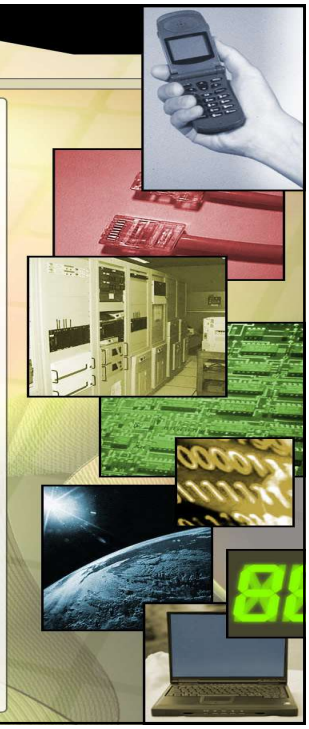
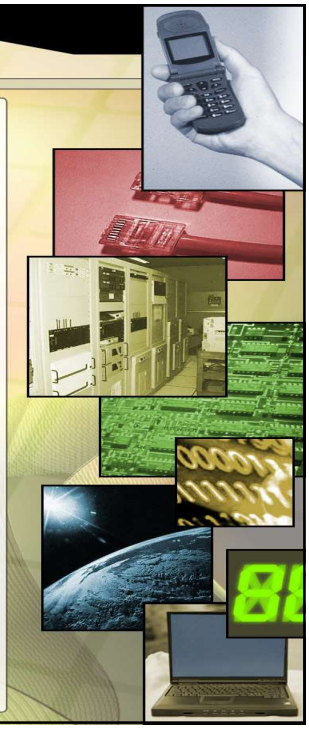
LẬP TRÌNH T-SQL



245

GIỚI THIỆU

- Transact-SQL (T-SQL) là một ngôn ngữ lập trình CSDL hướng thủ tục độc quyền của Microsoft sử dụng trong SQL Server.
- Được thiết kế để mở rộng khả năng của SQL

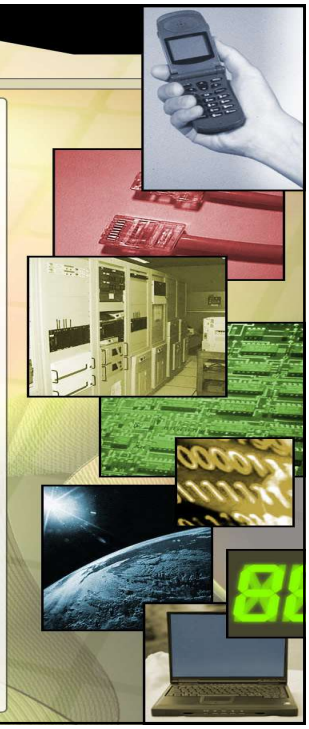


246

GIỚI THIỆU

- T-SQL tổ chức theo từng khối lệnh, bắt đầu bởi **BEGIN** và kết thúc bởi **END**, các lệnh trong khối ngăn cách nhau bởi dấu ";"

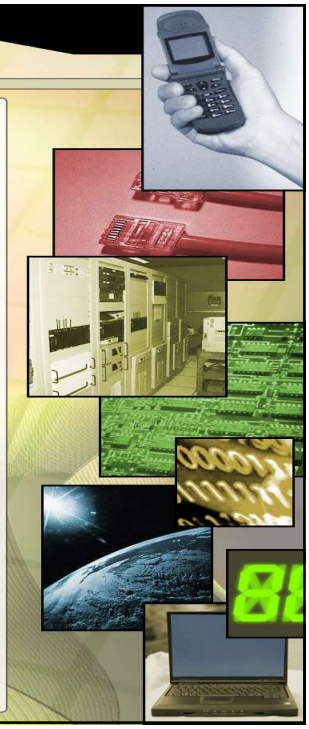
```
BEGIN
    -- Khai báo biến
    -- Các câu lệnh T-SQL
END;
```



247

BIẾN CỤC BỘ

- Dùng để lưu trữ các giá trị tạm thời trong quá trình tính toán.
- Biến phải có kiểu dữ liệu.
- Biến muốn sử dụng trong một batch phải khai báo trước.
- Một biến cục bộ chỉ có phạm vi hoạt động trong một Batch, Stored Procedure hay Trigger



248

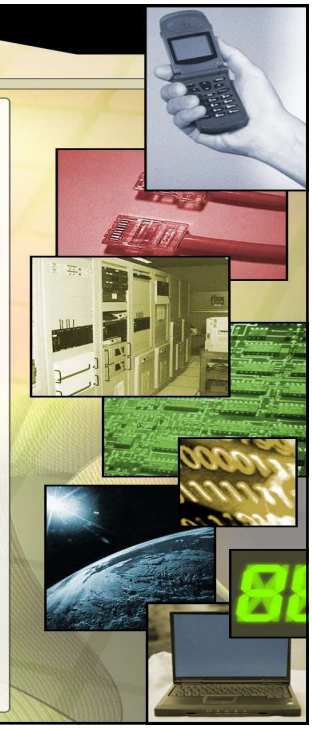
BIẾN CỤC BỘ

- **Khai báo**

```
DECLARE @Tên_biến Kiểu_dữ_liệu [ = Giá_trị ]
[ , ... ] ;
```

Ví dụ:

```
DECLARE @TenNV CHAR(50), @Ngaysinh DATE;
DECLARE @Tong INT = 0;
DECLARE @v_table TABLE (
   < Định nghĩa bảng >
);
```



249

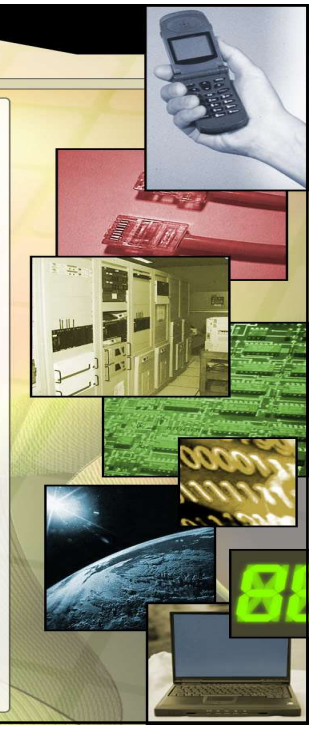
BIẾN CỤC BỘ

- **Gán giá trị cho biến**

```
SET @Tên_biến = Giá_trị ;
SELECT @Tên_biến = Tên_Cột [...]
   FROM Tên_Bảng [...] ;
```

Ví dụ:

```
SET @TenNV = N'Sơn';
SELECT
    @product_name = product_name,
    @list_price = list_price
FROM PRODUCTS
WHERE product_id = 100;
```



250

BIẾN CỤC BỘ

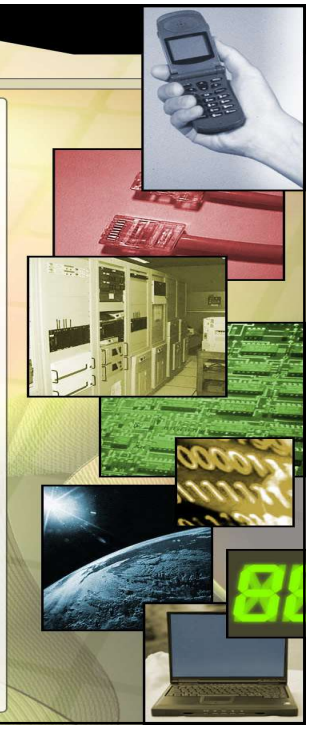
- **Xem giá trị hiện hành của biến:** sử dụng hàm CAST hoặc CONVERT

  CAST(Biểu_thức AS *Kiểu_dữ_liệu*)

  CONVERT(*Kiểu_dữ_liệu*, Biểu_thức [, Định_dạng])

  Ví dụ:

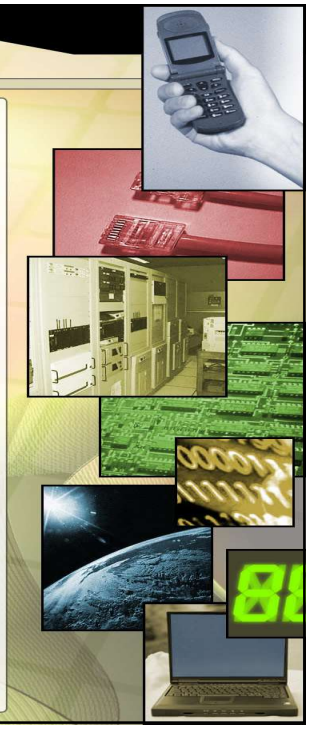
```
declare @Ngay datetime = '2019-11-26';
declare @Soluong int = 5;
PRINT N'Ngày: ' + CONVERT(nvarchar(20),@Ngay,103);
PRINT N'Số lượng: ' + CAST(@Soluong as char(4));
```



251

BIẾN HỆ THỐNG

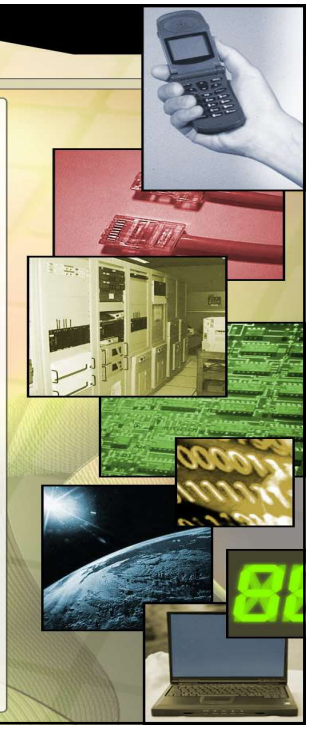
- **Cung cấp các thông tin hệ thống**
  - **@@Version** - Phiên bản SQL Server
  - **@@ServerName** - Tên Server
  - **@@Language** - Ngôn ngữ sử dụng
  - **@@Connections** - Tổng kết nối vào SQL Server
  - **@@Error** - Mã lỗi của lệnh lỗi gần nhất
  - **@@Fetch_Status** - Tình trạng đọc con trỏ (đọc thành công = 0)
  - **@@RowCount** - Tổng số mẩu tin tác động bởi câu lệnh truy vấn gần nhất.
- **Không cần khai báo**



252

RẼ NHÁNH IF...ELSE...

```
IF Biểu_thức_logic
   BEGIN
      < Code xử lý khi Biểu_thức_logic TRUE >
   END
[
ELSE
   BEGIN
      < Code xử lý khi Biểu_thức_logic FALSE >
   END
]
```

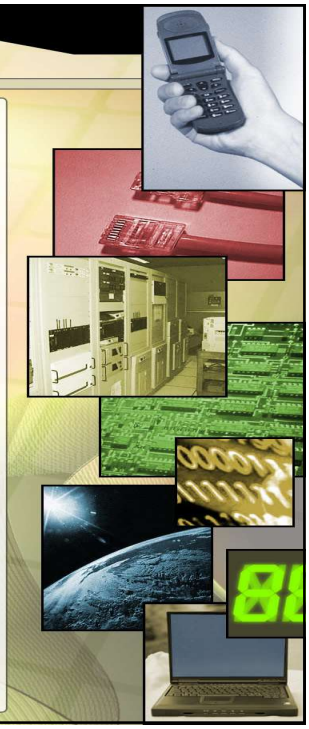


253

LẶP WHILE

```
WHILE Biểu_thức_logic
   BEGIN
      < Code xử lý khi Biểu_thức_logic TRUE >
   END
```

- Vòng lặp sẽ dừng khi Biểu_thức_logic có giá trị FALSE

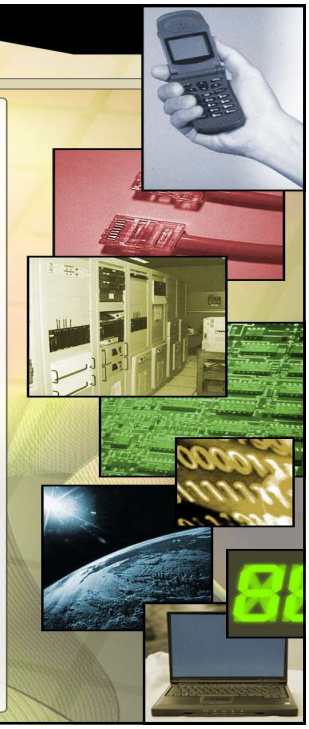


254

XỬ LÝ LỖI TRY...CATCH

- Thực hiện các lệnh trong khối **TRY**, nếu gặp lỗi sẽ chuyển qua xử lý bằng các lệnh trong khối **CATCH**

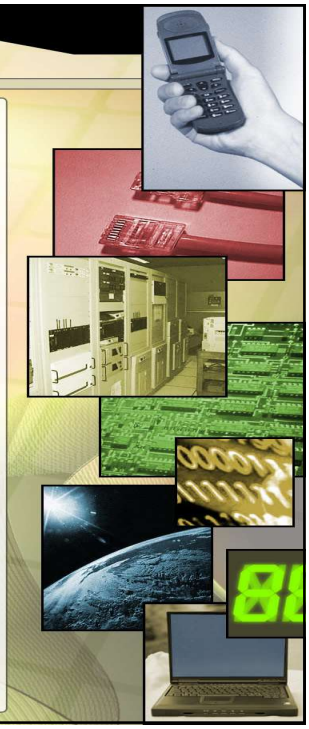
```
BEGIN TRY
    < Code xử lý >
END TRY
BEGIN CATCH
    < Code xử lý khi phần TRY có lỗi >
END CATCH
```



255

XỬ LÝ LỖI TRY...CATCH

- TRY và CATCH phải cùng lô xử lý
- Sau khối TRY phải là khối CATCH
- Có thể lồng nhiều cấp

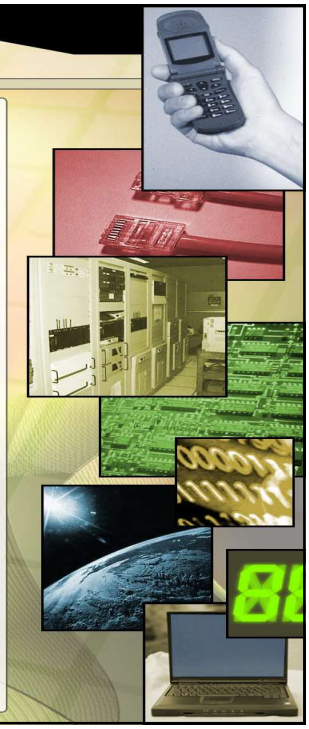


256

XỬ LÝ LỖI TRY...CATCH

Ví dụ:

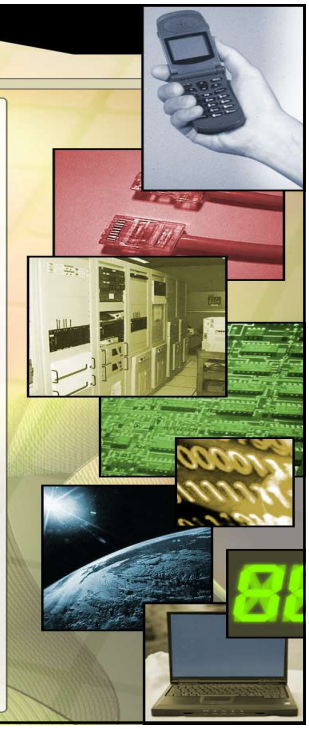
```
begin try
    select * from BangKhongTonTai;
end try
begin catch
    select
      ERROR_NUMBER() as ErrorNumber,
      ERROR_MESSAGE() as ErrorMessage;
end catch
```



257

XỬ LÝ LỖI TRY...CATCH

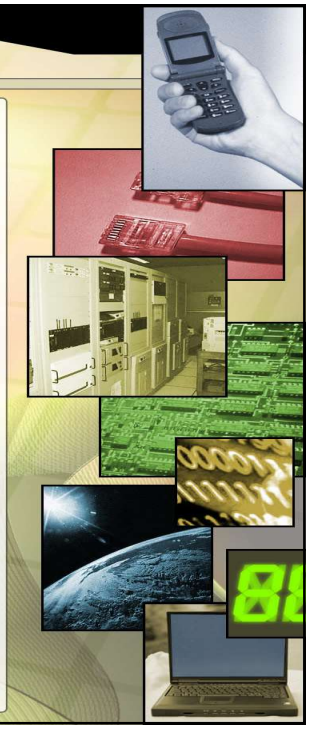
- **Một số hàm ERROR thường dùng**
  - `ERROR_NUMBER()` - Mã lỗi
  - `ERROR_MESSAGE()` - Thông báo lỗi
  - `ERROR_SEVERITY()` - Mức độ lỗi
  - `ERROR_STATE()` - Tình trạng lỗi
  - `ERROR_LINE()` - Dòng code gây ra lỗi
  - `ERROR_PROCEDURE()` - Tên thủ tục/ trigger gây ra lỗi



258

CON TRỎ (CURSOR)

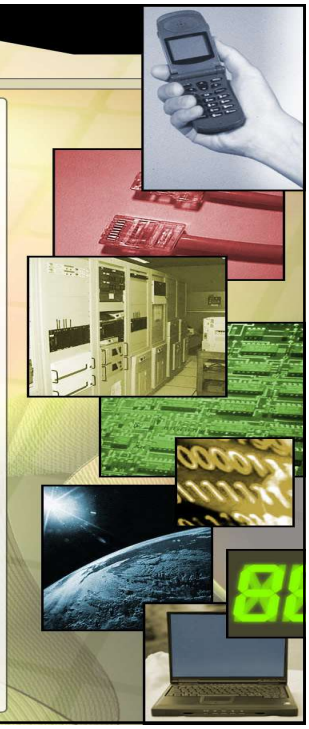
- Các lệnh của SQL Server làm việc trên một nhóm nhiều mẩu tin
- Cursor là cấu trúc giúp làm việc từng mẩu tin tại một thời điểm
  - Khai báo như một câu lệnh SELECT
  - Di chuyển giữa các mẩu tin trong cursor để làm việc
  - Cập nhật dữ liệu (Update, Delete)



259

SỬ DỤNG CON TRỎ

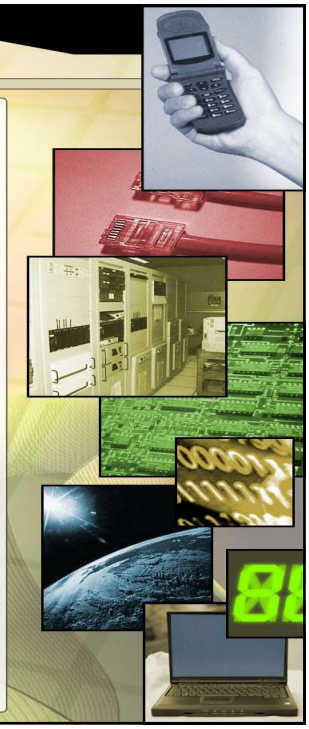
- Định nghĩa biến kiểu cursor bằng lệnh **DECLARE**
  - Có hai loại cursor: **LOCAL**, **GLOBAL**
  - Cách di chuyển mẩu tin trong cursor: **FORWARD_ONLY**, **SCROLL**
  - Cách quản lý dữ liệu của cursor: **STATIC**, **DYNAMIC**, **KEYSET**



260

SỬ DỤNG CON TRỎ

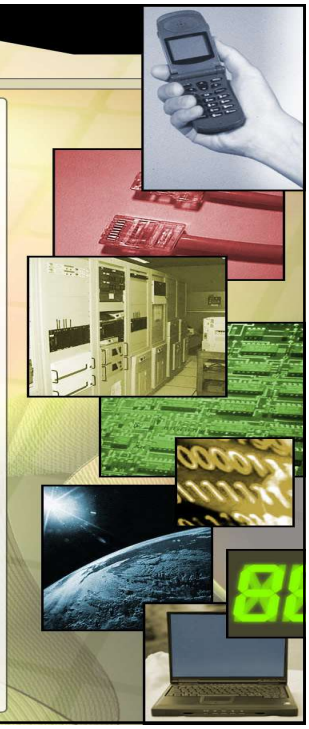
- Sử dụng lệnh **OPEN** để mở ra cursor
- Đọc và xử lý trên từng dòng dữ liệu trong cursor:
  - Sử dụng biến **@@Fetch_Status**, các lệnh **FETCH** và cấu trúc **WHILE**
- Đóng cursor lại bằng lệnh **CLOSE** và **DEALLOCATE**
  - Sau khi CLOSE, có thể mở lại
  - DEALLOCATE: hủy cursor khỏi bộ nhớ



261

SỬ DỤNG CON TRỎ

```
DECLARE Tên_cursor CURSOR
[ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | DYNAMIC | KEYSET ]
[ READ_ONLY | SCROLL_LOCK ]
FOR Câu_lệnh_SELECT
[ FOR UPDATE [ OF Danh_sách_cột_cập_nhật ] ]
;
```



262

SỬ DỤNG CON TRỎ

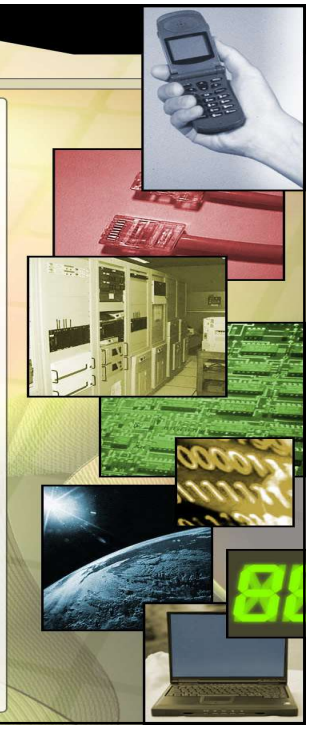
FETCH [NEXT | PRIOR | FIRST | LAST

| ABSOLUTE n | RELATIVE n]

FROM Tên_cursor

[INTO *Danh_sách_biến*] ;

- **Absolute n:** Đọc dòng thứ n trong cursor
- **Relative n:** Đọc dòng thứ n kể từ vị trí hiện hành

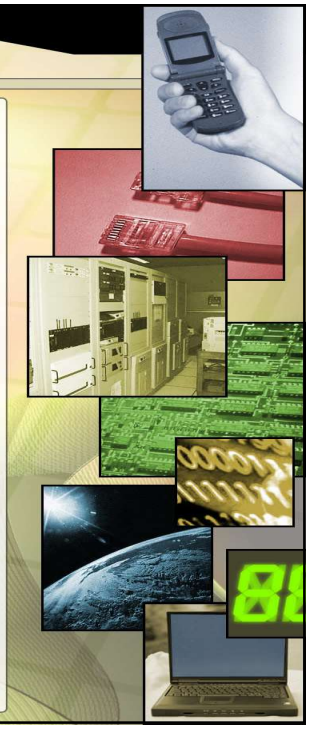


263

SỬ DỤNG CON TRỎ

Ví dụ:

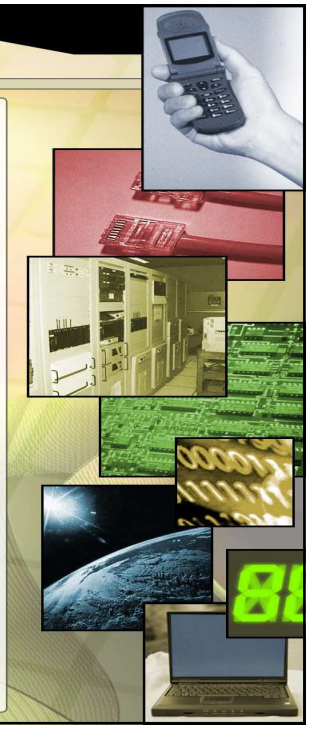
```
--1. Khai báo cursor
declare cur_Vattu cursor keyset
  for select * from VATTU
      where MAVTU like 'TV%'
      order by MAVTU ;

--2. Mở cursor
open cur_Vattu ;
```



264

SỬ DỤNG CON TRỎ

```
--3. Đọc dữ liệu
fetch next from cur_Vattu ;
while @@FETCH_STATUS = 0
  begin
    -- Xử lý dòng mới vừa đọc được
    -- Thực hiện đọc tiếp các dòng kế
    fetch next from cur_Vattu ;
  end ;

--4. Đóng cursor
close cur_Vattu ;
deallocate cur_Vattu ;
```

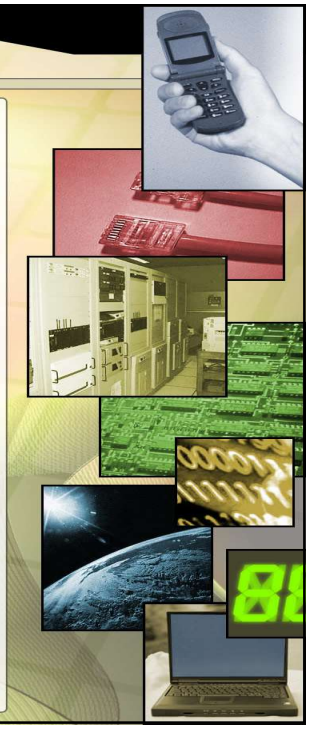


265

SỬ DỤNG CON TRỎ

Ví dụ: **Khai báo con trỏ dạng biến**

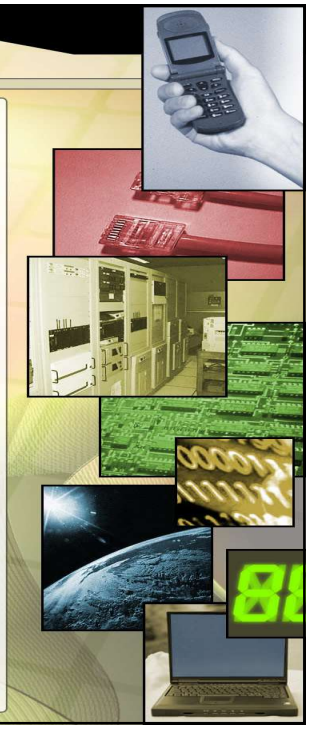
```
--1. Khai báo biến cursor
declare @cur_Vattu cursor ;
SET @cur_Vattu = CURSOR
  for select * from VATTU
      where MAVTU like 'TV%'
      order by MAVTU ;

--2. Mở cursor
open @cur_Vattu ;
```



266

SỬ DỤNG CON TRỎ

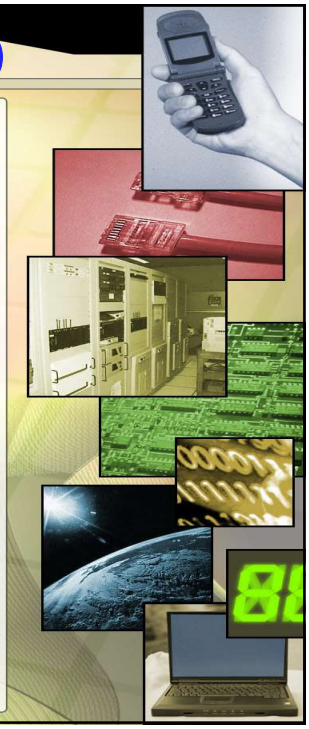
```
--3. Đọc dữ liệu
fetch next from @cur_Vattu ;
while @@FETCH_STATUS = 0
  begin
    -- Xử lý dòng mới vừa đọc được
    -- Thực hiện đọc tiếp các dòng kế
    fetch next from @cur_Vattu ;
  end ;

--4. Đóng cursor
close @cur_Vattu ;
deallocate @cur_Vattu ;
```



267

THỦ TỤC (STORED PROCEDURE)

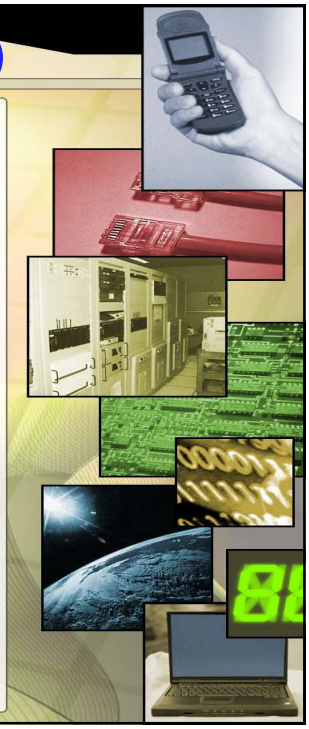
- Là tập hợp một hoặc nhiều câu lệnh T-SQL thành một nhóm đơn vị xử lý logic và được lưu trữ trên Database Server.
- Khi gọi stored procedure lần đầu tiên thì SQL Server sẽ thực thi và lưu trữ vào bộ nhớ đệm, gọi là **plan cache**, những lần gọi tiếp theo SQL Server sẽ sử dụng lại plan cache nên tốc độ xử lý tối ưu.



268

THỦ TỤC (STORED PROCEDURE)

```
CREATE PROCEDURE < Tên_thủ_tục>
  [ (
      @tham_số_1  Kiểu_dữ_liệu_1 [ OUTPUT ] ,
      @tham_số_2  Kiểu_dữ_liệu_2 [ OUTPUT ] ,
      …
    ) ]
AS
BEGIN
   -- Khai báo biến sử dụng
   -- Nội dung của thủ tục
END;
```

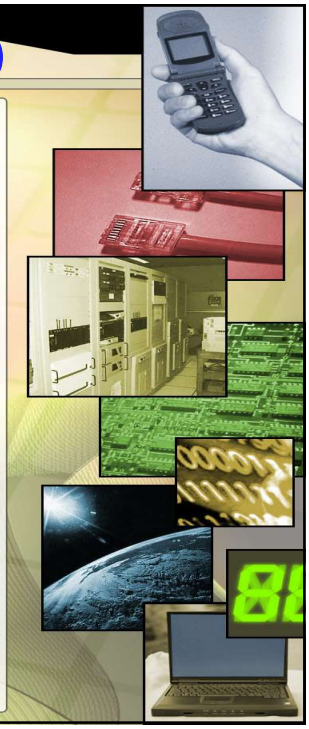


269

THỦ TỤC (STORED PROCEDURE)

Ví dụ:

```
create procedure FindProductByModel (
    @model_year smallint,
    @product_count int output
) as
begin
    select product_name, list_price
    from production.products
    where model_year = @model_year;

    select @product_count = @@ROWCOUNT;
end;
```



270

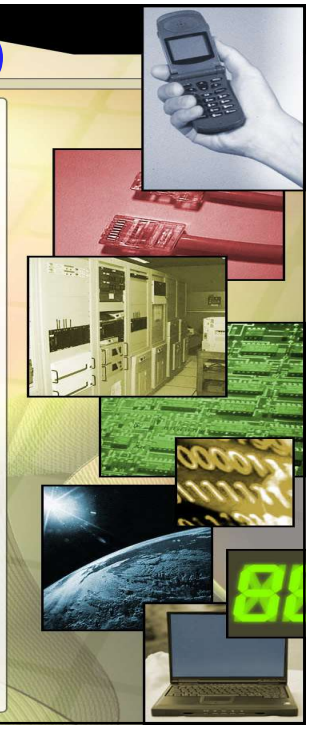
THỦ TỤC (STORED PROCEDURE)

- **Gọi thủ tục**

  EXECUTE | EXEC schema.Tên_thủ_tục
  [Giá_trị_tham_số_1, Giá_trị_tham_số_2, …]

  hoặc

  EXECUTE | EXEC schema.Tên_thủ_tục
  [@tham_số_1 = Giá_trị_tham_số_1,
  @tham_số_2 = Giá_trị_tham_số_2,
  …
  ]

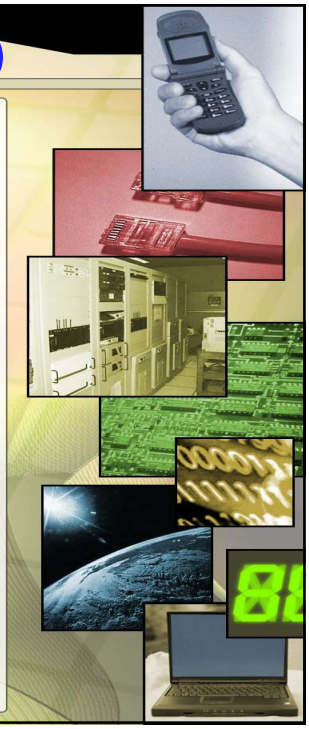


271

THỦ TỤC (STORED PROCEDURE)

Ví dụ:

```
declare @count int;

exec FindProductByModel
   @model_year = 2018
   @product_count = @count;

if (@count > 0)
begin
   print 'Number of products found';
end;
```



272

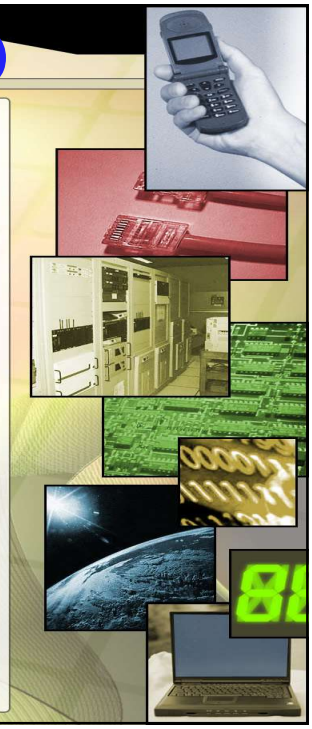
THỦ TỤC (STORED PROCEDURE)

- **Thay đổi thủ tục**

```
ALTER PROCEDURE < Tên_thủ_tục>
    [ (
      @tham_số_1  Kiểu_dữ_liệu_1 [ OUTPUT ] ,
      @tham_số_2  Kiểu_dữ_liệu_2 [ OUTPUT ] ,
    …
      ) ]
AS …
```

- **Xóa thủ tục**

```
DROP PROCEDURE < Tên_thủ_tục> ;
```

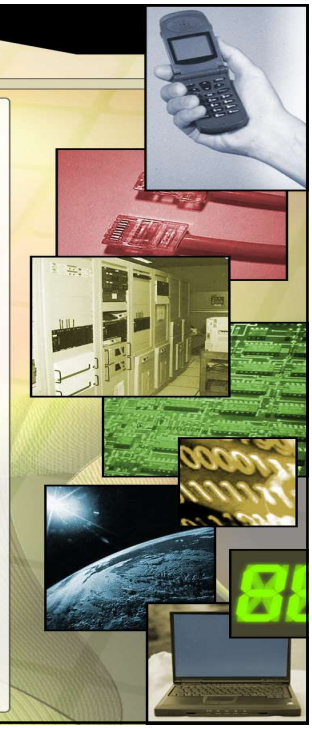


273

HÀM(FUNCTION)

- Tương tự như procedure nhưng Function có giá trị trả về

```
CREATE FUNCTION < Tên_hàm>
( [
     @tham_số_1  Kiểu_dữ_liệu_1 [ OUTPUT ] ,
     @tham_số_2  Kiểu_dữ_liệu_2 [ OUTPUT ] ,
     …
  ] )
RETURNS Kiểu_dữ_liệu AS
BEGIN
     -- Khai báo biến sử dụng
     -- Nội dung của hàm
    RETURN Giá_trị_hàm ;
END;
```

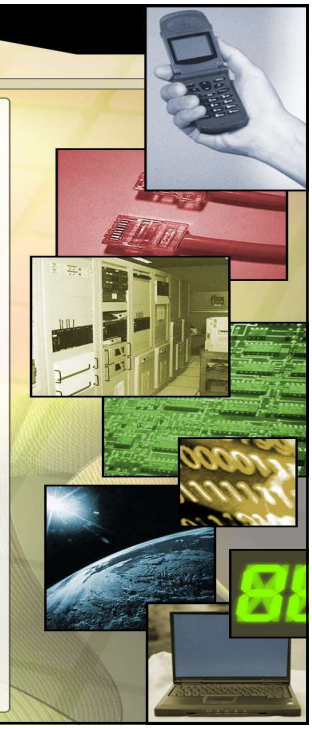


274

HÀM(FUNCTION)

- **Gọi hàm:** gọi trực tiếp trong câu lệnh

```
schema.Tên_hàm (
    Giá_trị_tham_số_1,
    Giá_trị_tham_số_2,
    …
)
```



275

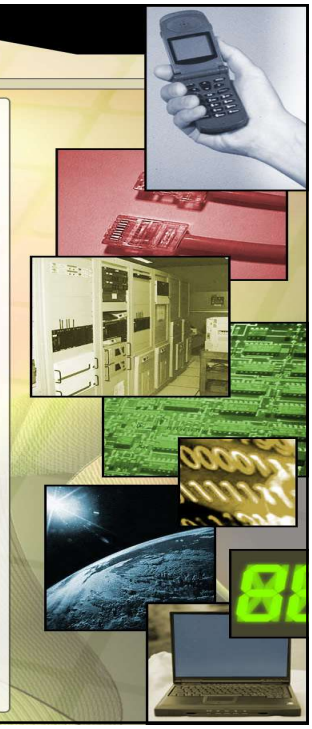
HÀM(FUNCTION)

- **Thay đổi hàm**

```
ALTER FUNCTION < Tên_thủ_tục>
   ( [
      @tham_số_1  Kiểu_dữ_liệu_1 [ OUTPUT ] ,
      @tham_số_2  Kiểu_dữ_liệu_2 [ OUTPUT ] ,
   …
      ] )
RETURNS …
```

- **Xóa hàm**

```
DROP FUNCTION < Tên_hàm> ;
```



276

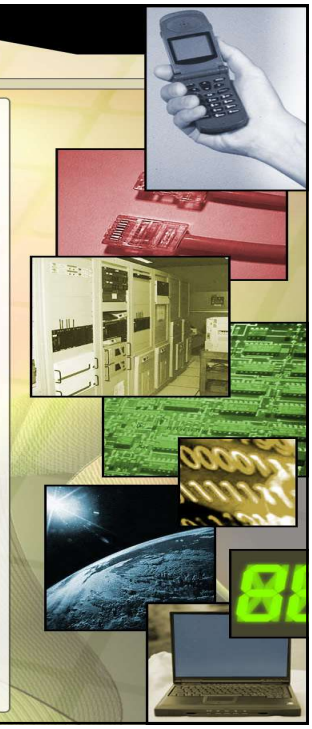
HÀM(FUNCTION)

- **Các hàm chuỗi cơ bản**
  - *Đổi một số thành chuỗi*

```
STR(Số_thực, Số_ký_tự [, Số_lẻ])
```

Ví dụ:

```
select str(12345.6789,8,2);
```



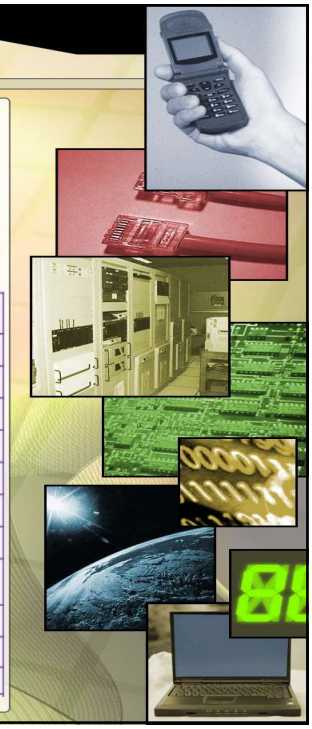
277

HÀM(FUNCTION)

- **Các hàm chuỗi cơ bản**
  - *Đổi kiểu dữ liệu và định dạng*

```
CONVERT(Kiểu_dữ_liệu, Biểu_thức [, Định_dạng])
```

yy	yyyy	Chuỗi kết quả
-	0 hoặc 100	mon dd yyyy hh:miAM (or PM)
1	101	mm/dd/yyyy
2	102	yy.mm.dd
3	103	dd/mm/yy
5	105	dd-mm-yy
6	106	dd mon yy
8	108	hh:mm:ss
9	109	mon dd yyyy hh:mi:ss:mmmAM (or PM)
10	110	mm-dd-yy
11	111	yy/mm/dd
12	112	yymmdd



278

HÀM(FUNCTION)

- **Các hàm chuỗi cơ bản**
  - *Cắt chuỗi*

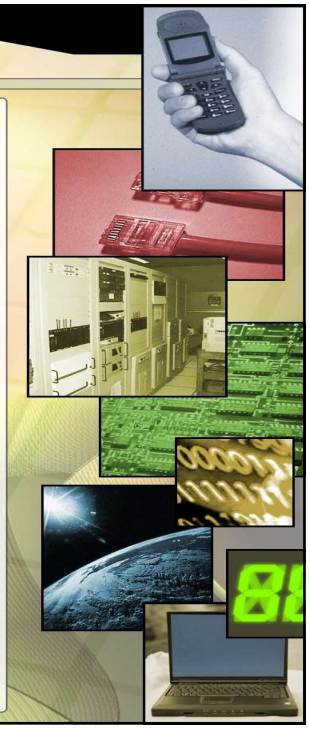
```
LEFT(Chuỗi_nguồn, Số_ký_tự)
RIGHT(Chuỗi_nguồn, Số_ký_tự)
SUBSTRING(Chuỗi_nguồn, Vị_trí, Số_ký_tự)
```

  - *Tính chiều dài chuỗi*

```
LEN(Chuỗi)
```

  - *Tìm và thay thế chuỗi*

```
REPLACE(Chuỗi_nguồn, Chuỗi_tìm, Chuỗi_thay_thế)
```



279

HÀM(FUNCTION)

- **Các hàm chuỗi cơ bản**
  - *Cắt khoảng trắng*

```
LTRIM(Chuỗi_nguồn, Số_ký_tự)
RTRIM(Chuỗi_nguồn, Số_ký_tự)
```

  - *Tạo chuỗi khoảng trắng*

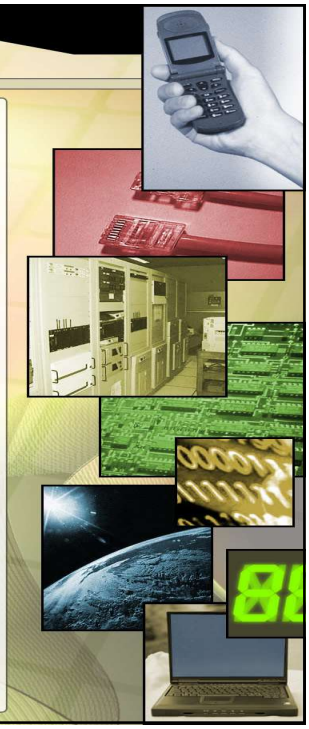
```
SPACE(Số_ký_tự)
```

  - *Đảo chuỗi*

```
REVERSE(Chuỗi)
```

  - *Đổi số mã ASCII thành ký tự và ngược lại*

```
CHAR(Số)      ASCII(Ký_tự)
```



280

HÀM(FUNCTION)

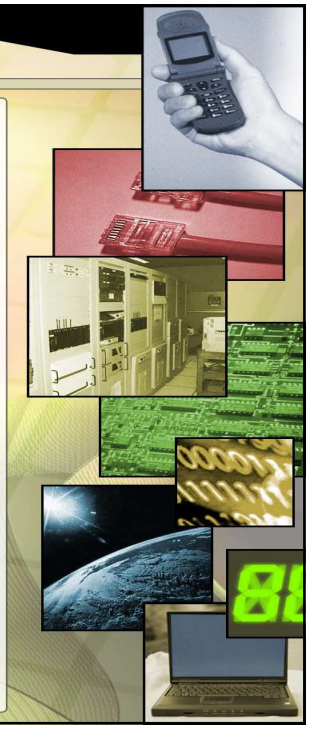
- **Các hàm ngày giờ**
  - *Lấy thời điểm hiện hành*

    `GETDATE()`
  - *Tính số chênh lệch giữa 2 mốc thời gian*

    `DATEDIFF(Đơn_vị, Thời_gian_1, Thời_gian_2)`
  - *Lấy tên đơn vị thời gian (cho kiểu chuỗi)*

    `DATENAME(Đơn_vị, Thời_gian)`
  - *Lấy đơn vị thời gian (cho kiểu số)*

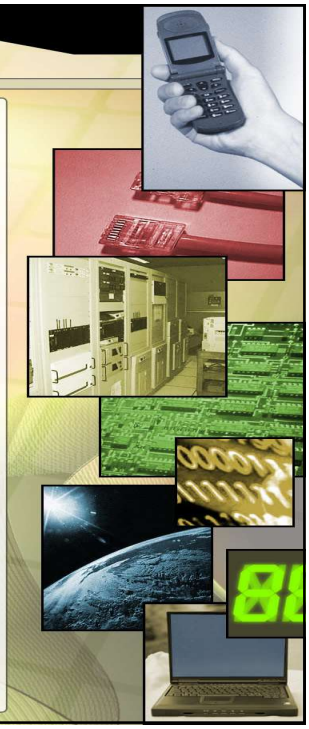
    `DATEPART(Đơn_vị, Thời_gian)`



281

HÀM(FUNCTION)

Đơn_vị (Thành phần của ngày)	Chữ viết tắt
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw
hour	hh
minute	mi, n
second	ss, s
millisecond	ms



282

HÀM(FUNCTION)

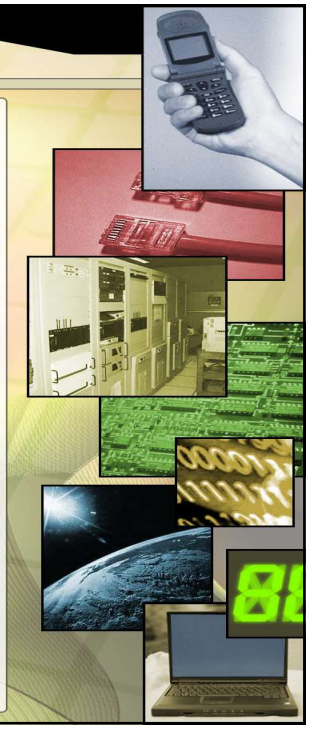
- **Các hàm toán học (số)**
  - *Làm tròn số*

    ```
    ROUND(Biểu_thức_số, Vị_trí_làm_tròn)
    ```

  - *Lấy số ngẫu nhiên >0 và <1*

    ```
    RAND([seed])
    ```

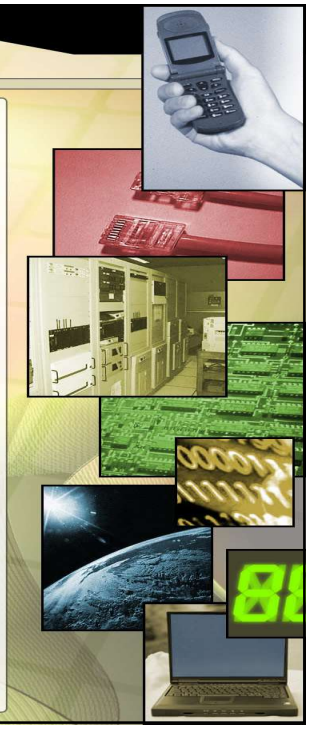
    (Nếu có số seed thì số ngẫu nhiên sẽ cố định)



283

TRIGGER

- Là thủ tục đặc biệt tự động thực hiện khi có sự kiện diễn ra trong CSDL server
- Có 3 loại Trigger: DML, DDL, Logon
  - DML Trigger: xảy ra khi có lệnh INSERT, UPDATE, DELETE trên bảng hoặc view.
  - DDL Trigger: thực thi khi xảy ra lệnh CREATE, ALTER hoặc DROP.
  - Logon Trigger: xảy ra khi session người dùng được thiết lập.



284

DML TRIGGER

- SQL Server cung cấp 2 bảng ảo dành riêng cho trigger tên là INSERTED và DELETED, hai bảng này sẽ lưu trữ dữ liệu của các row trước hoặc sau khi hành động xảy ra.

Event	INSERTED	DELETED
INSERT	Dữ liệu của row vừa insert	Rỗng
UPDATE	Dữ liệu mới của row vừa update	Dữ liệu cũ của row vừa update
DELETE	Rỗng	Dữ liệu của row bị xóa

285

DML TRIGGER

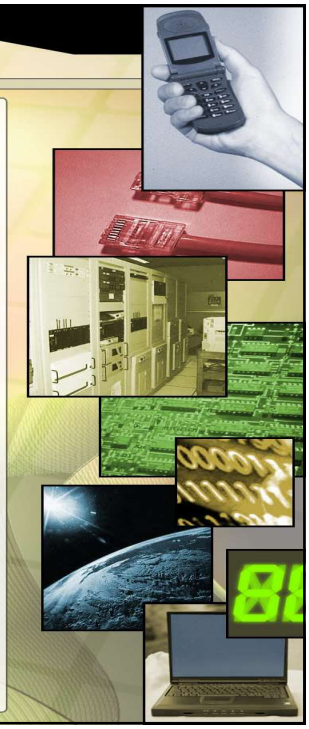
```
CREATE [ OR ALTER ] TRIGGER [ tên_schema . ] tên_trigger
ON tên_table | tên_view
FOR | AFTER | INSTEAD OF
[ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ]
AS
   BEGIN
   …
   END ;
```

286

DML TRIGGER

Ví dụ:

```
/* cập nhật hàng trong kho sau khi đặt hàng */
CREATE TRIGGER trg_DatHang ON tbl_DatHang
AFTER INSERT AS
BEGIN
   UPDATE tbl_KhoHang
   SET SoLuongTon = SoLuongTon - (
            SELECT SoLuongDat
            FROM inserted
            WHERE MaHang = tbl_KhoHang.MaHang
   )
   FROM tbl_KhoHang JOIN inserted ON
               tbl_KhoHang.MaHang = inserted.MaHang
END
```

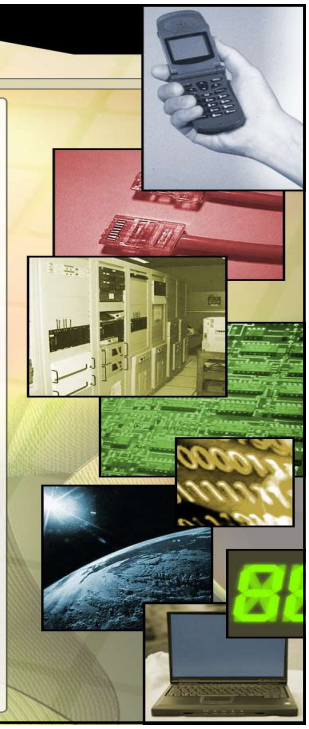


287

DML TRIGGER

Ví dụ:

```
/* Gửi mail sau khi thao tác trên CSDL */
CREATE TRIGGER reminder2 ON Sales.Customer
AFTER INSERT, UPDATE, DELETE
AS
   EXEC msdb.dbo.sp_send_dbmail
        @profile_name = 'James Bond',
        @recipients = 'jbond@mi5.com',
        @body = 'Mission completed',
        @subject = 'Reminder';
```

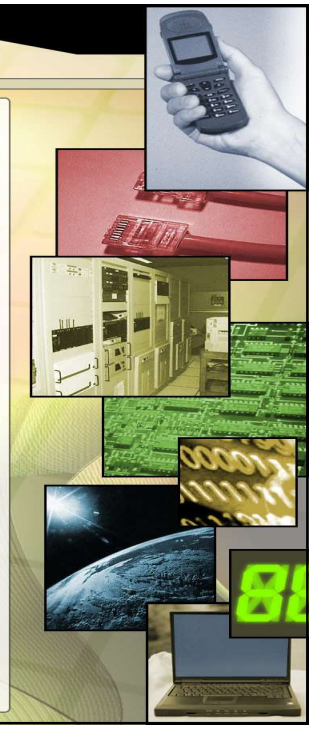


288

DML TRIGGER

Ví dụ:

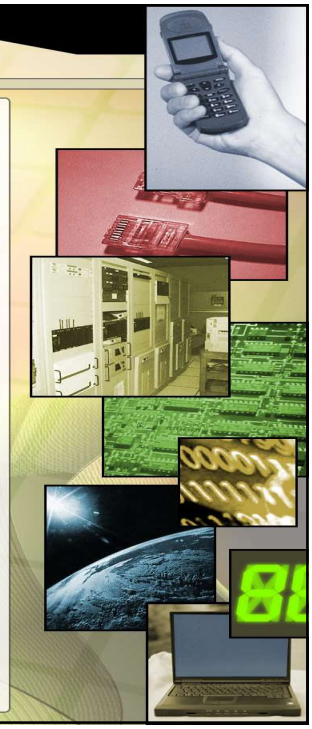
```
CREATE TRIGGER trg_InsteadOfInsert ON tbl_Origin
INSTEAD OF INSERT
AS
INSERT INTO tbl_MyTable VALUES
((SELECT TOP 1  inserted.ID FROM inserted), N'Đã thêm')
```



289

DDL TRIGGER

```
CREATE [ OR ALTER ] TRIGGER tên_trigger
ON ALL SERVER | DATABASE
FOR | AFTER tên_sự_kiện [ , ...n ]
AS
   BEGIN
        …
   END ;
```

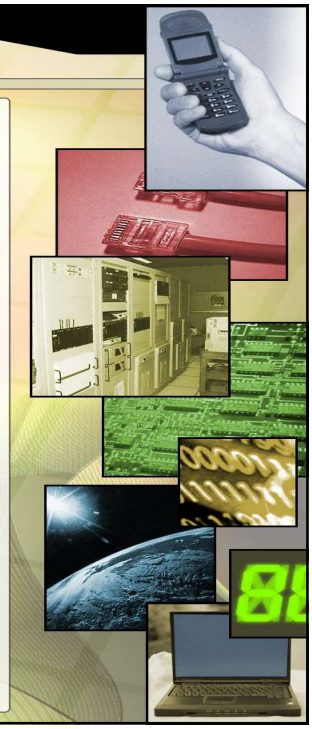


290

DDL TRIGGER

Ví dụ:

```
CREATE TABLE index_logs (
    log_id INT IDENTITY PRIMARY KEY,
    event_data XML NOT NULL,
    changed_by SYSNAME NOT NULL
);
```

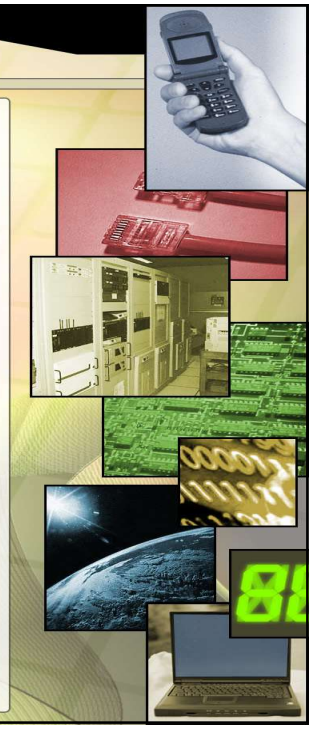


291

DDL TRIGGER

Ví dụ:

```
CREATE TRIGGER trg_index_changes
ON DATABASE
FOR
    CREATE_INDEX, ALTER_INDEX, DROP_INDEX
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO index_logs(event_data, changed_by)
    VALUES (EVENTDATA(), USER);
END;
```

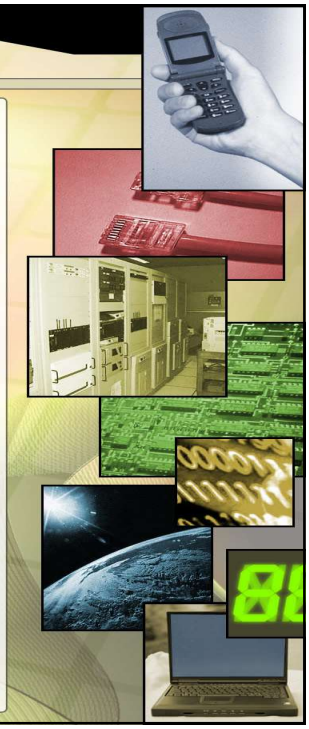


292

DDL TRIGGER

Ví dụ:

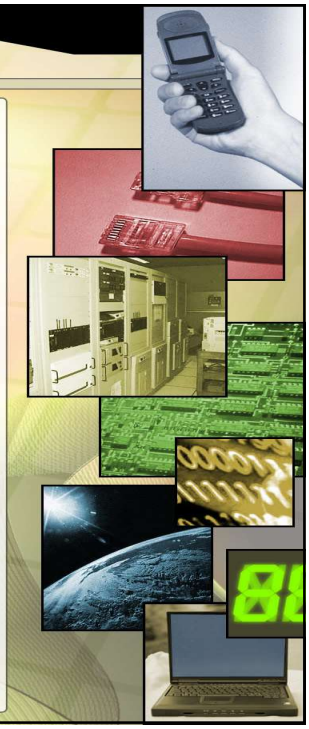
```
CREATE TRIGGER trRestrictDDLEvents
ON DATABASE
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
   PRINT 'You cannot create, alter or drop a table'
   ROLLBACK TRANSACTION
END
```



293

LOGON TRIGGER

```
CREATE [ OR ALTER ] TRIGGER tên_trigger
ON ALL SERVER [ WITH EXECUTE AS tên_login ]
FOR | AFTER  LOGON
AS
  BEGIN
     …
  END ;
```

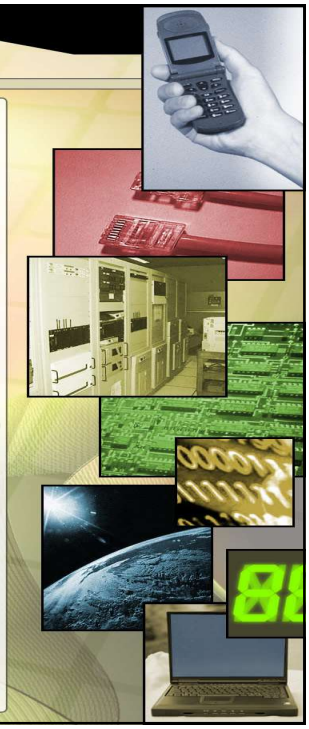


294

LOGON TRIGGER

Ví dụ: Giới hạn giờ đăng nhập tài khoản test_user từ *10AM* đến *6PM*

```
CREATE TRIGGER Logon_from_10AM_to_6PM
ON ALL SERVER
FOR LOGON
AS
BEGIN
  IF ((ORIGINAL_LOGIN() = 'test_user') and
  (   (DATEPART(hour, GETDATE()) between 18 and 24)
   or (DATEPART(hour, GETDATE()) between 0 and 9)
  ))
     BEGIN
        ROLLBACK
     END
END
```

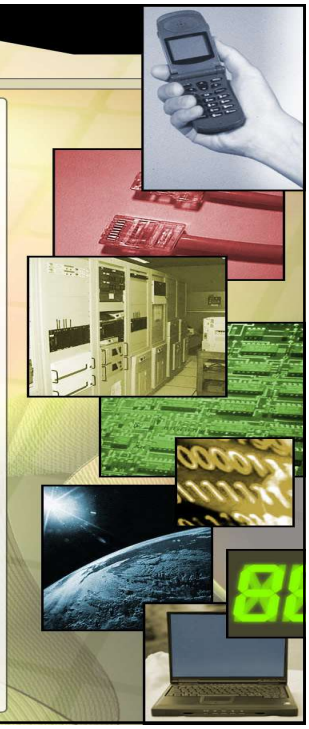


295

LOGON TRIGGER

Ví dụ: Giới hạn tài khoản đăng nhập vào server

```
CREATE TRIGGER Prevent_login
ON ALL SERVER WITH EXECUTE AS 'sa'
FOR LOGON
AS
BEGIN
  DECLARE @LoginName sysname
  DECLARE @LoginType sysname

  SET @LoginName = ORIGINAL_LOGIN()
  IF(@LoginName NOT IN ('sa', 'son'))
  BEGIN
    ROLLBACK; --Disconnect the session
  END
END
```

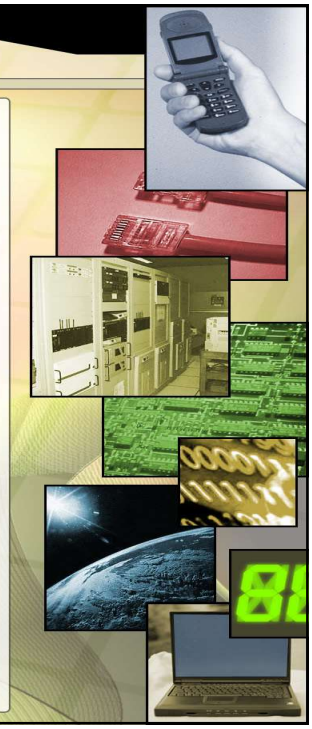


296

LOGON TRIGGER

Ví dụ: Giới hạn ứng dụng đăng nhập vào server

```
CREATE TRIGGER Prevent_login
ON ALL SERVER WITH EXECUTE AS 'sa'
FOR LOGON
AS
BEGIN
  DECLARE @AppName varchar(max)
  DECLARE @LoginName sysname
  DECLARE @LoginType sysname

  SET @AppName = APP_NAME()
  SET @LoginName = ORIGINAL_LOGIN()
  IF NOT (@LoginName = 'son' AND @AppName like 'SQLCMD')
  BEGIN
    ROLLBACK;
  END
END
```

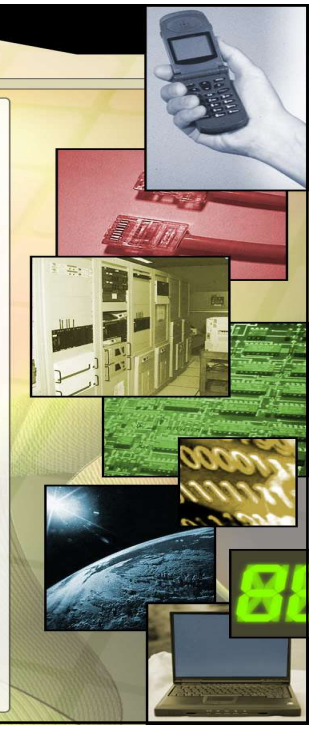


297

LOGON TRIGGER

Ví dụ: Giới hạn kết nối server

```
CREATE TRIGGER Limit_Connection
ON ALL SERVER
FOR LOGON
AS
BEGIN
  IF(
      select COUNT(*) from sys.dm_exec_sesions
      where is_user_process = 1
  ) > 5
  BEGIN
    ROLLBACK; --Disconnect the session
  END
END
```



298

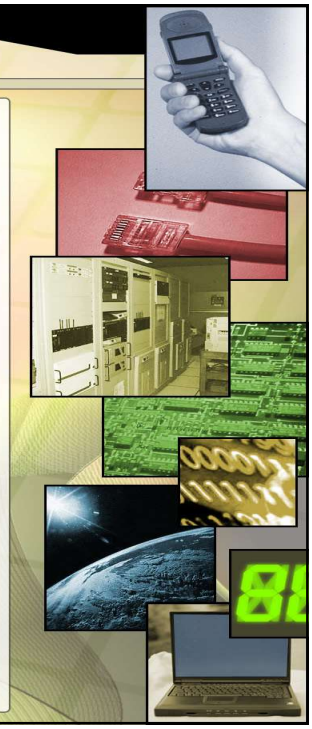
TRIGGER

- **Kích hoạt | vô hiệu hóa Trigger**

  ENABLE | DISABLE TRIGGER [*tên_schema* .] tên_trigger [, ...n] | ALL ON tên_đối_tượng | DATABASE | ALL SERVER [;]

- **Xóa Trigger**

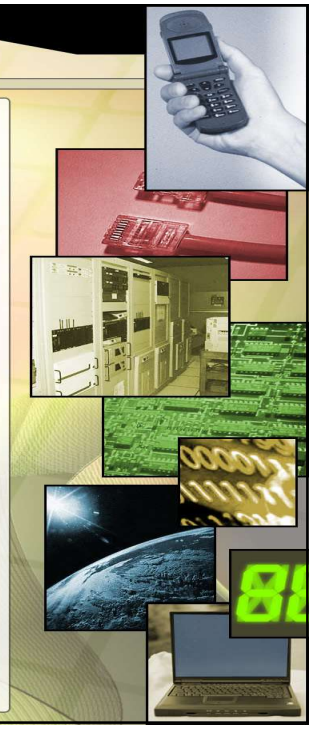
  DROP TRIGGER [IF EXISTS] [*tên_schema* .] tên_trigger [, ...n] ON DATABASE | ALL SERVER [;]



299

TRANSACTION

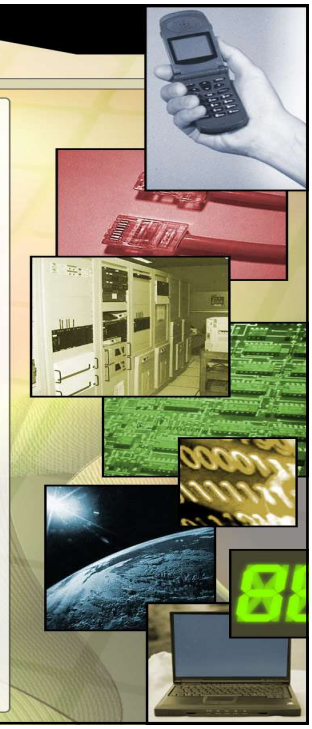
- Transaction được dùng để đảm bảo tính toàn vẹn dữ liệu khi xảy ra cập nhật (INSERT, UPDATE, DELETE…), ngăn chặn tình huống dữ liệu được cập nhật nửa chừng.
- Cấu trúc transaction
  - *Bắt đầu transaction*: **begin tran | begin transaction**
  - *Kết thúc transaction*: **commit | commit tran | commit transaction**



300

TRANSACTION

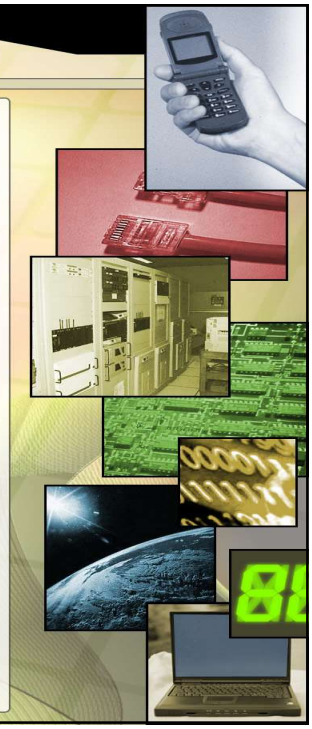
- *Quay lui transaction (Rollback transaction)*:
  rollback | rollback tran | rollback transaction
- *Đánh dấu một savepoint trong transaction*:
  save transaction tên_savepoint



301

TRANSACTION

- Lệnh **rollback tên_savepoint** có tác dụng quay lui (rollback) giao dịch đến vị trí đặt savepoint tương ứng (không có tác dụng kết thúc transaction).
- Biến **@@trancount**: cho biết số transaction hiện đang thực hiện (chưa được kết thúc với rollback hay commit)

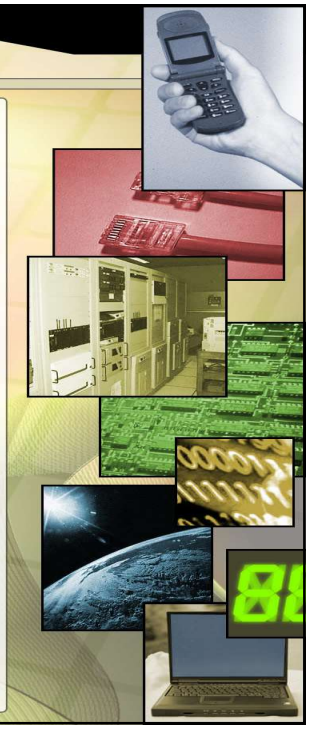


302

TRANSACTION

- Cú pháp

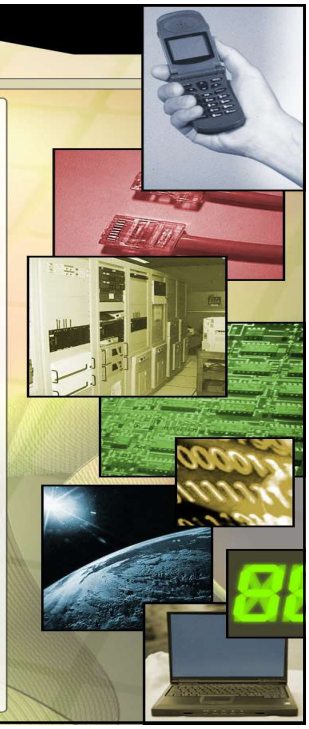
```
SET XACT_ABORT ON
BEGIN TRAN
    BEGIN TRY
            …
      COMMIT
    END TRY
    BEGIN CATCH
      ROLLBACK
      DECLARE @ErrorMessage VARCHAR(2000)
      SELECT @ErrorMessage = ERROR_MESSAGE()
      RAISERROR(@ErrorMessage, 16, 1)
    END CATCH
```



303

GO

- SQL Server thực thi các lệnh theo từng nhóm lệnh (batch)
- Các nhóm lệnh phân cách bởi lệnh GO
- Các lệnh nằm đầu gói và không kết hợp với các lệnh khác: **CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE RULE, CREATE SCHEMA, CREATE TRIGGER, CREATE VIEW**
- Các lệnh trong gói lệnh phải phù hợp với thứ tự logic

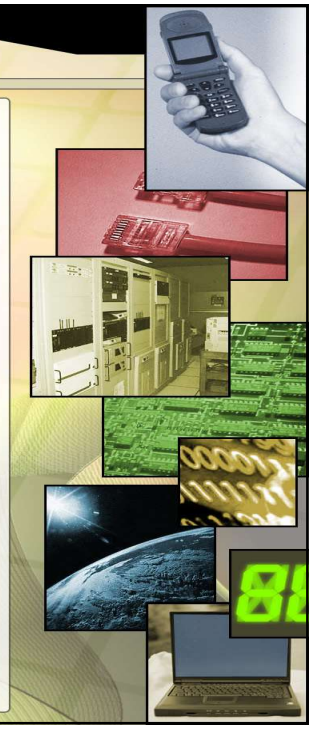


304

GO

Ví dụ:

```
use SON
IF OBJECT_ID('cities') IS NOT NULL
   DROP TABLE cities;
CREATE TABLE cities (
   name VARCHAR(90) PRIMARY KEY
)
GO
DROP TABLE cities
```



305