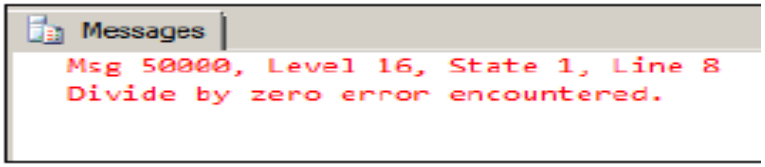
THROW Statement

- Exception handling is now made easier with the introduction of the THROW statement in SQL Server 2012.
- In previous versions, RAISERROR was used to show an error message.

Difference Between RaiseError and Throw

```
BEGIN TRY
DECLARE @MyInt int
SET @MyInt = 1 / 0
END TRY
BEGIN CATCH
DECLARE @ErrorMessage nvarchar(4000),
@ErrorSeverity int
SELECT @ErrorMessage =
ERROR_MESSAGE(),
@ErrorSeverity = ERROR_SEVERITY()
RAISERROR (@ErrorMessage,
@ErrorSeverity, 1)
END CATCH
```


```
BEGIN TRY
DECLARE @MyInt int
SET @MyInt = 1/0
END TRY
BEGIN CATCH
-- throw out the error
THROW
END CATCH
```



Advantages of THROW:

- THROW has now made the developer's life much easier and developers can now code independent of the Tester's input on the exception message.
- It can be used in a TRY..CATCH block.
- No restrictions on error message number to exist in sys.messages.

Best Practices

- Verify Input Parameters
- Design Each Stored Procedure to Accomplish a Single Task
- Validate Data Before You Begin Transactions
- Use the Same Connection Settings for All Stored Procedures
- Use WITH ENCRYPTION to Hide Text of Stored Procedures

Demo

- Stored Procedures

Definition

- Named Collections of Transact-SQL Statements
- Takes parameter and returns a single value
- Can be used as a part of expression

Note : Table data types are also singular value

Differences: Stored Procedure and Functions

Procedures	Function
Return single integer value represents return status	Return single value of any scalar data type supported by SQL server or Table type
Use execute statement to execute stored procedure	Can be called through select statement if it returns scalar value otherwise can be called through from statement if it returns table.
Use output parameter to pass values to caller	Use return statement to pass values to caller

Types of User-defined Function

- Scalar Functions
  - Similar to a built-in function
- Multi-Statement Table-valued Functions
  - returns a defined table as a result of operations
- In-Line Table-valued Functions
  - Returns a table value as the result of single SELECT statement

Creating a User-defined Function

```
CREATE Function  udf_GetProductcategory (@i_prodID  INT)
RETURNS  nvarchar(40)
as
 BEGIN
     declare @retvalue  nvarchar(40)
      if  @i_prodID is NULL or @i_prodID <= 0
          return null
     SELECT @retvalue=CategoryName From
      PRODUCTS , CATEGORIES
      WHERE PRODUCTS.CategoryID = CATEGORIES.CategoryID
       AND  PRODUCTS.ProductID=@i_prodID
        return @retvalue
END
```

Restrictions

- Restrictions on Functions
  - A function can return only single value at a time
  - The SQL statements within a function cannot include any nondeterministic system functions.
    - E.g getdate() function is nondeterministic hence cannot be used inside function but can be pass as argument

Creating a Function with Schema Binding

- Schema binding prevents the altering or dropping of any object on which the function depends.
- If a schema-bound function references TableA, then columns may be added to TableA, but no existing columns can be altered or dropped, and neither can the table itself.
- Schema binding not only alerts the developer that the change may affect an object, it also prevents the change. To remove schema binding so that changes can be made, ALTER the function so that schema binding is no longer included.

Altering and Dropping User-defined Functions

- Altering Functions

```
ALTER FUNCTION dbo.fn_NewRegion
<New function content>
```

  - Retains assigned permissions
  - Causes the new function definition to replace existing definition

- Dropping Functions

```
DROP FUNCTION dbo.fn_NewRegion
```

Using a Scalar User-defined Function

- RETURNS Clause Specifies Data Type
- Function Is Defined Within a BEGIN and END Block
- Return Type Is Any Data Type Except text, ntext, image, cursor, or timestamp

Example

- Notes Pages

In-Line Table-valued Functions

- An inline table-valued user-defi ned function retains the benefi ts of a view, and adds parameters.
- The inline table-valued user-defi ned function has no BEGIN/END body.

- **CREATE FUNCTION FunctionName (InputParameters)**
  RETURNS Table
  AS
  RETURN (Select Statement);.

Multi-Statement Table-valued Functions

- The multistatement table-valued, user-defi ned function combines the scalar function'scapability to contain complex code with the inline table-valued function's capability to return a result set. This type of function creates a table variable and then populates it within code. The table is then passed back from the function so that it may be used within SELECT statements.

```
CREATE FUNCTION FunctionName (InputParamenters)
RETURNS @TableName TABLE (Columns)
AS
BEGIN;
Code to populate table variable
RETURN;
END;
```

Refer to example In Notes Pages

View Definition of a Function

```
SELECT definition, type
FROM sys.sql_modules AS m
JOIN sys.objects AS o ON m.object_id = o.object_id
AND type IN ('FN', 'IF', 'TF');
GO
```

Types of Functions

- Scalar user-defi ned functions return a single value and must be deterministic.
- Inline table-valued user-defi ned functions are similar to views and return the results of a single SELECT statement.
- Multistatement, table-valued, user-defi ned functions use code to populate a table variable, which is then returned.

Best Practices

- Choose inline table-valued functions over multistatement table-valued functions whenever possible.
- Even if it looks like you need a scalar function, write it as an inline table-valued function avoid scalar functions wherever possible.
- If you need a multistatement table-valued function, check to see if a stored procedure might be the appropriate solution. This might require a broader look at query structure, but it’s worth taking the time to do it.

Demo

- Creating User-defined Functions

Summary

- In this lesson, you have learnt:
- Creating, Executing, Modifying, and Dropping Stored Procedures
- Using Parameters in Stored Procedures
- Using User defined Functions
  - Scalar User-defined Function
  - Multi-Statement Table-valued Function
  - In-Line Table-valued Function

Review Question

- Question 1: A stored procedure can return a single integer value
  - True
  - False
- Question 2: ---------- stored procedures call subroutines written in languages like c, c++,.NET
- Question 3: ---------- function includes only one select statement