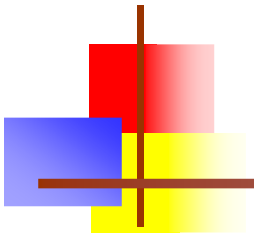
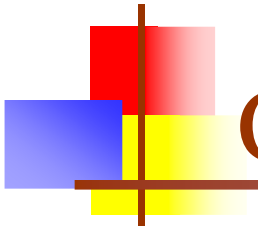
Thang Long University



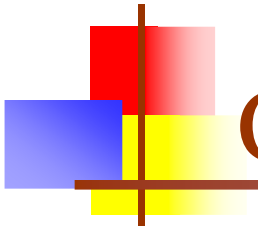
T-SQL Programming (Phần 3)

Trần Quang Duy



Content

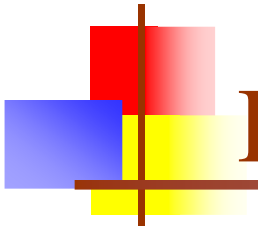
- Cursor
  - Working with Cursors
  - Fetch Data Options
  - Cursor with Stored Procedure
  - Cursor with Functions
- View
  - Working Views
  - Refresh Views
  - Modifications against View
  - View Options: ENCRYPTION, SCHEMABINDING, CHECK OPTION



Cursor

- Cursor: database query stored
- Working with Cursors

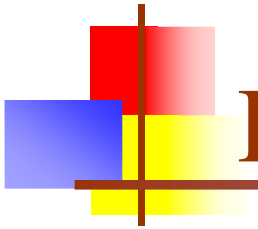
1. Before a cursor can be used, it must be declared (defined). it merely defines theSELECT statement to be used.
2. After it is declared, the cursor must be opened for use.
3. With the cursor populated with data, individual rows can be fetched (retrieved) as needed.
4. Once the desired data has been fetched, the cursor must be closed.
5. Finally, the cursor must be removed.



Declare Cursor

- SQL 92 Syntax

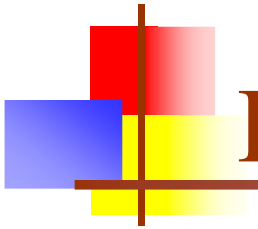
```
DECLARE cursor_name [ INSENSITIVE ] [
  SCROLL ]
CURSOR FOR select_statement
[ FOR { READ ONLY | UPDATE
[ OF column_name [ ,...n ] ] } ]
```



Declare Cursor

- Transact-SQL Extended Syntax

```
DECLARE cursor_name CURSOR[ LOCAL | GLOBAL]
[ FORWARD_ONLY | SCROLL ][ STATIC | KEYSET|
DYNAMIC | FAST_FORWARD ][ READ_ONLY |
  SCROLL_LOCKS |
OPTIMISTIC ][ TYPE_WARNING ]
FOR select_statement[ FOR UPDATE [ OF
  column_name [ ,...n ] ] ]
[;]
```

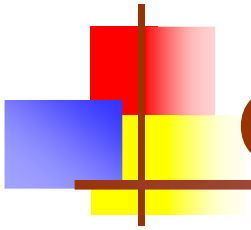


Declare and RemoveCursors

- Declare and Remove Cursors

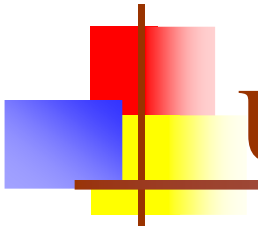
```
-- Define the cursor
DECLARE orders_cursor CURSOR
FOR
SELECT orderID FROM orders
  ORDER BY orderID;

-- Remove the cursor
DEALLOCATE orders_cursor;
```



Opening and Closing Cursors

```
-- Define the cursor
DECLARE orders_cursor CURSOR
FOR
SELECT orderID FROM orders  ORDER BY
  orderID;
-- Open cursor (retrieve data)
OPEN orders_cursor;
-- Close cursor
CLOSE orders_cursor
-- And finally, remove it
DEALLOCATE orders_cursor;
```



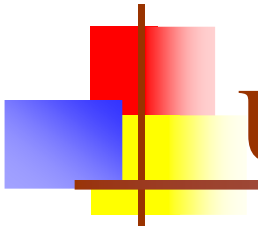
Using Cursor Data

- FETCH statement

```
FETCH   [ [ NEXT | PRIOR | FIRST | LAST
          | ABSOLUTE { n | @nvar }
          | RELATIVE { n | @nvar }]
FROM { { [ GLOBAL ] cursor_name } |
  @cursor_variable_name }
[INTO @variable_name [ ,...n ] ]
```

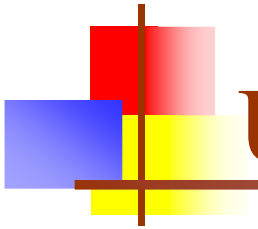
- Example

```
FETCH NEXT FROM orders_cursor INTO
  @order_num;
```



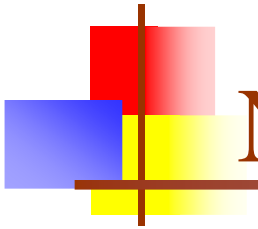
Using Cursor Data

```
-- Local variables
DECLARE @order_num INT;
-- Define the cursor
DECLARE orders_cursor CURSOR
FOR
SELECT orderID FROM orders ORDER BY orderID;
-- Open cursor (retrieve data)
OPEN orders_cursor;
-- Perform the first fetch (get first row)
FETCH NEXT FROM orders_cursor INTO @order_num;
-- Close cursor
CLOSE orders_cursor
-- And finally, remove it
DEALLOCATE orders_cursor;
```



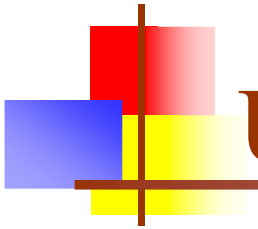
Using Cursor Data

```
DECLARE @order_num INT;
DECLARE orders_cursor CURSOR
FOR
SELECT orderID FROM orders ORDER BY orderID;
-- Open cursor (retrieve data)
OPEN orders_cursor;
-- Perform the first fetch (get first row)
FETCH NEXT FROM orders_cursor INTO @order_num;
WHILE @@FETCH_STATUS = 0
BEGIN
  PRINT @order_Num
FETCH NEXT FROM orders_cursor INTO @order_num;
END
-- Close cursor
CLOSE orders_cursor
-- And finally, remove it
DEALLOCATE orders_cursor;
```



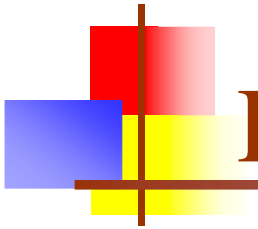
Nested Cursor

```
DECLARE CustomerCursor CURSOR FOR
   SELECT CustomerID, CompanyName FROM Customers;
OPEN CustomerCursor;
FETCH NEXT FROM CustomerCursor INTO @CustID, @CompanyName;
WHILE @@FETCH_STATUS=0
BEGIN
   DECLARE OrderCursor CURSOR FOR SELECT OrderID, OrderDate
   FROM Orders WHERE CustomerID=@CustID
   OPEN OrderCursor;
   FETCH NEXT FROM OrderCursor INTO @OrderID , @OrderDate;
   WHILE @@FETCH_STATUS=0
   BEGIN
        .......
   END
   CLOSE OrderCursor
   DEALLOCATE OrderCursor
   FETCH NEXT FROM CustomerCursor INTO @CustID, @CompanyName;
END
CLOSE CustomerCursor
DEALLOCATE CustomerCursor
```



Update Cursor

```
DECLARE OrderCursor CURSOR FOR
SELECT OrderID, CustomerID, ShipRegion  FROM Orders
FOR UPDATE OF ShipRegion
OPEN OrderCursor;
FETCH NEXT FROM OrderCursor INTO @OrderID , @CustomerID,
   @ShipRegion;
WHILE @@FETCH_STATUS=0
BEGIN
       IF @ShipRegion is null
       BEGIN
              UPDATE Orders
              SET ShipRegion ='N/A'
              WHERE CustomerID=@CustomerID
       END
       FETCH NEXT FROM OrderCursor INTO @OrderID,
   @CustomerID, @ShipRegion;
END
CLOSE OrderCursor
DEALLOCATE OrderCursor
```



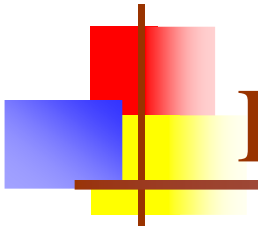
Fetch Data Options

- FETCH statement

```
FETCH   [ [ NEXT | PRIOR | FIRST | LAST
        | ABSOLUTE { n | @nvar }
        | RELATIVE { n | @nvar }]
FROM { { [ GLOBAL ] cursor_name } |
  @cursor_variable_name }
[INTO @variable_name [ ,...n ] ]
```

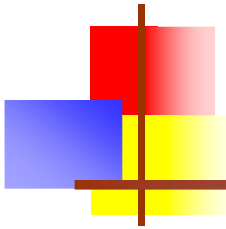
- Example

```
DECLARE orders_cursor SCROLL CURSOR
FOR SELECT orderID FROM orders ORDER BY
orderID;
FETCH FIRST FROM orders_cursor INTO
@order_num;
```



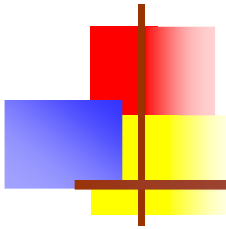
Fetch Data Options

```
-- get first row
FETCH FIRST FROM orders_cursor;
-- get last row
FETCH LAST FROM orders_cursor
-- get row 5th row from first
FETCH ABSOLUTE 5 FROM orders_cursor
-- get row 5th from last
FETCH ABSOLUTE -5 FROM orders_cursor
-- get row 3th next from current row
FETCH RELATIVE 3 FROM orders_cursor
-- get row 3th prior from current row
FETCH RELATIVE -3 FROM orders_cursor
```



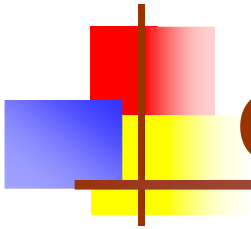
Cursor with Stored Procedure

```
CREATE PROC OutputCursor
  @CursorForOutput CURSOR VARYING
  OUTPUT
AS
  SET @CursorForOutput= CURSOR
  FORWARD_ONLY STATIC FOR
  SELECT OrderID,CustomerID FROM
  Orders;
  OPEN @CursorForOutput
GO
```



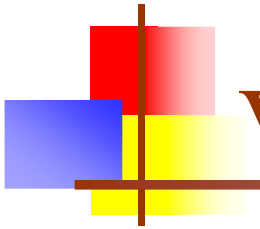
Cursor with Stored Procedure

```
DECLARE @OutputCursor CURSOR
EXEC OutputCursor @OutputCursor OUTPUT
--read data from Cursor
FETCH NEXT FROM @OutputCursor
WHILE @@FETCH_STATUS=0
BEGIN
     FETCH NEXT FROM @OutputCursor
END
CLOSE @OutputCursor
DEALLOCATE @OutputCursor
```



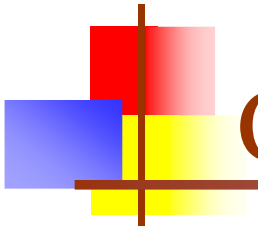
Cursor with Functions

```
CREATE FUNCTION dbo.GetOrderList(@CustID as nchar(5))
RETURNS varchar(1000) WITH EXECUTE AS caller
AS
BEGIN
   DECLARE @OrderList varchar(1000)
   DECLARE @OrderID int;
   DECLARE OrderCursor CURSOR FOR
   SELECT OrderID FROM Orders WHERE CustomerID=@CustID
   OPEN OrderCursor;
   FETCH NEXT FROM OrderCursor INTO @OrderID;
   WHILE @@FETCH_STATUS=0
   BEGIN
        SET @OrderList= @OrderList +  cast(@OrderID as
   varchar)  + ', '
        FETCH NEXT FROM OrderCursor INTO @OrderID
   END
   CLOSE OrderCursor
   DEALLOCATE OrderCursor
   RETURN @OrderList
END
```



Views

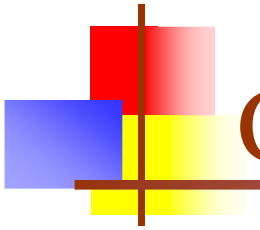
- What Are Views ?
  - A view is a named virtual table that is defined by a query and used as a table
- Using View
  - To provide a more or less normalized picture of the underlying data without changing the normalization of the actual data
  - Solving complex problems one step at a time
  - Use views as a security layer
  - To improve the performance



Create View

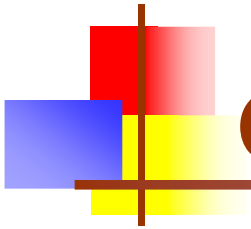
- Syntax

```
CREATE VIEW [schema_name].<view name>
  [(<column name list>)]
[WITH [ENCRYPTION] [, SCHEMABINDING] [,
  VIEW_METADATA]]
AS
<SELECT statement>
WITH CHECK OPTION
```



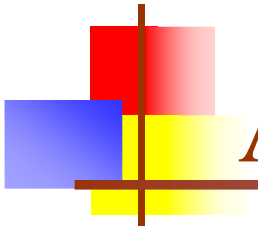
Create View

```
CREATE VIEW dbo.VCustsWithOrders
AS
SELECT CustomerID, CompanyName,
  ContactName, ContactTitle,
   Address, City, Region, PostalCode,
  Country, Phone, Fax
FROM Customers AS C
WHERE EXISTS
   (SELECT * FROM dbo.Orders AS O
    WHERE O.CustomerID = C.CustomerID);
GO
```



Create View

- Create view must meet three requirements
  - ORDER BY cannot be used in the view's query unless there is also a TOP.
  - All result columns must have names.
  - All result column names must be unique.



Alter and Drop View

- ALTER VIEW: the same CREATE VIEW

```
ALTER VIEW dbo.VCustsWithOrders
AS
SELECT CustomerID, CompanyName,
  ContactName, ContactTitle,
   Address, City, Region, PostalCode,
  Country, Phone, Fax
FROM Customers AS C
```

- DROP VIEW
  - Syntax

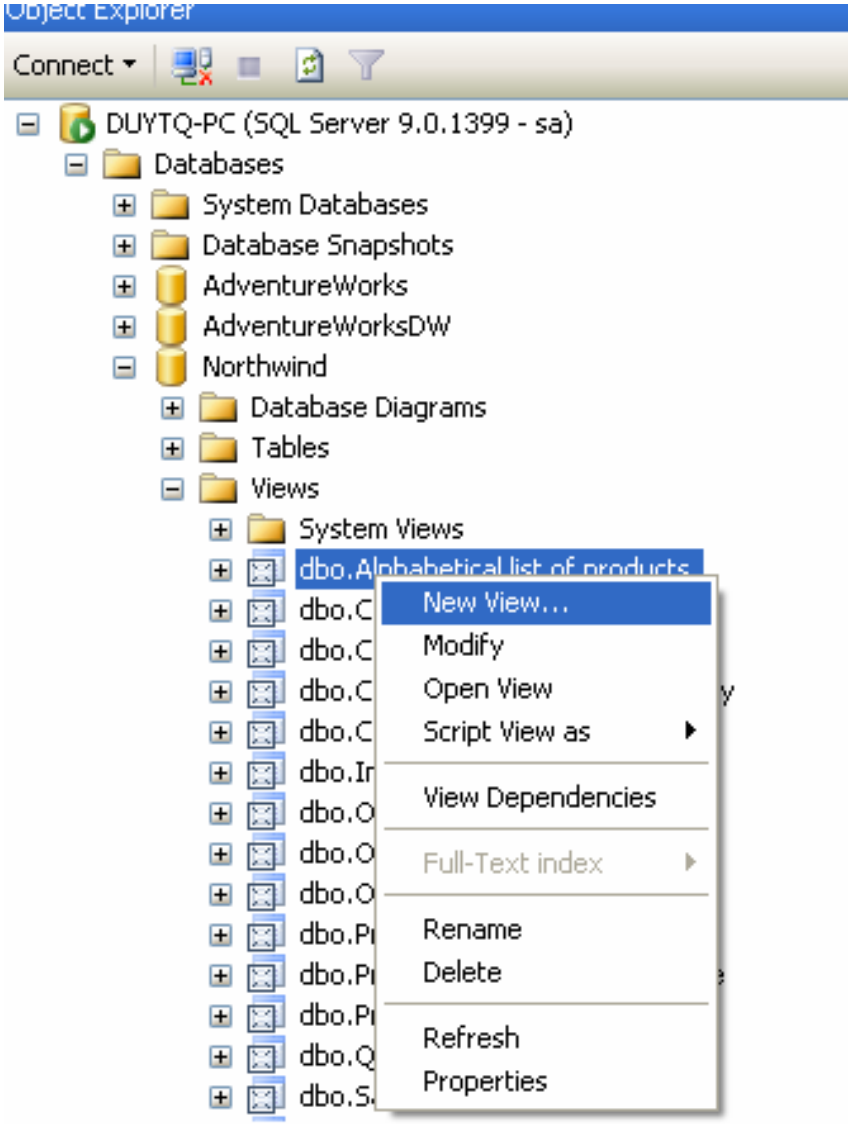
```
DROP VIEW <view name>, [<view name>,[...n]]
```

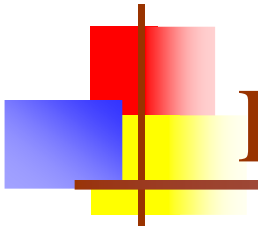
  - Example

```
DROP VIEW dbo.VCustsWithOrders
```

Working View with Management Studio

- Choose Database
- Right-click on Views

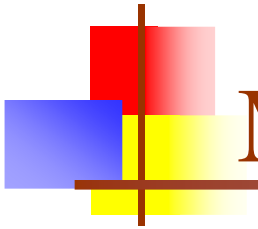




Refresh View

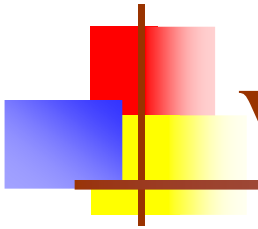
- Refreshing Views
  - When you create a view, SQL Server stores metadata information describing the view, its columns, security, dependencies
  - Schema changes in underlying objects are not reflected in the view's metadata information
- sp_refreshview
  - To refresh the view's metadata information
  - Syntax

    sp_refreshview 'View_Name'



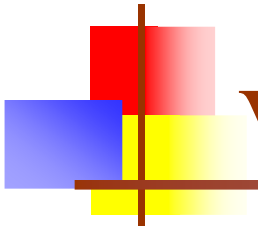
Modifications against View

- Can modify data against a view, SQL Server will modify the underlying tables
- Modifications against views have the following limitations
  - If the view is defined by a join query, an UPDATE or INSERT statement is allowed to affect only one side of the join
  - You cannot modify a column that is a result of a calculation
  - If WITH CHECK OPTION was specified when the view was created or altered, INSERT or UPDATE statements that conflict with the view's query filter will be rejected



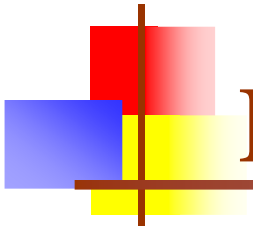
View Option

- ENCRYPTION
  - To encrypt your view
- SCHEMABINDING
  - Prevents drop underlying objects or make any schema modification to referenced columns
- WITH CHECK OPTION
  - Prevents INSERT and UPDATE statements that conflict with the view's query filter



View Option

```
ALTER VIEW dbo.VCustsWithOrders WITH
  ENCRYPTION, SCHEMABINDING
AS
SELECT CustomerID, CompanyName, ContactName,
  ContactTitle,
  Address, City, Region, PostalCode, Country,
  Phone, Fax
FROM dbo.Customers AS C
WHERE EXISTS
   (SELECT 1 FROM dbo.Orders AS O
    WHERE O.CustomerID = C.CustomerID)
WITH CHECK OPTION;
```



Reference

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