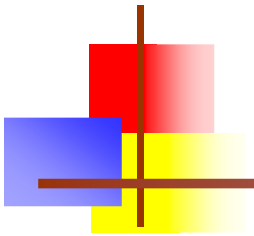
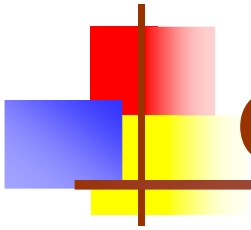
Thang Long University



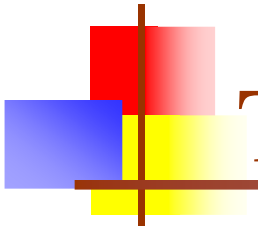
T-SQL Programming (Phần 4)

Trần Quang Duy



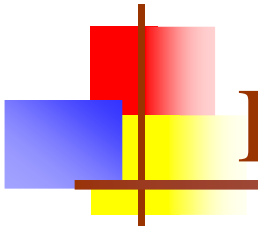
Content

- Transaction
  - What is Transaction
  - Working with Transaction.
- Trigger
  - What is Trigger
  - DML Triggers
    - DML AFTER Triggers
    - DML INSTEAD OF Triggers
  - DDL Triggers



Transaction

- Transaction
  - A transaction is a sequence of operations performed as a single logical unit of work.
- A logical unit of work must have four properties,
  - Atomicity (Tính nguyên tử)
  - Consistency (Tính nhất quán)
  - Isolation (Tính độc lập)
  - Durability (Tính bền bỉ)



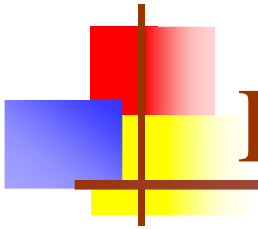
Declare Transaction

- Begin Transaction

```
BEGIN { TRAN | TRANSACTION }
   [ { transaction_name |
  @tran_name_variable }
  [ WITH MARK [ 'description' ] ]   ]
  [ ; ]
```

- Commit Transaction

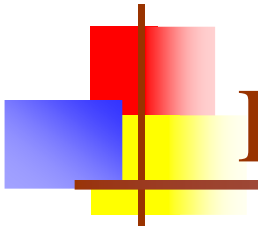
```
COMMIT { TRAN | TRANSACTION }
[ transaction_name | @tran_name_variable ]
  ]
[ ; ]
```



Declare Transaction

- Declare Transaction

```
BEGIN TRANSACTION
   T-SQL Statement
COMMIT TRANSACTION
--Or
BEGIN TRANSACTION TransactionName
   T-SQL Statement
COMMIT TRANSACTION TransactionName
```



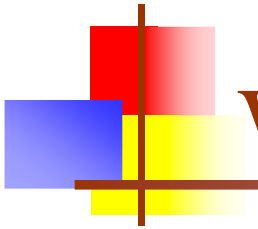
Declare Transaction

- Example

```
BEGIN TRANSACTION
  insert into dbo.Shippers
  values('VN Express', '084 12345678')
COMMIT TRANSACTION
```

Or

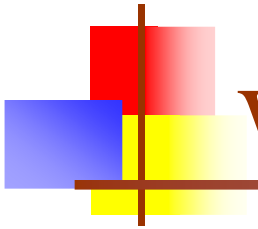
```
BEGIN TRANSACTION ShipperTran
  insert into dbo.Shippers
  values('VN Express', '084 12345678')
COMMIT TRANSACTION ShipperTran
```



Working with Rollback Tran

- Rollback Transaction

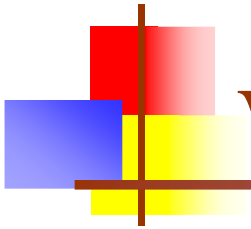
```
ROLLBACK { TRAN | TRANSACTION }
    [ transaction_name |
  @tran_name_variable
      savepoint_name | @savepoint_variable ]
[ ; ]
```



Working with Rollback Tran

- Example

```
BEGIN TRANSACTION
  DELETE FROM CustomerTrans
  INSERT INTO CustomerTrans
  VALUES('ABCDEF','VN Company')
IF @@ERROR !=0
  ROLLBACK TRAN
ELSE
  COMMIT TRAN
```



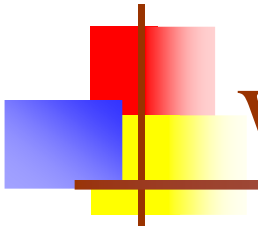
Working with Save Transaction

- Save Transaction

```
SAVE { TRAN | TRANSACTION }
    { savepoint_name | @savepoint_variable }[ ; ]
```

- Example

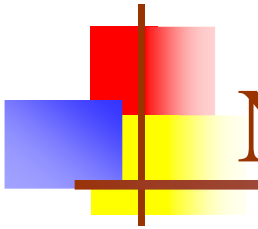
```
BEGIN TRANSACTION
  UPDATE CustomerTrans SET CompanyName = 'Test
  Company' WHERE CustomerID='ALFKI'
  SAVE TRAN UpdateCust
  DELETE FROM CustomerTrans
  INSERT INTO CustomerTrans VALUES('ABCDEF','VN
  Company')
IF @@ERROR !=0
  ROLLBACK TRAN UpdateCust
ELSE
  COMMIT TRAN
```



Withmark Options

- With Mark

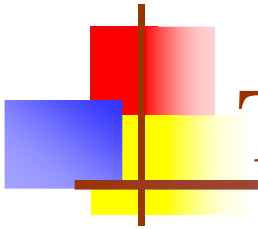
```
BEGIN TRANSACTION INS WITH MARK 'Working
  with CustomerTrans'
  DELETE FROM CustomerTrans
  INSERT INTO CustomerTrans
  VALUES('ABCDEF','VN Company')
COMMIT TRANSACTION INS
```



Nested Transaction

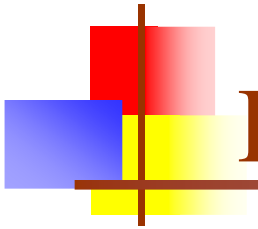
- Nested Transaction

```
BEGIN TRAN CustTran
  UPDATE CustomerTrans SET CompanyName =
  'Test Company' WHERE CustomerID='ALFKI'
  BEGIN TRAN OrderTran
        DELETE FROM OrderTrans WHERE
        CustomerID='ALFKI'
  COMMIT TRAN OrderTran
  Select * from OrderTrans WHERE
  CustomerID='ALFKI'
  Select * from CustomerTrans
COMMIT TRAN CustTran
```



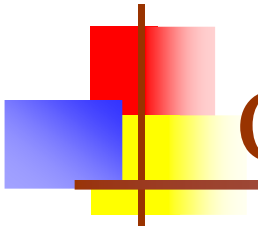
Trigger

- What is Trigger
  - A trigger is a special kind of stored procedure that automatically executes when an event occurs in the database
- Kinds of triggers
  - Data manipulation language (DML) triggers
  - Data definition language (DDL) triggers
- Trigger can fire After or Instead of the triggering event
  - AFTER triggers
  - INSTEAD OF triggers



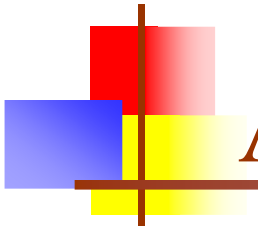
DML Trigger

- DML Triggers: is executed once for each modification statement.
  - Insert
  - Update
  - Delete
- DML Trigger
  - DML AFTER triggers
  - DML INSTEAD OF triggers



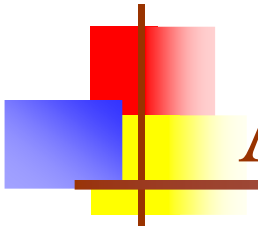
Create DML Triggers

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH  [ ENCRYPTION ] [ EXECUTE AS Clause ]
 [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [,] [UPDATE ] [,] [DELETE] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement  [ ; ] [ ...n ] | EXTERNAL
  NAME <method specifier [ ; ] > }
```



After DML Triggers

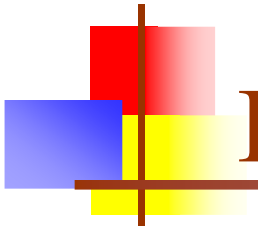
- After DML Triggers
  - An After trigger is fired after the modification statement finishes successfully
  - DML AFTER triggers can be created only on permanent tables. They cannot be created on views or temporary tables
  - AFTER triggers are fired per statement, not per row
  - Can create multiple AFTER triggers on each object for each statement type



After DML Triggers

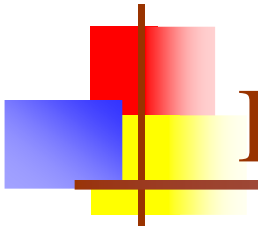
- Example

```
Create Trigger dbo.tr_CustomersTemp_I
On dbo.CustomersTemp
After insert  -- For Insert
As
      Print 'You have just inserted '
            + Cast(@@rowcount as varchar)
            + ' record(s)!'
Go
```



Inserted and Deleted Virtual Tables

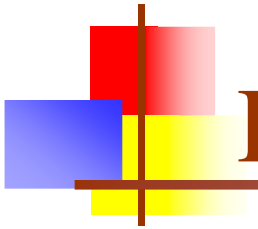
Modification statement	Deleted	Inserted
Insert	N/A	New records
Update	Old version of updated records	New version of updated records
Delete	Deleted records	N/A



Inserted and Deleted Virtual Tables

- Inserted and Deleted Virtual Tables

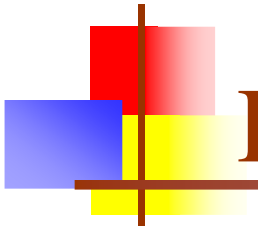
```
alter Trigger dbo.tr_CustomersTemp_I
On dbo.CustomersTemp
After insert  -- For Insert
As
      Print 'You have just inserted '
            + Cast(@@rowcount as varchar)
            + ' record(s)!'
  Select * from inserted
Go
```



Functions within the trigger

- UPDATE(Column Name):
  - returns TRUE?FALSE whether a particular column has been updated or no?
  - return TRUE for any column if you use it in an INSERT trigger.
- Example

```
Create Trigger dbo.tr_CustomersTemp_U
On dbo.CustomersTemp
After update  -- For Update
As
  if update(CompanyName)
Begin
  DECLARE @msg AS VARCHAR(100);
  SET @msg = 'Value: "'
   + CAST((SELECT CompanyName FROM inserted) AS
  VARCHAR(10)) + '" is updated into CompanyColunm';
   Print @msg
End
```



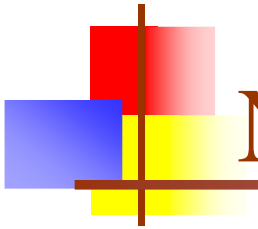
Functions within the trigger

- COLUMNS_UPDATED
  - Returns a binary string with a bit for each column

0	1	1	0	0	1	0	0

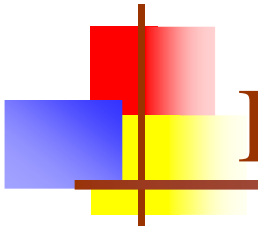
- Example

```
if Columns_Updated() & 2 = 2
      print 'Column 2 was updated!'
```



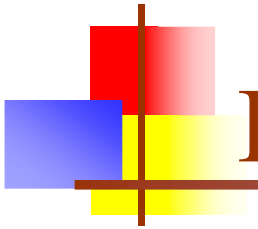
Nested and Recursive Triggers

- Nested triggers:
  - A trigger can fire other triggers on the same or other tables when it inserts, updates, or deletes records in them.
- Recursive triggers
  - Direct recursion
    - Example: an application updates table **T3**; this causes trigger **Trig3** to fire. **Trig3** updates table **T3** again; this causes trigger **Trig3** to fire again.
  - Indirect recursion
    - Example: an application updates table **T1**; this causes trigger **Trig1** to fire. **Trig1** updates table **T2**; this causes trigger **Trig2** to fire. **Trig2** in turn updates table **T1** that causes **Trig1** to fire again.



Instead of triggers

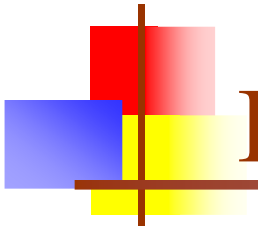
- Instead-of triggers
  - Are executed instead of the modification statement that has initiated them
  - Are executed after changes to base tables occur in Inserted and Deleted virtual tables but before any change to the base tables is executed.
  - INSTEAD OF triggers can be created on views
  - INSTEAD OF triggers is that they fire before constraints are checked



Instead of triggers

- Example

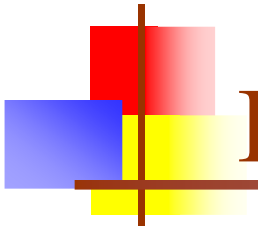
```
Create TRIGGER dbo.tr_CustomersTemp_I2
On dbo.CustomersTemp INSTEAD OF INSERT
As
  Select * from Inserted
  INSERT INTO dbo.CustomersTemp SELECT *
  FROM inserted
```



Instead of triggers on View

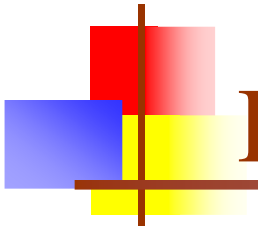
- Example

```
CREATE VIEW dbo.VOrderTotals
AS
SELECT orderid, SUM(quantity) AS totalqty
FROM [Order Details]
GROUP BY orderid;
GO
```



Instead of triggers on View

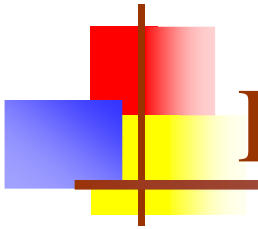
```
CREATE TRIGGER trg_VOrderTotals_ioi ON dbo.VOrderTotals INSTEAD
  OF UPDATE
AS
IF @@rowcount = 0 RETURN;
WITH UPD_CTE AS
(
  SELECT qty, ROUND(1.*OD.qty / D.totalqty * I.totalqty, 0) AS
  newqty
  FROM dbo.OrderDetails AS OD
    JOIN inserted AS I
      ON OD.oid = I.oid
    JOIN deleted AS D
      ON I.oid = D.oid
)
UPDATE UPD_CTE
  SET qty = newqty;
GO

------------------
UPDATE dbo.VOrderTotals
  SET totalqty = totalqty * 2;
```



DDL Triggers

- DDL triggers: new in SQL Server 2005
  - Create Table
  - Drop Table
  - Alter Procedure
  - Drop Schema
  - Create Login
- Syntax

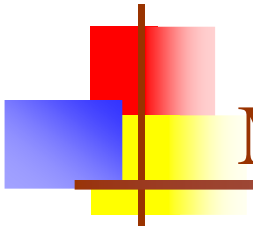
```
CREATE TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[[ WITH [ ENCRYPTION ] | [ EXECUTE AS ..]
{FOR| AFTER } { event_type | event_group } [ ,...n ]
AS
    { sql_statement  [ ; ] [ ...n ]
```



DDL Triggers

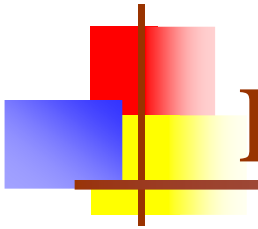
- Example

```
CREATE TRIGGER trdPreventTableChanges
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE, CREATE_TABLE
AS
RAISERROR('Changes to the tables are typically not
  allowed
If you do have a permission to change tables,
temporarily disablethe trigger by using:
DISABLE TRIGGER trdPreventTableChanges ON DATABASE
<your batch with table changes>
ENABLE TRIGGER trdPreventTableChanges ON DATABASE;
', 16, 1)
ROLLBACK;
GO
```



Managing Triggers in Management Studio

- Managing DML Triggers
  - Table\Trigger.
  - Right-click the trigger and choose Modify from the pop-up menu.
- Managing DDL Triggers
  - Server \ Databases \database \Programmability \Database Triggers
  - Server \ ServerObject \ Triggers.



Reference

- Books online
- Inside Microsoft® SQL Server™ 2005 T-SQL Programming, Microsoft Press, 2006
- Microsoft SQL Server 2005 Stored Procedure Programming in T-SQL & .NET, Third Edition, McGraw-Hill, 2006