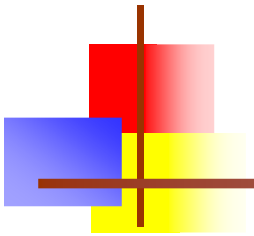
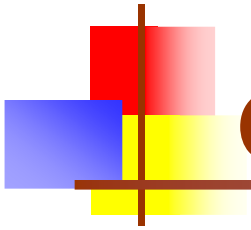
Thang Long University



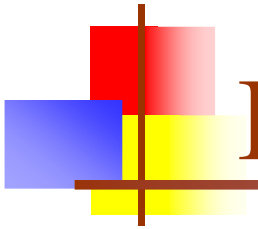
T-SQL Programming (Phần 1)

Giảng viên: Trần Quang Duy



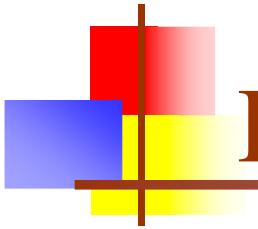
Content

- Basic Transact-SQL Programming Constructs
  - Identifiers
  - DataType
  - Variables
  - Flow-control Statements
- Batches and Scripts
- Temporary Tables and Table Variables
- Dynamic SQL
- Exception Handling



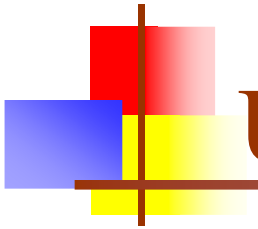
Identifiers

- Main rules:
  - Start with any letter: A–Z and a–z..
  - Up to 128 characters for normal objects, 116 for temporary objects.
  - Any names that are the same as SQL Server keywords or contain embedded spaces must be enclosed in double quotes (“”) or square brackets ([]).



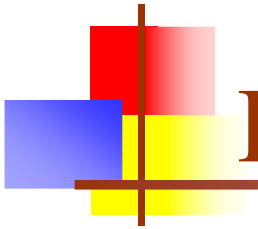
Data Types

- Character Strings
  - char, varchar, varchar(max), text
- Unicode Character Strings
  - nchar, nvarchar, nvarchar(max), ntext
- Date and Time
  - datetime, smalldatetime
- Intger Numbers
  - int, smallint, tinyint, bigint, bit
- Special Data Types
  - sql_variant



Using variables

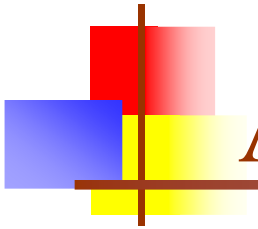
- Variable : stores values
- T-SQL variables:
  - Start with @
    - Local variables: @
    - Global variables: @@
  - DECLARE statement
  - Datatype: must be specified
  - Multiple variables: single DECLARE statement with “,”
  - There is no way to “undeclare” variables.



Declaring Variables

- Declaring Variables

```
DECLARE @age INT;
DECLARE @firstName CHAR(20), @lastName CHAR(20);
```



Assigning Values to Variables

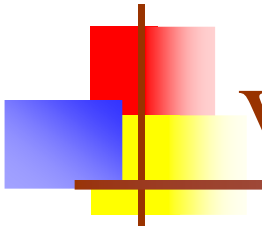
- Using the SET statement

```
DECLARE @age INT;
DECLARE @firstName CHAR(20), @lastName
  CHAR(20);
SET @lastName='Forta';
SET @firstName='Ben';
SET @age=21;
```

- Using the SELECT statement

```
SELECT @age=21;
```

- SET or SELECT?



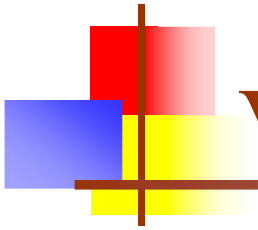
Viewing Variable Contents

- Input:

```
DECLARE @age INT;
DECLARE @firstName CHAR(20), @lastName
  CHAR(20);
SET @lastName='Forta';
SET @firstName='Ben';
SET @age=21;
SELECT @lastName, @firstName, @age
```

- Output:

Forta Ben 21



Viewing Variable Contents

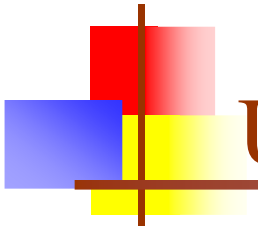
- Input

```
DECLARE @age INT;
DECLARE @firstName CHAR(20),@lastName
  CHAR(20);
SET @lastName='Forta';
SET @firstName='Ben';
SET @age=21;
PRINT @lastName + ', ' + @firstName;
PRINT @age;
```

- Output

Forta , Ben

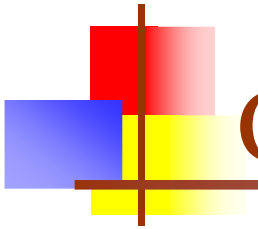
21



Using Variables in T-SQL Statements

- Using Variables:

```
DECLARE @cust_id nchar(5);
SET @cust_id = 'ALFKI';
-- Lay thong tin cong ty, dia chi
SELECT CompanyName, Address
FROM customers
WHERE CustomerID = @cust_id
--Lay thong tin hoa don
SELECT OrderID, OrderDate
FROM ORDERS
WHERE CustomerID = @cust_id
ORDER BY OrderDate
```



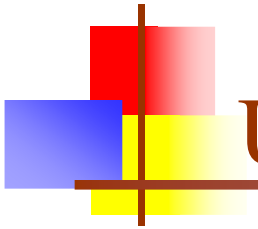
Comment

- Single-line Comments: --

```
-- This is a comment
-- Select * from Customers
```

- Multiline Comments: /*…*/

```
/* This is a comment.
 All these lines will be ignored.
*/
```



Using Conditional Processing

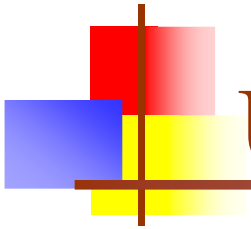
- IF statement

  IF <Boolean Expression>

  <SQL statement> | BEGIN <code series> END

  [ELSE

  <SQL statement> | BEGIN <code series> END]



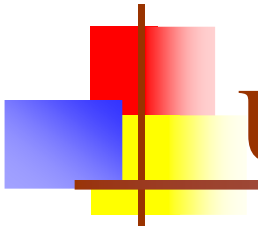
Using Conditional Processing

- Input

```
DECLARE @open BIT
IF DatePart(dw, GetDate()) = 1
  SET @open = 0
ELSE
  SET @open = 1
-- Output
SELECT @open AS OpenForBusiness
```

- Output

```
OpenForBusiness
---------------
1
```



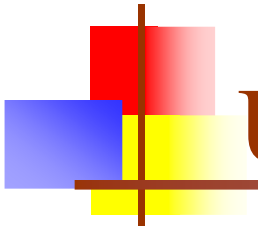
Using Conditional Processing

- Input

```
DECLARE @dow INT
DECLARE @open BIT
SET @dow = DatePart(dw, GetDate());
IF @dow = 1 OR @dow = 7
  SET @open = 0
ELSE
  SET @open = 1
SELECT @open AS OpenForBusiness
```

- Output

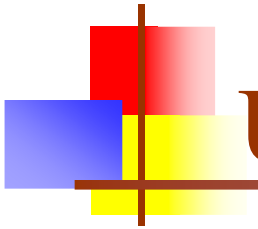
```
OpenForBusiness
---------------
1
```



Using Conditional Processing

- Input

```
DECLARE @dow INT
DECLARE @open BIT, @process BIT
-- Get the day of week
SET @dow = DatePart(dw, GetDate());
-- Open for business today?
IF @dow = 1 OR @dow = 7
BEGIN
  SET @open = 0
  SET @process = 0
END
ELSE
BEGIN
  SET @open = 1
  SET @process = 1
END
```



Using Looping

- WHILE statement

  WHILE <Boolean expression>

  <sql statement> |

  [BEGIN

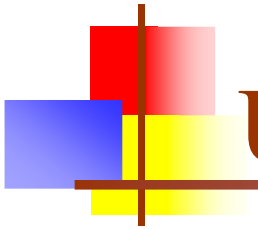
  <statement block>

  [BREAK]

  <sql statement> | <statement block>

  [CONTINUE]

  END]



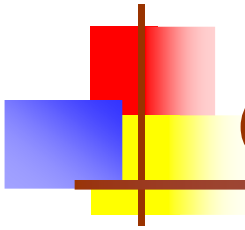
Using Looping

- Input

```
DECLARE @counter INT
SET @counter=1
WHILE @counter <= 10
BEGIN
  PRINT @counter
  SET @counter=@counter+1
END
```

- Output

1 2 3 4 5 6 7 8 9 10



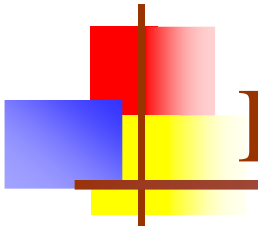
GoTo,WaitFor Statement

- GoTo label . . . label:

```
If @@Error <> 0 Goto ERR_1
  ..........
ERR_1:
  Print 'Error'
```

- WaitFor {Delay 'time' | Time 'time'}

```
WaitFor Delay '00:01:00'
        Select * from dbo.Customers

WaitFor Time '23:00'
 Backup Database Northwind To Northwind_bkp
```



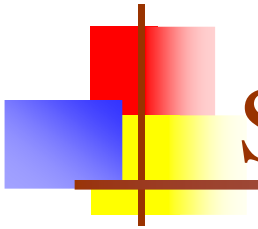
Batch and Scripts

- Batch?: is a set of SQL statements

```
SELECT CompanyName, Address
FROM customers WHERE CustomerID = N'ALFKI';
SELECT OrderID, OrderDate
FROM ORDERSWHERE CustomerID = N'ALFKI'
ORDER BY OrderDate
```

- Script: collection of Transact-SQL statements in the form of an external file.
- GO statements

```
SELECT CompanyName, Address
FROM customers WHERE CustomerID = N'ALFKI';
GO
SELECT OrderID, OrderDate
FROM ORDERS WHERE CustomerID = N'ALFKI'
ORDER BY OrderDate
```



SQLCMD

- SQLCMD: run scripts from a command prompt in a Windows command box

  SQLCMD

  [{ { -U <*login id*> [-P <*password*>] } | –E }]

  [-S <*server name*> [\<*instance name*>]] [-H <*workstation name*>] [-d <*db name*>]

  [-q "<*query*>"] [-Q "<*query*>"]

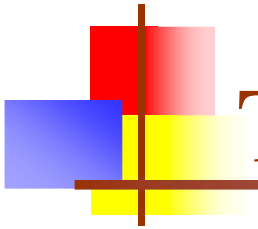
  [-i <*input file*>] [-o <*output file*>]

- Example

  C:\>SQLCMD -Usa -Pmypass -Q "SELECT * FROM Northwind.dbo.Shippers"

  -- run script in text file

  C:\> SQLCMD -Usa -Pmypass -i testsql.sql



Temporary Tables

- TempDB
- Local Temporary Tables
  - Prefixed with a number symbol (#).
  - A temporary table is owned by the creating session and visible only to it
  - Allows different sessions to create a temporary table with the same name

```
CREATE TABLE #T1(col1 INT);
```

- Global Temporary Tables
  - Prefixed with a two number symbol (##).
  - They are accessible by all sessions
  - Any session can even drop the table

```
CREATE TABLE ##T1(col1 INT);
```

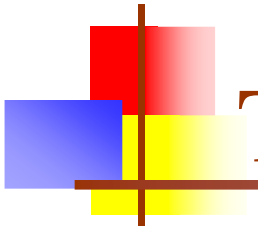
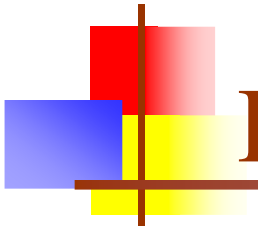


Table Variables

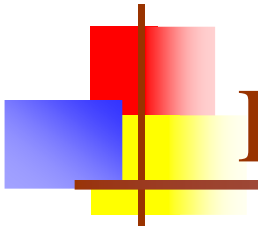
- Table Variables
  - Scope: in scope of a table variable is defined.
  - Cannot create explicit indexes on table variables, only PRIMARY KEY and UNIQUE constraints
  - Cannot alter the definition of a table variable
  - Cannot issue SELECT INTO
  - Cannot qualify a column name with a table variable name

```
DECLARE @T1 TABLE(col1 INT);
INSERT @T1 VALUES(1);
SELECT * FROM @T1;
```



Dynamic SQL

- Dynamic SQL
  - EXEC (EXECUTE)
  - sp_executesql
- Working with dynamic SQL
  - It runs under a separate scope than the code that calls it.
  - It runs under the same security context as the current user. Permissions to execute code if the code is within a stored procedure
  - It runs under the same connection and transaction context as the calling object
  - Can’t do the concatenation of function in EXEC call.
  - EXEC can not be used inside a User Defined Function.



Dynamic SQL

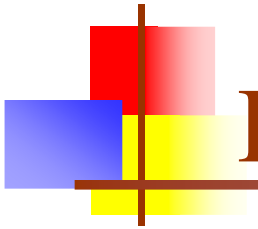
- EXEC (EXECUTE)

  EXEC <*procedure name and arguments*>

  Or EXEC ({<*string variable*>|'<*literal command string*>'})

- Example

```
EXEC ('select * from dbo.customers')
----------------------
DECLARE @InVar varchar(200)
SET @InVar = 'DECLARE @OutVar varchar(50)
SELECT @OutVar = FirstName FROM Employees
  WHERE EmployeeID = 1
SELECT ''The Value Is '' + @OutVar'
-- Now run it
EXEC (@Invar)
```



Dynamic SQL

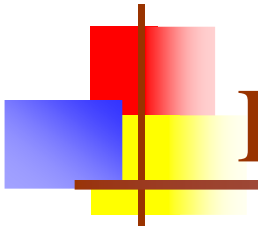
- sp_executesql
  - EXEC sp_executesql

    @stmt = <statement>, -- similar to proc's body

    @params = <params>, -- similar to proc's params declaration

    <params assignment> -- like in a procedure call
- Example

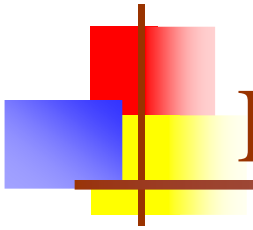
```
DECLARE @i AS INT;
SET @i = 10248;
DECLARE @sql AS NVARCHAR(46);
SET @sql = 'SELECT * FROM dbo.Orders WHERE OrderID =
  @oid;';
EXEC sp_executesql
  @stmt = @sql,
  @params = N'@oid AS INT',
  @oid = @i;
```



Exception Handling

- Prior to SQL Server 2005
  - @@Error: return an integer (0- success and <> 0: error code) of the last T-SQL statement executed.

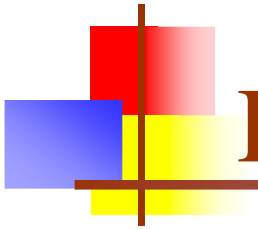
```
DECLARE @Error int
INSERT INTO [Order Details]
(OrderID, ProductID, UnitPrice, Quantity, Discount)
VALUES
(999999,11,10.00,10, 0)
SELECT @Error = @@ERROR
IF @Error!=0
  PRINT 'The Value of @Error is ' + CONVERT(varchar,
  @Error)
ELSE
  PRINT 'Successed'
```



Exception Handling

- In SQL Server 2005
  - TRY/CATCH

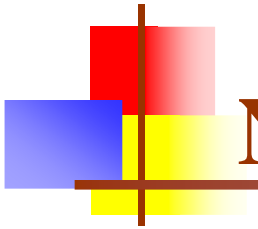
```
BEGIN TRY
….

END TRY
BEGIN CATCH
…

END CATCH
```



Exception Handling

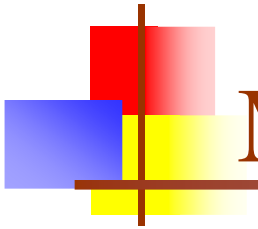
- TRY/CATCH

```
BEGIN TRY
 INSERT INTO [Order Details](OrderID, ProductID,
  UnitPrice, Quantity, Discount)
      VALUES(999999,11,10.00,10, 0)
  PRINT 'INSERT succeeded.';
END TRY
BEGIN CATCH
  PRINT 'INSERT failed.';
  /* handle error here */
END CATCH
```



New Exception-Handling Functions

Function	Purpose
Error_Message()	Returns the error message that would normally be returned to the caller application
Error_Number()	Returns the identifier of the error
Error_Severity()	Returns the severity
Error_State()	Returns the state
Error_Procedure()	Returns the name of the procedure (or other programmatic database object) in which the error has occurred
Error_Line()	Returns the line number of the procedure in which the error has occurred



Manually Raising Errors

- Raiserror

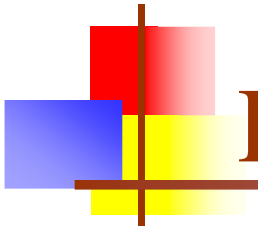
```
Raiserror  (<message ID | message string>, <severity>,
   <state> [, <argument>[,<...n>]] )[WITH option[,...n]]
Raiserror ('An error occurred!', 0, 1)
```

- sp_addmessage

```
Exec sp_addmessage 50001,16,'New Error'
```

- sp_dropmessage

```
Exec sp_dropmessage 50001
```



Reference

- Books online
- Inside Microsoft® SQL Server™ 2005 T-SQL Programming, Microsoft Press, 2006
- Sams Teach Yourself Microsoft SQL Server 2005 Express in 24 Hours, Alison Balter, Sams Publisher, 2006