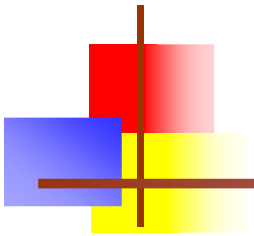
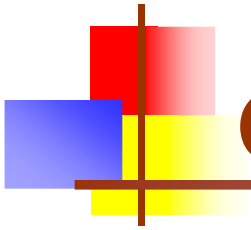
Thang Long University



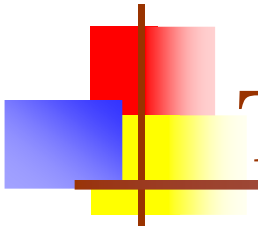
T-SQL Programming (Phần 2)

Trần Quang Duy



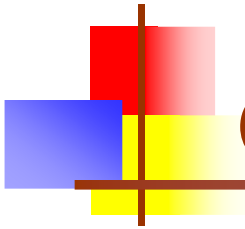
Content

- Stored Procedures
  - Types of Stored Procedures
  - Working Stored Procedures
  - Execute As
  - Resolution
  - Recursion
- User-Defined Functions (UDFs)
  - Scalar UDFs
  - Table-valued UDFs
  - UDFs with Schemabinding



Types of Stored Procedures

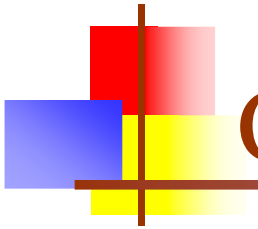
- User-defined Stored Procedures
  - T-SQL
    - Temporary Stored Procedures
    - Global Temporary Stored Procedures
  - CLR
- System Stored Procedures
- Extended Stored Procedures



Create Stored Procedures

- Stored Procedures

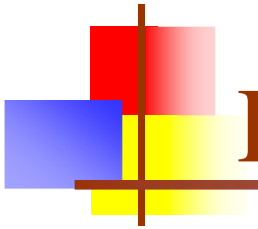
```
CREATE PROCEDURE|PROC <sproc name>
[<parameter name> [schema.]<data type> [VARYING] [=
  <default value>] [OUT[PUT]][,
<parameter name> [schema.]<data type> [VARYING] [=
  <default value>]
[OUT[PUT]][,
...
]]
[WITH
 RECOMPILE| ENCRYPTION |
[EXECUTE AS { CALLER|SELF|OWNER|<'user name'>}]
[FOR REPLICATION]
AS
<code>
```



Create Stored Procedures - Example

- Basic Example

```
CREATE PROC spShippers
AS
SELECT * FROM Shippers
```



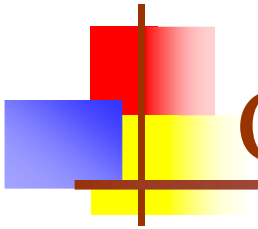
Execute Stored Procedures

- Execute Stored Procedures
  - Syntax:

```
EXECUTE | EXEC SPName
```

  - Example

```
EXECUTE spShippers
--or
EXECUTE spShippers
--or
spShippers
```



Change/Drop Stored Procedures

- Change Stored Procedures
  - Syntax:

```
ALTER PROCEDURE|PROC SPName[paralist]
```

  - Example

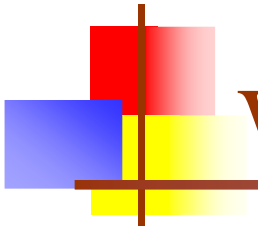
```
ALTER PROC spShippers
AS
SELECT * FROM Shippers
```

- Drop Stored Procedures
  - Syntax:

```
DROP PROCEDURE|PROC SPName
```

  - Example

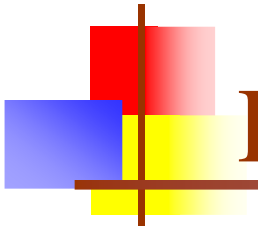
```
DROP PROC spShippers
```



Working with Parameters

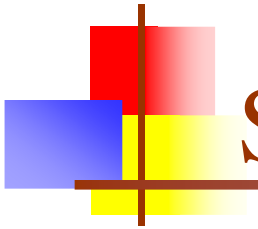
- Declaring Parameters

  @parameter_name [AS] datatype [= default|NULL]
  [VARYING] [OUTPUT|OUT]



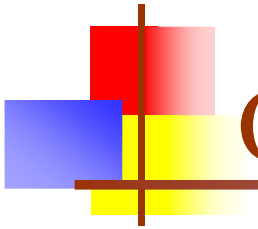
Parameters

```
CREATE PROC dbo.usp_GetCustOrders
  @custid   AS NCHAR(5),
  @fromdate AS DATETIME,
  @todate   AS DATETIME
AS
SELECT OrderID, CustomerID, EmployeeID, OrderDate
FROM dbo.Orders
WHERE CustomerID = @custid
  AND OrderDate >= @fromdate
  AND OrderDate < @todate;
--
EXEC dbo.usp_GetCustOrders N'ALFKI', '01-01-1997',
  '12-31-1999'
```



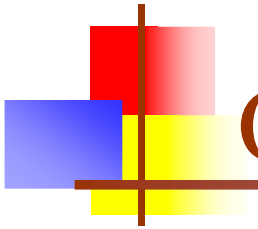
Supplying Default Values

```
ALTER PROC dbo.usp_GetCustOrders
  @custid   AS NCHAR(5),
  @fromdate AS DATETIME = '19000101',
  @todate   AS DATETIME = '99991231'
AS
SET NOCOUNT ON;
SELECT OrderID, CustomerID, EmployeeID, OrderDate
FROM dbo.Orders
WHERE CustomerID = @custid
  AND OrderDate >= @fromdate
  AND OrderDate < @todate;
GO
EXEC dbo.usp_GetCustOrders N'ALFKI'
```



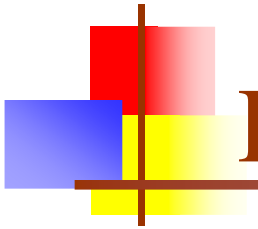
Output Parameters

```
ALTER PROC dbo.usp_GetCustOrders
  @custid   AS NCHAR(5),
  @fromdate AS DATETIME = '19000101',
  @todate   AS DATETIME = '99991231',
  @numrows  AS INT OUTPUT
AS
SET NOCOUNT ON
SELECT OrderID, CustomerID, EmployeeID, OrderDate
FROM dbo.Orders
WHERE CustomerID = @custid
  AND OrderDate >= @fromdate
  AND OrderDate < @todate;
SELECT @numrows = @@rowcount;
GO
```



Output Parameters

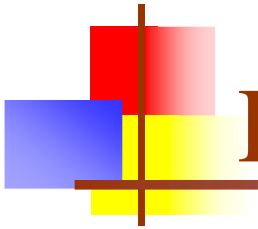
```
DECLARE  @mynumrows AS INT;
EXEC dbo.usp_GetCustOrders
  @custid   = N'ALFKI',
  @fromdate = '19970101',
  @todate   = '19980101',
  @numrows  = @mynumrows OUTPUT;
SELECT @mynumrows AS rc;
```



Return Statement

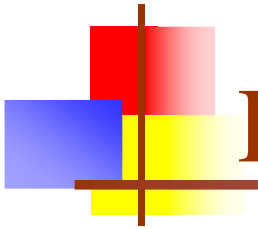
- Return Statement: to return the value
- Example

```
ALTER PROC dbo.usp_GetCustOrders
   @custid   AS NCHAR(5),
   @fromdate AS DATETIME = '19000101',
   @todate   AS DATETIME = '99991231',
   @numrows  AS INT OUTPUT
AS
SET NOCOUNT ON
DECLARE @err INT
SELECT OrderID, CustomerID, EmployeeID, OrderDate
FROM dbo.Orders
WHERE CustomerID = @custid
  AND OrderDate >= @fromdate
  AND OrderDate < @todate;
SELECT @numrows = @@rowcount; @err = @@error
Return @err
GO
```



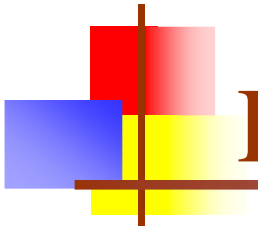
Return Statement

```
DECLARE @myerr AS INT, @mynumrows AS INT;
EXEC @myerr = dbo.usp_GetCustOrders
  @custid   = N'ALFKI',
  @fromdate = '19970101',
  @todate   = '19980101',
  @numrows  = @mynumrows OUTPUT;
SELECT @myerr AS err, @mynumrows AS rc;
```



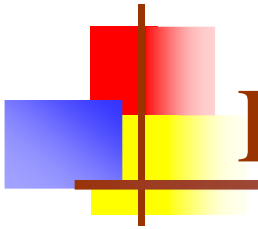
Resolution

- When create a stored procedure
  - SQL Server first parses the code to check for syntax errors.
  - The resolution process verifies the existence of object and column names, among other things.
  - If the referenced objects exist, the resolution process will take place fully—that is, it will also check for the existence of the referenced column names.
- Deferred name resolution
  - This process of postponing name resolution until run time is called *deferred name resolution*



Execute As

- EXECUTE permissions
  - Can grant EXECUTE permissions on the stored procedure without granting them direct access to the underlying objects
- To avoid requiring direct permissions from the caller,
  - The stored procedure and the underlying objects belong to the same schema.
  - The activity is static (as opposed to using dynamic SQL).
  - The activity is DML (SELECT, INSERT, UPDATE, or DELETE), or it is an execution of another stored procedure.



Execute As

- Example

```
DENY SELECT ON Shippers TO user1;
GRANT EXECUTE ON spShippers TO user1;
```

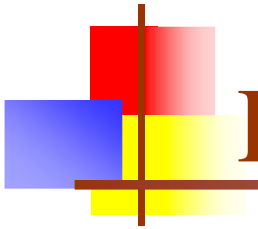
User1:

```
EXEC spShippers → OK
```

```
ALTER PROC spShippers
AS
BEGIN
   CREATE TABLE ShippersName (ID int, Company
   nvarchar(40));
   INSERT INTO ShippersName SELECT ShipperID, CompanyName
   FROM Shippers
   DROP TABLE ShippersName
END
```

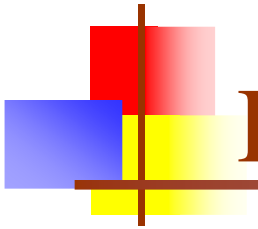
User1:

```
EXEC spShippers→ Fail
```



Execute As

- EXECUTE AS options:
  - **CALLER (default):** Security context of the caller
  - **SELF:** Security context of the user creating or altering the stored procedure
  - **OWNER:** Security context of the owner of the stored procedure
  - **'user_name‘:** Security context of the specified user name

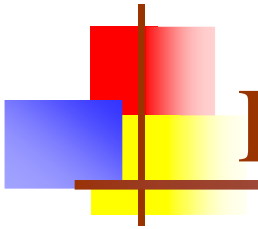


Execute As

```
ALTER PROC spShippers
WITH EXECUTE AS SELF
AS
BEGIN
  CREATE TABLE DBO.ShippersName (ID int, Company
  nvarchar(40));
  INSERT INTO ShippersName SELECT ShipperID,
  CompanyName FROM Shippers
  DROP TABLE DBO.ShippersName
END
```

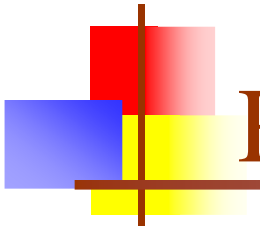
User1:

EXEC spShippers → Successed



Recursion

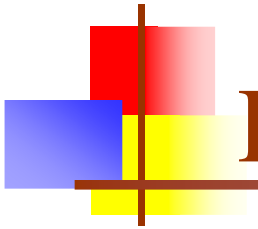
- Recursion? a piece of code calls itself.
- In SQL 2005
  - The 32 level recursion limit



Recursion

- Example: n!

```
CREATE PROC spFactorial @ValueIn int, @ValueOut int
  OUTPUT
AS
DECLARE @InWorking int
DECLARE @OutWorking int
IF @ValueIn != 1
BEGIN
  SELECT @InWorking = @ValueIn - 1
  EXEC spFactorial @InWorking, @OutWorking OUTPUT
  SELECT @ValueOut = @ValueIn * @OutWorking
END
ELSE
BEGIN
  SELECT @ValueOut = 1
END
RETURN
```



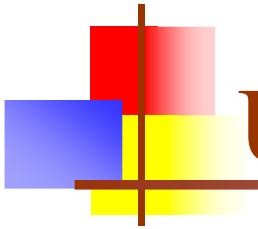
Recursion

- Input

```
DECLARE @WorkingOut int
DECLARE @WorkingIn int
SELECT @WorkingIn = 5
EXEC spFactorial @WorkingIn, @WorkingOut
  OUTPUT
PRINT CAST(@WorkingIn AS varchar) + '
  factorial is ' + CAST(@WorkingOut AS
  varchar)
```

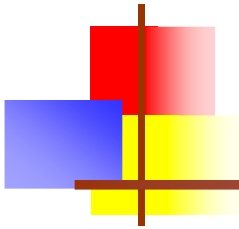
- OutPut

```
5 factorial is 120
```



User Defined Functions (UDF)

- User Defined Functions (UDF)
  - Queries
  - Computed columns
  - Constraints
- Types of UDFs
  - Scalar UDF: return a scalar value
  - Table-Valued UDFs: return a table



CREATE FUNCTION

- Syntax

CREATE FUNCTION [<schema name>.]<function name>

([<@parameter name> [AS] [<schema name>.]<scalar data type> [= <default

value>] [,...n]])

RETURNS {<scalar type>|TABLE [(<Table Definition>)]}

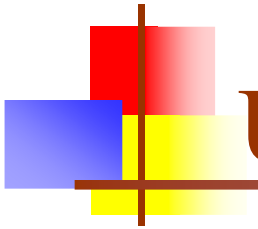
[WITH [ENCRYPTION]|[SCHEMABINDING]|CALLER|SELF|OWNER|<'user name'>}]]

BEGIN

[<function statements>]

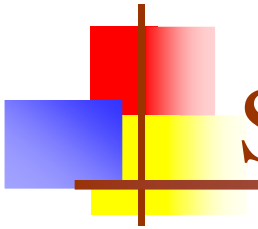
{RETURN <type as defined in RETURNS clause>|RETURN (<SELECT statement>)}

END }



UDFs Returning a Scalar Value

- Scalar UDFs
  - Must have a BEGIN/END
  - Must be schema qualified when invoked (unless invoked as stored procedures with EXEC, as in EXEC myFunction 3, 4).
  - Do not allow omitting optional parameters (ones that have default values) when invoked.



Scalar UDFs

```
CREATE FUNCTION dbo.DayOnly(@Date datetime)
RETURNS varchar(12)
AS
BEGIN
   RETURN CONVERT(varchar(12), @Date, 101)
END
----------
SELECT * FROM Orders
WHERE dbo.DayOnly(OrderDate) =
   dbo.DayOnly(GETDATE())
```

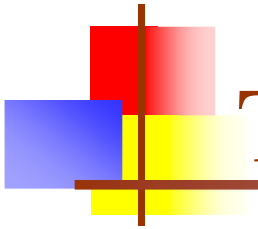


Table-valued UDFs

- Table-valued UDFs: return a table
  - Don't specify a BEGIN/END block in an inline UDF's body
  - Specify is a RETURN clause and a query
- Example

```
CREATE FUNCTION dbo.fn_GetCustOrders
   (@cid AS NCHAR(5)) RETURNS TABLE
AS
RETURN
   SELECT OrderID, CustomerID, EmployeeID, OrderDate,
   RequiredDate,ShippedDate, ShipVia, Freight,
   ShipName, ShipAddress, ShipCity,
     ShipRegion, ShipPostalCode, ShipCountry
   FROM dbo.Orders
   WHERE CustomerID = @cid;
```

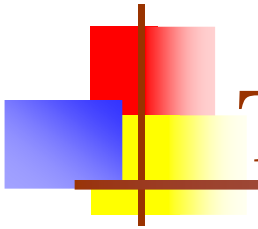
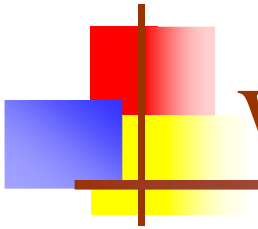


Table-valued UDFs

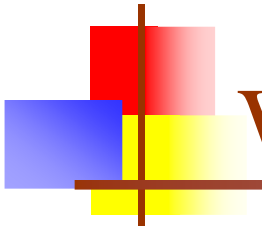
- Example

```
SELECT O.OrderID, O.CustomerID,
       OD.ProductID, OD.Quantity
FROM dbo.fn_GetCustOrders(N'ALFKI') AS O
  JOIN [Order Details] AS OD
    ON O.OrderID = OD.OrderID;
```



With Schemabinding

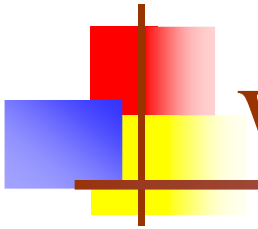
- SchemaBinding
  - Binds the function to the schema of any objects it references (tables, views, and other user-defined functions)
  - Alter or drop any object referenced by a schema-bound function fails.
- To alter or drop object referenced
  - The function is dropped.
  - The function is modified by using the ALTER statement with the SCHEMABINDING option not specified.



With Schemabinding

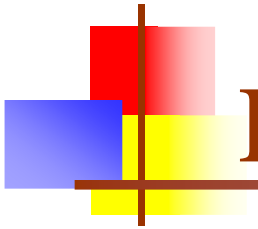
- Example

```
CREATE FUNCTION dbo.SumFreight()
RETURNS money
WITH SCHEMABINDING
AS
BEGIN
RETURN (SELECT SUM(Freight) FROM
  dbo.Orders)
END
```



With Schemabinding

- Conditions
  - All views and user-defined functions referenced by the function must be schema-bound.
  - All objects referenced by the function must be in the same database as the function. The objects must be referenced using either one-part or two-part names.
  - You must have REFERENCES permission on all objects (tables, views, and user-defined functions) referenced in the function.



Reference

- Books online
- Inside Microsoft® SQL Server™ 2005 T-SQL Programming, Microsoft Press, 2006