



# TÀI LIỆU HƯỚNG DẪN GIẢNG DẠY



## CHƯƠNG TRÌNH KỸ THUẬT VIÊN

### NGÀNH LẬP TRÌNH

#### Học phần 4

### SQL SERVER **2000**



# Mục lục

<b>MỤC LỤC</b>	<b>1</b>
<b>GIỚI THIỆU</b>	<b>6</b>
<b>GIÁO TRÌNH LÝ THUYẾT</b>	<b>7</b>
<b>TÀI LIỆU THAM KHẢO</b>	<b>7</b>
<b>HƯỚNG DẪN PHẦN LÝ THUYẾT</b>	<b>8</b>
<b>Bài 1 TỔNG QUAN VỀ MICROSOFT SQL SERVER</b>	<b>8</b>
I. Mô hình khách chủ (Client/Server)	9
I.1. Khái niệm về cấu trúc vật lý	9
I.2. Khái niệm về các xử lý	10
I.3. Vì sao phát triển ứng dụng mô hình khách chủ?	11
II. Microsoft SQL Server là gì?	12
II.1. Lịch sử ra đời Microsoft SQL Server	12
II.2. Cài đặt cơ sở dữ liệu SQL Server Desktop	13
III. Các tiện ích trong Microsoft SQL Server	21
III.1. Tiện ích Book Online	21
III.2. Tiện ích Client NetWork Utility	22
III.3. Tiện ích Enterprise Manager	23
III.4. Tiện ích Import and Export Data	23
III.5. Tiện ích Profiler	24
III.6. Tiện ích Query Analyzer	25
III.7. Tiện ích Server Network Utility	27
III.8. Tiện ích Service Manager	28
III.9. Định nghĩa cấu hình nối kết vào SQL	28
III.10. Nối kết từ Query Analyzer vào SQL	37
<b>Bài 2 CÁC ĐỐI TƯỢNG TRONG CƠ SỞ DỮ LIỆU</b>	<b>39</b>
I. Cơ sở dữ liệu của SQL Server	40
I.1. Khái niệm về cơ sở dữ liệu	40
I.2. Các tập tin vật lý lưu trữ cơ sở dữ liệu	41
I.3. Tạo mới cơ sở dữ liệu	42
I.4. Xóa cơ sở dữ liệu đã có	45
II. Bảng dữ liệu (Table)	46
II.1. Khái niệm về bảng	46
II.2. Các thuộc tính của bảng	47
II.3. Tạo cấu trúc bảng dữ liệu	49



II.4. Tạo cấu trúc bảng đơn giản .....	51
II.5. Xóa cấu trúc bảng .....	52
II.6. Tạo cấu trúc bảng có cột định danh .....	52
II.7. Thay đổi cấu trúc bảng .....	53
II.8. Thêm một cột mới trong bảng .....	53
II.9. Hủy bỏ cột hiện có bên trong bảng .....	54
II.10. Sửa đổi kiểu dữ liệu của cột .....	54
II.11. Đổi tên cột, tên bảng dữ liệu .....	55
III. Kiểu dữ liệu do người dùng định nghĩa .....	56
III.1. Khái niệm .....	56
III.2. Tạo mới kiểu dữ liệu do người dùng định nghĩa .....	56
III.3. Xóa kiểu dữ liệu do người dùng định nghĩa .....	58
IV. Bảng ảo (Virtual table – View) .....	60
IV.1. Khái niệm về bảng ảo .....	60
IV.2. Tạo bảng ảo bằng tiện ích Enterprise Manager .....	60
IV.3. Xem và cập nhật dữ liệu bảng ảo .....	63
IV.4. Cập nhật dữ liệu qua bảng ảo sử dụng trigger INSTEAD OF .....	64
IV.5. Hủy bỏ bảng ảo .....	64
IV.6. Tạo mới bảng ảo bằng lệnh CREATE VIEW .....	65
IV.7. Sửa đổi nội dung bảng ảo .....	69
<b>Bài 3 CÁC RÀNG BỘC TOÀN VỆN DỮ LIỆU .....</b>	<b>70</b>
I. Các ràng buộc toàn vẹn dữ liệu (Constraint) .....	71
I.1. Các quy định của công việc trong thực tế .....	71
I.2. Các ràng buộc toàn vẹn dữ liệu .....	72
I.3. Sử dụng constraint để kiểm tra toàn vẹn dữ liệu .....	73
II. Mô hình quan hệ dữ liệu (Diagram) .....	86
II.1. Khái niệm về mô hình quan hệ dữ liệu .....	86
II.2. Tạo mới mô hình quan hệ dữ liệu .....	87
II.3. Các chức năng trong mô hình quan hệ dữ liệu .....	90
III. Quy tắc kiểm tra miền giá trị dữ liệu (Rule) .....	92
III.1. Khái niệm .....	92
III.2. Tạo mới quy tắc kiểm tra miền giá trị dữ liệu .....	92
III.3. Áp dụng quy tắc kiểm tra miền giá trị dữ liệu .....	95
III.4. Xóa quy tắc kiểm tra miền giá trị dữ liệu .....	98
IV. Giá trị mặc định (Default) .....	99
IV.1. Khái niệm .....	99
IV.2. Tạo mới giá trị mặc định .....	100
IV.3. Liên kết giá trị mặc định vào cột dữ liệu .....	101
V. Xóa giá trị mặc định .....	105



<b>Bài 4 LẬP TRÌNH VỚI CƠ SỞ DỮ LIỆU .....</b>	<b>107</b>
I. Biến cục bộ .....	108
I.1. Khai báo biến cục bộ .....	108
I.2. Gán giá trị cho biến .....	109
I.3. Xem giá trị hiện hành của biến .....	110
I.4. Phạm vi hoạt động của biến .....	111
II. Biến hệ thống .....	113
II.1. Ý nghĩa sử dụng .....	113
II.2. Một vài biến hệ thống thường dùng .....	114
III. Các toán tử .....	114
III.1. Toán tử số học .....	114
III.2. Toán tử nối chuỗi .....	115
III.3. Toán tử so sánh .....	116
III.4. Toán tử luận lý .....	116
IV. Các câu lệnh truy vấn dữ liệu .....	117
IV.1. Lệnh SELECT FROM .....	117
IV.2. Truy vấn con .....	132
IV.3. Lệnh INSERT INTO .....	137
IV.4. Lệnh DELETE FROM .....	139
IV.5. Lệnh UPDATE SET .....	141
IV.6. Biểu thức CASE .....	143
V. Cấu trúc điều khiển .....	147
V.1. Cấu trúc rẽ nhánh IF...ELSE .....	147
V.2. Cấu trúc lặp WHILE .....	150
VI. Sử dụng biến kiểu dữ liệu cursor .....	153
VI.1. Khái niệm về cursor .....	154
VI.2. Các bước sử dụng kiểu dữ liệu cursor .....	154
VII. Các hàm thường dùng .....	162
VII.1. Các hàm chuyển đổi kiểu dữ liệu .....	162
VII.2. Các hàm ngày giờ .....	165
VII.3. Các hàm toán học .....	170
VII.4. Các hàm xử lý chuỗi .....	173
<b>Bài 5 THỦ TỤC NỘI TẠI .....</b>	<b>181</b>
I. Khái niệm về thủ tục nội tại .....	182
I.1. Thủ tục nội tại là gì? .....	182
I.2. Các thủ tục nội tại hệ thống .....	182
I.3. Các lợi ích khi sử dụng thủ tục nội tại .....	183
II. Các hành động cơ bản với thủ tục nội tại .....	183
II.1. Tạo mới một thủ tục nội tại .....	183
II.2. Gọi thực hiện thủ tục nội tại .....	186



II.3. Hủy bỏ thủ tục nội tại.....	187
II.4. Thay đổi nội dung của thủ tục nội tại .....	187
III. Tham số bên trong thủ tục nội tại .....	188
III.1. Tham số đầu vào .....	188
III.2. Tham số đầu ra .....	190
IV. Một số vấn đề khác trong thủ tục nội tại .....	192
IV.1. Mã hóa nội dung thủ tục.....	192
IV.2. Biên dịch thủ tục .....	193
IV.3. Thủ tục lồng nhau.....	194
IV.4. Sử dụng lệnh RETURN trong thủ tục .....	195
IV.5. Sử dụng bảng tạm trong thủ tục.....	197
IV.6. Tham số kiểu cursor bên trong thủ tục .....	199
IV.7. Thủ tục cập nhật bảng dữ liệu.....	200
IV.8. Thủ tục hiển thị dữ liệu.....	202
V. Giao tác (Transaction) .....	204
V.1. Khái niệm về giao tác.....	204
V.2. Giao tác không tường minh.....	204
V.3. Giao tác tường minh .....	205
<b>Bài 6 HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA .....</b>	<b>212</b>
I. Khái quát về hàm do người dùng định nghĩa .....	213
II. Làm việc với UDF .....	213
II.1. Tạo mới UDF.....	213
II.2. Quản lý UDF.....	217
III. Các thao tác trên UDF .....	219
III.1. Gọi thực hiện các UDF thuộc loại hàm đơn trị .....	219
III.2. Sử dụng các UDF thuộc loại hàm đọc bảng .....	223
III.3. Sử dụng các UDF thuộc loại hàm tạo bảng.....	223
<b>Bài 7 TRIGGER .....</b>	<b>225</b>
I. Khái quát về trigger .....	226
I.1. Trigger là gì?.....	226
I.2. Các xử lý bên trong trigger .....	226
I.3. Các hạn chế trên trigger .....	228
I.4. Các loại trigger .....	228
I.5. Các bảng trung gian Inserted và Deleted.....	228
II. Làm việc với trigger .....	229
II.1. Tạo mới trigger.....	229
II.2. Xóa trigger.....	233
II.3. Sửa nội dung trigger.....	233
II.4. Trigger lồng nhau .....	234
III. Trigger kiểm tra ràng buộc dữ liệu.....	235



III.1. Khi thêm mới mẫu tin.....	235
III.2. Khi hủy bỏ mẫu tin .....	237
III.3. Khi sửa đổi mẫu tin .....	239
IV. Trigger cập nhật giá trị tự động.....	242
IV.1. Khi thêm mới mẫu tin.....	243
IV.2. Khi hủy bỏ mẫu tin .....	245
IV.3. Khi sửa đổi mẫu tin .....	247
IV.4. Instead of trigger và cập nhật dữ liệu trên bảng ảo .....	249

#### **ĐỀ THI MẪU CUỐI MÔN**

#### **ĐỀ KIỂM TRA GIÁO VIÊN**



## GIỚI THIỆU

Sau khi hoàn thành khóa học, học viên sẽ có khả năng:

- Tạo lập cơ sở dữ liệu quan hệ (CSDL) có các thành phần chính như: các bảng (table), các ràng buộc toàn vẹn (constraint) trên CSDL, các bảng ảo (view), ...
- Thực hiện các thao tác cập nhật (thêm/sửa/hủy) dữ liệu trên bảng
- Thực hiện các câu truy vấn có chọn lọc, có nhóm, có thống kê dữ liệu.
- Lập trình với cơ sở dữ liệu bằng ngôn ngữ lập trình T-SQL để viết các xử lý tính toán, các xử lý kiểm tra tính đúng đắn của dữ liệu, ...
- Lập trình Visual basic (VB) kết nối với CSDL trên SQL Server.

Với thời lượng là 36 tiết LT và 60 tiết TH được phân bổ như sau:

STT	Bài học	Số tiết LT	Số tiết TH
1	Tổng quan về microsoft sql server	2	4
2	Các đối tượng trong cơ sở dữ liệu	4	12
3	Các ràng buộc toàn vẹn dữ liệu	6	6
4	Lập trình với cơ sở dữ liệu	6	8
5	Thủ tục nội tại	8	12
6	Hàm do người dùng định nghĩa	4	4
7	Trigger	6	14

Tổng số tiết :                      36                      60



## **GIÁO TRÌNH LÝ THUYẾT**

Sử dụng giáo trình “SQL Server 2000” của tác giả Nguyễn Thiện Tâm, Trần Xuân Hải, xuất bản lần thứ 1, nhà xuất bản Đại Học Quốc Gia Tp. HCM.

## **TÀI LIỆU THAM KHẢO**





# HƯỚNG DẪN PHẦN LÝ THUYẾT

## Bài 1

# TỔNG QUAN VỀ MICROSOFT SQL SERVER

### Tóm tắt

Lý thuyết 2 tiết - Thực hành 4 tiết

Mục tiêu	Các mục chính	Bài tập bắt buộc	Bài tập làm thêm
<ul style="list-style-type: none"><li>✓ Hướng dẫn học viên cài đặt SQL Server</li><li>✓ Định nghĩa cấu hình kết nối cho SQL Server</li><li>✓ Đăng ký quản trị SQL Server</li><li>✓ Kết nối Query Analyzer vào SQL Server</li></ul>	<ul style="list-style-type: none"><li>I. Mô hình khách chủ (Client/Server)</li><li>II. Microsoft SQL Server là gì?</li><li>III. Các tiện ích trong Microsoft SQL Server</li></ul>	<ul style="list-style-type: none"><li>1.1</li><li>1.2</li><li>1.3</li></ul>	<ul style="list-style-type: none"><li>1.4</li></ul>

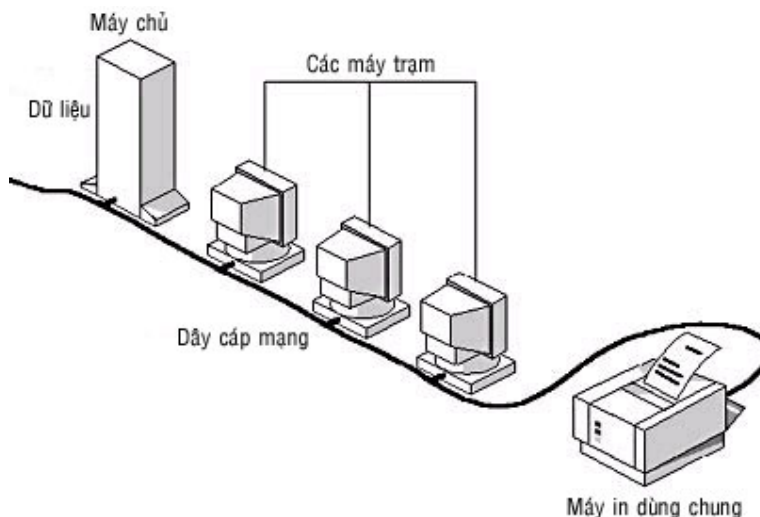
# I. Mô hình khách chủ (Client/Server)

Trước khi tìm hiểu bên trong hệ quản trị cơ sở dữ liệu quan hệ Microsoft SQL Server, chúng tôi muốn trình bày cho các bạn hiểu biết sơ bộ về mô hình khách chủ. Khác với các cơ sở dữ liệu khá đơn giản dễ hiểu và dễ dàng nắm bắt trước đây mà các bạn có thể đã làm quen như: Dbase, FoxBase, FoxPro, Microsoft Access ... thì Microsoft SQL Server là một hệ quản trị cơ sở dữ liệu quan hệ có khả năng hoạt động theo mô hình khách chủ rất mạnh. Vậy mô hình khách chủ là gì?

Mô hình khách chủ (client/server) được xây dựng với mục tiêu nâng cao khả năng khai thác hiệu quả các tài nguyên phần cứng và phần mềm trong một hệ thống thông tin. Hai tính năng quan trọng nhất của mô hình khách chủ là **chia sẻ tài nguyên** và **phối hợp xử lý**. Với việc chia sẻ tài nguyên, một hệ thống sẽ giảm bớt số lượng các thiết bị phần cứng và việc cài đặt các phần mềm. Nếu như việc chia sẻ các thiết bị phần cứng chỉ giảm bớt chi phí thì chia sẻ các tài nguyên phần mềm ngoài việc giảm chi phí còn giúp việc quản trị như nâng cấp hệ thống, bảo mật dữ liệu, chia sẻ đồng bộ dữ liệu được thực hiện đơn giản hơn. Khả năng phối hợp xử lý trong một mô hình khách chủ là khả năng một công việc có thể được chia ra và xử lý trên nhiều máy tính khác nhau để tốc độ thực hiện được nhanh hơn.

## I.1. Khái niệm về cấu trúc vật lý

Yếu tố căn bản đầu tiên trong mô hình khách chủ là phải có một hệ thống mạng các máy tính được nối kết chung với nhau theo một cách nào đó. Trong phần giáo trình này chúng tôi không muốn trình bày sâu vào chi tiết các kỹ thuật liên quan đến hệ thống mạng máy tính.



**Hình 1-1. Mô hình mạng máy tính đơn giản.**

Theo hình 1-1 bên trên, mạng máy tính là một hệ thống cho phép nối kết hai hoặc nhiều máy tính bằng một sợi cáp với mục đích cùng sử dụng chung các tài nguyên, dữ liệu giữa các máy tính. Trong mô hình này chúng ta sẽ có các khái niệm cơ bản như sau:

- ✓ **Máy chủ** (Server): Trước những năm 1990 những máy tính lớn (main frame) được sử dụng làm máy chủ, tuy nhiên ngày nay các máy tính cá nhân (personnel computer) vẫn được sử dụng như là một máy chủ. Một máy chủ phải có bộ xử lý với tốc độ cao (CPU), tài nguyên lớn (RAM, HardDisk) để hoạt động được tốt bởi vì cùng lúc sẽ có nhiều người dùng mạng truy xuất về máy chủ thông qua các máy trạm. Trong một hệ thống mạng máy tính được



phép có nhiều máy chủ với các chức năng độc lập riêng biệt khác nhau.

- ✓ **Máy trạm (Client):** là các máy tính được phép truy xuất các tài nguyên đã được chia sẻ trên mạng. Theo hình 1-1 các máy trạm sẽ được truy xuất phần dữ liệu dùng chung trên máy chủ và in ấn các tài liệu trên máy in chung.
- ✓ **Dây cáp mạng (Cable hoặc Media):** là một hệ thống dây cáp nối kết vật lý các máy tính, máy in lại với nhau.
- ✓ **Dữ liệu chung (Shared data):** là các tập tin, thư mục mà người sử dụng trong hệ thống mạng có thể truy xuất vào máy chủ từ các máy trạm.

## 1.2. Khái niệm về các xử lý

Trong mô hình khách chủ, các máy trạm hay khách sẽ gửi các yêu cầu xử lý tới máy chủ để máy chủ thực hiện và trả về kết quả. Một máy chủ có thể phải xử lý nhiều yêu cầu đến từ các máy trạm khác nhau. Để có một hệ thống khách chủ, ngoài hệ thống mạng với các máy tính được kết nối với nhau thì cần phải có hệ thống phần mềm đi kèm theo. Máy tính được cài phần mềm server để có thể thực hiện các xử lý các yêu cầu sẽ được gọi là máy chủ. Ví dụ, máy tính cài phần mềm SQL Server sẽ được gọi là máy chủ SQL Server. Các máy tính trạm muốn gửi được yêu cầu tới máy chủ sẽ cần phải có phần mềm client tương ứng. Các phần mềm hoạt động trên máy trạm có nhiều tên gọi khác nhau như client, desktop, workstation. Ví dụ trong một hệ thống SQL Server, máy trạm sẽ cần được cài SQL Client tools để có thể tương tác với máy SQL Server. Bạn cũng nên biết rằng tương tự như việc máy chủ có thể xử lý các yêu cầu tới từ nhiều máy trạm khác nhau, một máy trạm cũng có thể gửi yêu cầu tới những máy chủ khác nhau. Khả năng này giúp phòng ngừa hệ thống khách chủ bị ngừng hoạt động khi một máy chủ bị lỗi và không còn làm việc được nữa.

Trên một hạ tầng mạng gồm có các máy tính kết nối với nhau người ta có thể có nhiều hệ thống khách chủ làm việc đồng thời. Ví dụ, trên một mạng thường có các hệ thống mail, intranet, Internet và các phần mềm hoạt động theo mô hình khách chủ cùng làm việc. Với hệ thống mail ta có Mail server, intranet ta có Web server, Internet ta có firewall/proxy server và một hệ thống phần mềm làm việc theo mô hình khách chủ thường phải có database server và có thể có thêm application server. SQL Server là một phần mềm dành cho các máy database server và trong phạm vi môn học này, ta chỉ xét tới các ứng dụng làm việc theo mô hình khách chủ.

Có nhiều người suy nghĩ rằng mô hình khách chủ chỉ là việc tổ chức cơ sở dữ liệu trên một nơi dùng chung và cho phép nhiều người dùng trên mạng truy cập từ các máy trạm. Theo chúng tôi điều này chỉ đúng một phần mà thôi. Ngoài ra chúng ta cần phải phân biệt rõ ràng khi thiết kế các ứng dụng theo mô hình khách chủ. Chúng tôi chia các xử lý ra làm hai nhánh khác nhau: **nhánh máy trạm (client side)** và **nhánh máy chủ (server side)**.

### Nhánh máy trạm (client side)

- ✓ Các ứng dụng bên nhánh máy trạm thường sẽ thực hiện các công việc như là: **đọc và hiển thị dữ liệu** hiện có trong cơ sở dữ liệu, **tính toán dữ liệu** đang hiển thị trên các màn hình ứng dụng, **in dữ liệu** ra các kết xuất.
- ✓ Các ngôn ngữ được dùng để xây dựng ứng dụng cho bên nhánh khách thường là: Delphi, Visual Basic, C++... các ứng dụng này còn cho phép người dùng có thể thực hiện các hành động thêm, sửa, xóa dữ liệu hiện có trong cơ sở dữ liệu bên nhánh máy chủ.



- ✓ Khi đọc dữ liệu từ máy chủ về để xử lý bên nhánh máy trạm, các ứng dụng khi xây dựng **nhên tránh** việc đọc toàn bộ dữ liệu của bảng (table) mà chỉ nên lấy về đúng các thông tin cần thiết cho các xử lý. Điều này sẽ làm giảm đi lượng thông tin lưu thông trên hệ thống mạng máy tính. Khi đó chỉ những thông tin nào đúng với yêu cầu của nhánh máy trạm sẽ được máy chủ truyền tải trên hệ thống mạng, ngoài ra nó cũng làm cho ứng dụng được chạy nhanh hơn là vì khi đó các xử lý chọn lựa dữ liệu được thực hiện cục bộ tại máy chủ. Với việc trang bị cấu hình cao tại máy chủ thì đảm bảo rằng việc xử lý thông tin tại máy chủ phải ở tốc độ cao nhất có thể được.

**Ví dụ:**

Khi đọc thông tin khách hàng từ bảng khách hàng (KHACHANG) bên trong cơ sở dữ liệu quản lý bán hàng (QLBH), chúng ta có đọc khách hàng theo từng quận, hoặc theo thứ tự tên họ được chỉ định (A, B, C ...).

**Nhánh máy chủ (server side)**

- ✓ Các xử lý bên nhánh máy chủ sẽ được tổ chức và thực hiện **trực tiếp** ngay trên máy chủ. Xử lý quan trọng đầu tiên là việc đảm bảo việc truy cập của các người dùng trên mạng là **bảo mật** (security). Điều này có nghĩa là chỉ những người dùng nào được cấp quyền hạn truy cập thì mới có thể truy xuất được các dữ liệu dùng chung.
- ✓ Các xử lý liên quan đến việc thực hiện hoặc cập nhật dữ liệu, có thể **đồng thời**, giữa những người dùng hiện hành trên mạng (concurrent). Thí dụ như là hệ thống máy chủ phải cho phép cùng lúc cả hai người dùng cập nhật thông tin của một khách hàng trong bảng khách hàng.
- ✓ Các xử lý **sao lưu dữ liệu** (backup data) tự động để đảm bảo các dữ liệu không bị mất đi trong trường hợp các sự cố xấu nhất có thể tình cờ xảy ra.

**1.3. Vì sao phát triển ứng dụng mô hình khách chủ?**

Trước khi mô hình khách chủ ra đời, các hệ thống ứng dụng quản lý vận hành trên các loại cơ sở dữ liệu khác vẫn hoạt động tốt, tuy nhiên ngày nay phần đông các công ty đã chuyển hướng sang mô hình khách chủ là vì các lý do sau:

- ✓ **Giảm chi phí:** với mô hình khách chủ ngày nay cho phép các công ty có thể sử dụng các máy chủ là những máy tính cá nhân, thay vì phải sử dụng máy tính lớn. Các phần mềm sẽ không **é** quá đắt tiền khi chạy trên máy tính cá nhân bởi vì đã có khá nhiều công ty phát triển phần mềm hệ thống cho máy tính cá nhân.
- ✓ **Tốc độ nhanh:** việc phân chia các xử lý ra làm hai phần (nhánh máy chủ và nhánh máy trạm) sẽ làm giảm bớt việc tắc nghẽn thông tin trong hệ thống mạng máy tính. Các xử lý nào phức tạp tác động nhiều lên cơ sở dữ liệu sẽ được lưu trữ ngay trên máy chủ, các xử lý đơn giản sẽ được thực hiện ngay trong ứng dụng bên máy trạm. Khi đó sẽ làm cho hệ thống ứng dụng vận hành với hiệu quả cao hơn.
- ✓ **Tính tương thích cao:** với mô hình khách chủ, việc chọn lựa các phần mềm để phát triển ứng dụng có thể hoàn toàn độc lập từ ngôn ngữ lập trình, đến hệ cơ sở dữ liệu quan hệ và cả các thiết bị phần cứng khác. Chúng ta có thể chọn ra các thành phần tối ưu nhất theo đúng cái chúng ta cần khi xây dựng một hệ thống ứng dụng.



## II. Microsoft SQL Server là gì?

### II.1. Lịch sử ra đời Microsoft SQL Server

Sau khi có cái nhìn sơ bộ về các khái niệm trong mô hình khách chủ ở phần trên, bây giờ chúng ta sẽ đi vào tìm hiểu hệ quản trị cơ sở dữ liệu Microsoft SQL Server. Trong những phần kế tiếp, chúng tôi sẽ hướng dẫn cách các bạn sử dụng Microsoft SQL Server để tổ chức lưu trữ dữ liệu. Đầu tiên chúng ta sẽ đi tìm hiểu về lịch sử phát triển của phần mềm này.

Năm **1970** công ty máy tính **IBM** đã khởi tạo ra ngôn ngữ máy tính cho các truy vấn trong cơ sở dữ liệu có tên là **SEQUEL** (Structured English Query Language). Qua một thời gian sau, ngôn ngữ này đã phát triển rất nhanh không những chỉ thực hiện các truy vấn mà chúng còn cho phép người sử dụng xây dựng và quản trị cơ sở dữ liệu khá tốt. Sau đó IBM đã phổ biến ngôn ngữ này một cách công khai trên nhiều phạm vi mà ngày nay chúng ta biết đến với tên là **SQL**.

Năm **1985** hai công ty máy tính **IBM** và **Microsoft** đã loan báo rằng họ sẽ cùng nhau kết hợp để phát triển một số hệ điều hành và các phần mềm hệ thống khác. Mục tiêu đầu tiên là xây dựng hệ điều hành có tên là **OS/2** dựa theo hệ điều MS DOS của Microsoft. Ngày **16/12/1987** hệ điều hành OS/2 phiên bản 1.0 đã được chính thức phát hành. Nhưng sau đó IBM đã loan báo rằng họ sẽ đưa ra một phiên bản mới của OS/2 gọi là OS/2 mở rộng (Extended), hệ điều hành này sẽ mạnh hơn OS/2 phiên bản 1.0 bởi vì nó sẽ được tích hợp thêm một phần cơ sở dữ liệu SQL của IBM (ngày nay chính là hệ quản trị cơ sở dữ liệu **DB2**). Microsoft nhận thấy rằng nếu IBM có thể đưa ra giải pháp hoàn chỉnh cho OS/2 mở rộng thì liệu có khách hàng nào sẽ mua sản phẩm OS/2 của Microsoft không?

Vào thời điểm này Microsoft chưa hề có một sản phẩm thuộc loại quản trị cơ sở dữ liệu. Ngay sau đó Microsoft đã quay sang kết hợp với công ty **Sybase** để cùng hợp tác làm ra một sản phẩm thuộc loại hệ quản trị cơ sở dữ liệu (Database Management System). Với một sự hợp tác tốt đẹp hai công ty này đã thành công cho ra đời một sản phẩm thuộc loại cơ sở dữ liệu có tên khó nhớ là **Ashton-Tate** vào mùa thu năm **1988**, sản phẩm này hoạt động trên môi trường OS/2. Sau đó một thời gian Sybase đã phát triển sản phẩm này trên môi trường **UNIX** và đổi tên riêng là **DataServer** mà ngày nay còn có tên khác là **Sybase Adaptive Server**.

Microsoft quyết định không phát triển hệ điều hành OS/2 mà thay vào đó cho ra đời một hệ điều hành mạng máy tính có tên là **Windows NT Server**. Và thế là SQL Server chỉ hoạt động **độc lập** trên môi trường Windows NT Server mà thôi. Lần lượt các phiên bản của Microsoft SQL Server đã ra đời sau sự kiện này từ 4.2 sau đó được nâng cấp thành 4.21, 6.0, 6.5, 7.0 và hiện giờ là Microsoft SQL Server 2000. Trong phần giáo trình này chúng tôi trình bày Microsoft SQL Server phiên bản 2000.

Tóm lại Microsoft SQL Server là một **hệ quản trị cơ sở dữ liệu quan hệ mạng máy tính** hoạt động theo **mô hình khách chủ** cho phép đồng thời cùng lúc có **nhiều người dùng** truy xuất đến dữ liệu, quản lý việc truy nhập hợp lệ và các quyền hạn của từng người dùng trên mạng. Khái niệm này chỉ là một định nghĩa tương đối để giúp cho các bạn có cái nhìn tổng quát về Microsoft SQL Server, còn chi tiết bên trong nó là gì thì chúng ta sẽ xem ở các phần tiếp theo.



## II.2. Cài đặt cơ sở dữ liệu SQL Server Desktop

Khác với một số phần mềm khác : Microsoft Office, Visual Studio, Delphi... thì việc cài đặt Microsoft SQL Server không đơn giản, do đó chúng tôi sẽ hướng dẫn các bước để các bạn có thể tự cài đặt cơ sở dữ liệu **Microsoft SQL Server 2000 phiên bản Personal**. Đây là phiên bản làm việc trong môi trường Windows 9x, hoặc Windows 2000 professional, Windows XP cho phép chúng ta làm việc trên máy tính đơn (Stand – Alone) một cách để các bạn có thể làm việc với Microsoft SQL Server mà **không cần** phải cài đặt Windows NT Server hoặc **không cần** trang bị một hệ thống mạng máy tính.

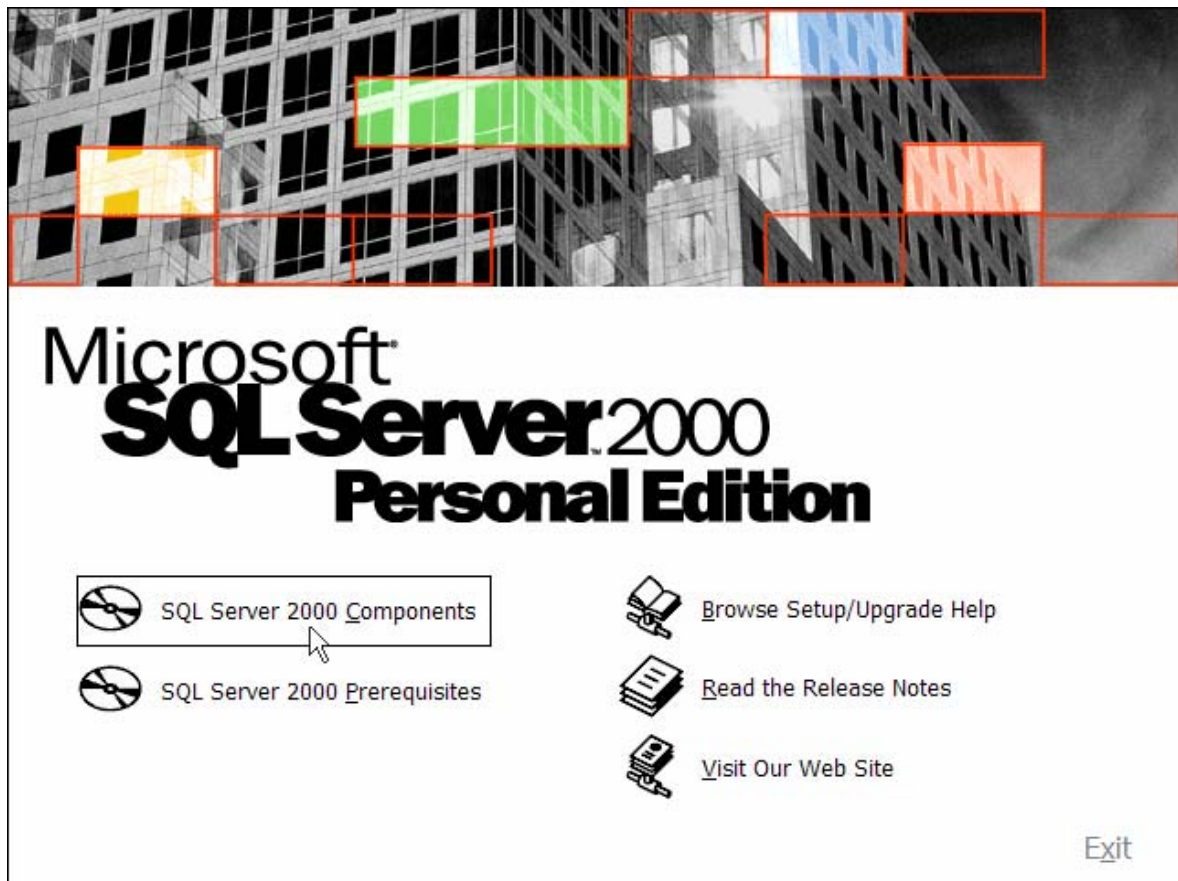
Để có thể cài đặt Microsoft SQL Server 2000 việc đầu tiên là chúng ta phải có một đĩa CDROM chứa phần mềm Microsoft SQL Server 2000 phiên bản Personal và một máy tính cá nhân có ổ đĩa CDROM với cấu hình tối thiểu như sau:

Thiết bị	Yêu cầu tối thiểu
CPU	Pentium 166 MHz hoặc Pentium Pro
RAM	32MB, 64MB thì tốt hơn.
Dung lượng đĩa trống	180MB với phiên bản đầy đủ 65MB với phiên bản tối thiểu
Hệ điều hành	Windows NT Workstation 4.0, Windows 95/98 Windows 2000 Professional
Internet browser	Microsoft Internet Explorer 4.01

Nếu máy tính của các bạn đã thỏa các yêu cầu tối thiểu bên trên thì chúng ta có thể bắt đầu việc cài đặt Microsoft SQL Server Desktop 2000 theo từng bước chỉ dẫn bên dưới.

- ✓ Bước 1: Đưa đĩa Microsoft SQL Server vào ổ đĩa CDROM. Nếu ổ đĩa không tự động chạy thì bạn sẽ nhấn đúp vào tập tin **Autorun.Exe** để khởi động chương trình cài đặt.

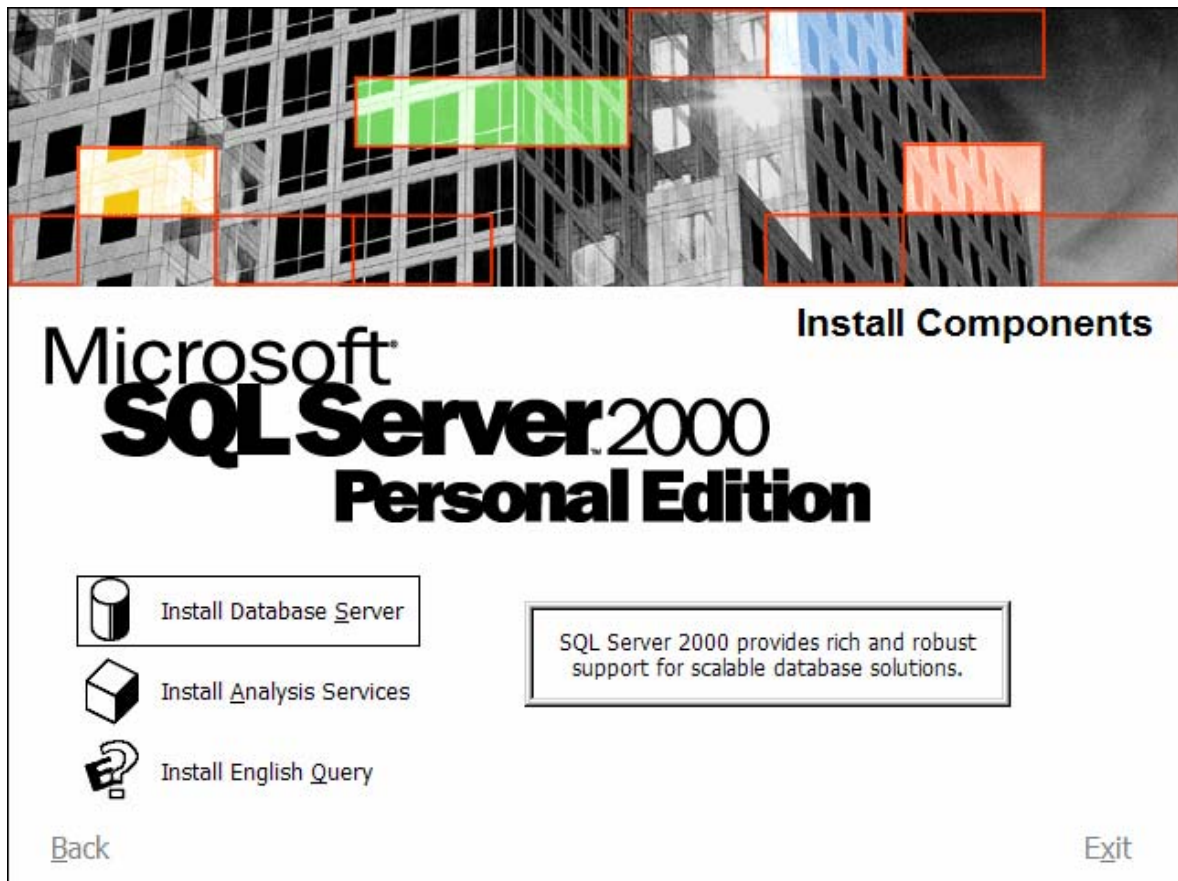
- ✓ Bước 2: Trong màn hình khởi động cài đặt Microsoft SQL Server, chúng ta sẽ chọn chức năng **SQL Server 2000 Components** để bắt đầu công việc cài đặt.



**Hình 1-2.** Màn hình khởi động cài đặt SQL.



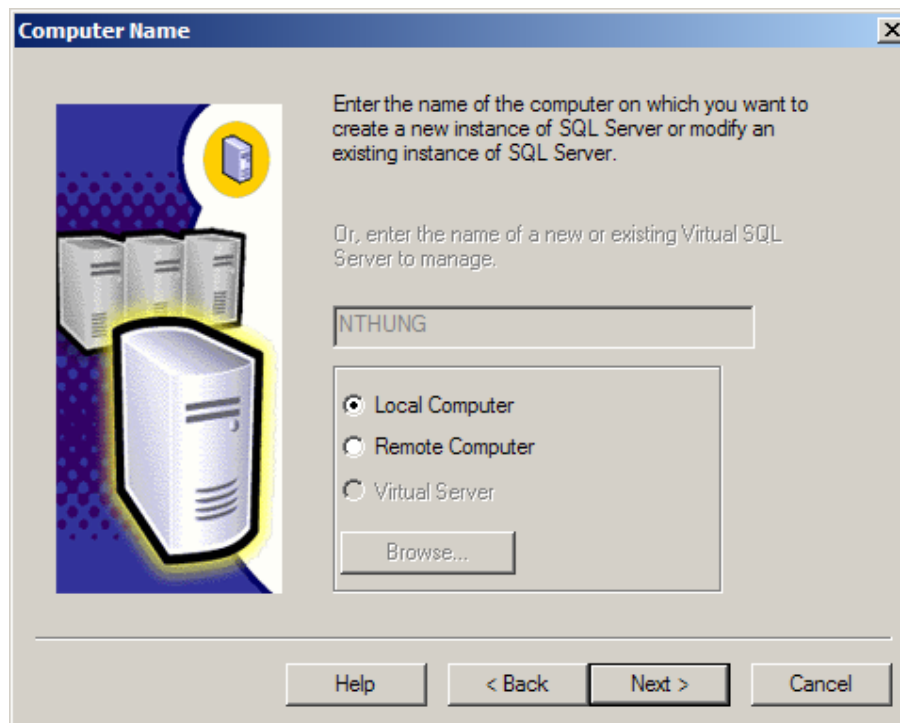
- ✓ Bước 3: Trong màn hình kế tiếp chúng ta sẽ chọn chức năng **Install Database Server** để chọn lựa cơ sở dữ liệu Microsoft SQL Server Desktop.



**Hình 1-3.** Màn hình chọn thành phần cài đặt SQL

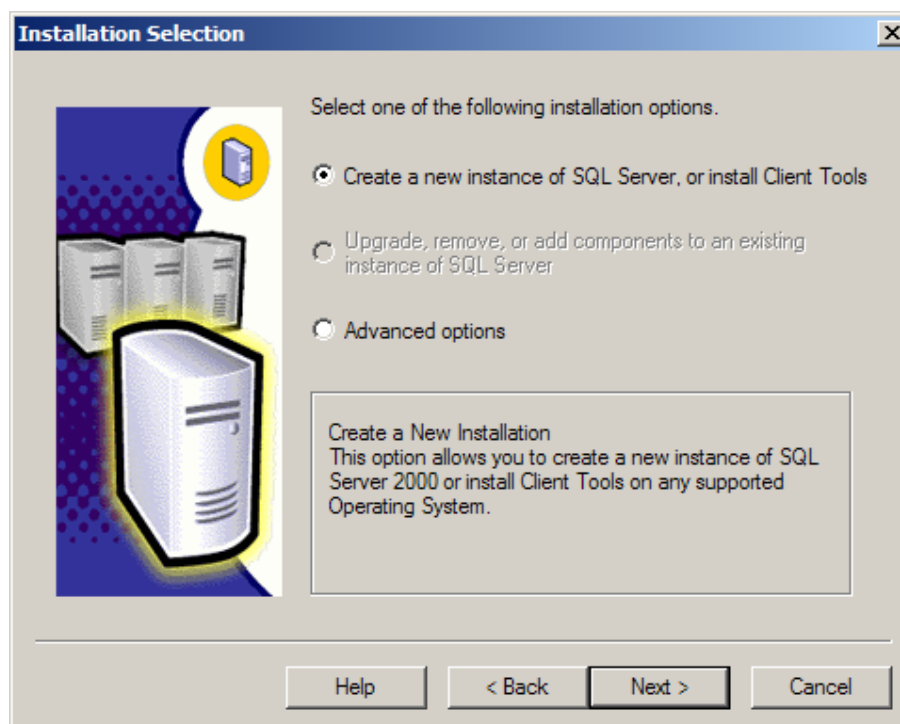
- ✓ Bước 4: Bỏ qua màn hình giới thiệu, trong màn hình chọn lựa cách thức cài đặt chúng ta chọn **Local computer** để cài đặt cơ sở dữ liệu Microsoft SQL Server ngay trên máy tính hiện hành, sau đó nhấn nút Next để tiếp tục.





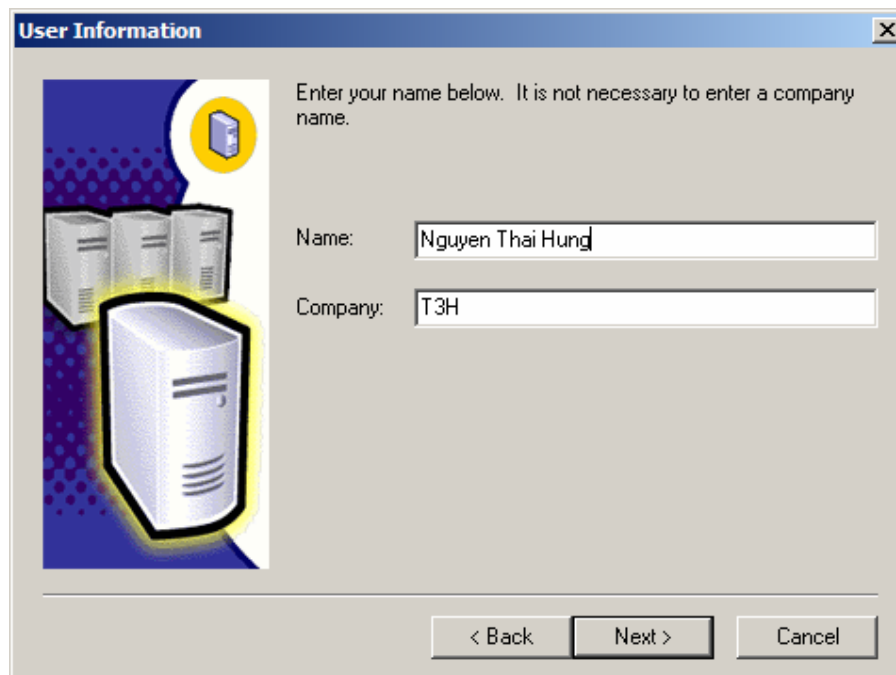
**Hình 1-4.** Màn hình chọn cách thức cài đặt SQL

- ✓ Bước 5: Màn hình tiếp theo yêu cầu bạn lựa chọn cách cài đặt SQL Server 2000. Trên một máy tính, SQL Server 2000 cho phép cài nhiều SQL Server khác nhau, mỗi server khi đó gọi là một instance. Do bạn chưa cài đặt SQL Server lần nào nên cần phải chọn **Create a new instance of SQL Server, or install Client tools**.



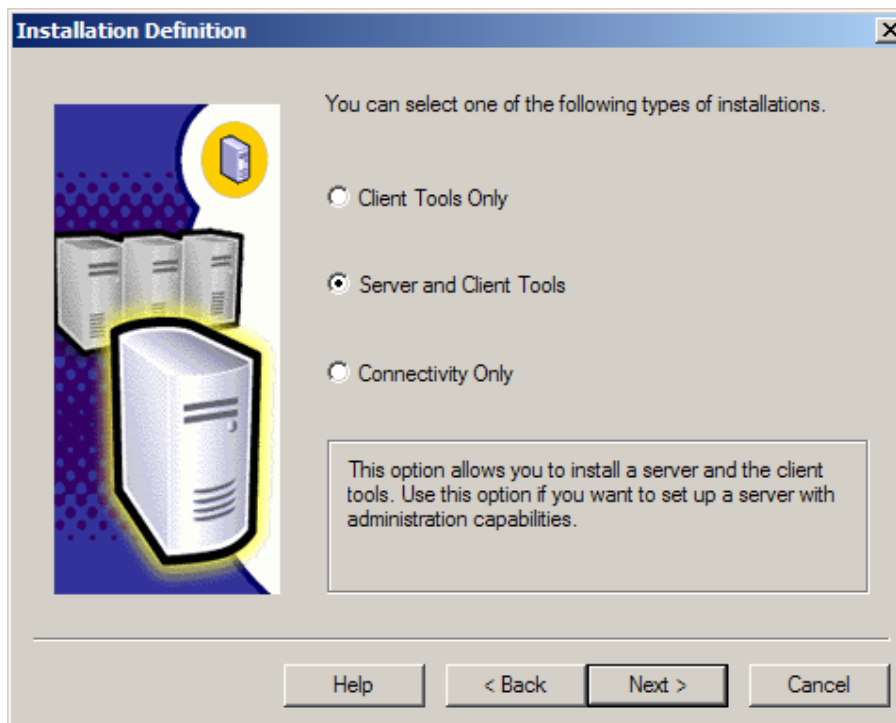
**Hình 1-5.** Màn hình chào mừng bắt đầu cài đặt SQL.

- ✓ Bước 6: Trong màn hình thông tin người dùng (User Information) các bạn sẽ nhập vào tên của mình và tên cơ quan. Có thể bỏ trống tên cơ quan. Sau đó nhấn nút **Next** để tiếp tục.



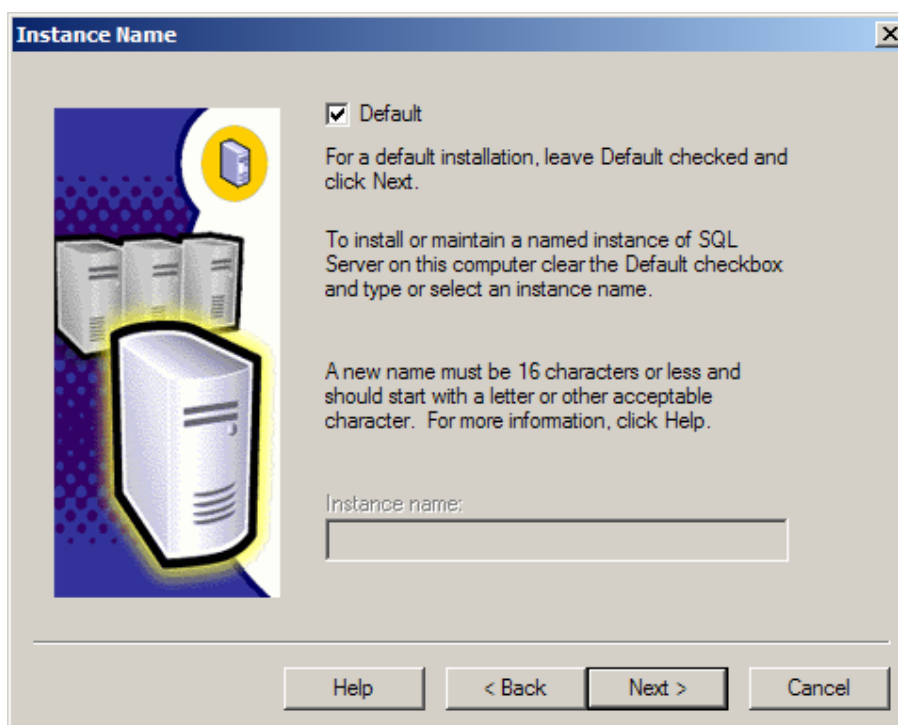
**Hình 1-6.** Màn hình thông tin người dùng.

- ✓ Bước 7: Ở màn hình chọn các thành phần cài đặt, bạn cần chọn **Server and Client Tools** để cài đầy đủ các chức năng cần thiết cho một database server.



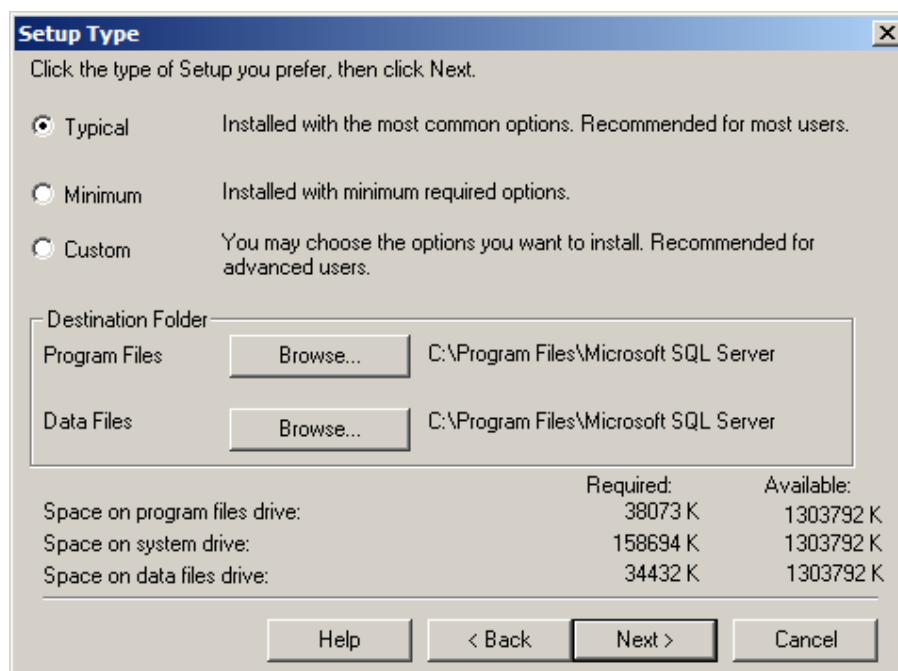
**Hình 1-7.** Màn hình thông tin người dùng.

- ✓ Bước 8: Trong màn hình chọn instance name, thông thường với lần cài đặt đầu tiên bạn sẽ chọn tên mặc định. Đây là tên của máy tính đang cài đặt. Ở những lần cài đặt sau, nếu bạn không gỡ bỏ SQL Server mà cài đặt thêm một SQL Server nữa trên máy thì bạn sẽ cần đặt lại tên instance.



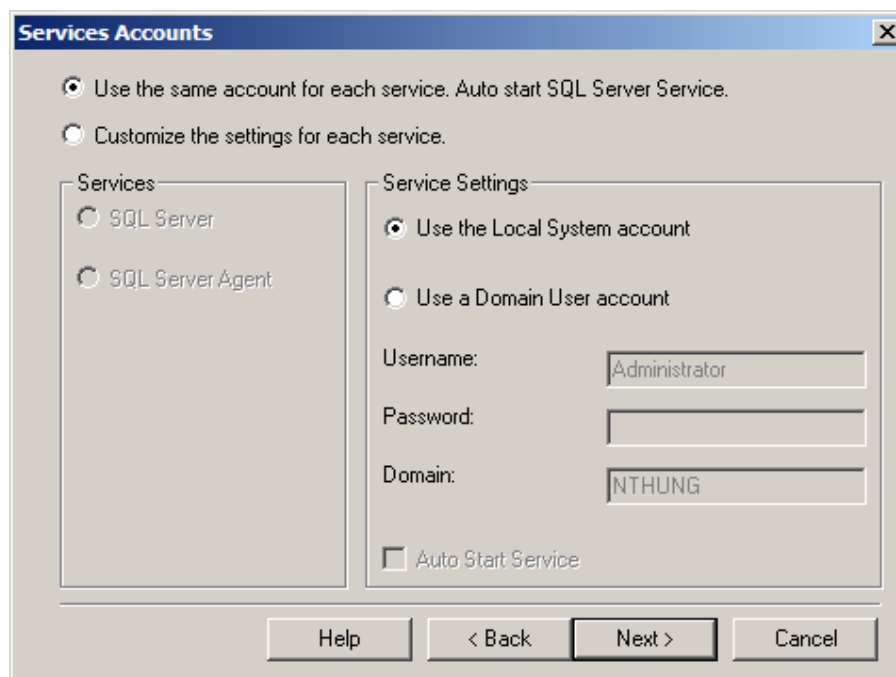
**Hình 1-8. Màn hình chọn loại cài đặt SQL.**

- ✓ Bước 9: Trong màn hình chọn loại cài đặt (Setup Type) chúng tôi khuyến cáo các bạn nên chọn loại thường dùng (**Typical**) nếu dung lượng đĩa cứng còn trống lớn hơn 200MB, ngược lại nếu dung lượng đĩa cứng còn trống nhiều thì các bạn nên chọn loại tối thiểu (**Minimum**).



**Hình 1-9. Màn hình chọn loại cài đặt SQL.**

- ✓ Bước 10: Trong màn hình chọn tài khoản người dùng khởi động dịch vụ (Services Accounts) cho phép các bạn chỉ định ra tên tài khoản người dùng để có thể khởi động hai dịch vụ chính yếu của Microsoft SQL Server khi hoạt động là: SQL Server và SQL Server Agent trong trường hợp Microsoft SQL Server chạy trong môi trường Windows 2000/XP.

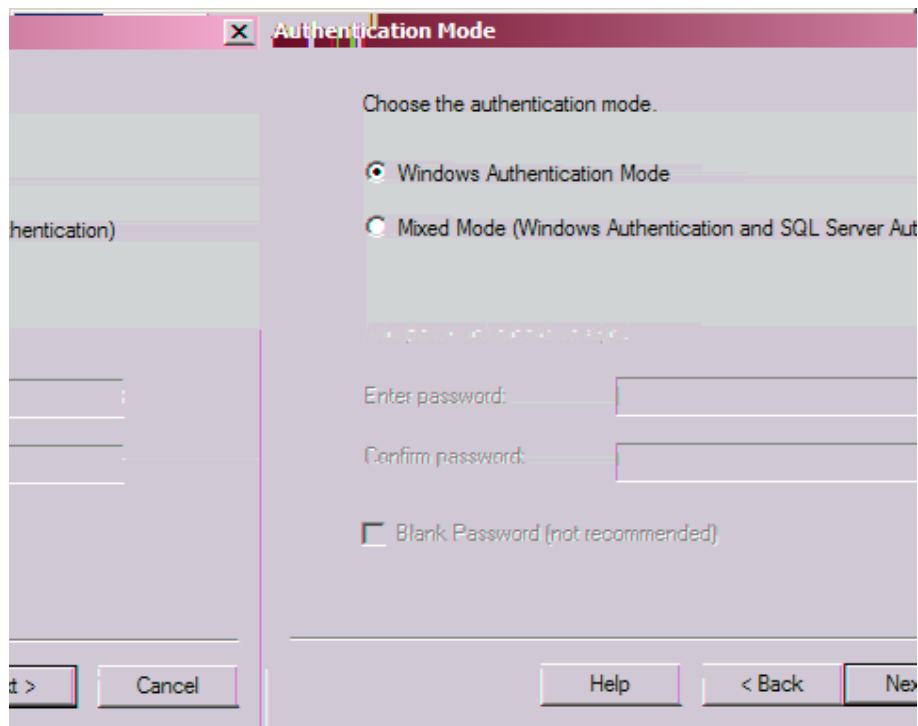


**Hình 1-10. Màn hình chọn tài khoản người dùng.**

Nếu hệ điều hành máy tính các bạn cài đặt là Windows 9x thì màn hình này sẽ không xuất hiện. Trên các hệ thống Windows 2000/XP, hai dịch vụ SQL Server và SQL Server Agent hoạt động dưới hình thức service vẫn cũng cần tài khoản đăng nhập vào máy tính như một người dùng bình thường. Local System account là tài khoản có quyền quản trị trên máy hiện hành và có thể thực hiện đa số các tính năng của SQL Server.

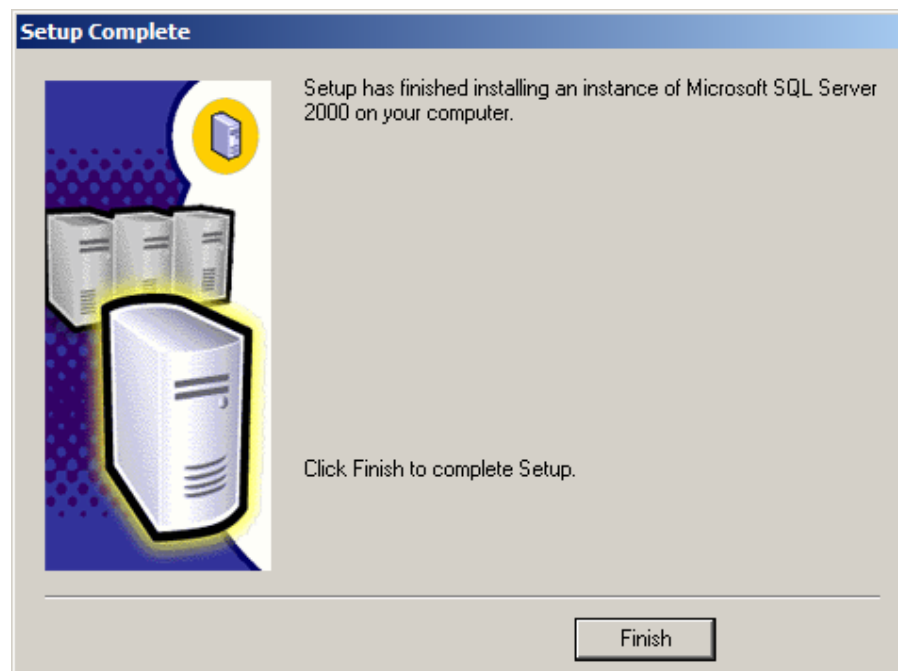
Trong một hệ thống mạng Windows NT, SQL Server có thể tham gia tích hợp với các SQL Server khác, các Domain controller hay các Mail Server. Để thực hiện điều này, SQL Server cần đăng nhập vào domain sử dụng một tài khoản có quyền quản trị trên toàn domain.

- ✓ Bước 11: Màn hình tiếp theo yêu cầu xác nhận chế độ kiểm tra người dùng khi đăng nhập SQL Server. Khi một máy trạm muốn làm việc với SQL Server, máy đó cần phải tạo ra một session để làm việc. SQL Server để bảo mật dữ liệu sẽ yêu cầu máy trạm cung cấp một tài khoản gồm username và password. SQL Server có hai hình thức kiểm tra tài khoản đăng nhập:
  - Trong một hệ thống mạng Windows NT, SQL Server có thể truy cập danh sách các tài khoản người dùng trên domain controller để kiểm tra. Chế độ này gọi là Windows Authentication Mode và người dùng trong mạng không cần phải cung cấp tài khoản khi đăng nhập SQL Server nữa.
  - SQL Server cũng có thể lưu trữ những tài khoản người dùng một cách độc lập với Windows. Chế độ này gọi là SQL Server Authentication Mode và người dùng bắt buộc phải nhập thông tin tài khoản khi đăng nhập.
  - Bạn nên chọn chế độ Mixed mode nếu không quen làm việc trong mạng và để đơn giản trong việc thực hành sau này. Tuy nhiên, bạn cũng nên biết rằng chế độ Windows Authentication mode an toàn hơn nhiều chế độ Mixed mode.



**Hình 1-11.** Màn hình bắt đầu sao chép tập tin.

- ✓ Bước 12: Các bạn phải chờ trong một thời gian để quá trình cài đặt thực hiện. Cuối cùng khi xuất hiện màn hình hoàn thành cài đặt (Setup Complete) thì các chúng ta sẽ nhấn nút **Finish** để kết thúc quá trình cài đặt cơ sở dữ liệu Microsoft SQL Server .



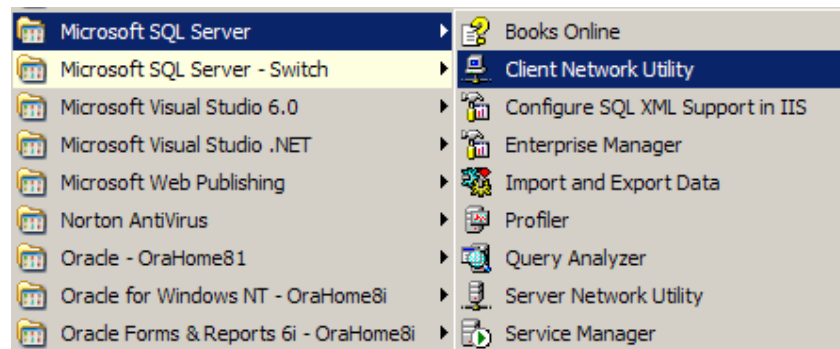
**Hình 1-12.** Màn hình hoàn thành cài đặt SQL.

Chúng tôi hy vọng là các bạn đã thực hiện thành công việc cài đặt phần mềm Microsoft SQL Server. Bây giờ chúng ta sẽ bắt đầu xem xét đến các tiện ích có trong Microsoft SQL Server.

### III. Các tiện ích trong Microsoft SQL Server

Sau khi cài đặt xong Microsoft SQL Server công việc kế tiếp mà chúng ta sẽ làm là tìm hiểu về các tiện ích bên trong Microsoft SQL Server nhằm để sử dụng một cách đúng đắn và có hiệu quả cao. Các tiện ích này đã được cài đặt chung với Microsoft SQL Server. Để kích hoạt các tiện ích này, chúng ta sẽ chọn những biểu tượng nằm trong đường dẫn như sau:

Start → Programs → Microsoft SQL Server



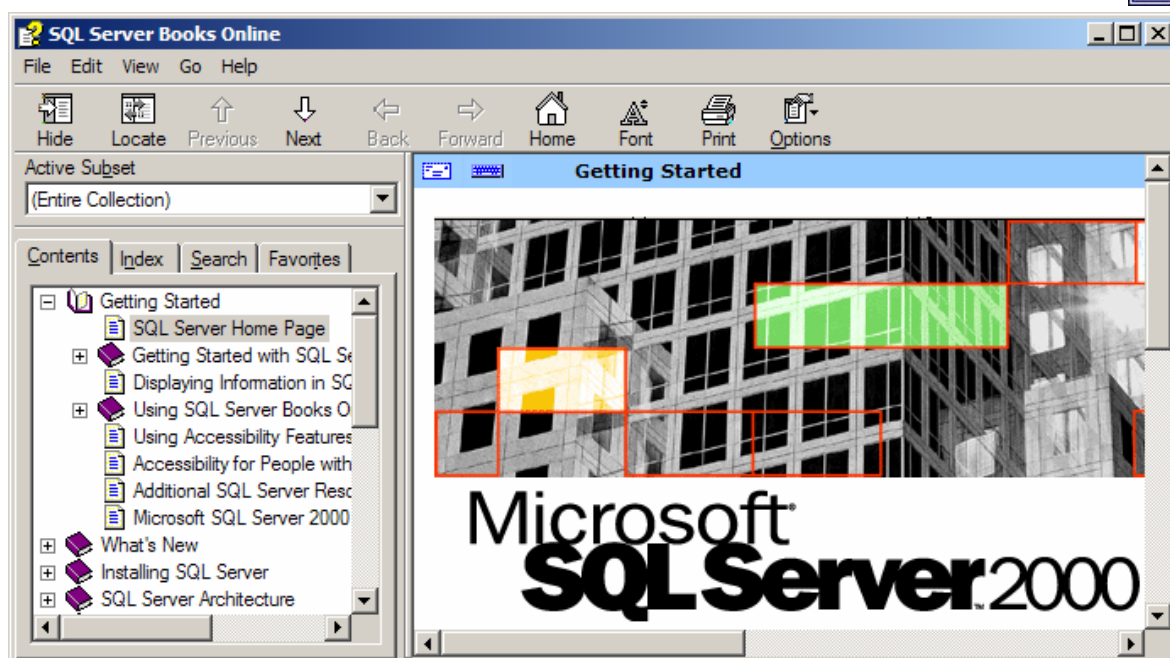
**Hình 1-13.** Các tiện ích bên trong Microsoft SQL Server.

#### III.1. Tiện ích Book Online

Ứng dụng này cho phép chúng ta có thể **tra cứu trực tuyến** tất cả các thông tin liên quan đến Microsoft SQL Server một cách đầy đủ với các tính năng tìm kiếm dễ dàng và một giao diện dễ sử dụng.

Nội dung được trình bày theo từng phần dễ dàng xem trong trang Contents. Ngoài ra còn các trang Index và Search cho phép các bạn tra cứu nhanh theo chỉ mục đã được sắp xếp trước đó hoặc gõ vào các từ khóa cần tìm.

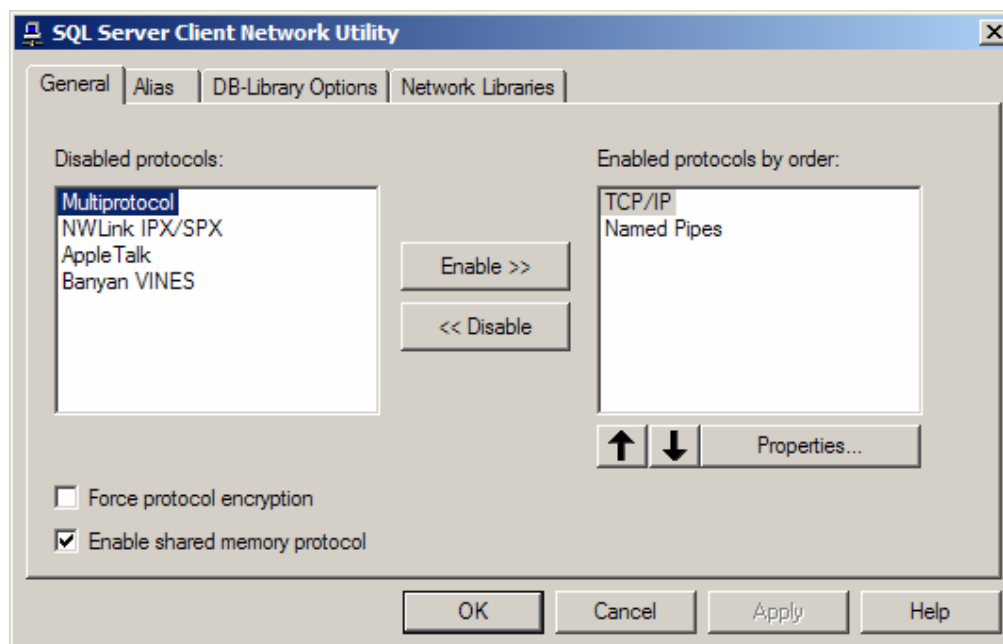
Khi cần tham khảo cú pháp các lệnh, hàm, biến hệ thống... một cách đầy đủ nhất thì chúng ta vào đây xem là hoàn toàn chính xác. Tuy nhiên nội dung trình bày hoàn toàn là bằng tiếng Anh.



Hình 1-14. Tiện ích Book Online.

### III.2. Tiện ích Client NetWork Utility

Tiện ích này cho phép chúng ta thay đổi, tạo mới và lưu lại các **nghi thức nối kết mạng** (network protocol) mặc định của máy trạm khi thực hiện nối kết vào Microsoft SQL Server tại các máy chủ.

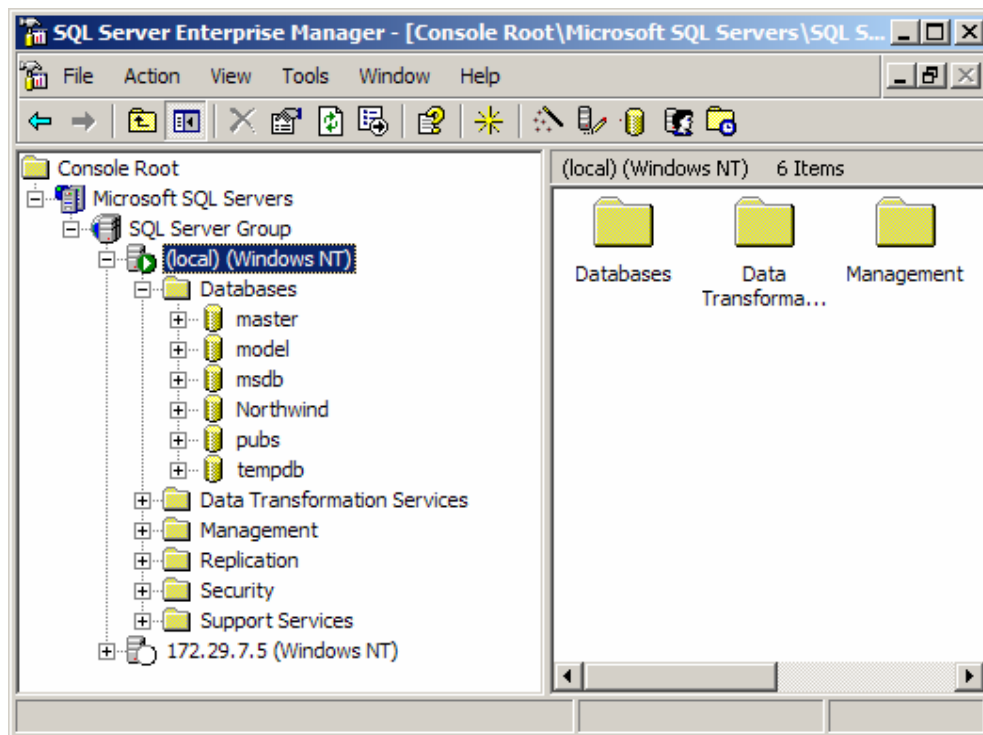


Hình 1-15. Tiện ích Client NetWork Utility.



### III.3. Tiện ích Enterprise Manager

Tiện ích này cho phép chúng ta khởi động hoặc tạm ngưng các dịch vụ của Microsoft SQL Server. Trong phần cài đặt ở bước 9 chúng tôi đã giới thiệu các dịch vụ của Microsoft SQL Server phải được khởi động trước thì các bạn mới có thể làm việc được với Microsoft SQL Server.



**Hình 1-16.** Tiện ích Enterprise Manager.

Ngoài ra tiện ích này còn giúp chúng ta **quản trị** một hoặc nhiều Microsoft SQL Server khác nhau, với giao diện đồ họa thân thiện (user friendly) tiện ích này sẽ giúp cho các bạn có thể tạo lập cơ sở dữ liệu và các thành phần bên trong Microsoft SQL Server một cách dễ dàng hơn. Tuy nhiên muốn quản trị Microsoft SQL Server thì chúng ta phải **đăng ký** (register) máy chủ vào tiện ích này, việc đăng ký như thế nào chúng tôi sẽ trình bày trong những phần tiếp theo.

### III.4. Tiện ích Import and Export Data

Tiện ích này cho phép chúng ta thực hiện các tính năng trong việc nhập (import), xuất (export) và **chuyển đổi dữ liệu** qua lại giữa Microsoft SQL Server và những loại cơ sở dữ liệu khác thường dùng như: Microsoft Access, Visual FoxPro, Microsoft Excel, tập tin văn bản ASCII...



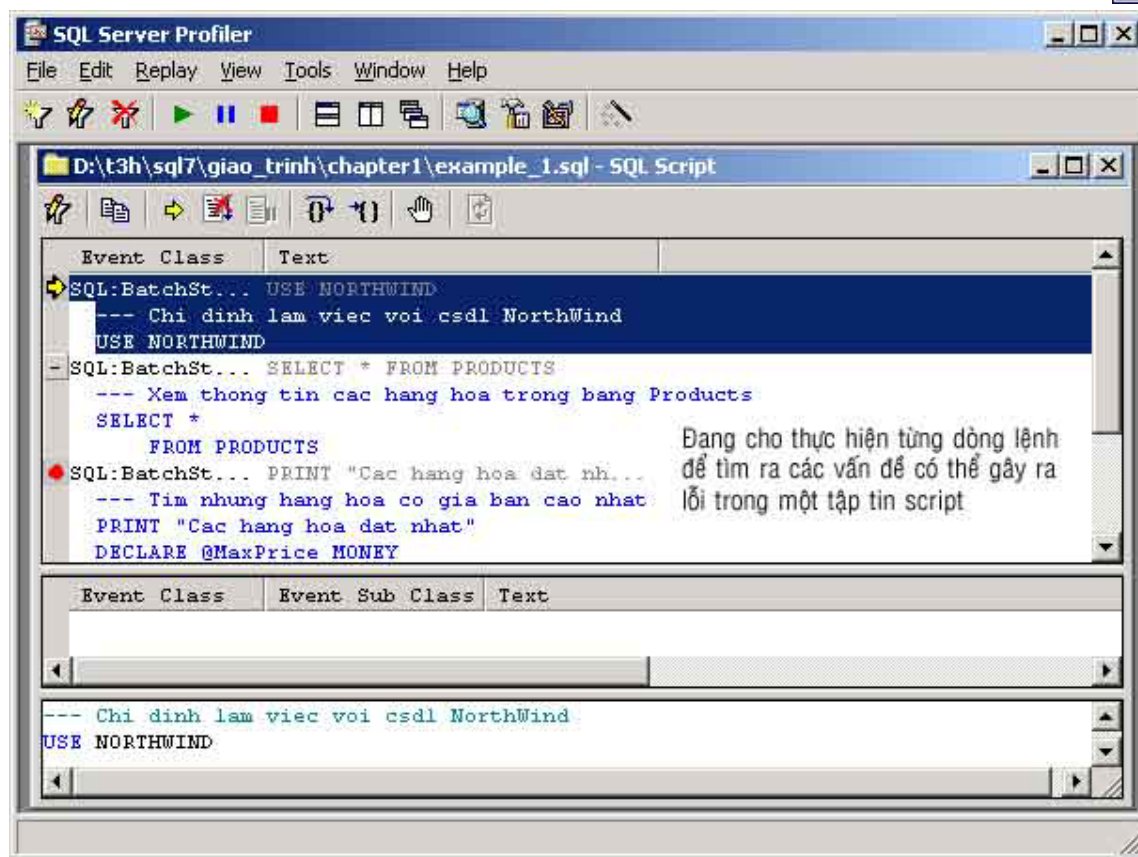


**Hình 1-17. Tiện ích Import and Export Data.**

Trong phần giáo trình này chúng tôi chỉ trình bày cách sử dụng Microsoft SQL Server nên không giới thiệu nhiều về tiện ích này, tuy nhiên nếu các bạn muốn trở thành người quản trị Microsoft SQL Server thì chắc rằng công cụ này là rất cần thiết cho người quản trị.

### III.5. Tiện ích Profiler

Tiện ích này cho phép chúng ta phát hiện ra những **biến cố** đã xảy ra của Microsoft SQL Server khi đang thực hiện một xử lý nào đó trên máy chủ. Các biến cố này có thể được ghi lại trong một tập tin lưu vết (trace file) để sau này sử dụng lại cho việc phân tích nhằm phát hiện ra những vấn đề khi thực hiện các câu lệnh truy vấn trong xử lý đó.

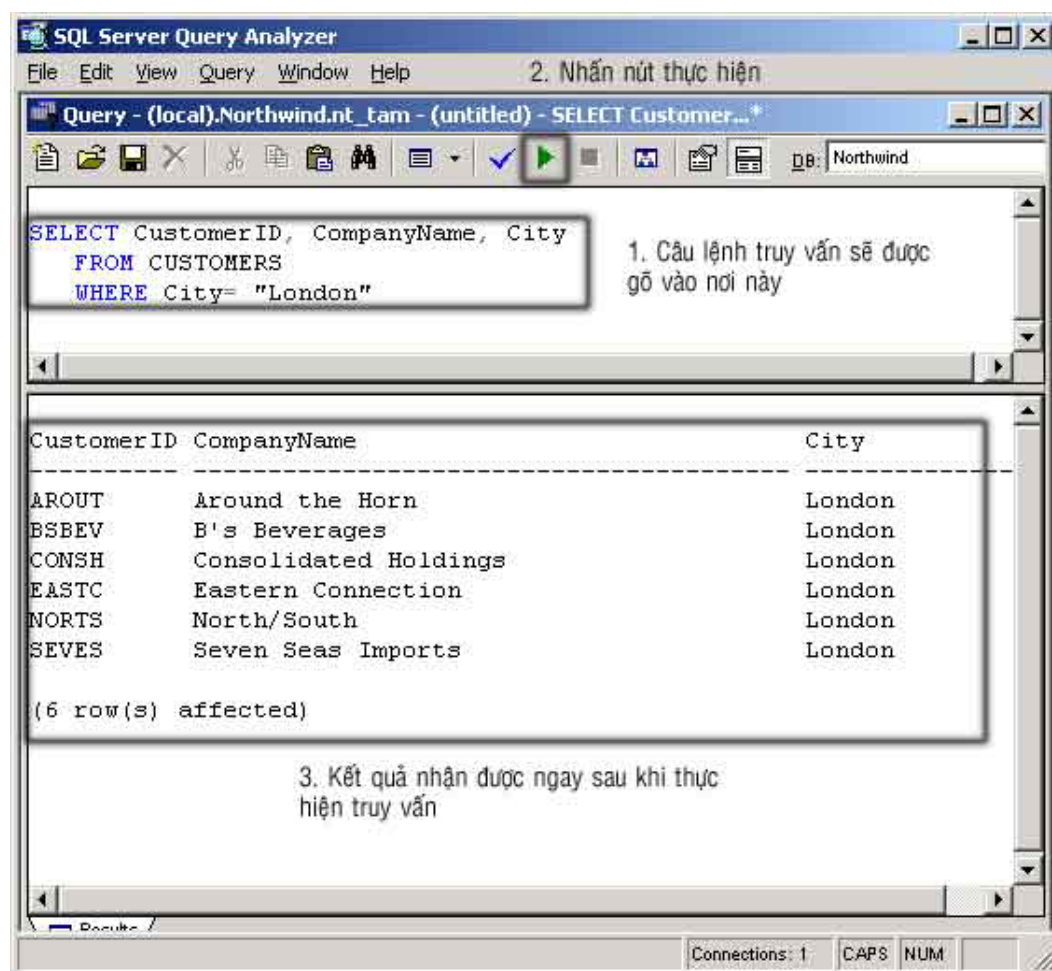


**Hình 1-18.** Tiện ích Profiler.

Hoạt động của tiện ích này có phần gần giống công cụ tìm lỗi trong các ngôn ngữ lập trình, có nghĩa là các bạn sẽ cho thực hiện tuần tự các câu lệnh trong một xử lý lô (batch) để có thể phát hiện ra lỗi của một câu lệnh nào đó (nếu có).

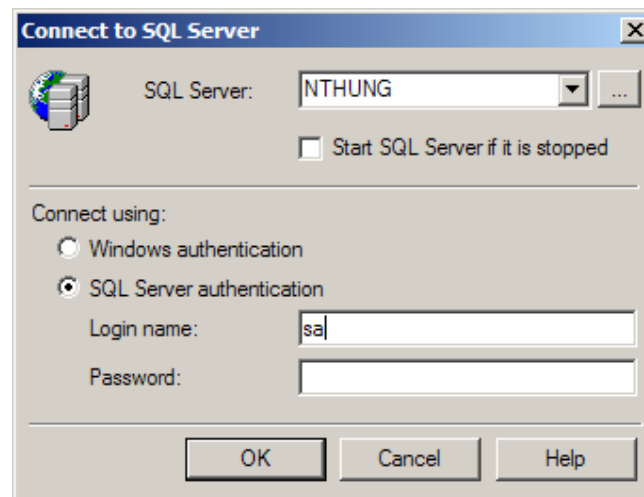
### III.6. Tiện ích Query Analyzer

Tiện ích này cho phép chúng ta **soạn thảo** các tập tin kịch bản (script file) – là tập tin văn bản ASCII chứa các câu lệnh SQL giao tác trên cơ sở dữ liệu Microsoft SQL Server hoặc có thể **thực hiện các truy vấn** trực tiếp trên cơ sở dữ liệu Microsoft SQL Server và nhận được kết quả trực tiếp ngay sau khi thực hiện truy vấn đó.



**Hình 1-19.** Tiện ích Query Analyzer.

Tuy nhiên trước khi vào được màn hình 1-19 hệ thống Microsoft SQL Server sẽ yêu cầu các bạn đăng nhập (login) vào hệ thống với tên tài khoản người dùng và mật khẩu hợp lệ bởi vì Microsoft SQL Server là một hệ quản trị cơ sở dữ liệu mạng máy tính. Trong đó tên tài khoản người dùng và mật khẩu này sẽ do người quản trị Microsoft SQL Server tạo ra trước đó, trong trường hợp làm việc trên Microsoft SQL Server Desktop thì chúng ta có thể nhập vào với tên là SA (System Administrator – người quản trị hệ thống cơ sở dữ liệu Microsoft SQL Server) và mật khẩu để trống.

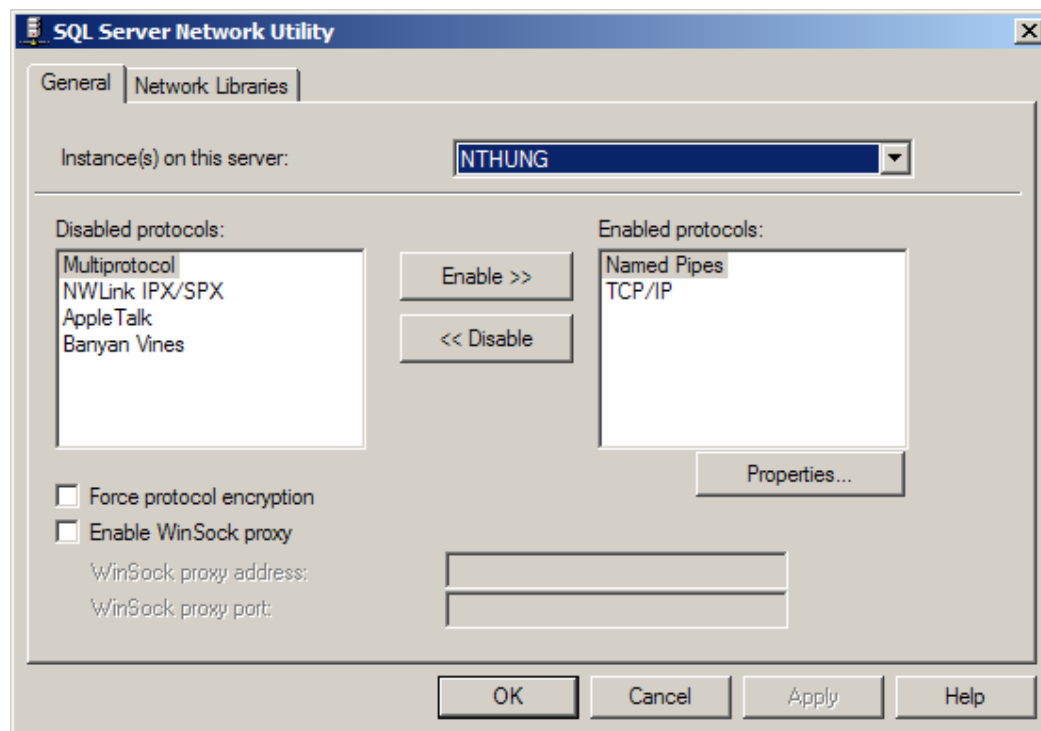


**Hình 1-20.** Đăng nhập đến Server.

Trong các chương còn lại chúng ta chủ yếu làm việc với tiện ích này để tạo ra các đối tượng lưu trữ dữ liệu bên trong Microsoft SQL Server.

### III.7. Tiện ích Server Network Utility

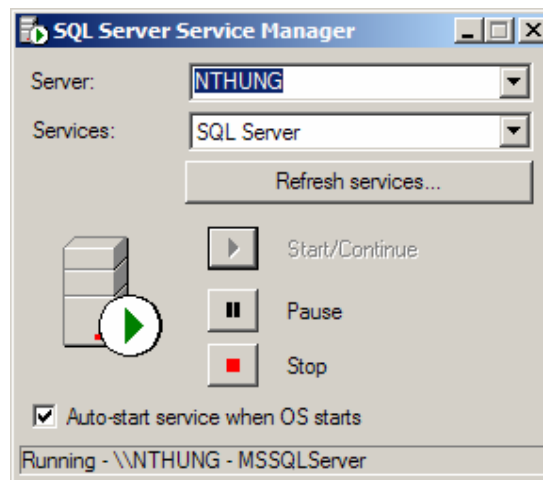
Tiện ích này cho phép chúng ta quản lý các **thư viện nghi thức nối kết mạng** của máy chủ dùng để lắng nghe các yêu cầu từ các máy trạm – có nghĩa các nghi thức nối kết mạng giữa máy chủ và máy trạm phải ăn khớp nhau để chúng có thể giao tiếp qua lại.



**Hình 1-21.** Tiện ích Server Network Utility

### III.8. Tiện ích Service Manager

Tiện ích này cho phép chúng ta **quản lý các dịch vụ** liên quan đến Microsoft SQL Server. Có thể thực hiện việc: khởi động (start), tạm dừng (pause) và ngưng lại (stop) các dịch vụ đó. Các dịch vụ (services) này được xem như là các ứng dụng chạy ngầm định bên dưới hệ thống trong môi trường Windows.



**Hình 1-22.** Tiện ích Service Manager.

Đối với dịch vụ MSSQLSever bắt buộc phải được khởi động để người dùng mới có thể làm việc với cơ sở dữ liệu trong Microsoft SQL Server. Do đó sau khi cài đặt hoàn thành Microsoft SQL Server các bạn phải tự khởi động dịch vụ này để chúng ta có thể làm việc với Microsoft SQL Server. Tuy nhiên kể từ các lần bật máy tính sau đó các dịch vụ này sẽ được tự khởi động nếu chúng ta đã chọn vào ô kiểm tra Auto-start như hình 1-22 ở phía trên.

Thông thường tiện ích Service Manager sẽ xuất hiện bên dưới thanh tác vụ (task tray) của Windows. Để kích hoạt tiện ích này, chúng ta có nhấn đúp vào biểu tượng Service Manager.



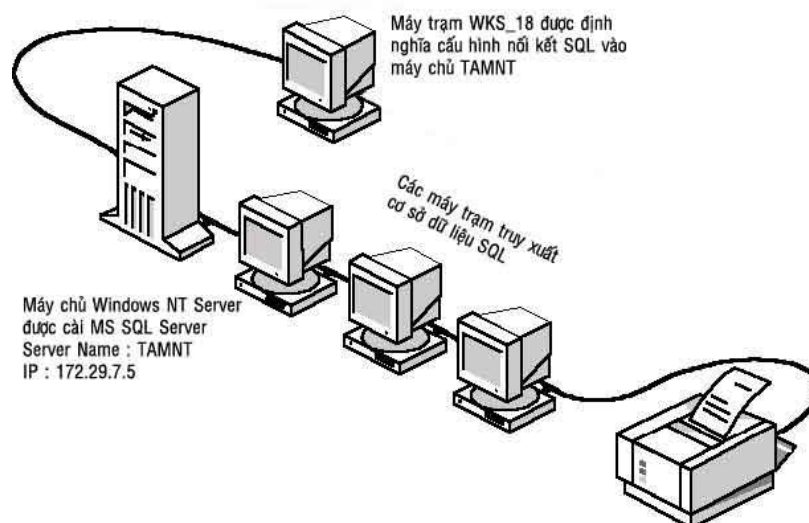
**Hình 1-23.** Tiện ích Service Manager xuất hiện trên thanh công cụ.

### III.9. Định nghĩa cấu hình nối kết vào SQL

Muốn truy xuất hoặc quản trị cơ sở dữ liệu Microsoft SQL Server từ một máy trạm thì công việc đầu tiên mà chúng ta cần phải thực hiện là định nghĩa cấu hình nối kết vào Microsoft SQL Server tại máy trạm đó. Để thực hiện công việc này các bạn sẽ sử dụng tiện ích Client

NetWork Utility mà chúng tôi đã trình bày trong phần các tiện ích trong Microsoft SQL Server.

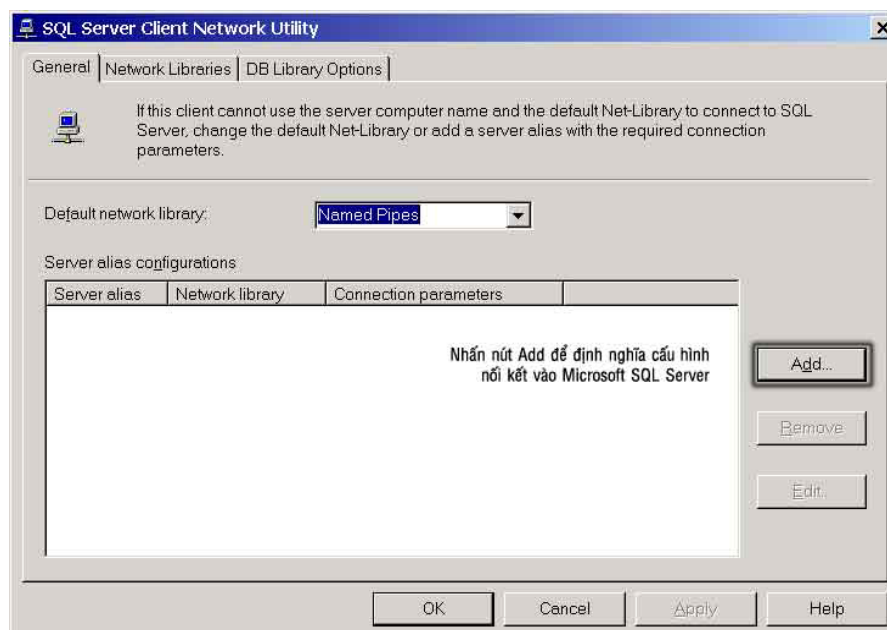
Chúng ta lấy thí dụ rằng trong một hệ thống mạng máy tính cục bộ (LAN–Local Network Area) theo như hình 1-24, máy chủ Windows NT Server đã được cài đặt Microsoft SQL Server phiên bản Standard hoặc Enterprise với tên máy chủ là TAMNT và địa chỉ IP (Internet Protocol) máy chủ là 172.29.7.5. Trên máy trạm WKS\_18 chúng ta sẽ định nghĩa cấu hình nối kết vào Microsoft SQL Server tại máy chủ NTSQL.



**Hình 1-24.** Hệ thống mạng máy tính cục bộ.

**III.9.1. Các bước chúng ta sẽ thực hiện tại máy trạm WKS\_18 như sau:**

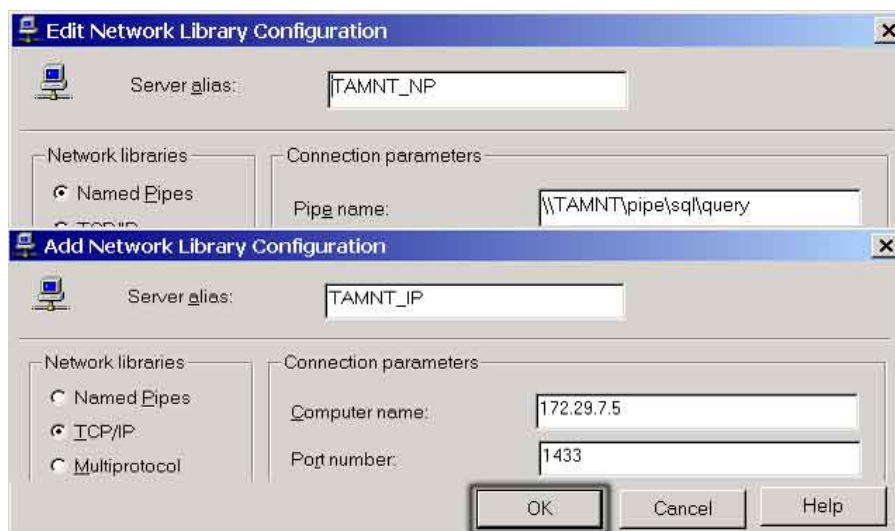
- ✓ Bước 1: Khởi động tiện ích Client NetWork Utility. Chuyển sang trang màn hình **General**, nhấn vào nút **Add** để định nghĩa cấu hình nối kết vào Microsoft SQL Server.



**Hình 1-25.** Tiện ích Client Network Utility.



- ✓ Bước 2: Trong màn hình thêm cấu hình thư viện mạng (Add Network Library Configuration) lần lượt chỉ định bí danh cho máy chủ (Server alias), thư viện mạng (Network library) sử dụng nối kết và các tham số nối kết (Connection parameters). Nhấn **OK** để kết thúc việc thêm mới cấu hình thư viện mạng.



**Hình 1-26.** Màn hình thêm cấu hình thư viện mạng.

- ✓ Bước 3: Quay về tiện ích Client NetWork Utility nhấn tiếp nút Apply hoặc OK để hoàn tất quá trình định nghĩa cấu hình nối kết vào Microsoft SQL Server.

Thư viện mạng sử dụng những tập tin thư viện liên kết động (DLL – Dynamic Linking Library) dùng định nghĩa cách thức liên lạc giữa những máy tính bên trong Microsoft SQL Server, các bạn có thể hình dung thư viện mạng như là một ngôn ngữ giao tiếp. Thí dụ nếu máy chủ đang sử dụng hai (2) thứ tiếng Anh và Pháp mà máy trạm lại sử dụng tiếng Đức thì chắc rằng máy chủ sẽ không thể nào hiểu và trả lời (response) các yêu cầu của máy trạm, thế là việc nối kết từ máy trạm vào máy chủ Microsoft SQL Server xem như là thất bại.

Do đó cả máy chủ và máy trạm phải sử dụng thư viện mạng giống nhau và chạy trên cùng nghi thức mạng (network protocol) giống nhau cho việc nối kết Microsoft SQL Server hợp lệ.



**Ví dụ:**

Nếu muốn ứng dụng tại máy trạm sử dụng nghi thức TCP/IP liên lạc với Microsoft SQL Server tại máy chủ thì tại máy trạm phải định nghĩa cấu hình thư viện mạng TCP/IP và tại máy chủ cũng phải định nghĩa cấu hình thư viện mạng TCP/IP.

Tất cả các thư viện mạng sẽ được cài đặt cùng với quá trình cài đặt Microsoft SQL Server. Bảng bên dưới mô tả một số thư viện mạng được định nghĩa cấu hình mặc định trên máy chủ, máy trạm sau khi cài đặt Microsoft SQL Server.

Windows NT Server	Windows 95/98
Tại máy chủ	
Named Pipes	Shared Memory
TCP/IP	TCP/IP

MultiProtocal	
Tại máy trạm	
Named Pipes	TCP/IP

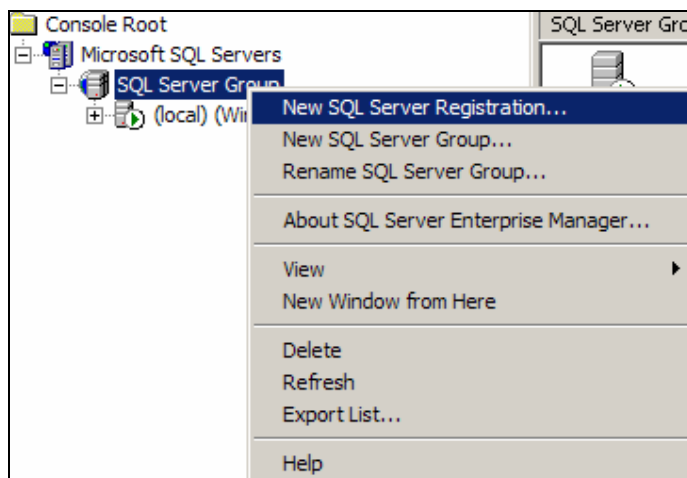
Nếu các bạn sử dụng Microsoft SQL Server Desktop thì chúng tôi khuyến cáo các bạn **không nên** vào ứng dụng Client Network Utility để định nghĩa lại cấu hình nối kết vào Microsoft SQL Server. Bởi vì lúc bây giờ việc nối kết vào Microsoft SQL Server sẽ xảy ra trên cùng một máy tính – máy chủ và máy trạm là một, nên thật sự không cần thiết.

### III.9.2. Đăng ký quản trị Microsoft SQL Server

Việc quản trị Microsoft SQL Server có thể được thực hiện trực tiếp tại máy chủ hoặc từ một máy trạm bất kỳ có thể nối kết vào Microsoft SQL Server. Lần đầu tiên khi khởi động tiện ích Enterprise Manager, hệ thống sẽ tự động đăng ký quản trị Microsoft SQL Server cục bộ vào tiện ích này.

Muốn quản trị Microsoft SQL Server, công việc đầu tiên mà chúng ta cần làm là phải đăng ký máy chủ đã được cài đặt Microsoft SQL Server vào tiện ích Enterprise Manager. Trong phần này chúng tôi sẽ hướng dẫn các bước để các bạn có thể đăng ký quản trị thành công một máy chủ Microsoft SQL Server. Tuy nhiên công việc này được thực hiện **sau khi** chúng ta đã định nghĩa cấu hình nối kết vào Microsoft SQL Server tại máy trạm và máy chủ là thành công. Chúng ta thực hiện các bước như sau:

- ✓ Bước 1: Khởi động tiện ích Enterprise Manager. Trong màn hình SQL Server Enterprise Manager nhấn chuột phải trên đối tượng SQL Server Group, chọn chức năng **New SQL Server Registration** trong thực đơn tắt để bắt đầu quá trình đăng ký quản trị máy chủ Microsoft SQL Server.



**Hình 1-27. Tiện ích Enterprise Manager.**

- ✓ Bước 2: Trong màn hình chào mừng đăng ký Microsoft SQL Server bằng Wizard hiển thị ba bước mà chúng ta sẽ lần lượt thực hiện ở các bước kế tiếp. Nhấn nút **Next** để tiếp tục.

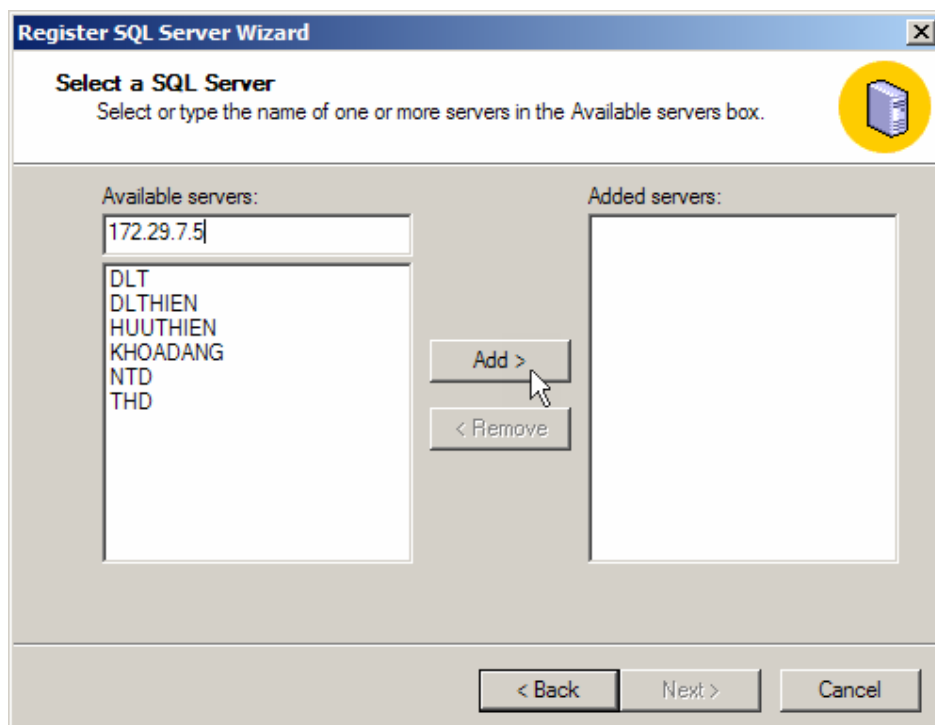




**Hình 1-28.** Màn hình chào mừng đăng ký quản trị SQL Server bằng Wizard.

Không nên nhấn vào ô kiểm tra bên dưới màn hình này để hệ thống sẽ bật chức năng trợ giúp thông minh trong quá trình đăng ký quản trị các máy chủ Microsoft SQL Server.

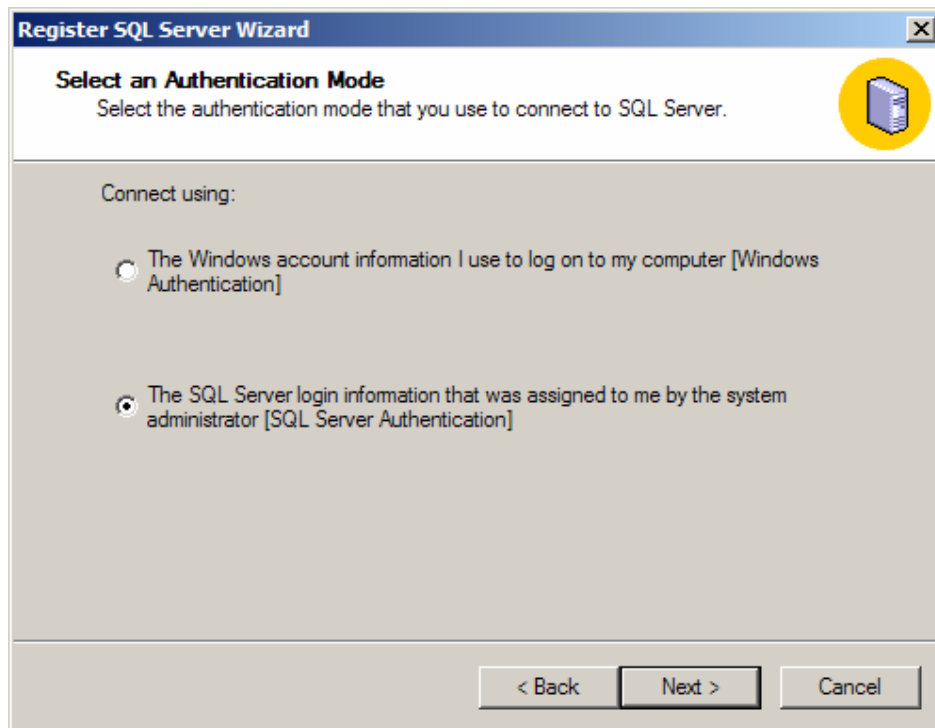
- ✓ Bước 3: Trong màn hình chọn máy chủ SQL Server (Select a SQL Server), chọn hoặc gõ vào tên của máy chủ muốn đăng ký quản trị trong hộp danh sách các máy chủ sẵn sàng dùng (Available servers), sau đó nhấn nút **Add** để chọn các máy chủ đó. Nhấn nút **Next** để tiếp tục.



**Hình 1-29.** Màn hình chọn máy chủ SQL Server.

Danh sách các máy chủ thông thường được hiển thị từ các bí danh máy chủ (server alias) mà chúng ta đã tạo ra trong tiện ích Client Network Utility.

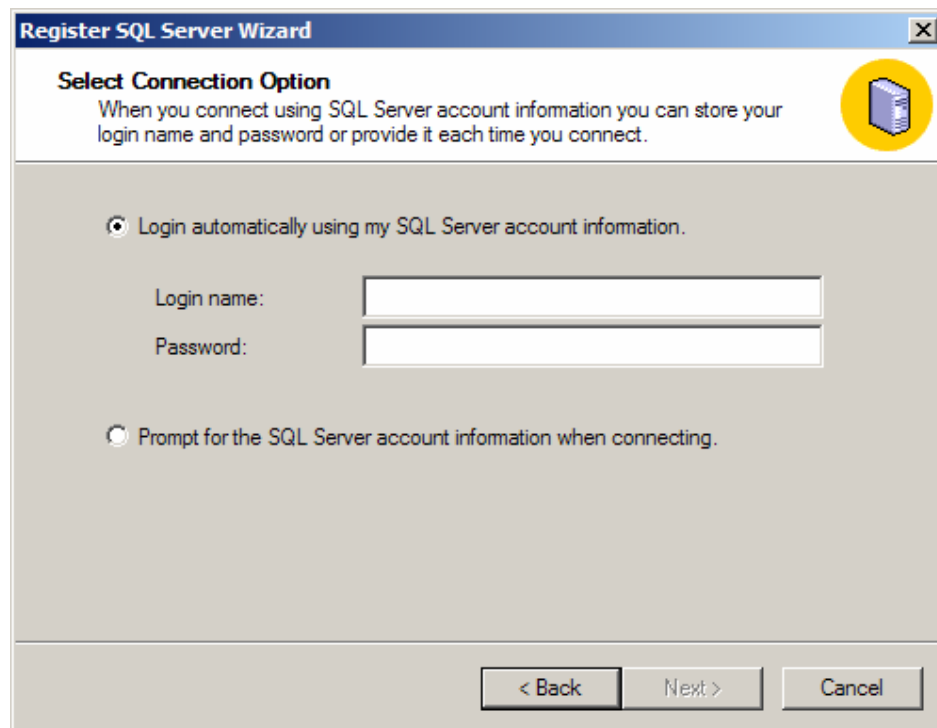
- ✓ Bước 4: Trong màn hình chọn chế độ xác thực (Authentication Mode) đăng nhập nối kết vào Microsoft SQL Server, chúng ta sẽ chọn hoặc là sử dụng tên tài khoản người dùng trong Windows NT Server (Windows NT Authentication) hoặc là tên tài khoản người dùng trong cơ sở dữ liệu Microsoft SQL Server (SQL Server Authentication).



**Hình 1-30.** Màn hình chọn chế độ xác thực.

Thông thường người quản trị Microsoft SQL Server sẽ tạo ra danh sách các tài khoản người dùng hoặc nhóm các người dùng trong Microsoft SQL Server để phân cấp quyền hạn truy cập dữ liệu của những người sử dụng đó. Vì thế chúng tôi sẽ chọn chế độ xác thực là sử dụng tài khoản người dùng trong Microsoft SQL Server.

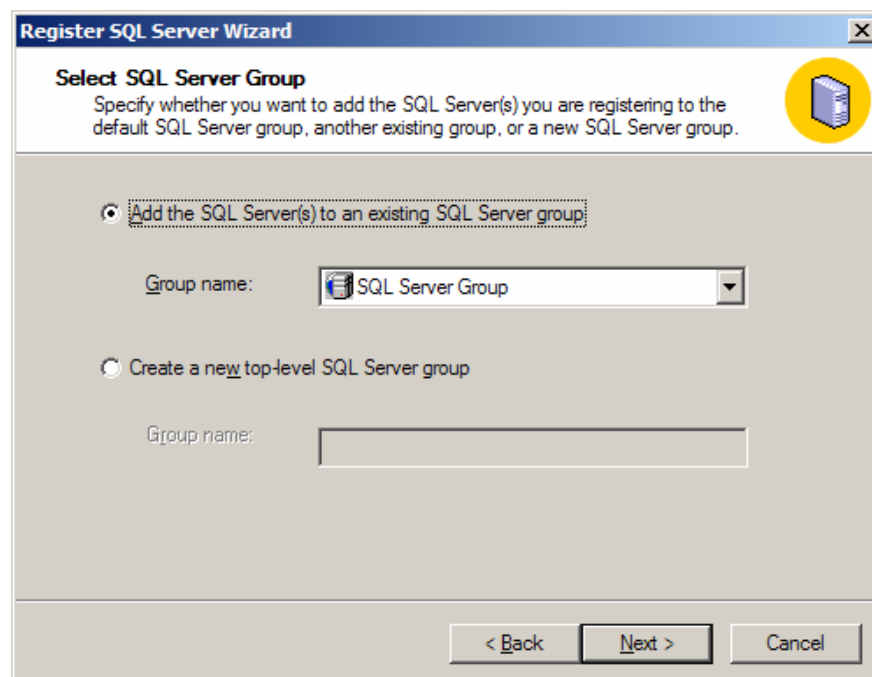
- ✓ Bước 5: Trong màn hình chọn chức năng nối kết (Connection Option), chọn chức năng đăng nhập tự động (Login automatically) sau đó gõ vào tên tài khoản người dùng và mật khẩu đã được người quản trị tạo ra trước đó trong Microsoft SQL Server. Nhấn nút **Next** để tiếp tục.



**Hình 1-31.** Màn hình chọn chức năng nối kết.

Trong trường hợp nếu chúng ta muốn đảm bảo tính bảo mật cao khi làm việc trong Enterprise Manager, chúng ta nên chọn chức năng **Prompt for the SQL Server...** để hệ thống sẽ xuất hiện hộp thoại xác thực việc đăng nhập mỗi lần nối kết vào cơ sở dữ liệu Microsoft SQL Server.

- ✓ Bước 6: Trong màn hình chọn nhóm SQL Server (Select SQL Group), chọn ra một (1) nhóm SQL mà chúng ta muốn máy chủ sẽ thuộc vào nhóm đó sau khi đăng ký. Hệ thống có một (1) nhóm mặc định là SQL Server Group. Nhấn nút **Next** để tiếp tục.



**Hình 1-32.** Màn hình chọn nhóm SQL Server.

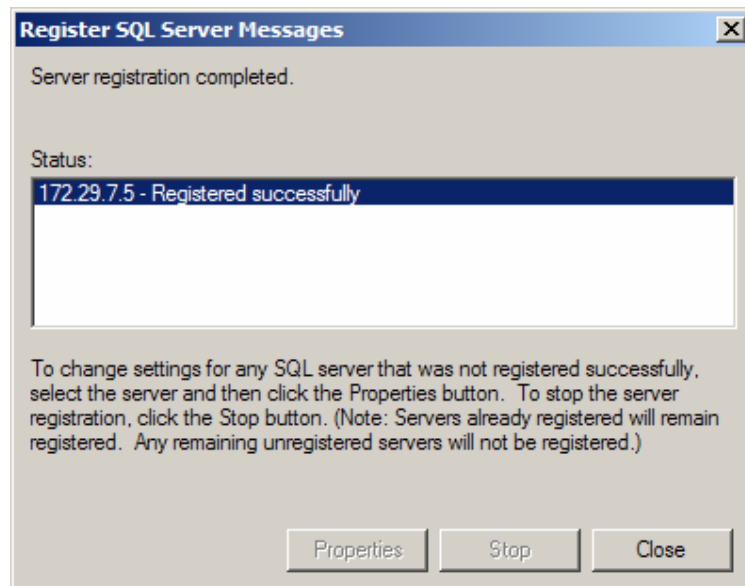
Chúng ta có thể tạo ra nhóm SQL Server mới bằng cách chọn chức năng **Create a new top-level...** và gõ vào tên của nhóm mới trong ô văn bản **Group name** để máy chủ Microsoft SQL Server sau khi đăng ký sẽ xuất hiện trong nhóm mới.

- ✓ Bước 7: Hệ thống thông báo tên các máy chủ Microsoft SQL Server đã chỉ định ở các bước trên. Nhấn nút Finish để hệ thống bắt đầu quá trình đăng ký quản trị máy chủ Microsoft SQL Server trong thời gian vài phút.



**Hình 1-33.** Màn hình thông báo danh sách máy chủ đăng ký.

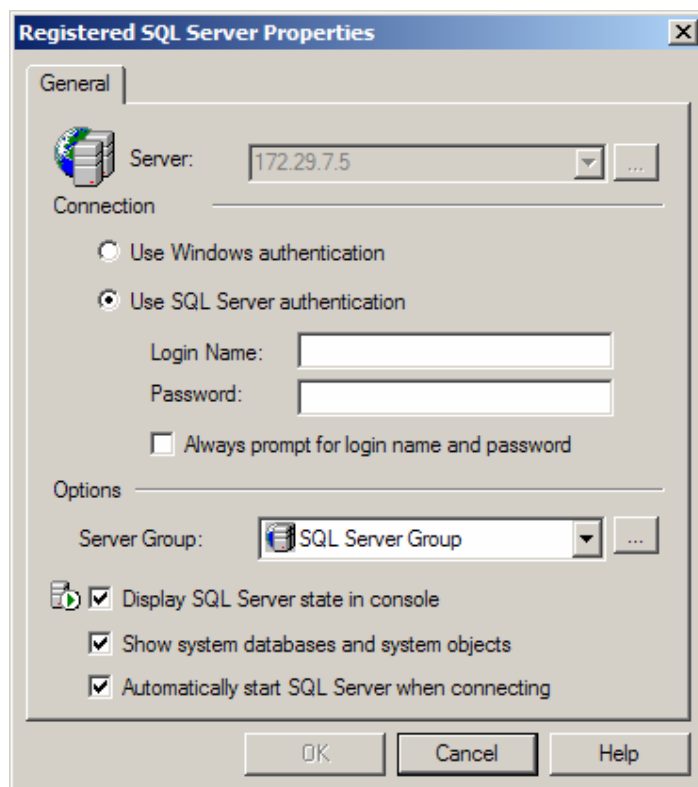
- ✓ Bước 8: Sau một thời gian ngắn màn hình thông báo việc đăng ký quản trị Microsoft SQL Server đã thành công. Nhấn Close hoàn tất quá trình đăng ký quản trị Microsoft SQL Server.



**Hình 1-34.** Màn hình thông báo việc đăng ký thành công.

Trong trường hợp khi đăng ký không thành công, các bạn phải xem lại việc định nghĩa nghi thức nối kết vào Microsoft SQL Server trong tiện ích Client Network Utility tại máy trạm đã đúng chưa? Hoặc dịch vụ MSSQLServer đã được khởi động chưa? Thực hiện lệnh **PING** đến máy chủ để kiểm tra hệ thống mạng bên dưới đã thông chưa?

Sau khi đăng ký quản trị Microsoft SQL Server thành công, các bạn có quay lại hiệu chỉnh một số các thuộc tính mà trong quá trình đăng ký chúng ta đã cung cấp trong các bước trước đó. Sử dụng chức năng **Edit SQL Server Registration properties** trong thực đơn tắt sau khi chọn chuột phải trên tên máy chủ Microsoft SQL Server đã có trong tiện ích Enterprise Manager.



**Hình 1-35.** Màn hình hiển thị các thuộc tính đăng ký của máy chủ.

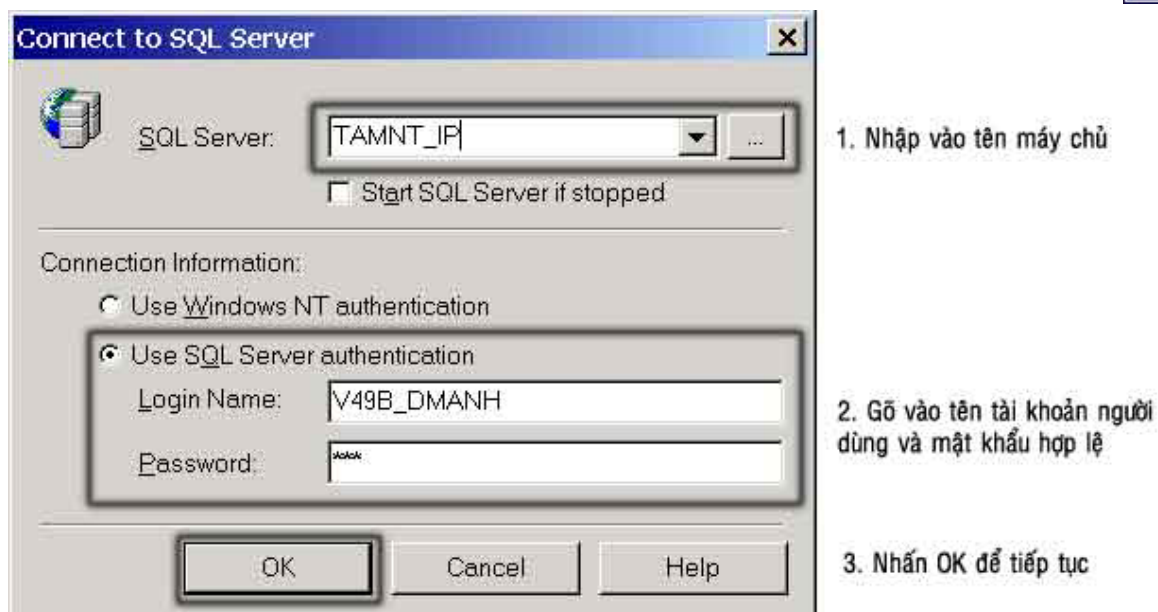
Ngoài ra chúng ta cũng được phép hủy bỏ ra khỏi Enterprise Manager các máy chủ đã đăng ký quản trị trước đó bằng chức năng **Delete SQL Server Registration** trong thực đơn tắt sau khi chọn chuột phải trên tên máy chủ Microsoft SQL Server muốn hủy bỏ có trong tiện ích Enterprise Manager. Nhấn nút Yes để đồng ý muốn hủy bỏ.

Bên cạnh đó chúng ta cũng được phép khởi động (Start), tạm dừng (Pause), ngưng lại (Stop) dịch vụ MSSQLServer của Microsoft SQL Server. Việc thực hiện nối kết (Connect) hoặc ngưng nối kết (Disconnect) vào Microsoft SQL Server cũng được thực hiện trong thực đơn tắt sau khi chọn chuột phải trên tên máy chủ Microsoft SQL Server có trong tiện ích Enterprise Manager. Trên đây là một số tính năng cơ bản mà Enterprise Manager cung cấp cho các bạn khi quản trị cơ sở dữ liệu Microsoft SQL Server. Các tính năng còn lại chúng tôi sẽ lần lượt trình bày trong các chương kế tiếp.

### III.10. Nối kết từ Query Analyzer vào SQL

Mục đích của phần trình bày này là chúng tôi muốn các bạn có thể thấy được **hoạt động cơ bản** nhất trong mô hình khách chủ. Chúng ta sẽ sử dụng tiện ích Query Analyzer từ một **máy trạm** bất kỳ nối kết vào Microsoft SQL Server để thực hiện các thao tác trên cơ sở dữ liệu SQL được lưu trữ tại máy chủ.

Trở lại mô hình mạng cục bộ ở hình 1-24, giả sử rằng việc định nghĩa cấu hình nối kết vào Microsoft SQL Server là thành công. Chúng ta sẽ khởi động tiện ích Query Analyzer tại máy trạm WKS\_18, sau đó đăng nhập vào máy chủ Microsoft SQL Server TAMNT\_IP với tên tài khoản người dùng và mật khẩu hợp lệ. Nhấn OK để tiếp tục.

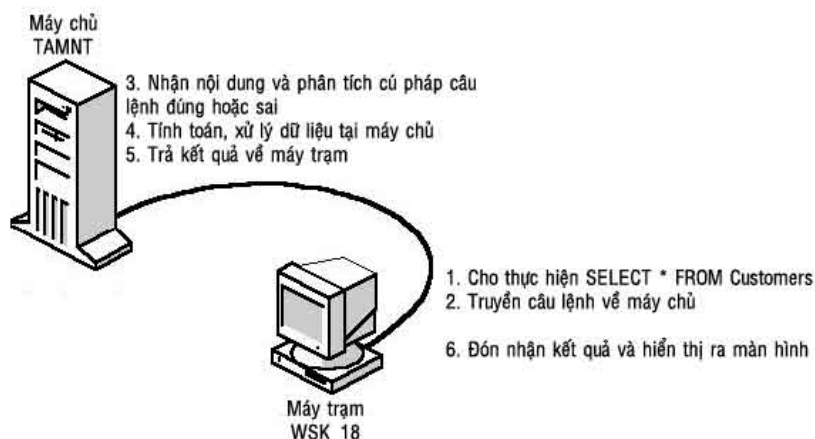


**Hình 1-36.** Màn hình đăng nhập khi khởi động tiện ích Query Analyzer.

Hệ thống sẽ kiểm tra xác thực tên tài khoản người dùng và mật khẩu có hợp lệ hay không? Nếu tất cả là hợp lệ thì các bạn có thể thực hiện các lệnh truy vấn tại máy trạm WKS\_18 vào cơ sở dữ liệu Microsoft SQL Server. Ngược lại khi không thực hiện thành công nối kết vào Microsoft SQL Server, các bạn cần phải xem lại mình đã định nghĩa cấu hình nối kết vào Microsoft SQL Server trong tiện ích Client Network Utility đúng chưa? Tên tài khoản người dùng và mật khẩu đã nhập vào đúng chưa?

Hình ảnh bên dưới mô tả thứ tự các xử lý được phân chia thực hiện trên máy chủ TAMNT và trên máy trạm WKS\_18 khi thực hiện một truy vấn đơn giản là liệt kê danh sách các khách hàng trong bảng CUSTOMERS trong cơ sở dữ liệu mặc định NorthWind của Microsoft SQL Server bằng câu lệnh truy vấn sau:

```
SELECT * FROM NorthWind..Customers
```



**Hình 1-37.** Các xử lý tại hai (2) nhánh trong mô hình khách chủ.

Tóm lại trong chương một chúng tôi đã giúp các bạn có cái nhìn tổng quan về mô hình khách chủ và làm quen với cơ sở dữ liệu Microsoft SQL Server thông qua việc sử dụng một số tiện ích có sẵn sau khi thực hiện cài đặt Microsoft SQL Server Desktop Database.



## Bài 2

# CÁC ĐỐI TƯỢNG TRONG CƠ SỞ DỮ LIỆU

## Tóm tắt

Lý thuyết 4 tiết - Thực hành 12 tiết

Mục tiêu	Các mục chính	Bài tập bắt buộc	Bài tập làm thêm
✓ Hiểu được cấu trúc của cơ sở dữ liệu SQL Server	I. Cơ sở dữ liệu của SQL Server	2.1	2.4
✓ Hiểu được các đối tượng có trong cơ sở dữ liệu bao gồm bảng (table) và bảng ảo (view).	IV. Bảng dữ liệu (Table)	2.2	2.5
	V. Kiểu dữ liệu do người dùng định nghĩa	2.3	2.9
	VI. Bảng ảo (Virtual table – View)	2.7	2.10
✓ Sử dụng các câu lệnh truy vấn SELECT để thực hiện các câu truy vấn trên bảng và bảng ảo		2.8	





# I. Cơ sở dữ liệu của SQL Server

## I.1. Khái niệm về cơ sở dữ liệu

Một cơ sở dữ liệu của Microsoft SQL Server là tập hợp dùng để chứa đựng các đối tượng: bảng (table), bảng ảo (view), thủ tục nội tại (stored procedure)... cho phép người sử dụng lưu trữ và khai thác các thông tin dữ liệu đã được tổ chức và lưu trữ bên trong đó. Thông thường sau khi cài đặt Microsoft SQL Server xong, hệ thống sẽ tự động tạo một vài cơ sở dữ liệu mặc định: Master, Model, Tempdb, Pubs, Northwind và Msdb. Một SQL Server được phép chứa tối đa có 32,767 các cơ sở dữ liệu khác nhau.

Một cơ sở dữ liệu của Microsoft SQL Server chỉ do một người dùng tạo ra, tuy nhiên trong cơ sở dữ liệu này có thể được phép có nhiều người truy cập và tạo ra nhiều đối tượng dữ liệu khác bên trong nó.

Trong phần này chúng tôi muốn giới thiệu cho các bạn biết sơ qua về ý nghĩa của từng loại cơ sở dữ liệu đã được tạo tự động trong quá trình cài đặt Microsoft SQL Server.

Tên dữ liệu	Ý nghĩa
Master	Là cơ sở dữ liệu chính yếu dùng để chứa thông tin của các bảng hệ thống nhằm quản lý tất cả các cơ sở dữ liệu khác hiện đang có trong SQL Server.
Model	Là cơ sở dữ liệu khuôn dạng mẫu (template) mà Microsoft SQL Server sử dụng để tạo lập ra các cơ sở dữ liệu mới. Trong đó cơ sở dữ liệu mới sẽ chứa đựng các đối tượng, quyền hạn hoàn toàn giống các đối tượng, quyền hạn hiện đang có trong cơ sở dữ liệu Model này.
Tempdb	Là cơ sở dữ liệu tạm thời (temporary database) mà Microsoft SQL Server sử dụng để chứa đựng các bảng tạm do người sử dụng tạo ra một cách tường minh hoặc là nơi lưu trữ các kết quả trung gian trong quá trình thực hiện các xử lý cho các truy vấn hoặc sắp xếp dữ liệu.
Pubs	Là cơ sở dữ liệu mẫu quản lý thông tin về các quyển sách (titles), tác giả (authors), nhà xuất bản (publishers)... phần lớn các thí dụ trong các sách, tài liệu tham khảo đều có sử dụng cơ sở dữ liệu này để minh họa.
Northwind	Là cơ sở dữ liệu mẫu rất quen thuộc với những người lập trình trong Microsoft Access. Cơ sở dữ liệu này quản lý thông tin của việc mua bán hàng bao gồm các bảng: hàng hóa (products), khách hàng (customers), nhà cung cấp (suppliers)...
Msdb	Là cơ sở dữ liệu được dùng cho dịch vụ SQL Server Agent, dịch vụ này sẽ tự động thực hiện các hành động mà người quản trị cơ sở dữ liệu đã đưa ra trong thời gian biểu như là: sao chép an toàn dữ liệu (backup), đồng bộ hóa dữ liệu (replicate)...

Chúng ta có thể hủy bỏ các loại cơ sở dữ liệu như là: Pubs, Northwind trong trường hợp không muốn sử dụng nữa mà không làm ảnh hưởng đến hệ thống Microsoft SQL Server.

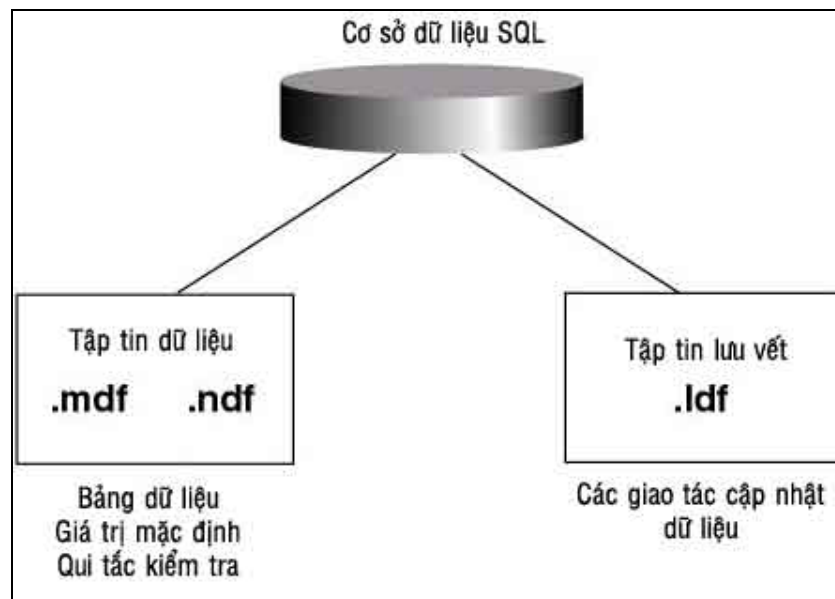
Tuy nhiên nếu dung lượng đĩa cứng của các bạn còn trống khá nhiều thì chúng ta không cần thiết hủy bỏ các cơ sở dữ liệu này làm gì.

## 1.2. Các tập tin vật lý lưu trữ cơ sở dữ liệu

Mặc dù phải quản lý nhiều đối tượng bên trong cơ sở dữ liệu nhưng Microsoft SQL Server chỉ tổ chức hai loại tập tin để lưu trữ.

Một cơ sở dữ liệu trong Microsoft SQL Server tối thiểu sẽ dùng hai (2) tập tin vật lý để lưu trữ dữ liệu:

- ✓ **Data file:** dùng lưu trữ dữ liệu.
- ✓ **Transaction log file:** dùng để lưu trữ các hành động thực hiện trên cơ sở dữ liệu trong quá trình sử dụng. Các hành động thực hiện trên CSDL gọi là các giao tác.



**Hình 2-1.** Các loại tập tin lưu trữ dữ liệu của SQL Server 2000.

Các tập tin lưu trữ cơ sở dữ liệu bên trong Microsoft SQL Server được phân chia thành ba loại tập tin vật lý khác nhau:

- ✓ **Tập tin dữ liệu chính** (Primary Data File): Đây là tập tin chính dùng để lưu trữ các thông tin hệ thống của cơ sở dữ liệu và phần còn lại dùng lưu trữ một phần dữ liệu. Phần mở rộng của tập tin này thông thường là \*.MDF.
- ✓ **Tập tin dữ liệu thứ yếu** (Secondary Data Files): Đây là tập tin dùng lưu trữ các đối tượng dữ liệu không nằm trong tập tin dữ liệu chính. Loại tập tin này không bắt buộc phải có khi tạo mới cơ sở dữ liệu. Phần mở rộng của tập tin này thông thường là \*.NDF.
- ✓ **Tập tin lưu vết** (Log Files): Đây là tập tin dùng lưu vết các giao tác – là những hành động cập nhật dữ liệu (thêm, sửa, xóa) vào các bảng do người sử dụng tác động trên cơ sở dữ liệu. Tập tin này sẽ hỗ trợ cho phép các bạn có thể hủy bỏ (rollback) các thao tác cập nhật dữ liệu đã được thực hiện hay giúp SQL Server phục hồi dữ liệu trong các trường hợp gặp sự cố như mất điện,... Phần mở rộng của tập tin này thông thường là \*.LDF.



Mặc định các tập tin dữ liệu của cơ sở dữ liệu SQL Server sẽ được lưu trữ trong thư mục: C:\MSSQL\DATA hoặc đường dẫn mà chúng ta đã chỉ định lại trong quá trình thực hiện cài đặt Microsoft SQL Server.

### 1.3. Tạo mới cơ sở dữ liệu

Sau khi có khái niệm về cách thức tổ chức các tập tin vật lý để lưu trữ dữ liệu trong Microsoft SQL Server, chúng ta sẽ tự tạo một cơ sở dữ liệu cho riêng mình nhằm lưu trữ các dữ liệu riêng biệt và đưa vào khai thác các dữ liệu đó.

Cách dễ nhất để các bạn tạo ra một cơ sở dữ liệu là sử dụng tiện ích Enterprise Manager. Chỉ những người với vai trò là quản trị hệ thống (sysadmin) thì mới có thể tạo lập cơ sở dữ liệu. Do đó các bạn có thể đăng nhập vào với tên tài khoản người dùng là **sa** để thực hiện việc tạo cơ sở dữ liệu mới cho ứng dụng của mình.

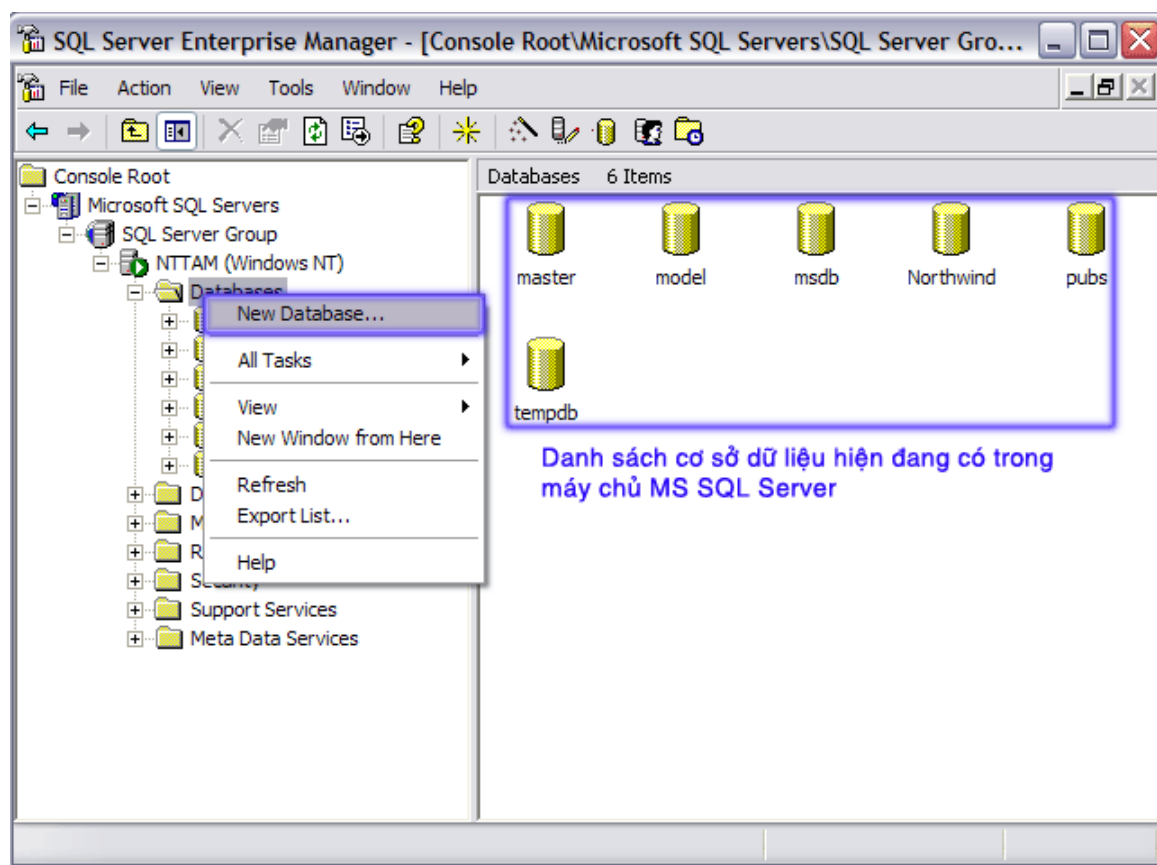
Trước khi giới thiệu từng bước tạo lập cơ sở dữ liệu, phần kế tiếp mà chúng tôi muốn trình bày là các thuộc tính của một cơ sở dữ liệu trong Microsoft SQL Server. Các thuộc tính nhằm giúp các bạn hiểu rõ thêm về bên trong cơ sở dữ liệu của Microsoft SQL Server, chúng gồm có:

- ✓ **Tên cơ sở dữ liệu** (database name): là duy nhất trong một Microsoft SQL Server, độ dài tối đa là 123 ký tự. Theo chúng tôi các bạn nên đặt tên cơ sở dữ liệu gợi nhớ. Thí dụ: QLBanhang (Quản lý bán hàng), QLBHocsinh (Quản lý học sinh)...
- ✓ **Vị trí tập tin** (File location): là tên và đường dẫn vật lý của các loại tập tin dữ liệu dùng để lưu trữ cơ sở dữ liệu của Microsoft SQL Server. Thông thường các tập tin này sẽ được lưu tại thư mục C:\MSSQL\DATA.
- ✓ **Tên tập tin** (File name): là tên luận lý của mỗi loại tập tin dữ liệu tương ứng mà hệ thống Microsoft SQL Server dùng để quản lý bên trong. Tương ứng mỗi loại tập tin dữ liệu sẽ có một tên tập tin riêng biệt.
- ✓ **Kích thước ban đầu** (Initial size): là kích thước khởi tạo của tập tin dữ liệu khi cơ sở dữ liệu mới được tạo lập. Đơn vị tính là MegaByte (MB). Thông thường kích thước ban đầu của một cơ sở dữ liệu mới tối thiểu phải bằng với kích thước của cơ sở dữ liệu Model, bởi vì Microsoft SQL Server sẽ lấy cơ sở dữ liệu Model làm khuôn dạng mẫu khi hình thành một cơ sở dữ liệu mới.
- ✓ **Việc tăng trưởng kích thước tập tin dữ liệu** (File growth): là các qui định cho việc tăng trưởng tự động kích thước tập tin dữ liệu, bởi vì các dữ liệu sẽ được lưu trữ ngày càng nhiều hơn so với kích thước ban đầu khi tạo lập. Việc tăng trưởng sẽ tự động làm tăng kích thước tập tin dữ liệu theo từng MB (in megabytes) hoặc theo tỷ lệ phần trăm (by percent) của kích thước hiện hành khi các dữ liệu bên trong Microsoft SQL Server lưu trữ gần đầy so với kích thước tập tin vật lý hiện thời. Mặc định kích thước tập tin dữ liệu sẽ được tăng tự động 10% khi dữ liệu lưu trữ gần đầy.
- ✓ **Kích thước tối đa tập tin dữ liệu** (Maximum file size): là việc qui định sự tăng trưởng tự động kích thước của các tập tin dữ liệu nhưng có giới hạn (restrict file growth) đến MB nào đó hoặc là không có giới hạn (un-restrict file growth). Trong trường hợp nếu các bạn chọn có giới hạn kích thước của tập tin dữ liệu thì chúng ta phải biết tự thêm vào các tập tin dữ liệu mới khi dữ liệu lưu trữ đã bằng với kích thước tối đa của tập tin dữ liệu. Các tập tin dữ liệu mới này chính là loại tập tin thứ yếu (Secondary data file) và chúng ta có thể lưu trữ các tập tin vật lý này tại các đĩa cứng khác có bên trong Microsoft SQL Server. Đây cũng là một trong nét đặc trưng của mô hình cơ sở dữ liệu phân tán (distributed database).

Đối với các CSDL thực tế, việc xác định các tham số về kích thước ban đầu rất quan trọng vì nhiều lý do. Để đảm bảo có đủ không gian lưu trữ dữ liệu, bạn cần dành trước cho CSDL phòng khi những ứng dụng hay CSDL khác sử dụng hết đĩa cứng. CSDL có kích thước nhỏ cũng sẽ ảnh hưởng tới tốc độ do SQL Server cần phải thực hiện nhiều lần thao tác mở rộng kích thước tập tin CSDL khi có dữ liệu thêm mới. Ngoài ra, đa số các dữ liệu trong CSDL thực tế theo thời gian không thể xóa bỏ mà cần phải lưu trữ (backup) lại trước. Việc lưu trữ và phục hồi (restore) dữ liệu cũng ảnh hưởng bởi kích thước các tập tin do chúng phải đủ nhỏ để lưu trên các đĩa CD-ROM hay băng từ.

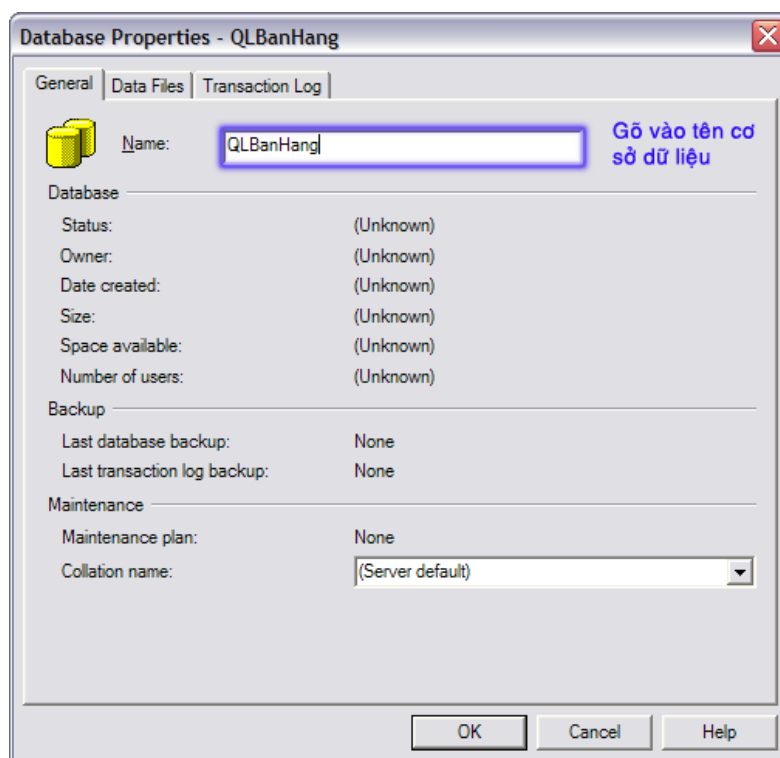
Các bước mà chúng tôi mô tả bên dưới sẽ giúp các bạn tạo ra một cơ sở dữ liệu mới bằng tiện ích Enterprise Manager.

- ✓ Bước 1: Khởi động ứng dụng Enterprise Manager, chọn một (1) Microsoft SQL Server đã được đăng ký quản trị trước đó. Chọn chức năng **New Database...** trong thực đơn tắt sau khi nhấn chuột phải trên đối tượng Database.



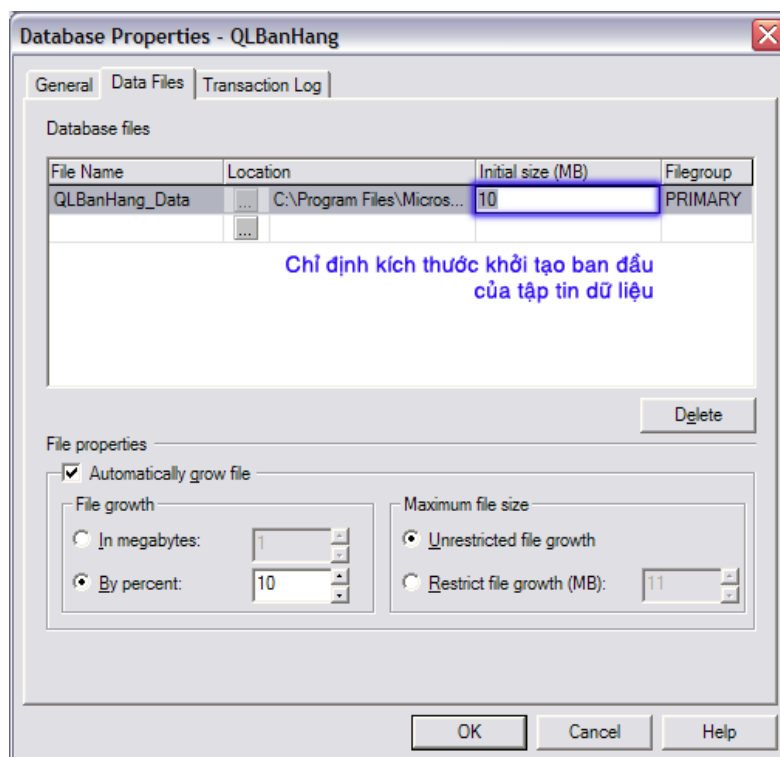
**Hình 2-2.** Chọn chức năng New Database.

- ✓ Bước 2: Trong màn hình các thuộc tính của cơ sở dữ liệu (Database Properties) tại trang General gõ vào tên cơ sở dữ liệu muốn tạo mới.



**Hình 2-3.** Màn hình các thuộc tính trang General.

- ✓ Bước 3: Trong màn hình các thuộc tính của cơ sở dữ liệu (Database Properties) tại trang Data Files, chỉ định kích thước ban đầu khi khởi tạo của tập tin dữ liệu chính, kế tiếp thay đổi các thuộc tính khác (nếu cần). Chuyển sang trang Transaction Log để thay đổi các thuộc tính của tập tin lưu vết theo cách tương tự.



**Hình 2-4.** Màn hình các thuộc tính trang Data Files.



Tùy thuộc vào kích thước của cơ sở dữ liệu mà thời gian thực hiện tạo cơ sở dữ liệu sẽ nhanh hoặc lâu. Ngoài ra chúng ta còn có thể tạo mới một cơ sở dữ liệu bằng câu lệnh **CREATE DATABASE** được thực hiện trong tiện ích Query Analyzer. Các thành phần trong câu lệnh này hoàn toàn giống với các thuộc tính của cơ sở dữ liệu mà chúng tôi đã giới thiệu trong phần trên.



**Ví dụ:**

Để tạo ra một cơ sở dữ liệu có tên *QLBanHang* với kích thước ban đầu lúc khởi tạo của tập tin dữ liệu chính là 50MB, tự động tăng kích thước lên 10% khi dữ liệu bị đầy, kích thước tăng trưởng tập tin dữ liệu tối đa không quá 200MB. Và tập tin lưu vết với kích thước ban đầu lúc khởi tạo là 10MB, tự động tăng kích thước tập tin lên 5MB khi dữ liệu bị đầy, kích thước tăng trưởng tập tin không giới hạn. Các bạn sẽ thực hiện câu lệnh **CREATE DATABASE** như sau:

```
CREATE DATABASE QLBanHang
ON PRIMARY
( NAME=QLBanHang_Data,
  FILENAME='C:\MSSQL7\DATA\QLBANHANG.MDF',
  SIZE=50MB,
  MAXSIZE=200MB,
  FILEGROWTH=10%)
LOG ON
( NAME=QLBanHang_Log,
  FILENAME='C:\MSSQL7\DATA\SAMPLE.LDF',
  SIZE=10MB,
  MAXSIZE=UNLIMITED,
  FILEGROWTH=5MB)
```

Trong phần giáo trình này hầu hết chúng tôi sẽ chỉ dẫn các bạn hai (2) cách để thực hiện các hành động trong Microsoft SQL Server: hoặc thực hiện các thao tác trong tiện ích Enterprise Manager hoặc thực hiện các câu lệnh T-SQL (Transaction SQL) trong tiện ích Query Analyzer để tạo ra các đối tượng bên trong Microsoft SQL Server. Tùy theo khả năng của cá nhân mình mà các bạn sẽ chọn tiện ích nào để làm việc. Tuy nhiên theo chúng tôi thì các bạn nên phối hợp cả hai để cho môi trường làm việc trở nên linh hoạt hơn.

## 1.4. Xóa cơ sở dữ liệu đã có

Một cơ sở dữ liệu sau khi tạo xong sau một thời gian dài mà các bạn không còn khai thác dữ liệu bên trong đó thì các bạn có thể hủy bỏ để làm cho dung lượng đĩa trống được tăng lên. Tuy nhiên phải chắc rằng các thông tin dữ liệu trong cơ sở dữ liệu mà các bạn dự định xóa sẽ không còn hữu ích về sau nữa. Bởi vì chúng ta không thể khôi phục khi đã xóa.

Để hủy bỏ cơ sở dữ liệu trong Microsoft SQL Server chúng ta có nhiều cách thực hiện: sử dụng câu lệnh **DROP DATABASE**, nhấn phím Delete hoặc nhấn chuột trên biểu tượng Delete và xác định đồng ý hủy bỏ cơ sở dữ liệu đã chọn trong tiện ích Enterprise Manager.

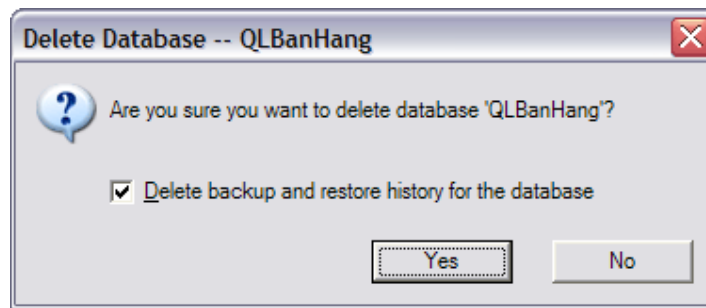


**Ví dụ:**

Để hủy bỏ cơ sở dữ liệu *QLBanHang*, thực hiện câu lệnh **DROP DATABASE** như sau:

```
DROP DATABASE QLBanHang
```

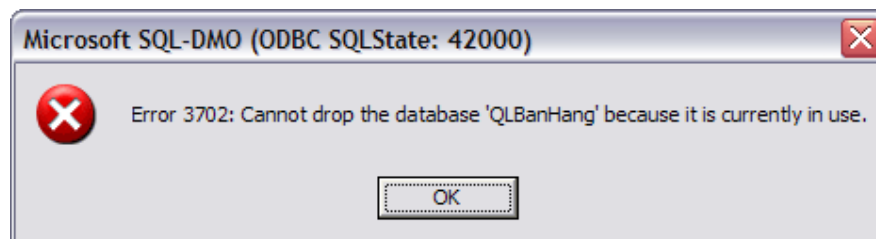
Hoặc nhấn phím Delete trên tên của cơ sở dữ liệu này trong tiện ích Enterprise Manager và xác nhận là đồng ý (chọn Yes) để hủy bỏ cơ sở dữ liệu QLBanHang.



**Hình 2-5.** Hộp thoại xác nhận đồng ý hủy bỏ cơ sở dữ liệu.

Hệ thống Microsoft SQL Server không cho người sử dụng có thể hủy bỏ các cơ sở dữ liệu hệ thống như là: Master, Model, Tempdb bởi vì các cơ sở dữ liệu luôn được hệ thống Microsoft SQL Server sử dụng.

Ngoài ra để hủy bỏ một cơ sở dữ liệu thành công thì phải đảm bảo không còn người sử dụng nào đang truy cập vào cơ sở dữ liệu đó. Trong trường hợp khi thực hiện hủy bỏ cơ sở dữ liệu đang còn người sử dụng truy cập thì hệ thống sẽ hiển thị thông báo bên dưới.



**Hình 2-6.** Hộp thoại thông báo hành động hủy bỏ cơ sở dữ liệu thất bại.



**Chú ý:**

Tuyệt đối không xóa cơ sở dữ liệu bằng cách sử dụng Windows Explorer hoặc Windows Commander để hủy bỏ các loại tập tin dữ liệu trong thư mục C:\MSSQL\DATA\ vì làm như thế sẽ ảnh hưởng trực tiếp đến hệ thống cơ sở dữ liệu Microsoft SQL Server.

## II. Bảng dữ liệu (Table)

### II.1. Khái niệm về bảng

Bên trong một cơ sở dữ liệu của Microsoft SQL Server có rất nhiều kiểu đối tượng khác nhau. Đối tượng đầu tiên mà chúng tôi muốn đề cập trong phần này cũng là đối tượng cơ sở của một cơ sở dữ liệu Microsoft SQL Server đó là đối tượng bảng dữ liệu.

Giống như các loại cơ sở dữ liệu khác, bảng trong Microsoft SQL Server cũng dùng cho việc lưu trữ các thông tin dữ liệu của những đối tượng, thực thể trong thế giới thực muốn được lưu trữ vào máy tính. Thí dụ như thông tin các khách hàng, nhà cung cấp, hóa đơn xuất hàng, hóa đơn nhập hàng... các thông tin này sẽ được tổ chức thành các dòng (row) và các cột (column) mà thông thường định dạng của nó sẽ gần giống như một danh sách trong bảng





tính Excel của các bạn thường dùng. Mỗi dòng dữ liệu trong bảng thường phải là duy nhất do đó sẽ có một hoặc nhiều cột bên trong bảng sẽ tham gia làm khóa chính (**primary key**). Giá trị dữ liệu tại các cột tham gia làm khóa chính là duy nhất không bị trùng lặp bên trong bảng.

Hầu như các bảng trong một cơ sở dữ liệu sẽ có các quan hệ với các bảng khác nhằm kiểm tra tính tồn tại dữ liệu (**foreign key**) và trao đổi, chia sẻ thông tin với nhau. Thí dụ trong cơ sở dữ liệu quản lý bán hàng, bảng nhà cung cấp sẽ có quan hệ với bảng đơn đặt hàng để nói rằng việc đặt hàng cho những nhà cung cấp cung ứng các loại vật tư nào.

Khi đọc đến đây chắc rằng các bạn đã có khái niệm cơ bản về bảng dữ liệu là gì. Do chúng tôi giả sử rằng các bạn đã biết qua về cơ sở dữ liệu Microsoft Access nên trong phần này chúng tôi không trình bày sâu về các khái niệm bên trong mô hình cơ sở dữ liệu quan hệ.

## II.2. Các thuộc tính của bảng

Trước khi thực hiện việc tạo cấu trúc bảng, chúng tôi muốn các bạn xem xét một số các thuộc tính liên quan đến bảng. Mục đích của phần trình bày này nhằm giúp các bạn hiểu rõ về các thuộc tính liên quan đến bảng để có thể thực hiện việc tạo cấu trúc bảng được tốt hơn. Các thuộc tính cơ bản của một bảng dữ liệu bao gồm:

- ✓ **Tên bảng** (table name): là tên của bảng dữ liệu do chúng ta qui định, độ dài không quá 128 ký tự. Tên bảng phải duy nhất bên **trong phạm vi** của người đã tạo ra nó trong một cơ sở dữ liệu. Điều này có nghĩa là nếu người sử dụng ND1 đã tạo bảng KHACHHANG thì ND2 **cũng được phép** tạo ra bảng KHACHHANG trong cùng một cơ sở dữ liệu.
- ✓ **Tên cột** (column name): là tên của các cột bên trong bảng, tên của các cột bên trong bảng phải duy nhất.
- ✓ **Kiểu dữ liệu** (data type): qui định kiểu dữ liệu mà cột sẽ lưu trữ bên trong bảng. Ngoài các kiểu dữ liệu cơ sở mô tả ở danh sách bên dưới, Microsoft SQL Server còn cho phép chúng ta định nghĩa ra các kiểu dữ liệu mới, cách thức định nghĩa kiểu dữ liệu mới sẽ được trình bày trong phần gần cuối của bài này.

Kiểu dữ liệu	Kích thước	Miền giá trị dữ liệu lưu trữ
<b>► Các kiểu dữ liệu dạng số nguyên</b>		
Int	4 Bytes	Từ -2,147,483,648 đến +2,147,483,647
Smallint	2 Bytes	Từ -32,768 đến +32,767
Tinyint	1 Byte	Từ 0 đến 255
Bit	1 Byte	0,1 Hoặc Null
<b>► Các kiểu dữ liệu dạng số thập phân</b>		
Decimal, Numeric	17 Bytes	Từ $-10^{38}$ đến $+10^{38}$
<b>► Các kiểu dữ liệu dạng số thực</b>		
Float	8 Bytes	Từ $-1.79E + 308$ đến $+1.79E + 308$





Kiểu dữ liệu	Kích thước	Miền giá trị dữ liệu lưu trữ
Real	4 Bytes	Từ $-1.79E + 308$ đến $+1.79E + 308$
<b>► Các kiểu dữ liệu dạng chuỗi có độ dài cố định (fixed)</b>		
Char	N Bytes	Từ 1 đến 8,000 ký tự, mỗi ký tự là 1 byte.
<b>► Các kiểu dữ liệu dạng chuỗi có độ dài biến đổi (variable)</b>		
Varchar	N Bytes	Từ 1 đến 8,000 ký tự, mỗi ký tự là 1 byte.
Text	N Bytes	Từ 1 đến 2,147,483,647 ký tự, mỗi ký tự là 1 byte.
<b>► Các kiểu dữ liệu dạng chuỗi dùng font chữ Unicode (national)</b>		
Nchar	2*N Bytes	Từ 1 đến 4,000 ký tự, mỗi ký tự là 2 bytes.
Nvarchar	2*N Bytes	Từ 1 đến 4,000 ký tự, mỗi ký tự là 2 bytes.
Ntext	2*N Bytes	Từ 1 đến 1,073,741,823 ký tự, mỗi ký tự là 2 bytes.
<b>► Các kiểu dữ liệu dạng tiền tệ</b>		
Money	8 Bytes	Từ $-922,337,203,685,477.5808$ đến $+922,337,203,685,477.5807$
Smallmoney	4 Bytes	Từ $-214,748.3648$ đến $+214,748.3647$
<b>► Các kiểu dữ liệu dạng ngày và giờ</b>		
Datetime	8 Bytes	Từ 01/01/1753 đến 31/12/9999
Smalldatetime	4 Bytes	Từ 01/01/1900 đến 06/06/2079
<b>► Các kiểu dữ liệu dạng chuỗi nhị phân (binary string)</b>		
Binary	N Bytes	Từ 1 đến 8,000 bytes
Varbinary	N Bytes	Từ 1 đến 8,000 bytes
Image	N Bytes	Từ 1 đến 2,147,483,647 bytes

**Danh sách các kiểu dữ liệu cơ bản trong Microsoft SQL Server.**



**Chú ý:**

Các kiểu dữ liệu *varchar*, *nvarchar*, *varbinary* là những kiểu dữ liệu mà hệ thống Microsoft SQL Server sẽ giúp cho việc lưu trữ dữ liệu được tối ưu hơn.

**Ví dụ:**

Giả sử cột tên khách hàng trong bảng khách hàng có kiểu dữ liệu là `varchar(30)`. Nếu giá trị dữ liệu tên khách hàng cần lưu là: “Cao Minh Trung” thì khi đó dữ liệu lưu trữ thật sự chỉ cần 14 bytes. Ngược lại nếu kiểu dữ liệu cột khách hàng là `char(30)` thì dữ liệu lưu trữ vẫn là 30 bytes bao gồm: 14 bytes dùng chứa chữ “Cao Minh Trung” và 16 bytes chứa các khoảng trắng (blank)

- ✓ **Độ dài dữ liệu (data length):** dùng để qui định độ dài dữ liệu mà cột sẽ lưu trữ đối với các kiểu dữ liệu dạng chuỗi, số.
- ✓ **Số ký số lưu trữ (precision):** là số ký số tối đa mà các kiểu dữ liệu dạng số có thể lưu trữ được.
- ✓ **Số lẻ lưu trữ (scale):** là số lẻ tối đa mà các kiểu dữ liệu dạng số thập phân dùng để lưu trữ.

**Ví dụ:**

Đối với các kiểu dữ liệu `Int` thì có số ký số lưu trữ tối đa là 10 số (từ -2,147,483,648 đến +2,147,483,647), không có chứa số lẻ (vì là số nguyên) và kích thước cần để lưu trữ là 4 bytes (do hệ thống qui định).

- ✓ **Cho phép để trống dữ liệu (allow null):** qui định dữ liệu có thể được phép để trống hay là không tại một cột bên trong bảng trong trường hợp thêm mới hoặc sửa đổi mẫu tin. Thí dụ như cột họ tên khách hàng là không được phép để trống, tuy nhiên cột số điện thoại di động của khách hàng là có thể để trống.
- ✓ **Cột định danh (identity):** qui định cột dữ liệu của một bảng sẽ làm cột định danh trong bảng. Giá trị trên cột định danh phải là một số nguyên không trùng lặp (nên được gọi là định danh) do hệ thống Microsoft SQL Server tự động cấp phát. Do đó chỉ có các kiểu dữ liệu của cột là số **int**, **smallint**, **tinyint**, **decimal** hoặc **numeric** mới được phép làm cột định danh. Trong một bảng chỉ được phép có duy nhất một cột làm cột định danh. Thuộc tính này có hoạt động gần giống kiểu dữ liệu **AutoNumber** của **Microsoft Access**, tuy nhiên người sử dụng có thể chỉ định giá trị của **số cấp phát đầu tiên** thông qua thuộc tính **Identity Seed** và số đơn vị sẽ được tăng cho các mẫu tin kế tiếp thông qua thuộc tính **Identity Increment**.
- ✓ **Giá trị mặc định (default value):** là giá trị mặc nhiên sẽ được gán vào cột dữ liệu khi người sử dụng thêm mới một mẫu tin nhưng lại để trống giá trị tại cột dữ liệu đó.

SQL Server còn có các kiểu dữ liệu đặc biệt khác. **Cursor** và **Table** là hai kiểu dữ liệu dùng để chứa các dữ liệu có cấu trúc dạng bảng, cursor đã được hỗ trợ từ phiên bản SQL Server trước trong khi table là kiểu dữ liệu mới của SQL Server 2000. Ngoài ra, kiểu **SQL\_variant** có thể dùng để lưu giá trị thuộc bất cứ kiểu nào trừ text, ntext, image. Kiểu **uniqueidentifier** dùng để lưu một giá trị duy nhất có thể dùng để làm khoá primary key. Giá trị của **uniqueidentifier** được phát sinh bằng hàm của SQL Server, đảm bảo rằng bất cứ hai giá trị nào trong bất cứ CSDL, table nào cũng khác nhau.

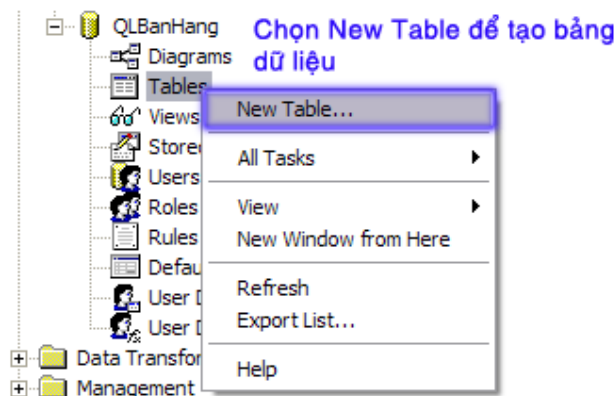
## II.3. Tạo cấu trúc bảng dữ liệu

Sau khi xem xét và hiểu được các thuộc tính liên quan đến cấu trúc của bảng, trong phần này chúng tôi sẽ hướng dẫn các bạn các cách để tạo cấu trúc bảng dữ liệu mới. Để tạo cấu trúc bảng chúng tôi hướng dẫn các bạn hai (2) cách thực hiện. Đầu tiên là tạo cấu trúc bảng bằng tiện ích Enterprise Manager. Kể từ bây giờ chúng tôi xem như các bạn đã đăng

ký quản trị một Microsoft SQL Server và bên trong Microsoft SQL Server này, cơ sở dữ liệu quản lý bán hàng (**QLBanHang**) đã được tạo lập. Các bảng dữ liệu và những đối tượng khác ở các phần trình bày kế tiếp sẽ được tạo ra bên trong cơ sở dữ liệu **QLBanHang** này.

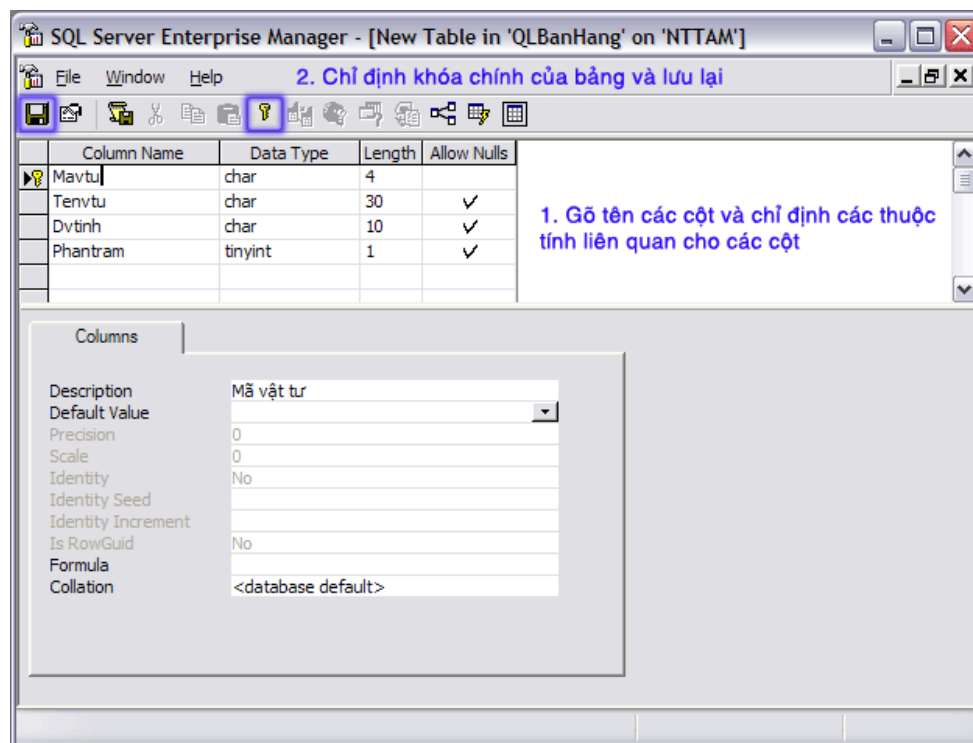
Các bước thực hiện việc tạo bảng dữ liệu trong Enterprise Manager như sau:

- ✓ Bước 1: Trong ứng dụng Enterprise Manager, mở rộng cơ sở dữ liệu để thấy các đối tượng bên trong. Nhấn chuột phải trên đối tượng **Tables**, chọn chức năng **New Table...** trong thực đơn tắt.



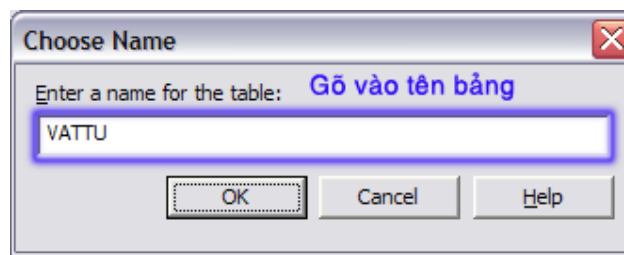
**Hình 2-7.** Chọn chức năng New Table để tạo mới bảng.

- ✓ Bước 2: Trong màn hình thiết kế cấu trúc bảng (design table), lần lượt gõ vào tên các cột bên trong bảng, chọn lựa các kiểu dữ liệu tương ứng thích hợp và chỉ định các thuộc tính cần thiết cho các cột bên trong bảng.



**Hình 2-8.** Màn hình chỉ định tên bảng mới.

- ✓ Bước 3: Định nghĩa khóa chính cho bảng và lưu lại cấu trúc bảng vừa định nghĩa. Đóng màn hình thiết kế cấu trúc bảng lại để kết thúc quá trình tạo cấu trúc bảng bằng tiện ích Enterprise Manager.



**Hình 2-9.** Màn hình thiết kế cấu trúc bảng.

Bên cạnh đó chúng ta cũng có thể tạo cấu trúc bảng bằng câu lệnh CREATE TABLE, cú pháp đầy đủ của câu lệnh này rất phức tạp. Do đó trong phần kế tiếp chúng tôi sẽ trình bày từng thành phần nhỏ bên trong câu lệnh CREATE TABLE để hướng dẫn các bạn từng bước làm quen với các cú pháp của câu lệnh này. Sau đó các bạn có thể kết hợp các thành phần này lại trong một câu lệnh CREATE TABLE để xây dựng cấu trúc bảng dữ liệu hoàn chỉnh.

## II.4. Tạo cấu trúc bảng đơn giản

Cú pháp CREATE TABLE cho phép các bạn tạo ra cấu trúc bảng gồm: tên bảng, tên các cột cùng với kiểu dữ liệu tương ứng và chỉ định giá trị dữ liệu tại các cột bên trong bảng không được phép bỏ trống. Mặc định giá trị dữ liệu tại một cột không tham gia làm khóa chính của bảng được phép bỏ trống.

Cú pháp:

```
CREATE TABLE Tên_bảng
(
    Tên_cột1    Kiểu_dữ_liệu [NOT NULL] ,
    Tên_cột2    Kiểu_dữ_liệu [NOT NULL] [, ...]
)
```

Trong đó:

- ✓ Tên bảng, tên cột và kiểu dữ liệu đã được mô tả ở phần II.2 các thuộc tính của bảng.
- ✓ Từ khóa **NOT NULL** chỉ định không cho phép dữ liệu tại cột được phép bỏ trống.



### Ví dụ:

Để tạo bảng có tên là VATTU (vật tư) gồm có những cột và kiểu dữ liệu mô tả như hình 2-9, chỉ có dữ liệu tại cột phần trăm (Phantram) là được phép bỏ trống. Chúng ta thực hiện câu lệnh CREATE TABLE như sau:

```
CREATE TABLE VATTU
(
    Mavtu      CHAR(4)  NOT NULL ,
    Tenvtu     CHAR(30) NOT NULL ,
    DvTinh     CHAR(10) NOT NULL ,
    Phantram    TinyInt
)
```

## II.5. Xóa cấu trúc bảng

Trong quá trình xây dựng cấu trúc bảng, chúng ta có thể hủy bỏ các bảng có cấu trúc chưa đúng để tạo lại bảng với cấu trúc mới. Tuy nhiên lệnh hủy bỏ cấu trúc bảng chỉ nên thực hiện khi bảng chưa hề chứa một dòng dữ liệu nào cả bởi vì khi xóa bảng thì chắc rằng dữ liệu mà bảng đang lưu trữ cũng sẽ bị mất.

Trong thực tế thì lệnh xóa bỏ cấu trúc bảng rất ít được thực hiện bởi vì nó rất nguy hiểm, khi đó nó sẽ làm mất toàn bộ các dữ liệu bên trong. Vì thế các bạn nên hạn chế sử dụng lệnh này.

Cú pháp:

```
DROP TABLE Danh_sách_tên_các_bảng
```

Trong đó:

- ✓ Danh sách tên bảng: là danh sách tên các bảng mà các bạn muốn hủy bỏ, tên các bảng muốn hủy bỏ được ngăn cách nhau bởi dấu phẩy (.). Thông thường muốn hủy bỏ các bảng có quan hệ trong cơ sở dữ liệu, chúng ta bắt buộc phải hủy bỏ các bảng bên nhánh quan hệ N trước và sau đó mới đến hủy bỏ các bảng bên nhánh 1.



### Ví dụ:

Để hủy bỏ bảng VATTU (vật tư) vừa tạo ở trên. Chúng ta thực hiện câu lệnh **DROP TABLE** như sau:

```
DROP TABLE VATTU
```



### Chú ý:

Trong các thí dụ kế tiếp, nếu bảng nào đã có tồn tại trong cơ sở dữ liệu rồi thì trước khi tạo lại cấu trúc mới của bảng đó, các bạn phải ra lệnh hủy bỏ bảng hiện có rồi mới thực hiện lệnh tạo lại bảng với cấu trúc mới. Nếu không thì Microsoft SQL Server sẽ thông báo rằng bảng đã có trong cơ sở dữ liệu rồi!



### Chú ý dành cho giáo viên:

Trong quá trình thực hành, lỗi mà học viên hay mắc phải nhất là tạo ra các đối tượng đã có trong CSDL. Giáo viên nên nhấn mạnh lỗi này, chỉ ra cách xác định các đối tượng trong CSDL bằng Enterprise Manager cũng như lỗi thông báo bởi Query Analyzer.

## II.6. Tạo cấu trúc bảng có cột định danh

Với cú pháp **CREATE TABLE** bên dưới cho phép các bạn tạo ra cấu trúc bảng có một cột định danh. Dữ liệu tại cột này sẽ do hệ thống Microsoft SQL Server cấp phát được đảm bảo là không trùng lặp và tất nhiên là phải khác trống và là kiểu số nguyên. Trong một bảng chỉ có tối đa một cột được chỉ định làm cột định danh.

Cú pháp:

```
CREATE TABLE Tên_bảng
(
    Tên_cột1 Kiểu_dữ_liệu_số
    IDENTITY [(Số_bắt_đầu, Chỉ_số_tăng)] ,
    Tên_cột2 Kiểu_dữ_liệu [NOT NULL] [, ...]
```

)

Trong đó:

- ✓ Kiểu dữ liệu số: gồm các kiểu dữ liệu dạng số nguyên như int, smallint, tinyint, numeric và decimal. Đối với hai kiểu cuối là numeric và decimal thì phải chỉ định không lấy số lẻ nào cả.
- ✓ Số bắt đầu: là số mà Microsoft SQL Server sử dụng để cấp phát cho mẫu tin đầu tiên. Mặc định là 1.
- ✓ Chỉ số tăng: là chỉ số mà Microsoft SQL Server cộng lên để cấp phát cho những mẫu tin kế tiếp. Mặc định là 1.



**Ví dụ:**

Để tạo bảng có tên là DONDH (đơn đặt hàng) gồm có những cột như: số đặt hàng có kiểu dữ liệu là số nguyên, ngày đặt hàng có kiểu dữ liệu là ngày, mã nhà cung cấp có kiểu dữ liệu là chuỗi và chiều dài 3 ký tự. Dữ liệu tại các cột không được phép trống và bảng này có cột định danh là cột số đặt hàng, giá trị khởi tạo đầu tiên là 1,000 và mỗi lần tăng kế tiếp là 5. Chúng ta thực hiện câu lệnh CREATE TABLE như sau:

```
CREATE TABLE DONDH
(
    Sodh      INT          IDENTITY (1000 , 5) ,
    Ngaydh    DATETIME    NOT NULL ,
    Manhacc   CHAR(3)     NOT NULL
)
```

## II.7. Thay đổi cấu trúc bảng

Trong thực tế việc thay đổi cấu trúc bảng vẫn thường được thực hiện đối với các bảng đã có chứa nhiều dữ liệu. Trong trường hợp này chúng ta **tuyệt đối không hủy bỏ** bảng hiện có và tạo lại cấu trúc mới bởi vì làm như thế là chúng ta sẽ mất đi tất cả các dữ liệu đang hiện có bên trong bảng.

Chúng ta có thể thay đổi cấu trúc bảng bằng tiện ích Enterprise Manager trong màn hình thiết kế bảng. Tuy nhiên trong phần này chúng tôi sẽ giới thiệu cho các bạn các tính năng mà câu lệnh **ALTER TABLE** sẽ mang đến cho các bạn trong khi thực hiện việc thay đổi cấu trúc bảng.

## II.8. Thêm một cột mới trong bảng

Với cú pháp **ALTER TABLE** bên dưới cho phép các bạn **thêm vào một hoặc nhiều cột mới** vào trong bảng hiện đang có. Tên các cột mới phải khác với tên các cột hiện đang có bên trong bảng. Mặc nhiên dữ liệu của các cột mới thêm vào phải được phép bỏ trống.

Cú pháp:

```
ALTER TABLE Tên_bảng
    ADD Tên_cột Kiểu_dữ_liệu [, ...]
```

Trong đó:

- ✓ Tên cột: tên của cột mới được thêm vào bảng.
- ✓ Kiểu dữ liệu: kiểu dữ liệu tương ứng của cột mới.

**Ví dụ:**

Để thêm vào bảng *DONDH* (đơn đặt hàng) một cột ngày dự kiến nhận hàng có kiểu dữ liệu là ngày. Chúng ta thực hiện câu lệnh **ALTER TABLE** như sau:

```
ALTER TABLE DONDH  
ADD Ngaydknh DATETIME
```

## II.9. Hủy bỏ cột hiện có bên trong bảng

Với cú pháp **ALTER TABLE** bên dưới cho phép các bạn **hủy bỏ một hoặc nhiều cột** hiện có bên trong bảng. Nên nhớ rằng việc hủy bỏ các cột đồng nghĩa với các dữ liệu mà cột hiện đang lưu trữ cũng sẽ bị mất theo. Do đó cần thận trọng khi sử dụng câu lệnh này.

Cú pháp:

```
ALTER TABLE Tên_bảng  
DROP COLUMN Tên_cột [, ...]
```

Trong đó:

- ✓ Tên cột: tên cột sẽ bị hủy bỏ ra khỏi bảng.

**Ví dụ:**

Để hủy bỏ cột ngày dự kiến nhận hàng trong bảng *DONDH* (đơn đặt hàng) vừa thêm ở thí dụ trên. Chúng ta thực hiện câu lệnh **ALTER TABLE** như sau:

```
ALTER TABLE DONDH  
DROP COLUMN Ngaydknh
```

## II.10. Sửa đổi kiểu dữ liệu của cột

Với cú pháp **ALTER TABLE** bên dưới cho phép các bạn **sửa đổi kiểu dữ liệu** của cột hiện có trong bảng. Đối với các kiểu dữ liệu dạng **text**, **ntext** hoặc **image** thì không thể đổi sang kiểu dữ liệu khác. Thông thường chúng tôi rất ít khi sửa đổi kiểu dữ liệu của các cột một cách trực tiếp mà thay vào đó chúng ta thêm một cột mới với kiểu dữ liệu như mong muốn vào bảng, sau đó cập nhật dữ liệu của cột hiện có sang cột mới vừa thêm vào, kiểm tra việc dữ liệu sau khi cập nhật có đúng theo như mong muốn không. Cuối cùng là hủy bỏ cột dữ liệu cũ và đổi tên cột dữ liệu mới thành tên của cột dữ liệu cũ trước đó.

Cú pháp:

```
ALTER TABLE Tên_bảng  
ALTER COLUMN Tên_cột Kiểu_dữ_liệu_mới
```

**Ví dụ:**

Để sửa lại kiểu dữ liệu và tăng độ dài lưu trữ của cột đơn vị tính lên 20 ký tự trong bảng VATTU (vật tư). Chúng ta thực hiện câu lệnh ALTER TABLE như sau:

```
ALTER TABLE VATTU
ALTER COLUMN Dvtinh VARCHAR(20)
```

## II.11. Đổi tên cột, tên bảng dữ liệu

Trong thực tế việc thay đổi tên cột hoặc tên bảng dữ liệu là **rất hạn chế** bởi vì chúng sẽ làm ảnh hưởng rất nhiều đến các đoạn chương trình có tham chiếu đến tên cột hoặc tên bảng ở đâu đó ứng dụng. Chúng ta chỉ thực hiện việc này thật sự khi cần thiết.

Để có thể đổi tên cột hoặc tên bảng chúng ta có thể vào trực tiếp tiện ích Enterprise Manager. Tuy nhiên chúng tôi muốn giới thiệu các bạn thủ tục nội tại hệ thống (system stored procedure) có tên là **sp\_rename**.

Các thủ tục nội tại hệ thống là một tập hợp các thủ tục do Microsoft SQL Server tạo ra và cung cấp cho người sử dụng thực hiện một số xử lý sẵn có bên trong Microsoft SQL Server.

Cú pháp:

```
EXEC sp_rename "Tên_bảng[Tên_cột]", "Tên_mới" [, "COLUMN"]
```

Trong đó:

- ✓ EXEC: lệnh dùng để thực thi (execute) các thủ tục nội tại bên trong Microsoft SQL Server.
- ✓ Tên bảng: tên của bảng mà chúng ta sẽ đổi tên hoặc tên bảng chứa tên cột mà chúng ta muốn đổi tên.
- ✓ Tên cột: tên cột mà chúng ta muốn đổi tên. Khi muốn đổi tên một cột trong bảng thì chúng ta phải chỉ ra đầy đủ tên bảng chứa tên cột muốn đổi tên.
- ✓ Tên mới: tên mới của cột hoặc tên mới của bảng sau khi đổi.
- ✓ COLUMN: từ khóa chỉ được sử dụng khi thay đổi tên cột.

**Ví dụ:**

Để thay đổi tên cột họ tên nhà cung cấp trong bảng NHACC (nhà cung cấp) từ cột cũ Tennhacc thành cột mới là Hotenncc. Chúng ta thực hiện câu lệnh như sau:

```
EXEC sp_rename "NHACC.Tennhacc", "Hotenncc", "COLUMN"
```

Để thay đổi tên bảng hiện có của NHACC thành tên mới là NHACCAP. Chúng ta thực hiện câu lệnh như sau:

```
EXEC sp_rename "NHACC", "NHACCAP"
```



## III. Kiểu dữ liệu do người dùng định nghĩa

### III.1. Khái niệm

Các phần còn lại trong chương này chúng ta sẽ lần lượt đi qua các đối tượng thường được sử dụng trong các cơ sở dữ liệu của những **ứng dụng lớn**, nhằm đảm bảo tính nhất quán về cấu trúc và dễ dàng có thể sửa đổi cấu trúc khi có yêu cầu. Đối tượng đầu tiên mà chúng ta cùng xem xét đó chính là kiểu dữ liệu do người dùng định nghĩa (UDDT).

Giống như những ngôn ngữ lập trình khác, Microsoft SQL Server cho phép chúng ta có thể định nghĩa ra các kiểu dữ liệu mới dựa trên các kiểu dữ liệu cơ sở của Microsoft SQL Server. Các kiểu dữ liệu mới này còn được gọi là kiểu dữ liệu do người dùng định nghĩa. Mục đích của việc này dùng để cho cấu trúc dữ liệu bên trong cơ sở dữ liệu được nhất quán và dễ sửa đổi.



**Ví dụ:**

*Trong một cơ sở dữ liệu gồm có các bảng: nhân viên, khách hàng, nhà cung cấp. Trong từng bảng này có các cột mà dữ liệu của chúng lưu trữ **hoàn toàn giống nhau** như là: họ, tên, địa chỉ, số điện thoại. Để kiểu dữ liệu và độ dài lưu trữ ở các cột trong mỗi bảng là giống nhau bắt buộc chúng ta phải ngẫm nhớ – cách làm việc này không khoa học lắm. Ngoài ra về sau khi có nhu cầu sửa đổi cấu trúc các bảng sẽ dễ dàng làm mất đi tính nhất quán về cấu trúc giữa các bảng trong cùng ứng dụng.*

Trong Microsoft SQL Server để có thể khắc phục được yếu điểm này, chúng ta có thể định nghĩa ra các kiểu dữ liệu mới và sau đó khi tạo cấu trúc bảng mới hoặc sửa đổi cấu trúc bảng thì phải chỉ rõ ra kiểu dữ liệu của các cột bên trong bảng là những kiểu mới vừa định nghĩa này.



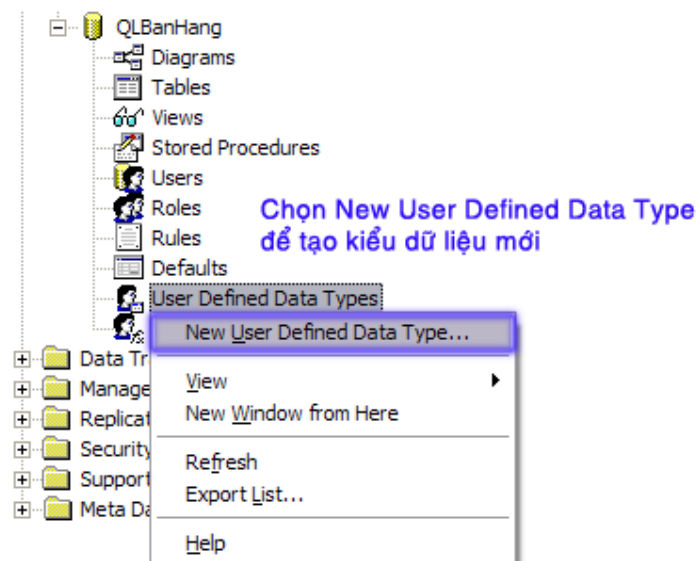
**Ví dụ:**

*Theo thí dụ trên chúng ta sẽ lần lượt tạo ra các kiểu dữ liệu mới như là: kiểu dữ liệu họ, kiểu dữ liệu tên, kiểu dữ liệu địa chỉ... để chỉ định cho các cột họ, tên, địa chỉ trong các bảng dữ liệu: nhân viên, khách hàng, nhà cung cấp.*

### III.2. Tạo mới kiểu dữ liệu do người dùng định nghĩa

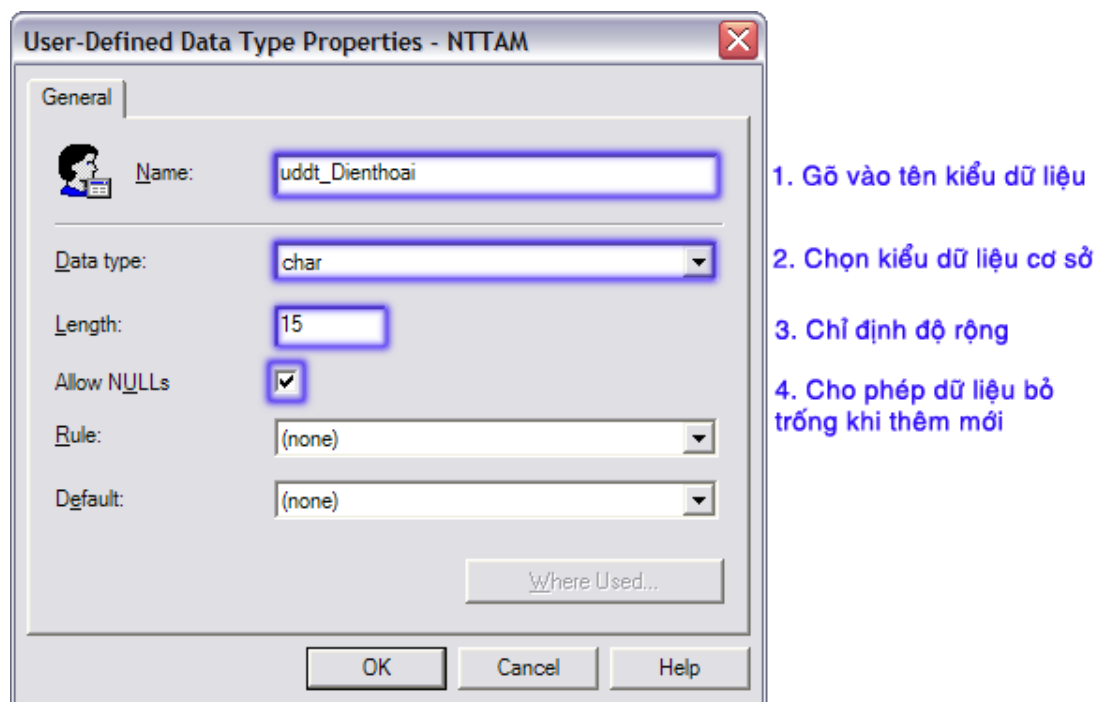
Giống như việc tạo mới các đối tượng khác mà các bạn đã làm quen trước đây trong Microsoft SQL Server, chúng ta có hai (2) cách để có thể tạo mới kiểu dữ liệu do người dùng định nghĩa. Các bước bên dưới sẽ hướng dẫn các bạn cách thức tạo ra kiểu dữ liệu mới do người dùng định nghĩa bằng tiện ích Enterprise Manager.

- ✓ Bước 1: Khởi động ứng dụng Enterprise Manager. Chọn chức năng **New User Defined Data Type...** trong thực đơn tắt sau khi nhấn chuột phải trên đối tượng **User Defined Data Types** để tạo mới kiểu dữ liệu do người dùng định nghĩa dùng chung cho các bảng trong cơ sở dữ liệu.



**Hình 2-10.** Chọn New User Defined Data Type để tạo kiểu dữ liệu mới.

- ✓ Bước 2: Trong màn hình định nghĩa kiểu dữ liệu mới lần lượt chỉ định các thuộc tính liên quan đến kiểu dữ liệu mới bao gồm: **tên** của kiểu dữ liệu mới, **kiểu dữ liệu cơ sở**, **chiều dài** dữ liệu, dữ liệu tại cột có **cho phép** bỏ trống. Sau cùng nhấn OK để lưu lại kiểu dữ liệu mới định nghĩa.



**Hình 2-11.** Các thuộc tính liên quan đến kiểu dữ liệu mới.

Ngoài ra chúng ta có cũng có thể tạo mới kiểu dữ liệu do người dùng định nghĩa bằng thủ tục nội tại hệ thống **sp\_addtype** khi gõ lệnh trong cửa sổ Query Analyzer.



Cú pháp:

```
EXEC sp_addtype Tên_kiểu_dl_mới, "Kiểu_dl_cơ_sở"
[,NULL | NOT NULL]
```

Trong đó:

- ✓ Tên kiểu dữ liệu mới: tên kiểu dữ liệu do người dùng định nghĩa, tên kiểu dữ liệu mới phải **duy nhất** trong một cơ sở dữ liệu.
- ✓ Kiểu dữ liệu cơ sở: tên của các kiểu dữ liệu hiện có trong Microsoft SQL Server. Thông thường đối với các kiểu dữ liệu có **chỉ định độ rộng** dữ liệu thì bắt buộc phải có mở và đóng nháy.
- ✓ NULL | NOT NULL: có cho phép hoặc không cho phép cột bỏ trống dữ liệu khi lưu trữ. Mặc định cho phép cột bỏ trống dữ liệu khi thêm mới dữ liệu.



**Ví dụ:**

Để định nghĩa kiểu dữ liệu có tên là **uddt\_Soluong** có kiểu dữ liệu cơ sở là **decimal** lưu trữ tối đa **20** ký số, lấy **2** số lẻ, **không** cho phép cột để trống dữ liệu khi lưu trữ. Các bạn sẽ thực hiện câu lệnh như sau:

```
EXEC sp_addtype uddt_Soluong, "Decimal(15, 2)", "NOT NULL"
```

Sau đó khi tạo lập cấu trúc bảng **TONKHO** (tồn kho) trong đó các cột số lượng đầu kỳ, tổng số lượng nhập, tổng số lượng xuất và số lượng cuối kỳ. Các bạn sẽ sử dụng kiểu dữ liệu mới vừa định nghĩa ở trên cho các cột này bởi vì chúng có cùng chung một kiểu dữ liệu là kiểu số lượng (**uddt\_Soluong**).

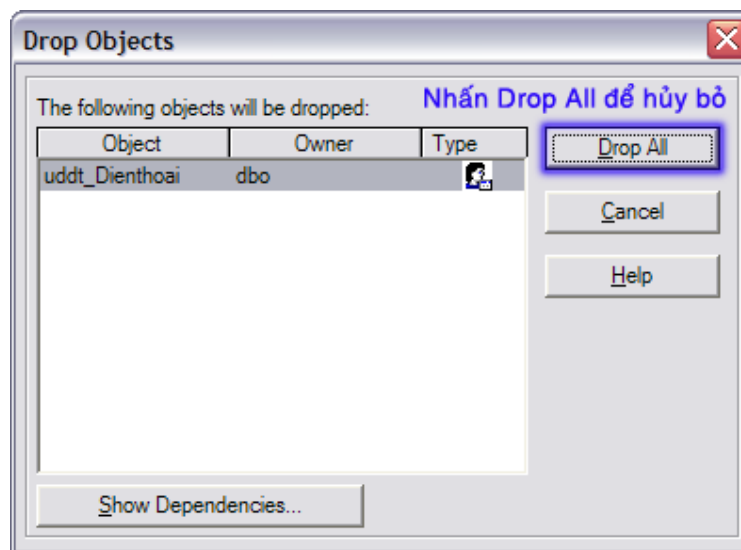
```
CREATE TABLE TONKHO
(
    Namthang CHAR(7) ,
    Mavtu     CHAR(4) ,
    Sldk      uddt_Soluong ,
    TSINhap   uddt_Soluong ,
    TSIXuat   uddt_Soluong ,
    Slick     uddt_Soluong
    CONSTRAINT PRK_TONKHO_NAMTHANG_MAVTU
        PRIMARY KEY (Namthang, Mavtu) ,
    CONSTRAINT FRK_TONKHO_MAVTU
        FOREIGN KEY (Mavtu) REFERENCES VATTU (Mavtu)
)
```

### III.3. Xóa kiểu dữ liệu do người dùng định nghĩa

Khi một kiểu dữ liệu do người dùng định nghĩa trong cơ sở dữ liệu không còn dùng đến nữa, chúng ta có thể hủy bỏ nó đi. Tuy nhiên nếu một kiểu dữ liệu do người dùng định nghĩa còn được sử dụng ít nhất cho một cột bên trong một bảng nào đó thì các bạn sẽ không thể nào hủy được nó.

Để hủy một kiểu dữ liệu do người dùng định nghĩa, chúng ta sẽ chọn chức năng **Delete** trên thực đơn tắt sau khi nhấn chuột phải vào tên kiểu dữ liệu do người dùng định nghĩa muốn hủy bỏ trong ứng dụng Enterprise Manager và xác nhận đồng ý hủy bằng cách chọn nút

**Drop All** trong màn hình hủy bỏ các đối tượng bên trong cơ sở dữ liệu của Microsoft SQL Server.



**Hình 2-12.** Màn hình xác nhận hủy bỏ kiểu dữ liệu mới.

Ngoài ra chúng ta cũng có thể sử dụng thủ tục nội tại hệ thống có tên **sp\_droptype** để hủy bỏ kiểu dữ liệu do người dùng định nghĩa trong cửa sổ Query Analyzer.

Cú pháp:

```
EXEC sp_droptype Tên_kiểu_dl
```

Trong đó:

- ✓ Tên kiểu dữ liệu: là tên kiểu dữ liệu do người dùng định nghĩa có trong cơ sở dữ liệu hiện hành muốn hủy bỏ (không còn dùng nữa trong cơ sở dữ liệu).
- ✓ Từ khóa ALL: được sử dụng khi các bạn muốn tạm ngưng việc kiểm tra tính toàn vẹn dữ liệu cho tất cả các constraint có liên quan đến bảng đã được định nghĩa trước đó.



**Ví dụ:**

Hủy kiểu dữ liệu *uddt\_Soluong* trong cơ sở dữ liệu *QLBanHang*.

```
EXEC sp_droptype uddt_Soluong
```

Tuy nhiên khi đó hệ thống sẽ xuất hiện thông báo lỗi bởi vì các cột số lượng trong bảng *TONKHO* đang sử dụng kiểu dữ liệu này nên không thể hủy được.

```
Server: Msg 15180, Level 16, State 1, Line 0
```

```
Cannot drop. The data type is being used.
```



## IV. Bảng ảo (Virtual table – View)

Nếu các bạn là người lập trình đã từng làm việc quen thuộc với cơ sở dữ liệu Microsoft Access thì chắc rằng các bạn đều biết đến đối tượng truy vấn chọn lựa (select query) trong Microsoft Access. Loại truy vấn này cho phép chúng ta chọn ra dữ liệu từ một hoặc nhiều bảng dùng để hiển thị, thống kê hoặc cho phép người sử dụng có thể cập nhật dữ liệu trực tiếp vào bên dưới các bảng mà nội dung của truy vấn có tham chiếu đến.

Gống như Microsoft Access, Microsoft SQL Server cũng có một đối tượng cho phép chúng ta có thể lựa chọn các cột, các dòng dữ liệu chính xác từ một hoặc nhiều bảng và sau đó hiển thị ra cho người sử dụng xem hoặc cập nhật trên các dữ liệu đó. Đối tượng này chính là đối tượng bảng ảo (view)

### IV.1. Khái niệm về bảng ảo

Bảng ảo thật chất là một đối tượng mà bên trong nó chỉ **lưu trữ duy nhất một câu lệnh SELECT** dùng để chỉ định các cột, các dòng dữ liệu bên dưới các bảng dữ liệu mà nó chọn lựa để hiển thị cho người sử dụng xem hoặc cập nhật. Với nguyên tắc này chúng ta có thể hiển thị ra đúng các thông tin tối thiểu mà người sử dụng cần dùng, không cần thiết phải hiển thị ra tất cả các thông tin hiện đang được lưu trữ bên trong bảng (**đáp ứng được tính bảo mật thông tin**). Ngoài ra còn giúp những người sử dụng dễ dàng truy xuất đến các thông tin mà họ đang cần, khi đó đơn giản sẽ thông qua việc thực hiện các truy vấn trực tiếp đến các bảng ảo mà không cần quan tâm các thông tin này đang được lưu trữ trong những bảng dữ liệu nào (**đáp ứng được tính dễ sử dụng**).

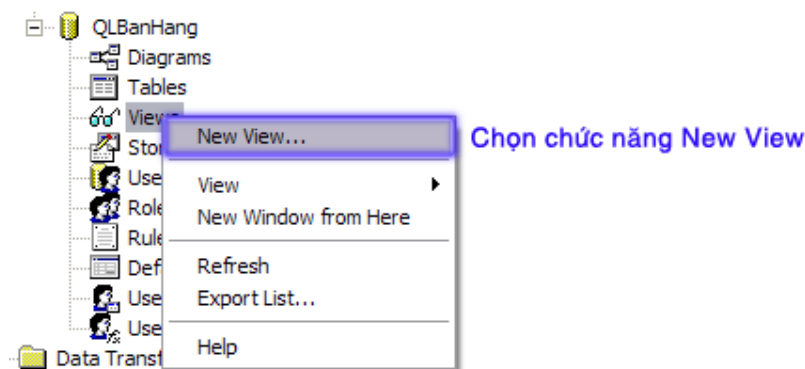
Trong thực tế chúng tôi thường tạo ra các bảng ảo để lưu trữ các **thông tin cho các loại báo cáo đơn giản** hoặc dữ liệu của các màn hình nhập liệu phức tạp có liên kết dữ liệu với nhiều bảng khác hoặc các màn hình tra cứu thông tin cho các người sử dụng.

Trước khi qua phần kế tiếp để hướng dẫn các bạn từng bước cách tạo một bảng ảo trong tiện ích Enterprise Manager, chúng tôi muốn các bạn hiểu rõ rằng: bảng ảo hoàn toàn không lưu trữ dữ liệu một cách riêng rẽ. Các dữ liệu được hiển thị trong bảng ảo sẽ được lấy từ bên dưới dữ liệu của các bảng cơ sở (underlying table) trong cơ sở dữ liệu hiện hành. Tuy nhiên chúng ta vẫn có thể cập nhật (thêm, sửa, xóa) dữ liệu trong các bảng ảo như là đang cập nhật dữ liệu trong các bảng cơ sở.

### IV.2. Tạo bảng ảo bằng tiện ích Enterprise Manager

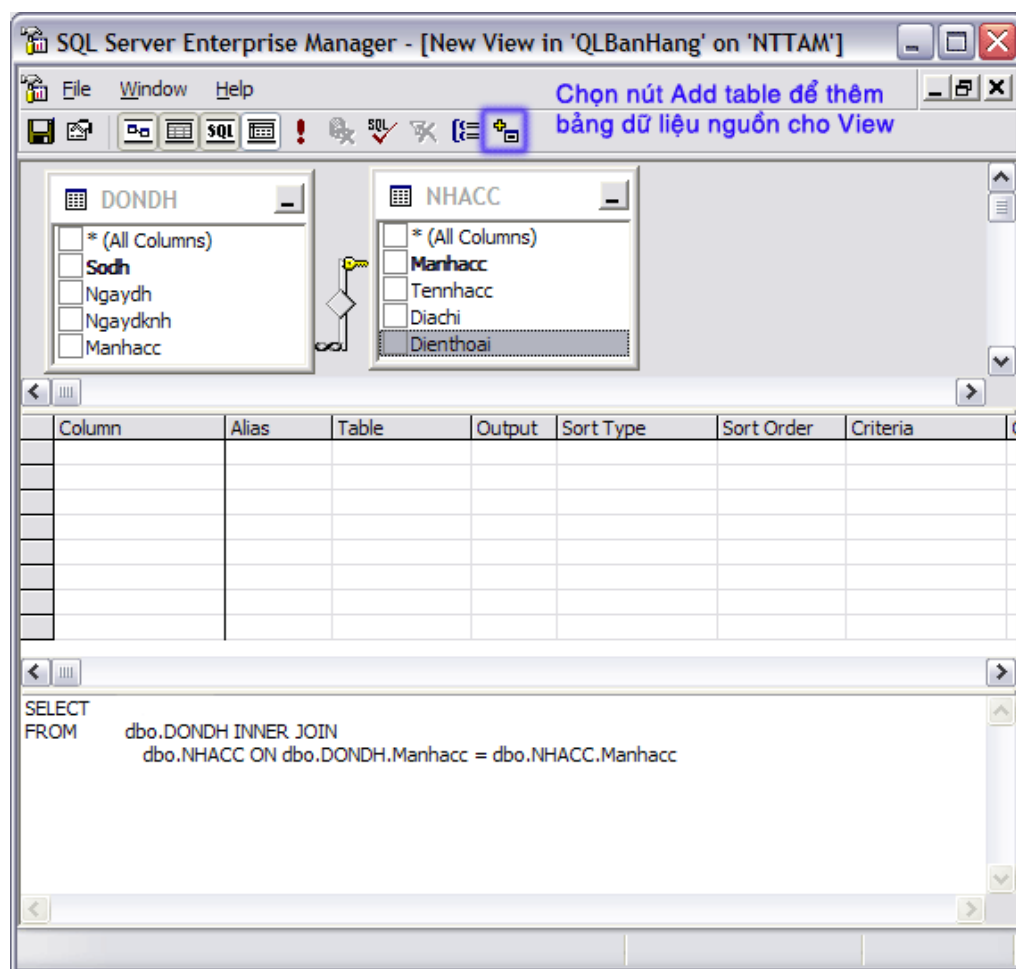
Để tạo bảng ảo, chúng ta có thể sử dụng tiện ích Enterprise Manager. Các bước tạo mới bảng ảo như sau:

- ✓ Bước 1: Khởi động ứng dụng Enterprise Manager. Chọn chức năng **New View...** trong thực đơn tắt sau khi nhấn chuột phải trên đối tượng Views.



**Hình 2-13.** Tạo mới bảng ảo trong Enterprise Manager.

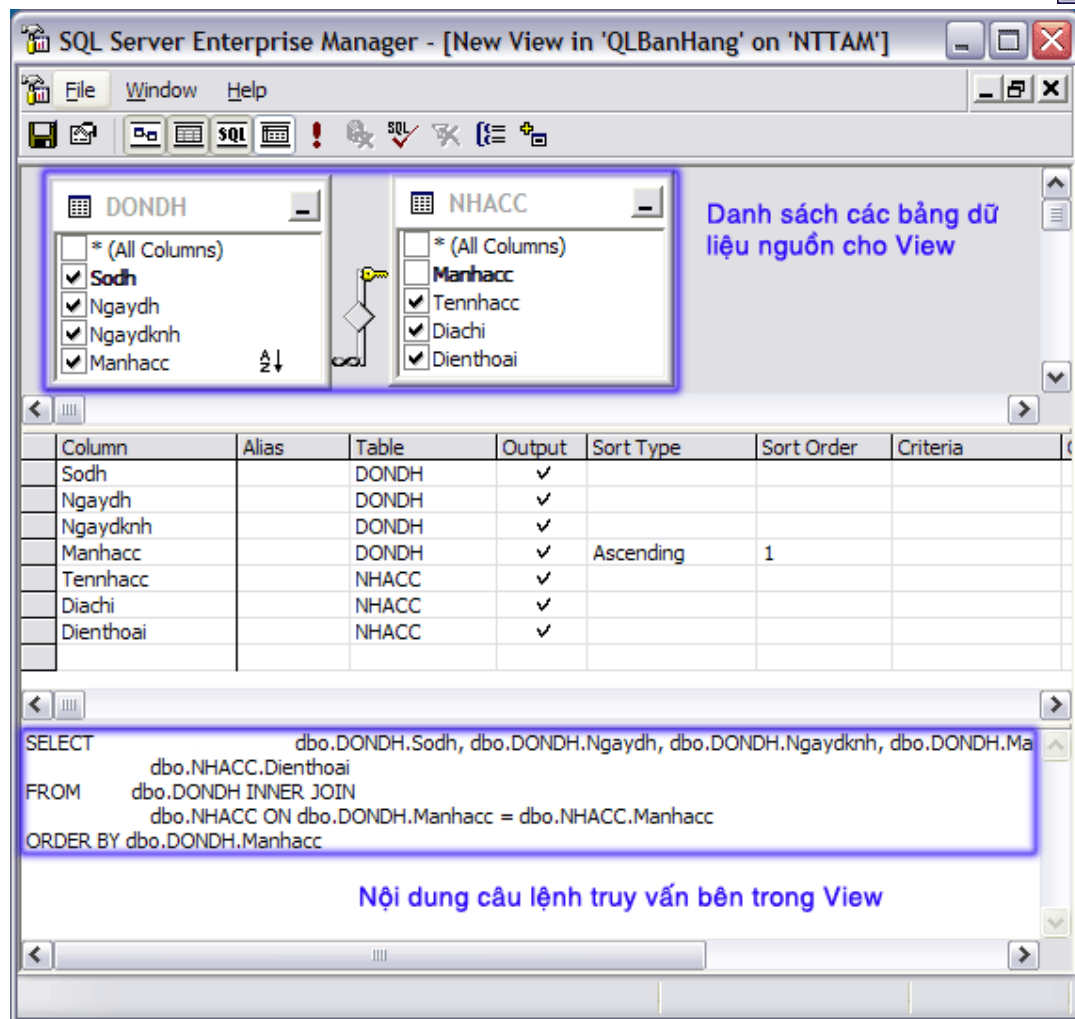
- ✓ Bước 2: Trong màn hình thiết kế dữ liệu bảng ảo, nhấn vào biểu tượng **Add Table** trên thanh công cụ để đưa các bảng dữ liệu làm dữ liệu nguồn cho bảng ảo.



**Hình 2-14.** Màn hình thiết kế dữ liệu bảng ảo.

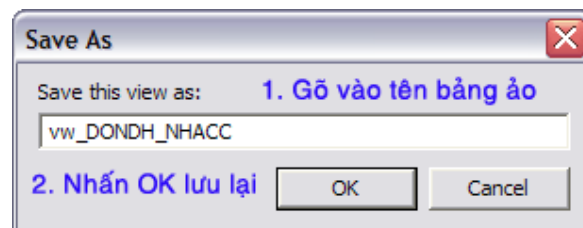
Màn hình này có cách trình bày gần giống như màn hình tạo truy vấn bằng thí dụ (QBE – Query by Example) trong Microsoft Access, nó sẽ giúp các bạn **phát sinh nội dung** của câu lệnh SELECT bằng các hành động kéo thả chuột của chính các bạn trên màn hình này.

- ✓ Bước 3: Trong màn hình chọn các dữ liệu cho bảng ảo, bằng cách chọn vào ô kiểm tra (check box) phía trước tên của các cột hoặc thao tác kéo thả (drag drop) để chọn các cột muốn hiển thị dữ liệu trong bảng ảo. Sửa lại bí danh (alias) các cột để gợi nhớ.



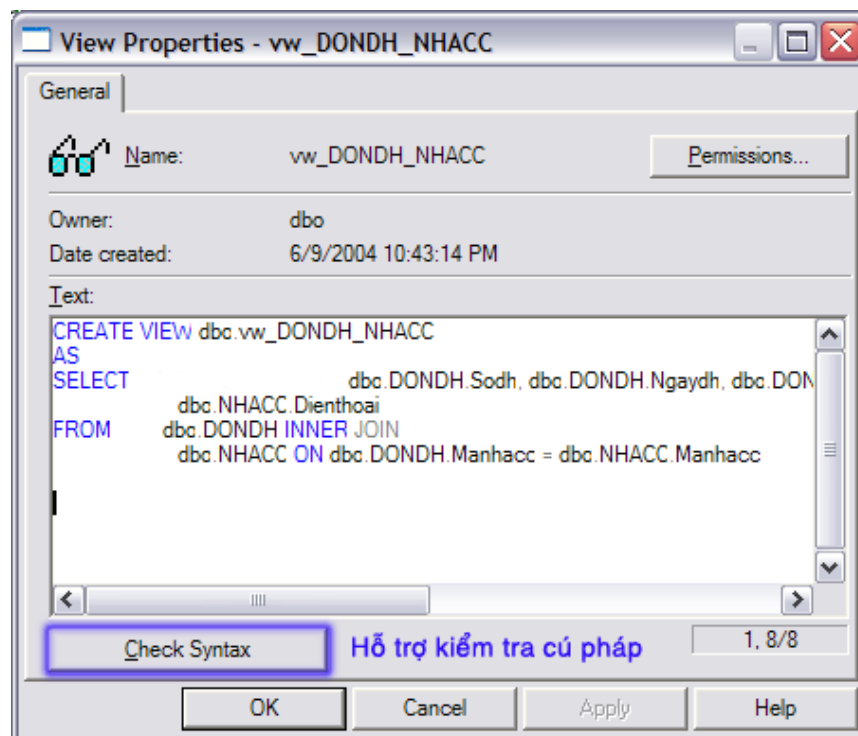
**Hình 2-15.** Chỉ định các cột hiển thị dữ liệu bên trong bảng ảo.

- ✓ Bước 4: Nhấn vào biểu tượng Save trên thanh công cụ và gõ vào tên của bảng ảo, sau đó nhấn OK để kết thúc quá trình tạo bảng ảo bằng tiện ích Enterprise Manager.



**Hình 2-16.** Gõ vào tên bảng ảo muốn lưu lại.

Sau khi tạo xong bảng ảo, các bạn cũng có thể quay lại để sửa đổi nội dung câu lệnh SELECT trong bảng ảo bằng cách chọn chức năng **Design View** để quay lại màn hình thiết kế dữ liệu bảng ảo trước đó hoặc chọn chức năng **Properties** để có thể sửa trực tiếp câu lệnh SELECT bên trong bảng ảo. Các chức năng này hiển thị trong thực đơn tắt sau khi nhấn chuột phải trên tên của bảng ảo cần sửa đổi.



**Hình 2-17.** Màn hình hiển thị câu lệnh SELECT trong bảng ảo.

Qua các bước thực hiện ở trên các bạn có thấy rằng các thao tác ở bước 2 và bước 3 gần giống như việc xây dựng các truy vấn bằng thí dụ (QBE – Query by example) trong Microsoft Access. Nội dung mà bảng ảo lưu trữ chính là nội dung câu lệnh truy vấn SELECT mà hệ thống phát sinh từ các chọn lựa trong bước 2 và bước 3. Trong các phần kế tiếp chúng tôi sẽ hướng dẫn các bạn tạo bảng ảo bằng lệnh **CREATE VIEW**.

### IV.3. Xem và cập nhật dữ liệu bảng ảo

Sau khi tạo xong bảng ảo, chúng ta có thể xem dữ liệu mà bảng ảo chứa đựng có đúng theo mong muốn hay không bằng cách thực hiện chức năng **Open View ► Return all rows** trong thực đơn tắt sau khi nhấn chuột phải trên tên của bảng ảo cần xem dữ liệu.

Hoặc thực hiện lệnh như sau:

```
SELECT *IDanh_sách_cột FROM tên_bảng_ảotên_bảng
```



**Ví dụ:**

Để xem nội dung dữ liệu của bảng ảo vw\_DONDH\_NHACC vừa tạo ở trên. Chúng ta thực hiện câu lệnh như sau:

```
SELECT * FROM vw_DONDH_NHACC
```

Việc cập nhật dữ liệu bảng ảo có thể được thực hiện bằng các lệnh **INSERT**, **UPDATE**, **DELETE** thông qua việc tham chiếu đến **tên các bảng ảo**. Mặc dù dữ liệu trong bảng ảo được lấy ra từ nhiều bảng khác nhau nhưng việc cập nhật dữ liệu trên bảng ảo chỉ được phép tác động trên **một và chỉ một** bảng mà thôi. Tuy nhiên đối với bảng ảo có tính chất thống kê tổng hợp (sử dụng các hàm tính toán: **MIN**, **MAX**, **SUM**, **COUNT**...) thì dữ liệu bên trong bảng ảo chỉ có tính chất để xem.



**Ví dụ:**

Để thêm thông tin của một số đơn đặt hàng mới trên bảng ảo vw\_DONDH\_NHACC. Chúng ta thực hiện câu lệnh như sau:

```
INSERT INTO vw_DONDH_NHACC (Sodh, Ngaydh, Ngaydknh, Manhacc)
VALUES ("D007", "2002-03-15", "2003-03-18", "C01")
```

Nhận xét thấy rằng các thông tin còn lại của bảng ảo như là: tên nhà cung cấp, điện thoại và địa chỉ của nhà cung cấp không cần thiết phải cung cấp khi nhập dữ liệu vào bởi vì dữ liệu chỉ được thêm trên bảng DONDH mà thôi.

**Ví dụ:**

Để sửa lại thông tin mã nhà cung cấp là C02 của số đơn đặt hàng D007. Chúng ta thực hiện câu lệnh như sau:

```
UPDATE vw_DONDH_NHACC
SET Manhacc = "C02"
WHERE Sodh = "D007"
```

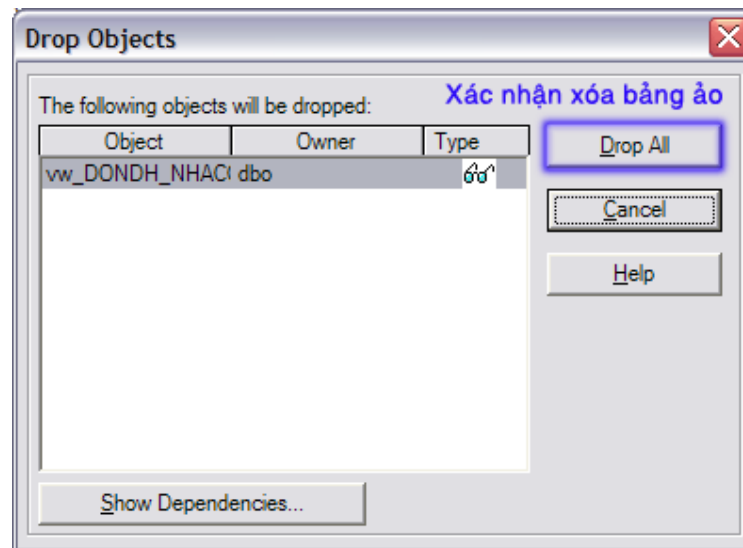
#### IV.4. Cập nhật dữ liệu qua bảng ảo sử dụng trigger INSTEAD OF

Việc cập nhật dữ liệu trên bảng ảo theo cách thông thường gặp nhiều giới hạn chủ yếu là do bảng ảo là sự kết hợp hoặc trích lược của một hay nhiều bảng trong CSDL. SQL Server 7.0 chỉ cho phép cập nhật dữ liệu qua bảng ảo một cách trực tiếp. Việc cập nhật dữ liệu qua câu lệnh INSERT, UPDATE, DELETE sẽ tác động tới các bảng liên quan. SQL Server 2000 mở rộng khả năng cập nhật dữ liệu trên bảng ảo với việc cung cấp trigger INSTEAD OF.

Trigger INSTEAD OF là một trong hai loại trigger của SQL Server 2000, bạn sẽ học chi tiết cách làm việc với trigger trong bài học về đối tượng này.

#### IV.5. Hủy bỏ bảng ảo

Giống như các đối tượng khác, chúng ta cũng được phép hủy bỏ các bảng ảo sau khi tạo ra chúng nếu không còn tiếp tục sử dụng nữa. Các bảng được tham chiếu trong câu lệnh SELECT của bảng ảo sẽ không bị hủy bỏ. Để hủy bỏ bảng ảo chúng ta sẽ chọn chức năng **Delete** trong thực đơn tắt sau khi nhấn chuột phải trên tên bảng ảo muốn hủy trong tiện ích Enterprise Manager. Sau đó chọn nút Drop All để đồng ý hủy bỏ.



**Hình 2-18.** Hộp thoại xác nhận đồng ý hủy bỏ bảng ảo.

Hoặc có thể sử dụng lệnh **DROP VIEW** với cú pháp như sau:

```
DROP VIEW Tên_bảng_ảo [, ...]
```



**Ví dụ:**

Để hủy bỏ bảng ảo vw\_DONDH\_NHACC. Chúng ta thực hiện câu lệnh như sau:

```
DROP VIEW vw_DONDH_NHACC
```

Cẩn thận sử dụng lệnh này bởi vì sau khi hủy bỏ bảng ảo, nếu các lệnh trong truy vấn nào đó vẫn còn tham chiếu tên của bảng ảo thì khi nó thực hiện, các bạn sẽ nhận được thông báo lỗi của hệ thống như bên dưới.

```
Server: Msg 208, Level 16, State 1, Line 1
```

```
Invalid object name 'vw_DONDH_NHACC'.
```

## IV.6. Tạo mới bảng ảo bằng lệnh CREATE VIEW

Trong phần này chúng tôi sẽ trình bày lệnh **tạo bảng ảo** và các hạn chế của câu lệnh SELECT bên trong lệnh **CREATE VIEW**.

Cú pháp:

```
CREATE VIEW Tên_bảng_ảo
[(Tên_các_cột)]
[WITH ENCRYPTION]
AS
    Câu_lệnh_SELECT
[WITH CHECK OPTION]
```

Trong đó:

- ✓ Tên bảng ảo: tên của bảng ảo muốn tạo mới.
- ✓ Tên các cột: danh sách tên các cột sẽ được sử dụng về sau bên trong bảng ảo khi tham chiếu đến các cột trong bảng ảo. Thông thường được sử dụng trong bảng ảo có sử dụng các hàm tính toán, các biểu thức tính toán, hoặc các cột trùng tên trong các bảng khác nhau.
- ✓ Từ khóa WITH ENCRYPTION: dùng để mã hóa nội dung câu lệnh SELECT bên trong bảng ảo. Không ai có thể biết được nội dung của câu lệnh SELECT trong bảng ảo là gì.
- ✓ Câu lệnh SELECT: câu lệnh truy vấn chọn lựa dữ liệu từ một hoặc nhiều bảng có liên kết để hiển thị dữ liệu trong bảng ảo. Một số từ khóa có trong câu lệnh SELECT chuẩn sẽ không được dùng kèm theo trong khi tạo bảng ảo, như là: **ORDER BY** (dùng sắp xếp dữ liệu), **COMPUTE** (thống kê dữ liệu cuối cùng), **COMPUTE BY** (thống kê dữ liệu theo từng nhóm), **SELECT INTO** (sao chép cấu trúc và dữ liệu sang bảng dữ liệu mới). Thông thường chúng ta nên thực hiện câu lệnh SELECT này trước để xem kết quả đúng như mong muốn hay không trước khi đưa nó lồng vào câu lệnh CREATE VIEW.
- ✓ Từ khóa WITH CHECK OPTION: dùng để ngăn cản các thao tác cập nhật dữ liệu (thêm, sửa) tác động trực tiếp vào bảng ảo có làm ảnh hưởng đến dữ liệu đối với các bảng ảo có sử dụng mệnh đề **WHERE** trong câu lệnh SELECT.



**Ví dụ:**

Để tạo một bảng ảo đơn giản có tên là vw\_DONDH\_NHACC dùng để hiển thị thông tin các tất cả các cột trong bảng DONDH (đơn đặt hàng) và hai (2) cột địa chỉ và tên nhà cung cấp trong bảng NHACC (nhà cung cấp). Chúng ta thực hiện câu lệnh CREATE VIEW như sau:

```
CREATE VIEW vw_DONDH_NHACC
AS
    SELECT DONDH.*, NHACC.Diachi AS Diachi, NHACC.Tennhacc AS Hoten
    FROM DONDH INNER JOIN NHACC ON DONDH.Manhacc = NHACC.Manhacc
GO
```

Khi muốn xem nội dung của câu lệnh SELECT bên trong bảng ảo, chúng ta có thể sử dụng thủ tục nội tại hệ thống có tên là **sp\_helptext** để xem. Cú pháp của nó khá đơn giản nên chúng tôi minh họa trực tiếp bằng thí dụ bên dưới.



**Ví dụ:**

Để xem nội dung câu lệnh SELECT của bảng ảo vừa được tạo ở thí dụ trên. Chúng ta gọi thực hiện thủ tục hệ thống **sp\_helptext** như sau:

```
EXEC sp_helptext vw_DONDH_NHACC
```

Kết quả được trả về là:

Text

```
-----
CREATE VIEW vw_DONDH_NHACC
AS
```

```
SELECT DONDH.*, NHACC.Diachi AS Diachi, NHACC.Tennhacc AS Hoten
FROM DONDH INNER JOIN NHACC ON DONDH.Manhacc = NHACC.Manhacc
```

Tuy nhiên khi chúng ta sử dụng từ khóa **WITH ENCRYPTION** bên trong câu lệnh **CREATE VIEW** lúc tạo ra bảng ảo thì khi đó nội dung của câu lệnh **SELECT** sẽ được mã hóa và các bạn không thể nào xem được nội dung của câu lệnh **SELECT** cho đến khi phải hủy bỏ và tạo lại bảng ảo bằng lệnh **CREATE VIEW** mà không sử dụng từ khóa **WITH ENCRYPTION**.



**Ví dụ:**

Để tạo lại bảng ảo có tên là **vw\_DONDH\_NHACC** giống thí dụ trên nhưng có bổ sung thêm cột điện thoại và từ khóa **WITH ENCRYPTION** để chủ động mã hóa nội dung câu lệnh **SELECT** bên trong bảng ảo. Chúng ta thực hiện câu lệnh các lệnh như sau:

```
DROP VIEW vw_DONDH_NHACC
GO

CREATE VIEW vw_DONDH_NHACC
WITH ENCRYPTION
AS
    SELECT DONDH.*, Diachi , Tennhacc, Dienthoai
    FROM DONDH INNER JOIN NHACC ON DONDH.Manhacc = NHACC.Manhacc
GO
```

Sau đó thực hiện lại câu lệnh để xem nội dung câu lệnh **SELECT** bên trong bảng ảo.

```
EXEC sp_helptext vw_DONDH_NHACC
```

Kết quả được trả về là một thông báo: Nội dung đối tượng đã được mã hóa

```
The object comments have been encrypted.
```

Khi thực hiện các bảng ảo dùng để tính toán dữ liệu cho các thống kê, chúng ta thường sử dụng các hàm tính toán **SUM**, **MIN**, **MAX**, **COUNT**, **AVG** đi kèm theo câu lệnh **SELECT**. Tuy nhiên dữ liệu của các bảng ảo dạng này chỉ có tính chất thống kê hoặc hiển thị dữ liệu cho người sử dụng xem mà **không cho phép** các hành động cập nhật dữ liệu trực tiếp ngay trên bảng ảo. Để chỉ định đến các cột dữ liệu tổng hợp này chúng ta có thể sử dụng hai (2) cách: hoặc đặt tên các cột ngay sau lệnh **CREATE VIEW** hoặc sử dụng bí danh để chỉ tên cột trong câu lệnh **SELECT**.



**Ví dụ:**

Để tạo một bảng ảo có tên là **vw\_TINH\_TONGSLDAT** dùng để tính tổng số lượng đặt hàng của các vật tư. Dữ liệu hiển thị gồm các cột: mã vật tư, tên vật tư và tổng số lượng đặt. Chúng ta thực hiện câu lệnh **CREATE VIEW** để tạo bảng ảo như sau:

```
CREATE VIEW vw_TINH_TONGSLDAT
(Mavtu, Tenvtu, Tong_sl-dat)
AS
    SELECT CTDH.Mavtu, Tenvtu, SUM(SLDAT)
    FROM CTDONDH CTDH INNER JOIN VATTU VT ON CTDH.MAVTU=VT.MAVTU
    GROUP BY CTDH.Mavtu, Tenvtu
GO
```



Hoặc:

```
CREATE VIEW vw_TINH_TONGSLDAT
AS
    SELECT CTDH.Mavtu, Tenvtu , SUM(SLDAT) AS Tong_slдат
    FROM CTDONDH CTDH INNER JOIN VATTU VT ON CTDH.MAVTU=VT.MAVTU
    GROUP BY CTDH.Mavtu, Tenvtu
GO
```

Để hiểu rõ về từ khóa **WITH CHECK OPTION** bên trong câu lệnh **CREATE VIEW**, chúng ta sẽ xem xét các thí dụ bên dưới.



**Ví dụ:**

Để tạo một bảng ảo có tên là vw\_VATTU\_TIVI gồm các cột dữ liệu của bảng VATTU: mã vật tư, tên vật tư, đơn vị tính nhưng các dòng dữ liệu chỉ lọc ra các loại hàng tivi. Chúng ta thực hiện câu lệnh CREATE VIEW như sau:

```
CREATE VIEW vw_VATTU_TIVI
AS
    SELECT Mavtu, Tenvtu, DvTinh
    FROM VATTU
    WHERE Mavtu LIKE "TV%"
GO
```

Kế tiếp chúng ta sử dụng lệnh **INSERT** để thêm một hàng hóa mới thuộc loại **máy hát nhac** vào bảng ảo. Câu lệnh này được thực hiện thành công.

```
INSERT INTO vw_VATTU_TIVI
VALUES ("MH01", "Máy hát Sony đời IK-2002", "Bộ")
```

Nhưng khi xem lại dữ liệu của bảng ảo bằng câu lệnh **SELECT** bên dưới các bạn sẽ ngạc nhiên vì khi đó sẽ **không thấy** thể hiện của mẫu tin vừa thêm mới ở trên.

```
SELECT * FROM vw_VATTU_TIVI
```

Tuy nhiên khi xem dữ liệu của bảng VATTU chúng ta vẫn thấy rằng dữ liệu đã **được lưu trữ** bên dưới bảng VATTU.

```
SELECT * FROM VATTU
```

Các bạn có thể lý giải được điều này không? Điều này sẽ làm cho một số người sử dụng rất khó hiểu. Lý do mà dữ liệu không được hiển thị bên trong bảng ảo bởi vì mẫu tin mà chúng ta thêm vào đã không thỏa điều kiện lọc bên trong mệnh đề **WHERE** của câu lệnh SELECT trong bảng ảo. (mã của hàng hóa này không bắt đầu bằng chữ TV)

Do đó để tránh những hành động có thể làm cho người sử dụng khó hiểu, Microsoft SQL Server cho phép chúng ta ngăn cản các mẫu tin **không thỏa điều kiện WHERE** được cập nhật vào bảng ảo bằng mệnh đề **WITH CHECK OPTION**.



**Ví dụ:**

Trở lại thí dụ trên nhưng chúng ta có thêm từ khóa **WITH CHECK OPTION** khi tạo bảng ảo.

```
DROP VIEW vw_VATTU_TIVI
GO
```



```
CREATE VIEW vw_VATTU_TIVI
AS
    SELECT Mavtu, Tenvtu, DvTinh
    FROM VATTU
    WHERE Mavtu LIKE "TV%"
WITH CHECK OPTION
GO
```

Bây giờ khi chúng ta sử dụng lệnh **INSERT** để thêm một hàng hóa mới thuộc loại **máy hát nhạc** vào bảng ảo.

```
INSERT INTO vw_VATTU_TIVI
VALUES ("MH02", "Máy hát Sony đời AK-2002", "Bộ")
```

Hoặc sửa đổi mã của hàng hóa ti vi Sony 14 inches thành "TI14" bằng câu lệnh **UPDATE** như bên dưới.

```
UPDATE vw_VATTU_TIVI
SET Mavtu="TI14"
WHERE Mavtu="TV14"
```

Mặc dù dữ liệu và cú pháp các câu lệnh này hoàn toàn đúng nhưng nó sẽ không được thực hiện thành công bởi vì mã vật tư của nó đã vi phạm mệnh đề **WHERE** (không bắt đầu bằng chữ TV).

Tóm lại khi sử dụng mệnh đề **WITH CHECK OPTION** trong lệnh **CREATE VIEW** dùng để ngăn cản các hành động cập nhật dữ liệu trong bảng ảo không thỏa điều kiện trong mệnh đề **WHERE** của câu lệnh **SELECT**.

## IV.7. Sửa đổi nội dung bảng ảo

Màn hình thuộc tính của bảng ảo ở hình 2-24 cho phép chúng ta có thể sửa đổi nội dung câu lệnh **SELECT** trong bảng ảo. Bên cạnh đó chúng ta cũng có thể sửa đổi nội dung câu lệnh **SELECT** trong bảng ảo bằng lệnh **ALTER VIEW**. Cú pháp của câu lệnh này có phần hoàn toàn giống như câu lệnh **CREATE VIEW** mà chúng tôi đã trình bày trong phần trước.

Cú pháp:

```
ALTER VIEW Tên_bảng_ảo
[(Tên_các_cột)]
[WITH ENCRYPTION]
AS
    Câu_lệnh_SELECT_mới
[WITH CHECK OPTION]
```

Trong đó:

- ✓ Tên bảng ảo, tên các cột: giống như câu lệnh **CREATE VIEW**.
- ✓ Câu lệnh **SELECT** mới: nội dung câu lệnh truy vấn chọn lựa mới.

Các bạn có thể sử dụng lệnh **DROP VIEW** để hủy bỏ bảng ảo rồi sau đó tạo lại mới bằng lệnh **CREATE VIEW** thay vì phải sử dụng lệnh **ALTER VIEW**.



## Bài 3

# CÁC RÀNG BUỘC TOÀN VỆN DỮ LIỆU

### Tóm tắt

Lý thuyết 6 tiết - Thực hành 6 tiết

Mục tiêu	Các mục chính	Bài tập bắt buộc	Bài tập làm thêm
Làm việc với các đối tượng có trong Database: Ràng buộc toàn vẹn, Mô hình quan hệ dữ liệu, Quy tắc kiểm tra dữ liệu, Giá trị mặc định	I. Các ràng buộc toàn vẹn dữ liệu (Constraint)  VII. Mô hình quan hệ dữ liệu (Diagram)  VIII. Quy tắc kiểm tra miền giá trị dữ liệu (Rule)  IX. Giá trị mặc định (Default)	3.1  3.2  3.6	3.3  3.4  3.5



# I. Các ràng buộc toàn vẹn dữ liệu (Constraint)

## I.1. Các quy định của công việc trong thực tế

Mục đích đầu tiên của một cơ sở dữ liệu được tạo ra là để lưu trữ những thông tin phục vụ cho một công việc cụ thể nào đó trong thực tế. Thông tin được lưu trữ sau đó có thể được truy cập và xử lý theo nhiều cách khác nhau dựa vào các chức năng mà một hệ quản trị CSDL cung cấp. Tuy nhiên, tất cả các chức năng truy xuất và xử lý dữ liệu đều phải dựa trên yếu tố cơ bản là dữ liệu sẵn có trong CSDL. Nếu các dữ liệu lưu trữ thông tin trong thực tế bị sai thì việc truy cập tới chúng chẳng có ý nghĩa gì hay thậm chí có thể gây ra nhiều thiệt hại.

Mỗi một công việc trong thực tế đều có những quy định nào đó chi phối. Ví dụ như việc quản lý quá trình đặt/giao hàng, một số quy định có thể nhận thấy rất rõ như:

- ✓ Số lượng đặt hàng phải lớn hơn 0
- ✓ Các số hoá đơn giao hàng không được trùng nhau
- ✓ Ngày dự kiến nhận hàng phải sau ngày đặt hàng
- ✓ ...

Một số quy định khác có thể tương đối khó nhận ra bởi tính chất logic của nó gần như là hiển nhiên đối với người thực hiện công việc như:

- ✓ Một đơn đặt hàng phải do một khách hàng lập ra
- ✓ Mỗi một mặt hàng phải có nhà cung cấp (mỗi mặt hàng phải có xuất xứ)
- ✓ Số lượng mặt hàng giao cho khách phải nhỏ hơn hay tối đa bằng với số lượng đặt
- ✓ Hai nhà cung cấp có thể trùng tên nhưng là hai nhà cung cấp khác nhau

Trong thực tế, những quy định là cần thiết để công việc được thực hiện chính xác và suôn sẻ do đó, khi mô hình hoá những thông tin này thành một hệ thống CSDL, những quy định này vẫn cần được đảm bảo nhằm tránh việc nhập liệu sai hay thực hiện các tính toán sai làm cho thông tin không còn chính xác và không sử dụng được nữa.

Khi một CSDL ở trạng thái mà tất cả các dữ liệu lưu trữ đều chính xác so với thực tế và thỏa mãn đầy đủ tất cả các quy định thì ta nói CSDL đó có tính toàn vẹn dữ liệu. Tính toàn vẹn dữ liệu có thể mất đi do các yếu tố khách quan như mất điện đột xuất lúc đang ghi dữ liệu, ổ cứng bị hỏng,... nhưng phần nhiều là do những nguyên nhân chủ quan như nhập liệu sai, thực hiện các hành động cập nhật dữ liệu sai do tính toán phức tạp,...

SQL Server cung cấp cơ chế đảm bảo tính toàn vẹn dữ liệu trong các trường hợp khách quan nói trên nhờ vào khả năng sử dụng các log file và các cơ chế lưu trữ/phục hồi dữ liệu, cơ chế transaction,... Tuy nhiên, những quy định cụ thể như ví dụ ở trên thì SQL Server hoàn toàn không thể tự động thực hiện được. Thay vào đó, SQL Server cung cấp các công cụ giúp người dùng mô tả các quy định này trong hệ thống CSDL. Khi đã được mô tả, SQL Server sẽ đảm bảo thực hiện đúng các quy định trên để bảo vệ dữ liệu hiện có cũng như những dữ liệu sắp được thêm vào.



## 1.2. Các ràng buộc toàn vẹn dữ liệu

Trong CSDL, các quy định về dữ liệu được gọi chung là các **ràng buộc toàn vẹn dữ liệu** hay vắn tắt là các **ràng buộc toàn vẹn** hoặc các **ràng buộc**. Để đảm bảo tính toàn vẹn của dữ liệu trong CSDL SQL Server cung cấp hai loại đối tượng là **constraint** và **trigger**.

Về nguyên tắc, các đối tượng constraint và trigger khi được tạo ra sẽ được xử lý **trước khi** dữ liệu được cập nhật (thêm, sửa hay xóa). Một constraint hay trigger cần được liên kết với một bảng cụ thể trong CSDL. Tuy nhiên, trigger có thể tham chiếu đến các bảng và nhiều đối tượng khác trong CSDL trong khi constraint chỉ có thể tham chiếu các cột trong bảng mà nó liên kết tới mà thôi. Chúng ta sử dụng constraint để mô tả các quy định và SQL Server sẽ tự động thực hiện các kiểm tra dữ liệu cần thiết. Đối với trigger, chúng ta phải lập trình để tự kiểm soát dữ liệu thông qua việc lập trình. Trong bài học này các bạn sẽ làm việc với các loại constraint khác nhau, phần trigger sẽ được trình bày ở bài học sau.

Chúng ta chia các quy tắc ra thành từng loại khác nhau tùy theo tính năng kiểm tra dữ liệu của nó. Các loại quy tắc kiểm tra tính toàn vẹn dữ liệu bao gồm:

### 1.2.1. Kiểm tra duy nhất

Cho phép chúng ta có thể kiểm tra tính duy nhất của dữ liệu bên trong bảng. Điều này ngăn cản việc người sử dụng nhập trùng giá trị dữ liệu, ví dụ mỗi nhân viên có một mã số duy nhất. Các bạn có thể sử dụng **PRIMARY KEY** hoặc **UNIQUE** trong câu lệnh **CREATE TABLE** để thực hiện việc kiểm tra tính duy nhất của dữ liệu.



**Ví dụ:**

Trong bảng **VATTU** (vật tư) kiểm tra quy tắc mã vật tư phải là duy nhất. Có nghĩa là ứng với mỗi vật tư mới chúng ta phải tạo ra một mã vật tư hoàn toàn khác biệt với các vật tư đã được cấp trước đó.

### 1.2.2. Kiểm tra tồn tại

Cho phép chúng ta có thể kiểm tra tính tồn tại của dữ liệu, bắt buộc phải có bên một bảng khác còn gọi là bảng tham chiếu. Điều này ngăn cản việc người sử dụng nhập một giá trị dữ liệu không có trong một bảng khác, ví dụ với mỗi vật tư phải xác định được nhà cung cấp vật tư đó. Các bạn có thể sử dụng thành phần **FOREIGN KEY ... REFERENCES** trong câu lệnh **CREATE TABLE** để thực hiện việc kiểm tra tính tồn tại của dữ liệu.



**Ví dụ:**

Trong bảng **PNHAP** (phiếu nhập hàng) phải kiểm tra tính tồn tại của cột số đơn đặt hàng bên bảng tham chiếu **DONDH** (đơn đặt hàng). Có nghĩa là những phiếu nhập hàng sẽ phản ánh việc nhập hàng theo đúng các số đơn đặt hàng mà công ty đã đặt hàng cho các nhà cung cấp trước đó.

### 1.2.3. Kiểm tra miền giá trị

Cho phép chúng ta có thể kiểm tra miền giá trị của dữ liệu bên trong bảng. Điều này ngăn cản



việc người sử dụng nhập một giá trị dữ liệu vượt quá phạm vi qui định trước đó, ví dụ số lượng đặt vật tư chỉ giới hạn từ 0 đến 1000. Các bạn có thể sử dụng thành phần **CHECK** hoặc **DEFAULT** trong câu lệnh **CREATE TABLE** để thực hiện việc kiểm tra tính tồn tại của dữ liệu.

**Ví dụ:**

Trong bảng CTDONDH (chi tiết đơn đặt hàng) kiểm tra quy tắc giá trị dữ liệu của cột số lượng đặt tối thiểu là 10 và không lớn hơn 50. Có nghĩa là cột số lượng đặt  $\geq 10$  và số lượng đặt  $\leq 50$ , giá trị mặc định sẽ là 10 khi người sử dụng không cung cấp giá trị cột số lượng đặt.

### 1.3. Sử dụng constraint để kiểm tra toàn vẹn dữ liệu

Các loại constraint PRIMARY KEY, UNIQUE, FOREIGN KEY, CHECK và DEFAULT cần phải được gắn với một bảng trong CSDL. Do đó, ta có thể tạo ra constraint bằng hai cách:

- ✓ Sử dụng câu lệnh **CREATE TABLE** để tạo ra các constraint ngay khi tạo mới một bảng
- ✓ Sử dụng câu lệnh **ALTER TABLE** để thêm các constraint vào một bảng đã có

**Chú ý:**

Không có câu lệnh **CREATE CONSTRAINT**, **DROP CONSTRAINT** hay **ALTER CONSTRAINT**.

#### 1.3.1. Tạo cấu trúc bảng có khóa chính

Với cú pháp **CREATE TABLE** bên dưới cho phép các bạn tạo ra cấu trúc bảng có chứa **khóa chính** của bảng. Khóa chính của bảng dùng để kiểm tra tính duy nhất của dữ liệu khi lưu trữ. Trong một bảng chỉ có **một khóa chính**, tuy nhiên được phép có **nhiều cột** tham gia vào làm khóa chính của bảng. Dữ liệu tại các cột tham gia làm khóa chính **không** được phép bỏ trống.

Cú pháp:

```
CREATE TABLE Tên_bảng
(
    Tên_cột1    Kiểu_dữ_liệu [NOT NULL] ,
    Tên_cột2    Kiểu_dữ_liệu [NOT NULL] [, ...]
    PRIMARY KEY (Danh_sách_cột_khóa_chính)
)
```

Trong đó:

- ✓ Danh sách cột khóa chính: là danh sách tên của các cột tham gia làm khóa chính, tên các cột được ngăn cách nhau bởi dấu phẩy (,).

**Ví dụ:**

Để tạo lại cấu trúc mới của bảng có tên là DONDH (đơn đặt hàng) gồm có những cột như: số đặt hàng có kiểu dữ liệu là chuỗi và chiều dài 4 ký tự, ngày đặt hàng có kiểu dữ liệu là ngày, mã nhà cung cấp có kiểu dữ liệu là chuỗi và chiều dài 3 ký tự. Dữ liệu tại các cột không được phép trống và bảng này có khóa chính là cột số đặt hàng, nghĩa là giá trị của các số đặt hàng là duy nhất trong bảng DONDH. Chúng ta thực hiện câu lệnh **CREATE TABLE** như sau:

```
CREATE TABLE DONDH
```



```
(
    Sodh      CHAR(4) ,
    Ngaydh    DATETIME NOT NULL ,
    Manhacc   CHAR(3) NOT NULL
    PRIMARY KEY (Sodh)
)
```

### 1.3.2. Tạo cấu trúc bảng có khóa ngoại

Với cú pháp CREATE TABLE bên dưới cho phép các bạn tạo ra cấu trúc bảng có chứa khóa ngoại của bảng. Khóa ngoại của bảng dùng để kiểm tra tính tồn tại của dữ liệu dựa trên dữ liệu của một bảng tham chiếu. Trong một bảng được phép có nhiều khóa ngoại, tuy nhiên các cột tham chiếu bên bảng tham chiếu phải là khóa chính của bảng tham chiếu.

Cú pháp:

```
CREATE TABLE Tên_bảng
(
    Tên_cột1    Kiểu_dữ_liệu [NOT NULL] ,
    Tên_cột2    Kiểu_dữ_liệu [NOT NULL] [, ...]
    FOREIGN KEY (Danh_sách_cột_khoá_ngoại)
    REFERENCES Tên_bảng_tham_chiếu (Danh_sách_cột_tham_chiếu)
    [ON DELETE { CASCADE | NO ACTION }] [ ON UPDATE { CASCADE | NO ACTION } ]
)
```

Trong đó:

- ✓ Danh sách cột khóa ngoại: là danh sách tên các cột tham gia làm khóa ngoại, tên các cột được ngăn cách nhau bởi dấu phẩy(,).
- ✓ Tên bảng tham chiếu: là tên của bảng tham chiếu để kiểm tra tính tồn tại dữ liệu bên bảng tham chiếu.
- ✓ Danh sách cột tham chiếu: là danh sách tên các cột tham chiếu trong bảng tham chiếu, tên các cột ngăn cách nhau bởi dấu phẩy (,). Danh sách các cột khóa ngoại và danh sách các cột tham chiếu phải tương xứng nhau.
- ✓ ON DELETE: chỉ định cách SQL Server 2000 thi hành khi hành động xoá xảy ra trên bảng được tham chiếu (bảng cha).
  - Nếu chỉ định NO ACTION thì lỗi sẽ xảy ra trong trường hợp mối quan hệ giữa hai bảng cha – con có dữ liệu
  - Nếu chỉ định CASCADE thì khi xoá một dòng dữ liệu trên bảng cha thì các dòng dữ liệu liên quan trên bảng con cũng sẽ bị xoá.
- ✓ ON UPDATE: chỉ định cách SQL Server 2000 thi hành khi hành động cập nhật khoá chính xảy ra trên bảng được tham chiếu (bảng cha).
  - Nếu chỉ định NO ACTION thì lỗi sẽ xảy ra trong trường hợp mối quan hệ giữa hai bảng cha – con có dữ liệu
  - Nếu chỉ định CASCADE thì khi cập nhật giá trị khoá chính của một dòng dữ liệu trên bảng cha thì giá trị các cột khoá ngoại của các dòng dữ liệu liên quan trên bảng con

cũng sẽ được cập nhật.



**Ví dụ:**

Để tạo lại bảng có tên là CTDONDH (chi tiết đơn đặt hàng) gồm có những cột như: số đặt hàng có kiểu dữ liệu là chuỗi và chiều dài 4 ký tự, mã vật tư có kiểu dữ liệu là chuỗi và chiều dài 4 ký tự, số lượng đặt có kiểu số nguyên. Dữ liệu tại các cột không được phép trống. Khóa chính gồm có hai (2) cột là số đặt hàng và mã vật tư. Khóa ngoại gồm số đặt hàng tham chiếu qua bảng DONDH, nghĩa là giá trị dữ liệu tại cột số đặt hàng phải tồn tại trong bảng DONDH và mã vật tư tham chiếu qua bảng VATTU, nghĩa là giá trị dữ liệu tại cột mã vật tư phải tồn tại trong bảng VATTU. Chúng ta thực hiện câu lệnh CREATE TABLE như sau:

```
CREATE TABLE CTDONDH
(
    Sodh    CHAR(4) ,
    Mavtu   CHAR(4) ,
    SIDat   SMALLINT
    PRIMARY KEY (Sodh, Mavtu) ,
    FOREIGN KEY (Sodh) REFERENCES DONDH (Sodh) ,
    FOREIGN KEY (Mavtu) REFERENCES VATTU (Mavtu)
)
```

### 1.3.3. Tạo cấu trúc bảng kiểm tra miền giá trị

Với cú pháp CREATE TABLE bên dưới cho phép các bạn tạo ra cấu trúc bảng có chứa các quy tắc dùng để kiểm tra dữ liệu tại các cột bắt buộc nằm trong một miền giá trị chỉ định. Trong trường hợp nếu dữ liệu vi phạm quy tắc chỉ định thì dữ liệu sẽ không được lưu trữ vào trong bảng.

Cú pháp:

```
CREATE TABLE Tên_bảng
(
    Tên_cột1    Kiểu_dữ_liệu [NOT NULL] ,
    Tên_cột2    Kiểu_dữ_liệu [NOT NULL] [, ...]
    CHECK (Biểu_thức_luận_lý)
)
```

Trong đó:

- ✓ Biểu thức luận lý: là một biểu thức chỉ định quy tắc kiểm tra dữ liệu trong bảng. Thông thường biểu thức luận lý sẽ là biểu thức so sánh.



**Ví dụ:**

Trong bảng CTDONDH (chi tiết đơn đặt hàng) ngoài các yêu cầu như thí dụ trên, có thêm một quy tắc kiểm tra dữ liệu là số lượng đặt hàng phải thuộc trong phạm vi 10 đến 50. Chúng ta thực hiện câu lệnh CREATE TABLE như sau:

```
CREATE TABLE CTDONDH
(
    Sodh    CHAR(4) ,
```



```
Mavtu CHAR(4) ,
SIDat SMALLINT
PRIMARY KEY (Sodh, Mavtu) ,
FOREIGN KEY (Sodh) REFERENCES DONDH (Sodh) ,
FOREIGN KEY (Mavtu) REFERENCES VATTU (Mavtu) ,
CHECK (SIDat BETWEEN 10 AND 50)
)
```

### I.3.4. Tạo cấu trúc bảng kiểm tra dữ liệu duy nhất

Với cú pháp CREATE TABLE bên dưới cho phép các bạn tạo ra cấu trúc bảng có kiểm tra dữ liệu duy nhất tại các cột không tham gia làm khóa chính. Trong trường hợp nếu có các cột không là khóa chính của bảng nhưng chúng ta muốn kiểm tra tính duy nhất của dữ liệu tại các cột đó thì phải sử dụng câu lệnh bên dưới.

Cú pháp:

```
CREATE TABLE Tên_bảng
(
    Tên_cột1 Kiểu_dữ_liệu [NOT NULL] ,
    Tên_cột2 Kiểu_dữ_liệu [NOT NULL] [, ...]
    UNIQUE (Danh_sách_các_cột)
)
```



#### Ví dụ:

Trong bảng VATTU (vật tư) ngoài các yêu cầu như các thí dụ trên, cần kiểm tra tên vật tư là duy nhất mặc dù khóa chính đã là cột mã vật tư. Chúng ta thực hiện câu lệnh CREATE TABLE như sau:

```
CREATE TABLE VATTU
(
    Mavtu CHAR(4) NOT NULL ,
    Tenvtu CHAR(30) NOT NULL ,
    DvTinh CHAR(10) NOT NULL ,
    Phantram TinyInt DEFAULT 20
    PRIMARY KEY (Mavtu) ,
    UNIQUE (Tenvtu)
)
```

### I.3.5. Tạo cấu trúc bảng có giá trị mặc định

Với cú pháp **CREATE TABLE** bên dưới cho phép các bạn tạo ra cấu trúc bảng với định nghĩa giá trị mặc định cho cột. Giá trị mặc định là những giá trị sẽ được gán vào bên trong cột dữ liệu của bảng trong trường hợp dữ liệu tại cột đó bỏ trống khi thêm mới dòng dữ liệu vào bảng.

Cú pháp:

```
CREATE TABLE Tên_bảng
(
```



```
Tên_cột1  Kiểu_dữ_liệu  DEFAULT  Giá_trị|Hàm ,
Tên_cột2  Kiểu_dữ_liệu [NOT NULL] [, ...]
)
```

Trong đó:

- ✓ Giá trị: là giá trị cụ thể sẽ được gán vào cột.
- ✓ Hàm: là tên hàm cùng với các tham số (nếu có), tuy nhiên kiểu dữ liệu mà hàm trả về phải là cùng với kiểu dữ liệu mà cột sẽ lưu trữ.



**Ví dụ:**

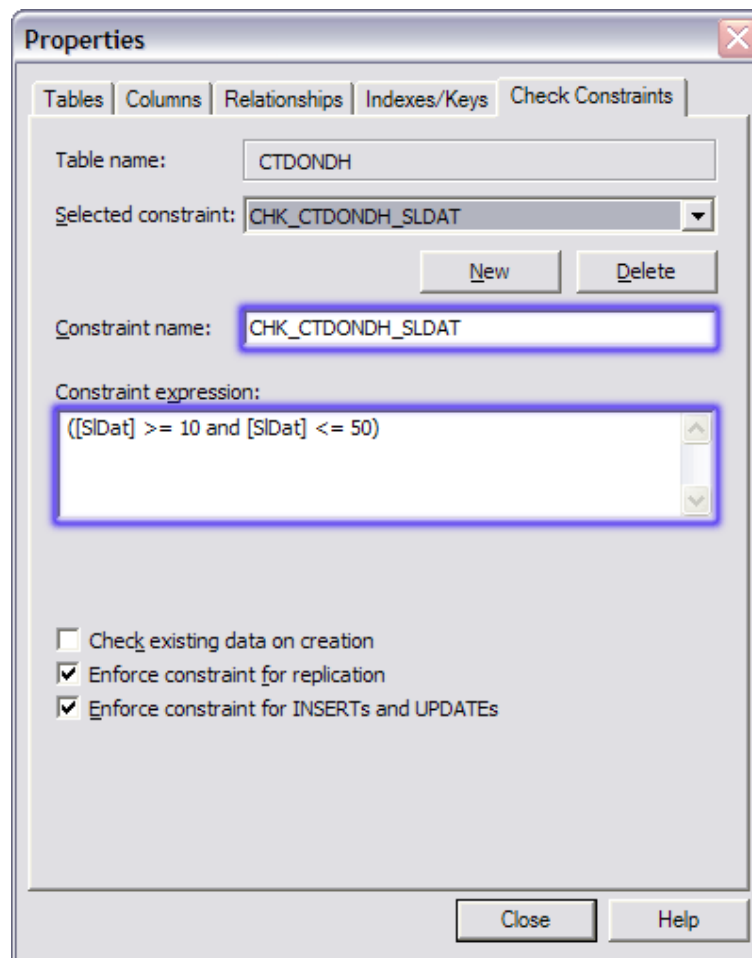
Để tạo bảng có tên là VATTU (vật tư) gồm có những cột và kiểu dữ liệu mô tả như hình 2-9, có thêm yêu cầu là tạo giá trị mặc định tại cột phần trăm (Phantram) là 20. Chúng ta thực hiện câu lệnh **CREATE TABLE** như sau:

```
CREATE TABLE VATTU
(
    Mavtu      CHAR(4)  NOT NULL ,
    Tenvtu     CHAR(30) NOT NULL ,
    DvTinh     CHAR(10) NOT NULL ,
    Phantram    TinyInt  DEFAULT 20
)
```

Tóm lại chúng tôi đã trình bày một cách riêng rẽ về các thành phần có trong câu lệnh **CREATE TABLE** để giúp các bạn dễ hiểu, dễ nhớ, dễ thực hiện. Thực tế các bạn có thể kết hợp tất cả các thành phần riêng rẽ ở trên để hình thành một câu lệnh **CREATE TABLE** với đầy đủ các tính năng trong việc tạo lập cấu trúc bảng.

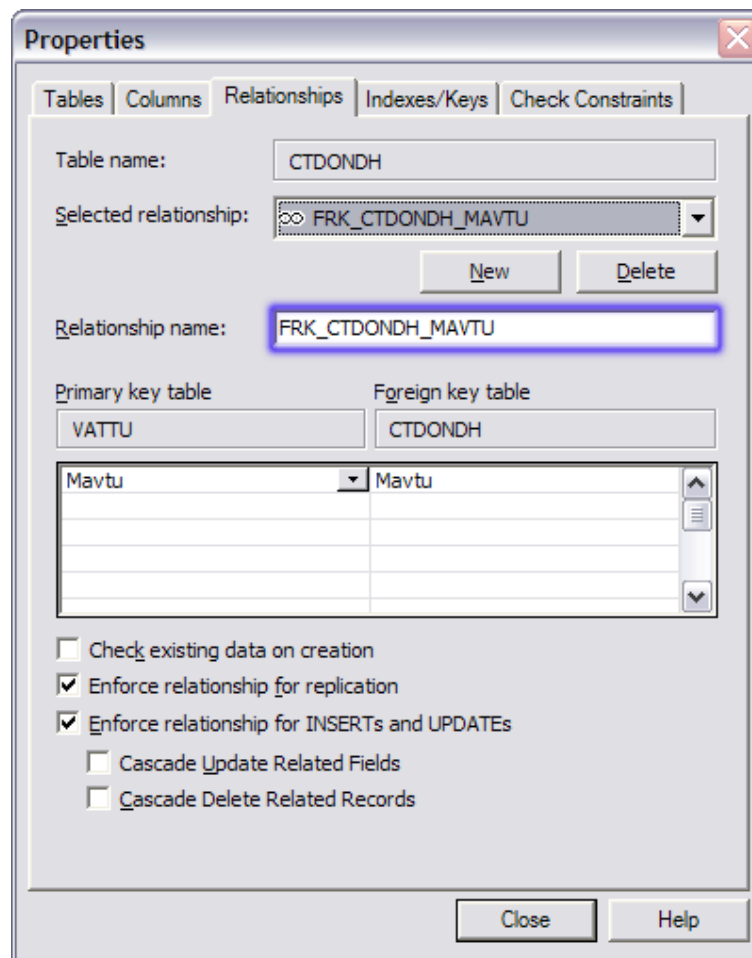
Việc cấp phát tên cho các constraint trong cơ sở dữ liệu là hoàn toàn tự động và tên của các constraint trong một cơ sở dữ liệu phải được đảm bảo là *duy nhất không trùng lặp*.

Trong các hình bên dưới cho chúng ta thấy được tên của những constraint kèm theo các quy tắc kiểm tra toàn vẹn dữ liệu trên bảng CTDONDH (chi tiết đơn đặt hàng) khi tạo lập cấu trúc bảng bằng lệnh **CREATE TABLE** có sử dụng các thành phần **PRIMARY KEY**, **FOREIGN KEY** và **CHECK** mà thí dụ trước đây chúng ta đã thực hiện.



**Hình 3-1. Màn hình thuộc tính bảng.**

Tương tự chúng ta có thể di chuyển qua các trang Relationships hoặc Indexes/Keys để thấy được các quy tắc kiểm tra dữ liệu về khóa ngoại, khóa chính.



**Hình 3-2. Màn hình thuộc tính bảng.**

Chúng ta nhận thấy rằng tên của các constraint luôn kết thúc bằng các con số khó nhớ, hệ thống làm điều này nhằm **đảm bảo** tên của các constraint trong một cơ sở dữ liệu **là duy nhất**. Tuy nhiên các bạn vẫn có thể chủ động đưa ra tên của các constraint thay vì lấy tên mặc nhiên do hệ thống Microsoft SQL Server tự tạo, điều này sẽ giúp chúng ta dễ nhớ và có thể tham chiếu đến trong các câu lệnh khác có liên quan đến tên constraint. Cú pháp bên dưới mô tả cho việc đặt tên các constraint.

Cú pháp:

```
CONSTRAINT Tên_constraint LOẠI Các_tham_số
```

Trong đó:

- ✓ **Tên constraint:** Phải là duy nhất trong cơ sở dữ liệu. Thông thường chúng tôi qui định tên constraint gồm có ba (3) phần. Bắt đầu bằng các chữ: PRK, FRK, UNQ, CHK, DEF dùng chỉ các loại constraint tương ứng, kế tiếp là tên bảng và cuối cùng là tên cột áp dụng quy tắc kiểm tra dữ liệu.
- ✓ **Loại:** là các từ khóa tương ứng cho các loại constraint như: kiểm tra tính duy nhất (**PRIMARY KEY, UNIQUE**), kiểm tra khóa ngoại (**FOREIGN KEY**), kiểm tra miền giá trị (**CHECK, DEFAULT**).
- ✓ **Các tham số:** là các tham số cần thiết đi kèm theo với các loại constraint tương ứng.



**Ví dụ:**

Tạo lại cấu trúc bảng CTDONDH (chi tiết đơn đặt hàng) với các quy tắc kiểm tra dữ liệu như các thí dụ ở trên nhưng tên các constraint do chúng ta qui định. Chúng ta thực hiện câu lệnh **CREATE TABLE** như sau:

```
CREATE TABLE CTDONDH
(
    Sodh CHAR(4) ,
    Mavtu CHAR(4) ,
    SIDat SMALLINT
    CONSTRAINT DEF_CTDONDH_SLDAT DEFAULT 10 ,
    CONSTRAINT PRK_CTDONDH_SODH
        PRIMARY KEY (Sodh, Mavtu) ,
    CONSTRAINT FRK_CTDONDH_SODH
        FOREIGN KEY (Sodh) REFERENCES DONDH (Sodh),
    CONSTRAINT FRK_CTDONDH_MAVTU
        FOREIGN KEY (Mavtu) REFERENCES VATTU (Mavtu),
    CONSTRAINT CHK_CTDONDH_SLDAT
        CHECK (SIDat BETWEEN 10 AND 50)
)
```

Với từ khóa **CONSTRAINT** được sử dụng bên trong câu lệnh **CREATE TABLE** sẽ giúp các bạn có thể chủ động đặt tên của các constraint cho những quy tắc trong việc kiểm tra các toàn vẹn dữ liệu. Thí dụ bên dưới minh họa việc sử dụng các lệnh **CREATE TABLE** với cú pháp khá đầy đủ để thực hiện việc tạo cấu trúc các bảng.

**Ví dụ:**

Sử dụng bốn (4) câu lệnh **CREATE TABLE** đầy đủ nhất để tạo ra cấu trúc các bảng: VATTU (vật tư), NHACC (nhà cung cấp), DONDH (đơn đặt hàng) và CTDONDH (chi tiết đặt hàng) với các yêu cầu như sau:

Bảng	Các quy tắc kiểm tra toàn vẹn dữ liệu
VATTU	-Khóa chính là cột mã vật tư. -Giá trị dữ liệu duy nhất tại cột tên vật tư. -Giá trị mặc định cho cột phần trăm là 20. -Kiểm tra phạm vi giá trị của phần trăm từ 5 đến 100
<b>Câu lệnh thực hiện</b>	



<pre> CREATE TABLE VATTU (     Mavtu    CHAR(4) NOT NULL,     Tenvtu   CHAR(30) NOT NULL,     DvTinh   CHAR(10) NOT NULL,     Phantram  TinyInt     CONSTRAINT DEF_VATTU_PHANTRAM DEFAULT 20 ,     CONSTRAINT PRK_VATTU_MAVTU         PRIMARY KEY (Mavtu) ,     CONSTRAINT UNQ_VATTU_TENVTU  UNIQUE (Tenvtu),     CONSTRAINT CHK_VATTU_PHANTRAM         CHECK (PHANTRAM BETWEEN 5 AND 100) ) </pre>	
NHACC	<ul style="list-style-type: none"> <li>-Khóa chính là cột mã nhà cung cấp.</li> <li>-Giá trị dữ liệu duy nhất tại cột địa chỉ.</li> <li>-Giá trị mặc định cho cột điện thoại là “Chưa có”.</li> </ul>
<b>Câu lệnh thực hiện</b>	
<pre> CREATE TABLE NHACC (     Manhacc  CHAR(3) NOT NULL,     Tennhacc CHAR(30) NOT NULL,     Diachi   VARCHAR(50) NOT NULL,     Dienthoai CHAR(15) NOT NULL     CONSTRAINT DEF_NHACC_DIENHOTOAI         DEFAULT "Chưa có" ,     CONSTRAINT PRK_NHACC_MANHACC         PRIMARY KEY (MANHACC),     CONSTRAINT UNQ_NHACC_DIACHI UNIQUE (DIACHI) ) </pre>	
DONDH	<ul style="list-style-type: none"> <li>-Khóa chính là cột số đơn đặt hàng.</li> <li>-Giá trị mặc định cho các cột ngày đặt hàng và ngày dự kiến nhận hàng là ngày hiện hành.</li> <li>-Ngày dự kiến nhận hàng phải sau ngày đặt hàng.</li> <li>-Kiểm tra tính tồn tại dữ liệu của cột mã nhà cung cấp bên bảng NHACC.</li> </ul>
<b>Câu lệnh thực hiện</b>	



```
CREATE TABLE DONDH
(
    Sodh    CHAR(4) ,
    Ngaydh  DATETIME NOT NULL
        CONSTRAINT DEF_DONDH_NGAYDH DEFAULT GETDATE(),
    Ngaydknh DATETIME NOT NULL
        CONSTRAINT DEF_DONDH_NGAYDKNH DEFAULT GETDATE(),
    Manhacc CHAR(3) NOT NULL
        CONSTRAINT PRK_DONDH_SODH PRIMARY KEY (SODH),
    CONSTRAINT CHK_DONDH_NGAYDKNH
        CHECK (NGAYDH<=NGAYDKNH) ,
    CONSTRAINT FRK_DONDH_MANHACC
        FOREIGN KEY (MANHACC)
        REFERENCES NHACC (MANHACC)
)
```

CTDONDH

- Khóa chính là cột số đặt hàng và mã vật tư.
- Giá trị mặc định cho cột số lượng đặt là 10.
- Kiểm tra phạm vi giá trị của số lượng đặt từ 10 đến 50
- Kiểm tra tính tồn tại dữ liệu của cột số đặt hàng trong bảng DONDH và cột mã vật tư trong bảng VATTU.

#### Câu lệnh thực hiện

```
CREATE TABLE CTDONDH
(
    Sodh    CHAR(4) ,
    Mavtu   CHAR(4) ,
    SIDat   SMALLINT
        CONSTRAINT DEF_CTDONDH_SLDAT DEFAULT 10 ,
    CONSTRAINT PRK_CTDONDH_SODH
        PRIMARY KEY (Sodh, Mavtu) ,
    CONSTRAINT FRK_CTDONDH_SODH
        FOREIGN KEY (Sodh) REFERENCES DONDH (Sodh) ,
    CONSTRAINT FRK_CTDONDH_MAVTU
        FOREIGN KEY (Mavtu) REFERENCES VATTU (Mavtu) ,
    CONSTRAINT CHK_CTDONDH_SLDAT
        CHECK (SIDat BETWEEN 10 AND 50)
)
```

**Chú ý:**

Đối với những cột nào đã có định nghĩa các quy tắc kiểm tra toàn vẹn dữ liệu thông qua các constraint thì chúng ta không thể hủy bỏ hay thay đổi kiểu dữ liệu. Trong các trường hợp này để có thể hủy bỏ hay đổi kiểu dữ liệu của các cột đó, trước tiên chúng ta phải hủy bỏ các constraint có liên quan đến chúng. **Thêm vào constraint mới trong bảng**

Với cú pháp **ALTER TABLE** bên dưới cho phép các bạn **thêm vào các quy tắc** kiểm tra toàn vẹn dữ liệu thông qua các đối tượng constraint bên trong bảng.

Cú pháp:

```
ALTER TABLE Tên_bảng
ADD CONSTRAINT Tên_constraint LOẠI Các_tham_số
[,...]
```

Trong đó:

- ✓ Tên constraint: Phải là duy nhất bên trong cơ sở dữ liệu. Thông thường chúng tôi qui định tên của một constraint bao gồm ba (3) phần nhỏ. Bắt đầu bằng các chữ: PRK, FRK, UNQ, CHK, DEF chỉ các loại constraint tương ứng là khóa chính; khóa ngoại; duy nhất; miễn giá trị; giá trị mặc định, kế sau đó là tên của bảng và cuối cùng là tên cột sẽ áp dụng quy tắc kiểm tra dữ liệu.
- ✓ Loại: là các từ khóa tương ứng cho các loại constraint như: kiểm tra tính duy nhất (**PRIMARY KEY, UNIQUE**), kiểm tra khóa ngoại (**FOREIGN KEY**), kiểm tra miễn giá trị (**CHECK, DEFAULT**).
- ✓ Các tham số: là các tham số cần thiết đi kèm theo với các loại constraint tương ứng.

Trong trường hợp FOREIGN KEY constraint, tham số ON DELETE chỉ định cách SQL Server 2000 thi hành khi hành động xóa xảy ra trên bảng được tham chiếu (bảng cha).

- Nếu chỉ định NO ACTION thì lỗi sẽ xảy ra trong trường hợp mối quan hệ giữa hai bảng cha – con có dữ liệu
- Nếu chỉ định CASCADE thì khi xóa một dòng dữ liệu trên bảng cha thì các dòng dữ liệu liên quan trên bảng con cũng sẽ bị xóa.

Trong trường hợp FOREIGN KEY constraint, tham số ON UPDATE chỉ định cách SQL Server 2000 thi hành khi hành động cập nhật khoá chính xảy ra trên bảng được tham chiếu (bảng cha).

- Nếu chỉ định NO ACTION thì lỗi sẽ xảy ra trong trường hợp mối quan hệ giữa hai bảng cha – con có dữ liệu
- Nếu chỉ định CASCADE thì khi cập nhật giá trị khoá chính của một dòng dữ liệu trên bảng cha thì giá trị các cột khoá ngoại của các dòng dữ liệu liên quan trên bảng con cũng sẽ được cập nhật.

**Ví dụ:**

Sau khi tạo cấu trúc đơn giản của bảng NHACC (Nhà cung cấp). Chúng ta có thể thêm vào các quy tắc kiểm tra duy nhất của dữ liệu tại khóa chính là cột mã nhà cung cấp. Sử dụng các lệnh như sau:



```
CREATE TABLE NHACC
(
    Manhacc CHAR(3) NOT NULL ,
    Tennhacc CHAR(30) NOT NULL ,
    Diachi VARCHAR(50) NOT NULL ,
    Dienthoai CHAR(15)
)
GO

ALTER TABLE NHACC
ADD CONSTRAINT PRK_NHACC_MANHACC
PRIMARY KEY (MANHACC)
```

Chúng ta có thể thêm một quy tắc mới dùng để kiểm tra tính duy nhất dữ liệu tại cột địa chỉ của bảng nhà cung cấp và đưa vào giá trị mặc định cho cột điện thoại của bảng nhà cung cấp là "Chưa có" bằng các câu lệnh như bên dưới:

```
ALTER TABLE NHACC
ADD CONSTRAINT UNQ_NHACC_DIACHI UNIQUE (DIACHI),
CONSTRAINT DEF_NHACC_DIENTHOAI
DEFAULT "Chưa có" FOR DIENTHOAI
```



**Chú ý:**

Đối với việc định nghĩa một giá trị mặc định cho một cột dữ liệu bên trong bảng bắt buộc chúng ta phải có thêm mệnh đề **FOR <tên cột>** để chỉ định giá trị mặc định được tạo ra sẽ áp dụng cho cột dữ liệu nào.

### 1.3.6. Hủy bỏ constraint đã có trong bảng

Với cú pháp **ALTER TABLE** bên dưới cho phép các bạn **hủy bỏ** các quy tắc kiểm tra toàn vẹn dữ liệu thông qua các đối tượng constraint bên trong bảng. Các quy tắc mà chúng ta hủy bỏ sẽ không còn áp dụng để kiểm tra toàn vẹn dữ liệu bên trong của bảng cho đến khi chúng ta cần tạo mới lại nó.

Cú pháp:

```
ALTER TABLE Tên_bảng
DROP CONSTRAINT Tên_constraint [...]
```

Trong đó:

- ✓ Tên constraint: Phải tồn tại trong cơ sở dữ liệu mà các bạn muốn hủy bỏ. Do đó đối với tên của các constraint mà chúng ta chủ động đặt khi sử dụng từ khóa **CONSTRAINT** thì sẽ giúp các bạn dễ dàng gọi nhớ.



**Ví dụ:**

Với các constraint mà chúng ta đã tạo ra trong các thí dụ trước đây có bên trong bảng NHACC (Nhà cung cấp), chúng ta có thể chọn hủy bỏ quy tắc kiểm tra tính duy nhất dữ liệu tại



cột địa chỉ nhà cung cấp. Sử dụng lệnh như sau:

```
ALTER TABLE NHACC
DROP CONSTRAINT UNQ_NHACC_DIACHI
```



**Chú ý:**

Đối với những constraint là khóa chính của bảng mà đã được một bảng khác định nghĩa có khóa ngoại tham chiếu đến thì chúng ta **không thể** nào hủy bỏ khóa chính cho đến khi phải hủy bỏ các constraint khóa ngoại tham chiếu đến từ các bảng khác.



**Ví dụ:**

Hệ thống sẽ không hủy bỏ được constraint khóa chính trong bảng VATTU (vật tư) bởi vì bảng CTDONDH (chi tiết đặt hàng) đã định nghĩa khóa ngoại tham chiếu qua bảng VATTU. Do đó trước tiên, chúng ta cần phải hủy bỏ constraint khóa ngoại của bảng CTDONDH và sau đó mới tiếp tục hủy bỏ constraint khóa chính của bảng VATTU. Lần lượt các bạn thực hiện các lệnh như sau:

```
ALTER TABLE CTDONDH
DROP CONSTRAINT FRK_CTDONDH_MAVTU
GO

ALTER TABLE VATTU
DROP CONSTRAINT PRK_VATTU_MAVTU
```

### 1.3.7. Tắt bỏ các quy tắc kiểm tra toàn vẹn dữ liệu

Với cú pháp **ALTER TABLE** bên dưới cho phép các bạn **tạm thời bỏ qua** việc kiểm tra toàn vẹn dữ liệu thông qua các đối tượng constraint bên trong bảng bằng cách tắt các quy tắc. Các quy tắc mà chúng ta chỉ định sẽ không còn được áp dụng dùng để kiểm tra tính toàn vẹn dữ liệu cho đến khi nào chúng ta bật cho nó hoạt động trở lại. Tuy nhiên các quy tắc này sẽ hoàn toàn không bị xóa đi ra khỏi cơ sở dữ liệu.

Cú pháp:

```
ALTER TABLE Tên_bảng
NOCHECK CONSTRAINT ALL | Tên_constraint [...]
```

Trong đó:

- ✓ Tên constraint: tên của các constraint phải có tồn tại bên trong cơ sở dữ liệu mà các bạn muốn tạm ngưng việc kiểm tra tính toàn vẹn dữ liệu trên đó.
- ✓ Từ khóa ALL: được sử dụng khi các bạn muốn tạm ngưng việc kiểm tra tính toàn vẹn dữ liệu cho tất cả các constraint có liên quan đến bảng đã được định nghĩa trước đó.



**Ví dụ:**

Chúng ta tạm thời tắt việc kiểm tra dữ liệu trong cột số lượng đặt hàng của bảng CTDONDH (chi tiết đặt hàng) nằm trong miền giá trị từ 10 đến 50 khi người dùng thêm hoặc sửa dữ liệu trong bảng CTDONDH. Chúng ta thực hiện câu lệnh ALTER TABLE như sau:

```
ALTER TABLE CTDONDH
NOCHECK CONSTRAINT CHK_CTDONDH_SLDAT
```

**Chú ý:**

Chúng ta chỉ được phép tắt các quy tắc kiểm tra toàn vẹn dữ liệu của các constraint là khóa ngoại (**FOREIGN KEY**), miền giá trị (**CHECK**) và giá trị mặc định (**DEFAULT**). Hai loại constraint còn lại là khóa chính (**PRIMARY KEY**) và duy nhất (**UNIQUE**) hệ thống Microsoft SQL Server hoàn toàn **không cho phép tắt** là bởi vì hệ thống cần phải kiểm tra tính duy nhất của dữ liệu bên trong bảng có định nghĩa khóa chính.

**1.3.8. Bật lại các quy tắc kiểm tra toàn vẹn dữ liệu**

Với cú pháp **ALTER TABLE** bên dưới cho phép các bạn **bật lại** việc áp dụng các quy tắc kiểm tra toàn vẹn dữ liệu thông qua các đối tượng constraint bên trong bảng sau khi mà chúng ta tạm thời đã tắt đi trước đó.

Cú pháp:

```
ALTER TABLE Tên_bảng
CHECK CONSTRAINT ALL | Tên_constraint [...]
```

Trong đó:

- ✓ Tên constraint: tên của các constraint phải tồn tại trong cơ sở dữ liệu mà các bạn muốn bật trở lại sau khi đã tắt tạm thời.
- ✓ Từ khóa ALL: được sử dụng khi muốn bật việc kiểm tra toàn vẹn dữ liệu cho toàn bộ tất cả các constraint liên quan đến bảng.

**Ví dụ:**

Chúng ta bật lại việc kiểm tra dữ liệu trong cột số lượng đặt hàng của bảng CTDONNDH (chi tiết đặt hàng) đã bị tắt tạm thời ở thí dụ trên. Chúng ta thực hiện lệnh **ALTER TABLE** như sau:

```
ALTER TABLE CTDONNDH
CHECK CONSTRAINT CHK_CTDONNDH_SLDAT
```

**II. Mô hình quan hệ dữ liệu (Diagram)**

Đến thời điểm này sau khi xem xét qua hai đối tượng cơ sở trong Microsoft SQL Server là **cơ sở dữ liệu** và **bảng**, việc tiếp theo trong chương này là chúng tôi sẽ trình bày tiếp một số các đối tượng còn lại bên trong cơ sở dữ liệu của Microsoft SQL Server vẫn thường sử dụng. Đối tượng mà chúng tôi muốn đề cập trong phần này chính là mô hình quan hệ dữ liệu (diagram).

**II.1. Khái niệm về mô hình quan hệ dữ liệu**

Đối với người sử dụng đã quen thuộc với loại cơ sở dữ liệu Microsoft Access, màn hình **Relationship** trong Microsoft Access sẽ giúp cho chúng ta hiển thị và tạo lập các mối quan hệ dữ liệu giữa các bảng trong tập tin cơ sở dữ liệu MDB. Thực ra việc tạo mối quan hệ giữa các bảng là nhằm trao đổi và chia sẻ thông tin qua lại giữa các bảng với nhau, đồng thời cũng giúp kiểm tra tính tồn tại dữ liệu (khóa ngoại).

Riêng trong môi trường cơ sở dữ liệu của Microsoft SQL Server thì đến lúc này thì chúng ta chỉ biết được việc tạo cấu trúc các bảng có thể tạo thêm các mối quan hệ dữ liệu ngầm định giữa các bảng thông qua quy tắc kiểm tra ràng buộc về **khóa ngoại** (foreign key constraint). Tuy nhiên các mối quan hệ này không thể hiện một cách trực quan để giúp cho người sử dụng dễ dàng quan sát.

Thấy được sự thiếu sót này trong các phiên bản cũ trước đây. Vì thế kể từ phiên bản 7.0, Microsoft đã đưa ra một đối tượng mới trong cơ sở dữ liệu của Microsoft SQL Server – đó chính là đối tượng mô hình quan hệ dữ liệu (diagram). Trong mô hình quan hệ dữ liệu này các bạn có thể dễ dàng nhìn thấy mối quan hệ dữ liệu giữa các bảng có liên quan với nhau, thực hiện một số các hành động tác động trực tiếp ngay trong các bảng như là: tạo cấu trúc bảng mới, xóa cấu trúc bảng đã có, chỉ định khóa chính cho bảng, thêm các bảng hiện có vào bên trong mô hình quan hệ dữ liệu...



**Chú ý:**

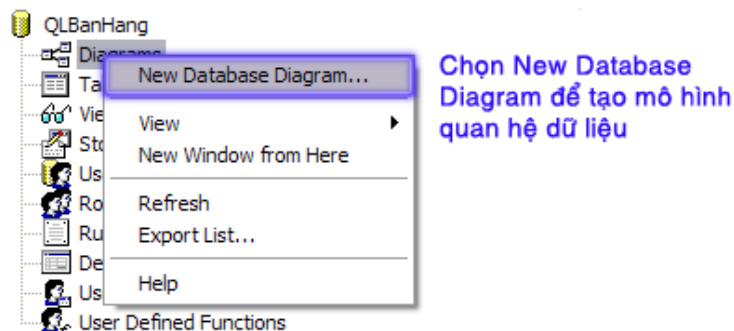
Mô hình quan hệ dữ liệu chỉ có ý nghĩa thể hiện mối tương quan, liên kết giữa các bảng trong một CSDL chứ không có ý nghĩa về mặt lưu trữ hay xử lý dữ liệu. Do đó, trong một CSDL có thể có hay không có mô hình quan hệ đều được.

## II.2. Tạo mới mô hình quan hệ dữ liệu

Đối tượng mô hình quan hệ dữ liệu chỉ được tạo ra và hiển thị trong tiện ích Enterprise Manager nhằm tổng hợp các ràng buộc dữ liệu khóa ngoại trong cơ sở dữ liệu và giúp cho người dùng dễ dàng thấy được các mối liên kết dữ liệu giữa các bảng một cách trực quan hơn.

Thông thường các mối kết hợp bên trong mô hình quan hệ dữ liệu sẽ được tự động tạo ra khi chúng ta chèn các bảng vào mô hình này. Để tạo mới mô hình quan hệ dữ liệu bên trong một cơ sở dữ liệu các bạn lần lượt thực hiện các bước như sau:

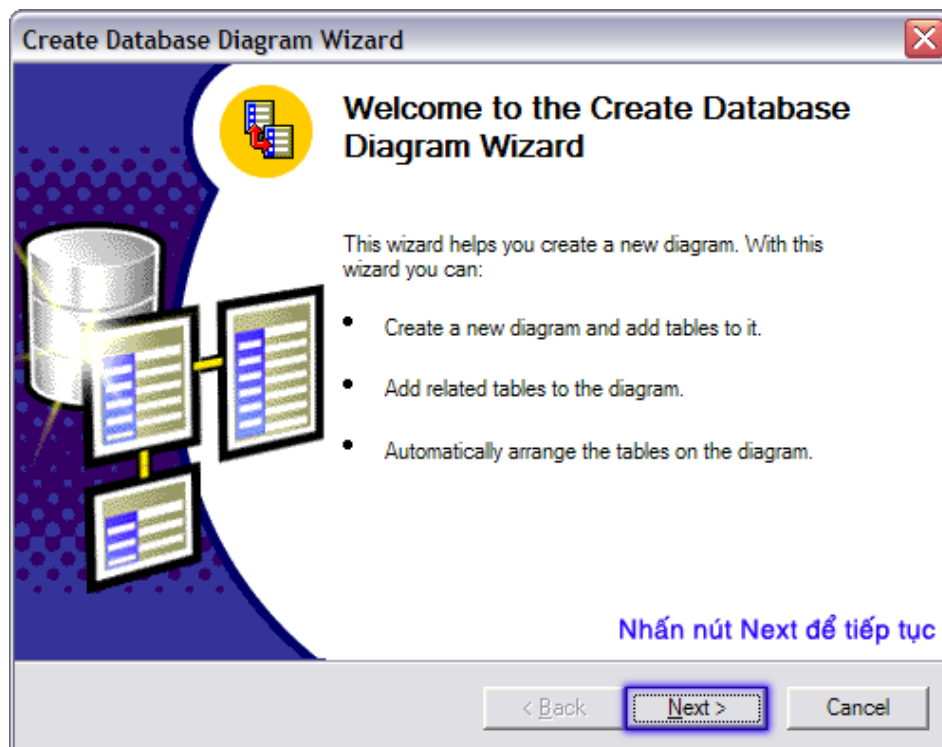
- ✓ Bước 1: Khởi động ứng dụng Enterprise Manager, chọn cơ sở dữ liệu có chứa các bảng cần tạo mới mô hình quan hệ dữ liệu. Chọn chức năng **New Database Diagram...** trong thực đơn tắt sau khi nhấn chuột phải trên đối tượng Diagrams.



**Hình 3-3.** Tạo mới mô hình quan hệ dữ liệu.

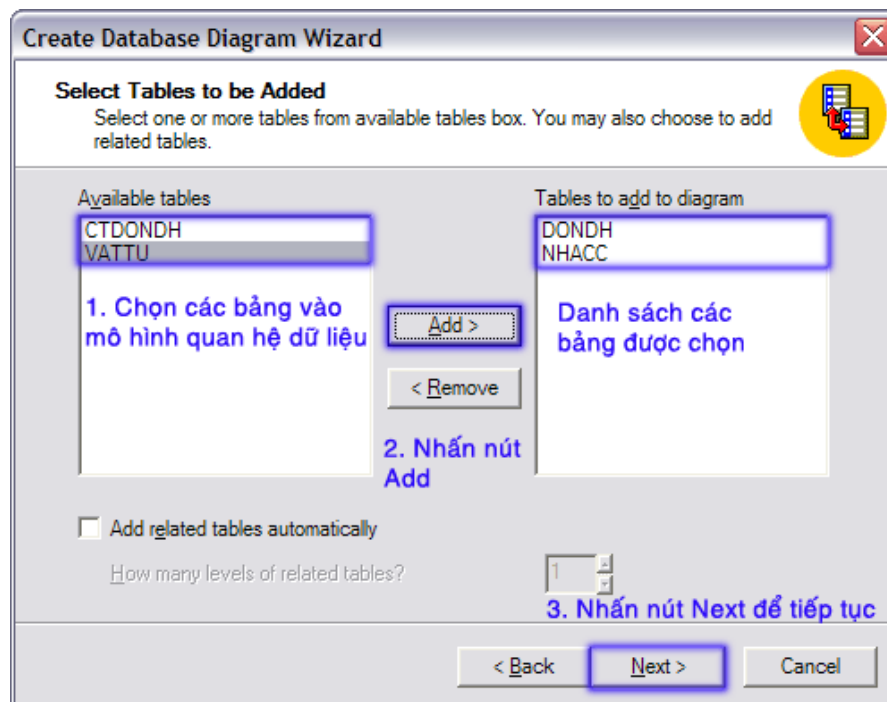
- ✓ Bước 2: Trong màn hình chào đón (Welcome) việc tạo mới mô hình quan hệ dữ liệu bằng trình trợ giúp thông minh, hệ thống hiển thị cho các bạn biết các bước sẽ thực hiện trong quá trình tạo mới mô hình quan hệ dữ liệu. Nhấn nút **Next** để tiếp tục.





**Hình 3-4.** Màn hình chào đón tạo mới mô hình quan hệ dữ liệu.

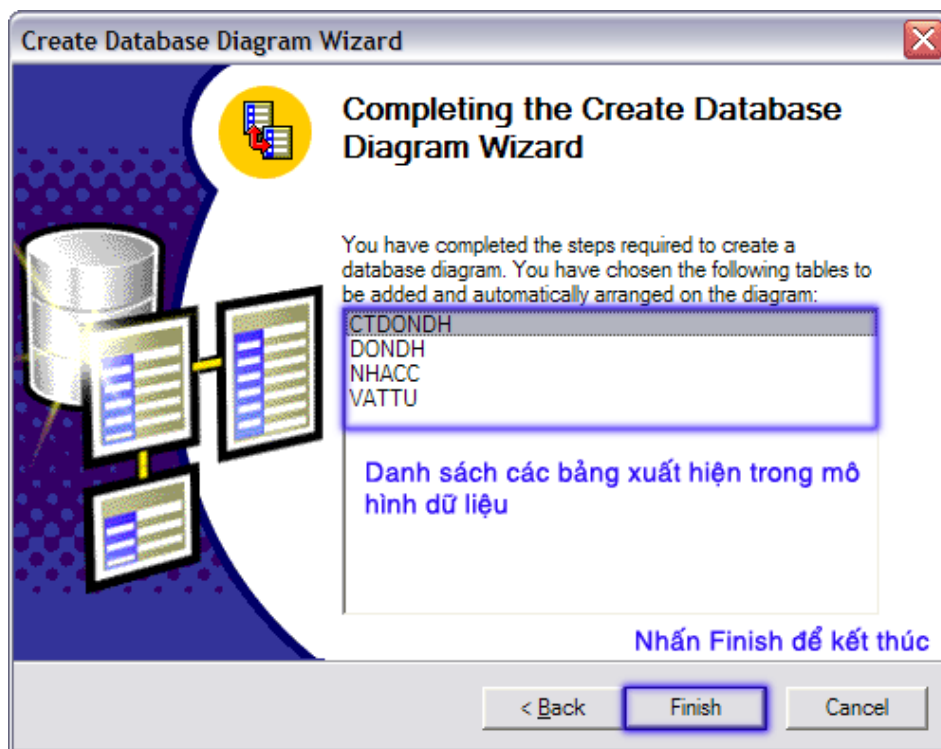
- ✓ Bước 3: Trong màn hình chọn các bảng (Select Tables), lần lượt chọn các bảng trong danh sách bảng đang có trong cơ sở dữ liệu (available tables) mà chúng ta muốn đưa chúng vào mô hình quan hệ dữ liệu, sau đó nhấn nút Add. Nhấn nút Next để tiếp tục.



**Hình 3-5.** Màn hình chọn các bảng đưa vào mô hình quan hệ dữ

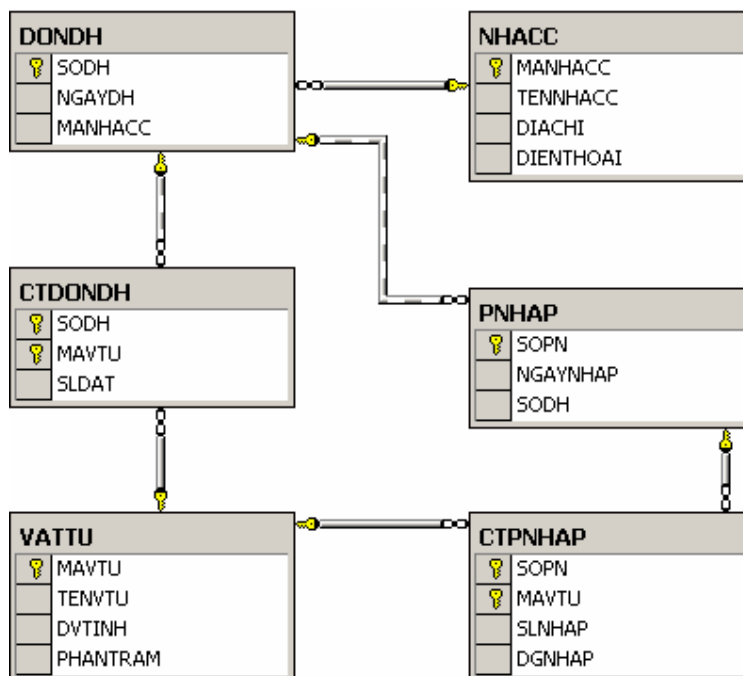
- ✓ Ngược lại khi muốn bỏ đi các bảng có trong mô hình quan hệ dữ liệu thì chúng ta sẽ nhấn vào nút Remove sau khi đã chọn các bảng cần xóa trong danh sách bên phải (**Tables to add to diagram**).

- ✓ Bước 4: Trong màn hình kết thúc (Completing) hệ thống sẽ hiển thị danh sách các bảng mà các bạn đã chọn ở bước 3, các bảng này sẽ xuất hiện đầy đủ trong mô hình quan hệ dữ liệu. Nếu các bạn thấy còn thiếu bảng nào cần bổ sung thêm thì chúng ta có thể nhấn nút Back để quay lại bước 3. Nhấn nút Finish để kết thúc quá trình tạo mới mô hình quan hệ dữ liệu.



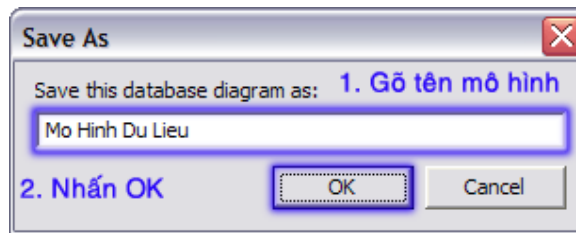
**Hình 3-6.** Màn hình kết thúc quá trình tạo mới mô hình quan hệ dữ liệu.

Bên trong cơ sở dữ liệu Microsoft SQL Server, một mô hình quan hệ dữ liệu đã được tạo lập theo như mô hình bên dưới.



**Hình 3-7.** Màn hình hiển thị mối liên kết giữa các bảng trong mô hình quan hệ dữ liệu.

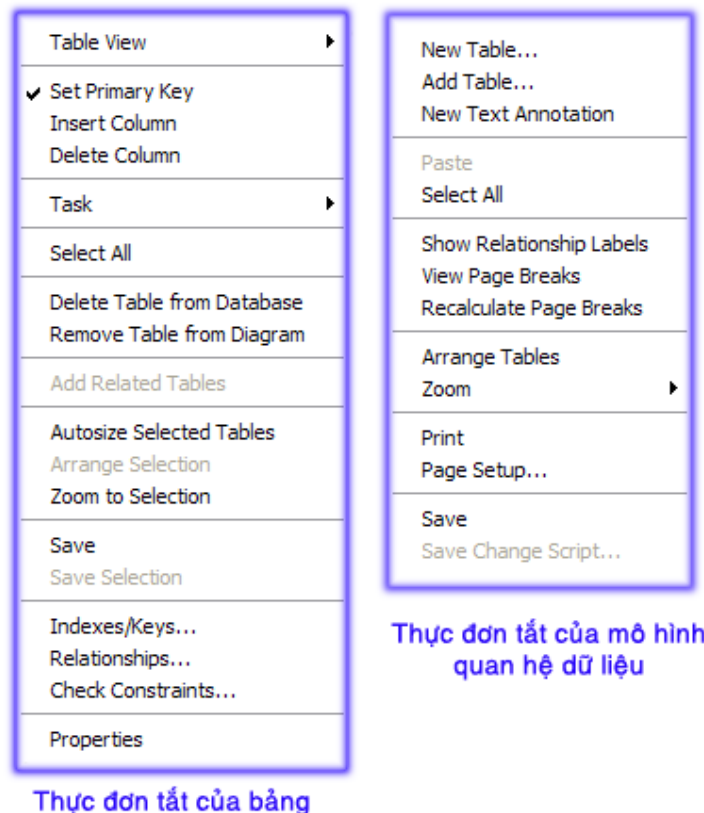
Cuối cùng các bạn nhấn vào biểu tượng nút Save trên thanh công cụ và gõ vào tên của mô hình quan hệ dữ liệu dùng để lưu lại mô hình quan hệ dữ liệu vừa mới tại ở các bước trên.



**Hình 3-8.** Hộp thoại lưu lại tên mô hình quan hệ dữ liệu.

### II.3. Các chức năng trong mô hình quan hệ dữ liệu

Trong mô hình quan hệ dữ liệu có **rất nhiều chức năng** để người sử dụng có thể thực hiện khi đang làm việc bên trong mô hình này. Tuy nhiên trong phần này, chúng tôi chỉ hướng dẫn một số chức năng chính tác động vào các bảng dữ liệu bên trong cơ sở dữ liệu. Các chức năng này sẽ thông qua hai (2) thực đơn tắt khi các bạn nhấn **chuột phải** trên tên một bảng bất kỳ hoặc trên một vùng trống trong mô hình quan hệ dữ liệu.



**Hình 3-9.** Các chức năng trong mô hình quan hệ dữ liệu

#### II.3.1. Sửa đổi cấu trúc bảng hiện có

Các bạn có thể chọn chức năng **Insert column** dùng để thêm cột, hoặc chức năng **Delete Column** dùng để xóa cột, hoặc chức năng **Set Primary Key** dùng để định nghĩa lại khóa chính



của bảng đang được chọn. Trong trường hợp thêm vào các cột mới, để có thể chỉ định được các thuộc tính liên quan đến các cột mới như là: tên cột, kiểu dữ liệu, độ rộng, dữ liệu tại cột có cho phép dữ liệu bỏ trống... chúng ta **nên** chọn chức năng **Column Properties** để có thể thấy được đầy đủ các thuộc tính liên quan đến các cột dữ liệu giống như màn hình thiết kế bảng trong tiện ích Enterprise Manager. Mặc định trong mô hình quan hệ dữ liệu chỉ hiển thị ra tên các cột có bên trong bảng.

### II.3.2. Tạo mới, hủy bỏ bảng trong cơ sở dữ liệu

Các bạn có thể chọn chức năng **New Table** để tạo mới bảng và chèn bảng đó vào mô hình quan hệ dữ liệu hoặc chức năng **Delete Table from Database** để hủy bỏ bảng hiện có ra khỏi cơ sở dữ liệu. Xin cẩn thận khi sử dụng chức năng **Delete Table from Database** bởi vì sau khi đồng ý hủy bỏ thì ngay sau đó dữ liệu và cấu trúc của bảng sẽ không còn được lưu trữ bên trong cơ sở dữ liệu nữa và các bạn sẽ không thể nào khôi phục lại.

### II.3.3. Chèn, xóa bảng trong mô hình quan hệ dữ liệu

Các bạn có thể chọn chức năng **Add Existing Table** để có thể chọn các bảng có trong cơ sở dữ liệu và chèn nó vào mô hình quan hệ dữ liệu hoặc chức năng **Add Related Tables** để hệ thống Microsoft SQL Server tự động chèn các bảng có quan hệ với bảng hiện hành đang chọn vào bên mô hình quan hệ dữ liệu. Các chức năng này vẫn được thường sử dụng để chèn các bảng mới vừa tạo nhưng lại có quan hệ với các bảng hiện đang có trong mô hình quan hệ dữ liệu.

Ngoài ra các bạn cũng có thể chọn chức năng **Remove Table from Diagram** để xóa bảng đang chọn ra khỏi mô hình quan hệ dữ liệu, tuy nhiên cấu trúc và dữ liệu của bảng này hoàn toàn không hề mất đi trong cơ sở dữ liệu.

### II.3.4. Thay đổi cách trình bày

Các bạn có thể chọn chức năng **Arrange Tables** để hệ thống tự động sắp xếp lại vị trí hiển thị của các bảng hiện có trong mô hình quan hệ dữ liệu. Việc sắp xếp lại các bảng này nhằm giúp cho chúng ta dễ dàng thấy rõ ràng về mối liên kết giữa các bảng.

Đối với một mô hình quan hệ dữ liệu lớn có chứa rất nhiều bảng thì chúng ta có thể chọn chức năng **Zoom** với các tỷ lệ khác nhau (10%, 25%, 50%, 75%, 100%, 150% và 200%) để có thể thu nhỏ hoặc phóng to các bảng và các mối liên kết của chúng cho dễ nhìn.

Ngoài ra chúng ta có thể chọn chức năng **Show Relationship Lables** để hiển thị tên của các quy tắc kiểm tra dữ liệu khóa ngoại liên kết quan hệ giữa các bảng.

### II.3.5. In mô hình quan hệ dữ liệu

Các bạn có thể chọn chức năng **View Page Breaks** để thấy được các bảng dữ liệu trong mô hình quan hệ dữ liệu lớn được trình bày trên nhiều trang giấy khác nhau.

Ngoài ra để in mô hình quan hệ dữ liệu ra máy in, chúng ta có thể sử dụng chức năng **Print**

hoặc nhấn vào biểu tượng nút Print trên thanh công cụ.

Tóm lại sau khi tạo xong cấu trúc bảng và thiết lập mô hình quan hệ dữ liệu, công việc **kế tiếp** mà các bạn cần thực hiện trước khi đi tiếp các phần sau là việc nhập **dữ liệu mẫu** vào cho các bảng (xin xem thêm phần phụ lục). Để làm được việc này nhanh nhất, các bạn sẽ chọn chức năng **Open Table ► Return all rows** để nhập dữ liệu trực tiếp vào bảng. Xin nhớ một nguyên tắc trong khi nhập dữ liệu là luôn luôn **nhập dữ liệu của bảng bên quan hệ nhánh 1 trước tiên và nhánh N sau đó**. Thí dụ nhập dữ liệu cho các bảng trong ứng dụng theo thứ tự như sau: VATTU, NHACC, DONDH và CTDONDH

Một lần nữa chúng tôi đề nghị các bạn nên nhập đầy đủ các dữ liệu mẫu vào trong các bảng dữ liệu của cơ sở dữ liệu Quản lý bán hàng trước khi xem xét qua các phần kế tiếp.

### III. Quy tắc kiểm tra miền giá trị dữ liệu (Rule)

#### III.1. Khái niệm

Trong khi giới thiệu các lệnh tạo lập hoặc thay đổi cấu trúc bảng, chúng tôi đã trình bày cho các bạn hiểu được tính năng sử dụng của các đối tượng **constraint** bên trong cơ sở dữ liệu là dùng để **kiểm tra** các loại **toàn vẹn dữ liệu** trong cơ sở dữ liệu khi người sử dụng cập nhật dữ liệu bằng các lệnh **INSERT** hoặc **UPDATE**.

Bên cạnh đó Microsoft SQL Server cũng cung cấp thêm một đối tượng khác mà tính năng hoạt động của nó gần giống như **CHECK constraint**, đối tượng này dùng để kiểm tra giá trị dữ liệu của các cột bên trong bảng phải thỏa một điều kiện nào đó khi dữ liệu bị sửa đổi hoặc thêm vào. Đó chính là đối tượng **rules** mà trong giáo trình này chúng tôi tạm dịch là quy tắc kiểm tra miền giá trị dữ liệu.

Ưu điểm của đối tượng này là khi có các quy tắc kiểm tra miền giá trị dữ liệu **giống nhau** thì chúng ta chỉ cần định nghĩa **một** ra một quy tắc mới trong đối tượng rules và sau đó chỉ định áp dụng quy tắc này cho các cột bên trong các bảng khác nhau hoặc các kiểu dữ liệu mới do chúng ta đã định nghĩa trước đó.



#### Ví dụ:

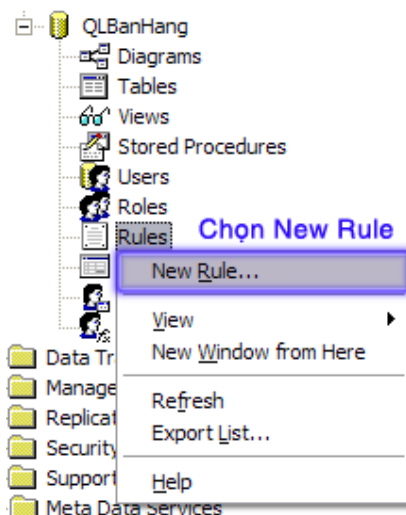
*Trong cấu trúc bảng TONKHO (tồn kho), bắt buộc giá trị dữ liệu tại các cột số lượng đầu kỳ, tổng số lượng nhập, tổng số lượng xuất và số lượng cuối kỳ phải là dương (>0). Thay vì chúng ta cần phải định nghĩa bốn (4) CHECK constraint riêng rẽ bên trong bảng để kiểm tra các ràng buộc miền giá trị tại các cột này, thì chúng ta chỉ cần định nghĩa ra một quy tắc kiểm tra miền giá trị dữ liệu hợp lý và sau đó có thể áp dụng quy tắc đó cho các cột trên trong bảng TONKHO hoặc hay hơn chúng ta sẽ áp dụng quy tắc đó cho đúng kiểu dữ liệu mới mà chúng ta sẽ sử dụng cho các cột số lượng trong bảng dữ liệu TONKHO.*

#### III.2. Tạo mới quy tắc kiểm tra miền giá trị dữ liệu

Giống như các đối tượng khác trong Microsoft SQL Server, chúng ta có hai (2) cách để có thể tạo mới quy tắc kiểm tra miền giá trị dữ liệu. Các bước bên dưới mà chúng tôi trình bày sẽ

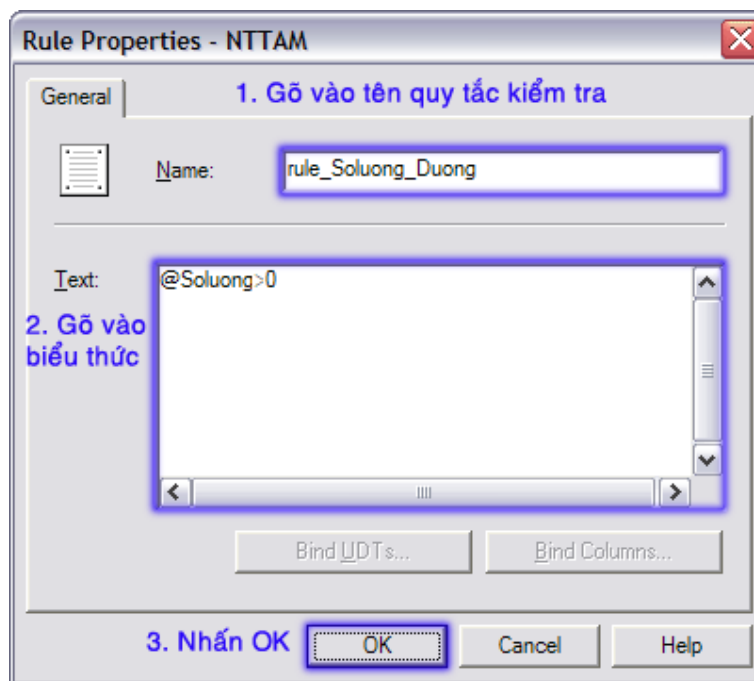
hướng dẫn các bạn cách thức để tạo ra một quy tắc kiểm tra miền giá trị dữ liệu mới bằng tiện ích Enterprise Manager.

- ✓ Bước 1: Khởi động ứng dụng Enterprise Manager. Chọn chức năng **New Rule...** trong thực đơn tắt sau khi nhấn chuột phải trên đối tượng **Rules** để tạo mới một quy tắc kiểm tra dữ liệu.



**Hình 3-10.** Chọn New Rule để tạo quy tắc kiểm tra miền giá trị dữ liệu.

- ✓ Bước 2: Trong màn hình định nghĩa quy tắc kiểm tra dữ liệu lần lượt chỉ định các thuộc tính liên quan đến quy tắc kiểm tra dữ liệu bao gồm: **tên của quy tắc** kiểm tra dữ liệu, **biểu thức** điều kiện áp dụng quy tắc kiểm tra miền giá trị dữ liệu – thông thường là một biểu thức so sánh bao gồm: tên của một biến hình thức, phép toán so sánh và một giá trị/hàm.



**Hình 3-11.** Các thuộc tính liên quan đến quy tắc kiểm tra dữ liệu.

Sau cùng nhấn OK để lưu lại quy tắc kiểm tra dữ liệu mới vừa định nghĩa ở trên. Lúc bây giờ trong cơ sở dữ liệu Quản lý bán hàng của chúng ta sẽ có thêm một đối tượng quy tắc mới.



Ngoài ra chúng ta có cũng có thể tạo mới quy tắc kiểm tra miền giá trị dữ liệu bằng lệnh **CREATE RULE** có cú pháp mô tả như bên dưới.

Cú pháp:

```
CREATE RULE Tên_quy_tắc
AS
    Biểu_thức
```

Trong đó:

- ✓ Tên quy tắc: tên quy tắc kiểm tra miền giá trị dữ liệu được tạo mới, tên quy tắc kiểm tra này phải là **duy nhất** trong một cơ sở dữ liệu.
- ✓ Biểu thức: thông thường là một biểu thức **luận lý** hoặc biểu thức **so sánh** gần giống như biểu thức được sử dụng sau mệnh đề WHERE trong các câu lệnh truy vấn. Biểu thức này gồm có ba (3) thành phần: tên biến hình thức (phải bắt đầu bằng ký tự @), toán tử so sánh, giá trị so sánh. Trong đó tên biến hình thức chính là tên dùng để chỉ định giá trị đại diện cho cột dữ liệu mà quy tắc sẽ áp dụng kiểm tra.



**Ví dụ:**

Để tạo quy tắc kiểm tra miền giá trị dữ liệu cột số lượng phải dương. Chúng ta thực hiện câu lệnh CREATE RULE như sau:

```
CREATE RULE rule_Soluong_Duong
AS
    @Soluong>0
```

Để tạo quy tắc kiểm tra miền giá trị dữ liệu cột đơn vị tính của các vật tư chỉ thuộc trong các từ: **cái, bộ, kg, m2, m3**. Chúng ta thực hiện câu lệnh CREATE RULE như sau:

```
CREATE RULE rule_Dvtinh_Hople
AS
    @Dvt IN ("Cái", "Bộ", "Kg", "m2", "m3")
```



**Chú ý:**

Có thể sử dụng các toán tử như: **LIKE, IN, BETWEEN AND** như là các phép toán so sánh trong lúc xây dựng biểu thức dùng để kiểm tra miền giá trị dữ liệu. Trong biểu thức này **không thể tham chiếu** đến các cột khác bên trong bảng và cũng **không thể** đưa ra các quy tắc **kiểm tra phức tạp** khác.



**Ví dụ:**

Bằng đối tượng quy tắc kiểm tra miền giá trị dữ liệu, ta không thể nào tạo ra một quy tắc như sau: giá trị của cột tổng số lượng xuất trong bảng TONKHO của các vật tư trong một tháng bằng tổng các giá trị tại cột số lượng xuất trong bảng CTPXUAT (chi tiết phiếu xuất) của các vật tư tương ứng trong tháng đó. Ta sẽ kiểm tra các quy tắc tính toán phức tạp dạng này bằng đối tượng **trigger** trong bảng dữ liệu, đối tượng trigger này chúng ta sẽ xem xét ở chương 5.

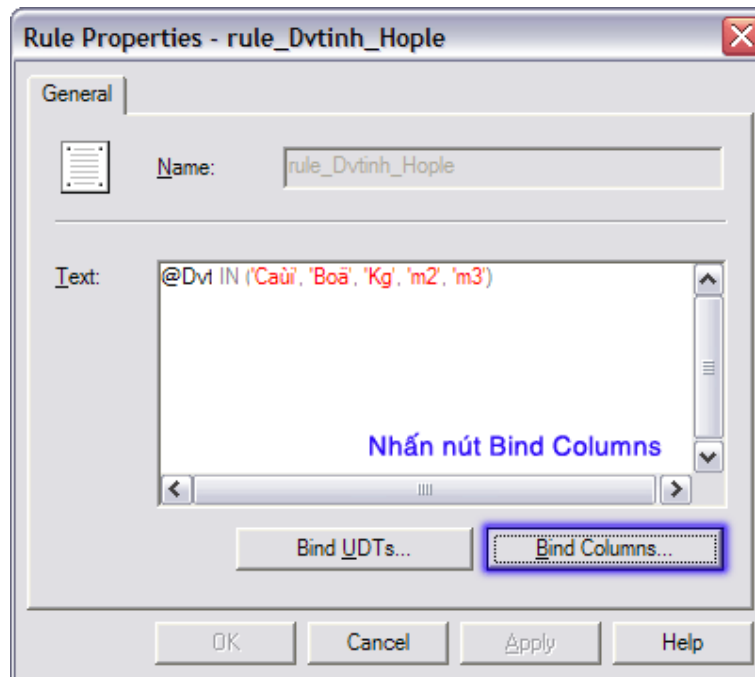


### III.3. Áp dụng quy tắc kiểm tra miền giá trị dữ liệu

Sau khi tạo ra các quy tắc kiểm tra miền giá trị dữ liệu bằng tiện ích Enterprise Manager hoặc câu lệnh CREATE RULE trong cửa sổ Query Analyzer, các quy tắc này vẫn chưa hoạt động – chưa thể thực hiện kiểm tra miền giá trị dữ liệu **cho đến khi** nào chúng ta chỉ định rõ ràng sẽ **áp dụng** các quy tắc đó cụ thể vào những **cột** nào trong bảng hoặc những **kiểu dữ liệu do người dùng định nghĩa**. Ngay sau khi áp dụng quy tắc kiểm tra miền giá trị dữ liệu cho cột bên trong bảng, các quy tắc này chỉ có hiệu lực đối các dòng dữ liệu khi **thêm mới** hoặc khi **sửa đổi** các dòng dữ liệu hiện có bên trong bảng.

Các bước bên dưới sẽ hướng dẫn các bạn cách thức để áp dụng một quy tắc kiểm tra miền giá trị dữ liệu mà chúng ta đã tạo trước đó vào một cột trong bảng trong tiện ích Enterprise Manager.

- ✓ Bước 1: Khởi động ứng dụng Enterprise Manager. Chọn chức năng **Properties** trong thực đơn tắt sau khi nhấn chuột phải trên **tên của quy tắc** muốn áp dụng vào các **cột** trong bảng hoặc **kiểu dữ liệu do người dùng định nghĩa**.

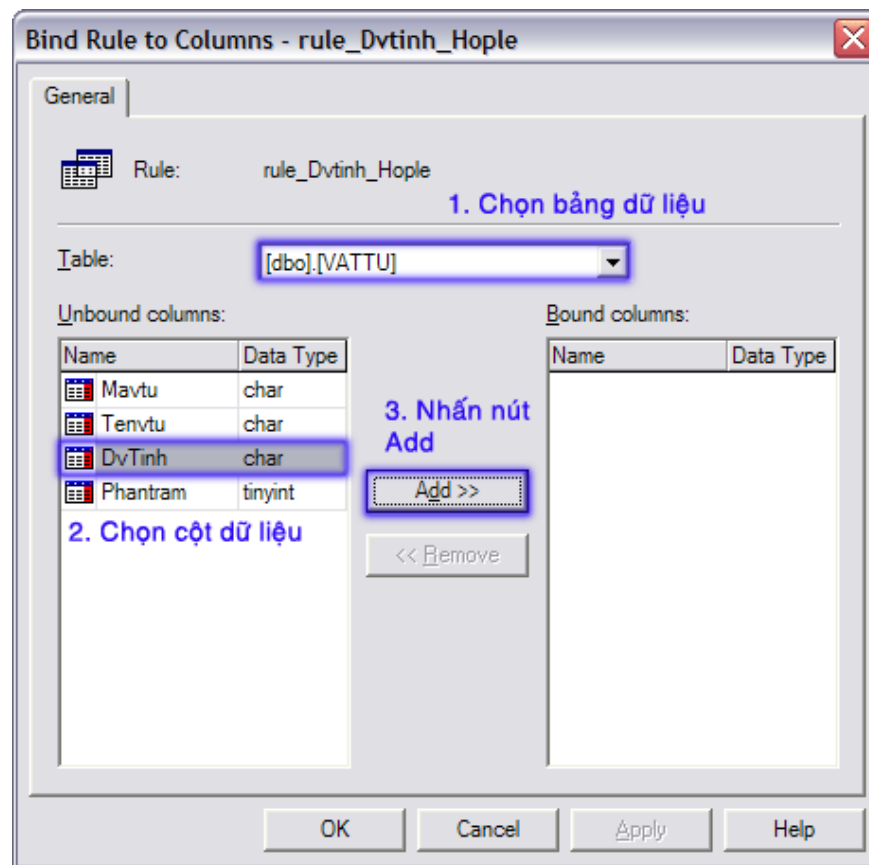


**Hình 3-12.** Chọn quy tắc kiểm tra dữ liệu áp dụng cho các cột.

Trong màn hình các thuộc tính của quy tắc, nhấn vào nút **Bind Columns** để chọn ra các cột trong bảng mà quy tắc kiểm tra miền giá trị dữ liệu hiện hành sẽ áp dụng ở bước 2.

- ✓ Bước 2: Trong màn hình chọn các cột bên trong bảng, lần lượt chọn bảng dữ liệu, các cột có trong bảng đó để chỉ định việc áp dụng quy tắc kiểm tra miền giá trị dữ liệu hiện hành cho các cột. Sau đó nhấn nút **Add**.

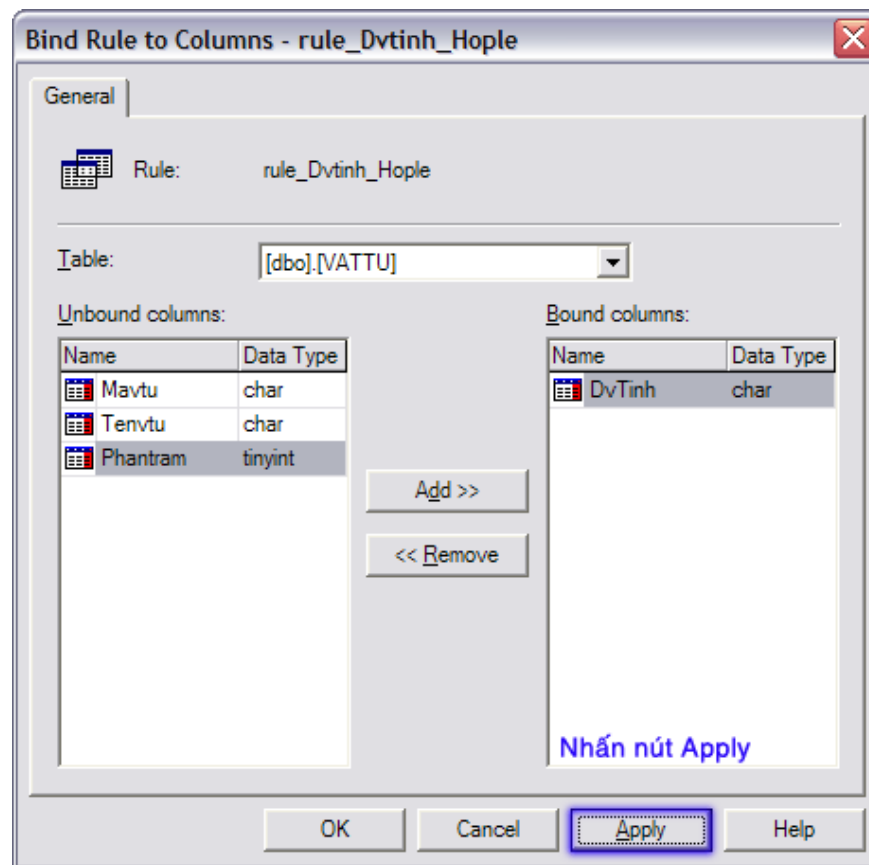




**Hình 3-13.** Chọn các cột để áp dụng quy tắc kiểm tra dữ liệu.

Chúng ta có thể áp dụng **cùng lúc** một quy tắc kiểm tra miền giá trị dữ liệu cho nhiều cột khác nhau trên nhiều bảng.

- ✓ Bước 3: Trong màn hình chọn các cột, nhấn nút Apply để áp dụng quy tắc kiểm tra dữ liệu hiện hành cho các cột đã chọn. Chúng ta có thể quay lại thực hiện lại các bước 2 và bước 3 cho các cột nằm trong các bảng khác.



**Hình 3-14.** Nhấn Apply để áp dụng quy tắc hiện hành cho cột đã chọn.

- ✓ Bước 4: Sau cùng nhấn OK để đóng màn hình chọn các cột lại. Quay lại màn hình các thuộc tính của quy tắc, lại nhấn OK một lần nữa để kết thúc quá trình chỉ định các cột sẽ áp dụng quy tắc kiểm tra miền giá trị dữ liệu.

Ngoài ra chúng ta có cũng có áp dụng quy tắc kiểm tra miền giá trị dữ liệu cho các cột trong bảng hoặc các kiểu dữ liệu do người dùng định nghĩa bằng thủ tục nội tại hệ thống **sp\_bindrule**.

Cú pháp:

```
EXEC sp_bindrule Tên_quy_tắc, Tên_đối_tượng
```

Trong đó:

- ✓ Tên quy tắc: tên quy tắc kiểm tra miền giá trị dữ liệu đã được tạo ra trước đó.
- ✓ Tên đối tượng: tên cột của bảng hoặc tên kiểu dữ liệu do người dùng định nghĩa. Thông thường nếu là tên cột của bảng bắt buộc chúng ta phải theo định dạng là: **Tên\_bảng.tên\_cột**.



**Ví dụ:**

Để áp dụng quy tắc kiểm tra có tên là rule\_Dvtinh\_Hople cho cột đơn vị tính trong bảng VATTU. Chúng ta thực hiện câu lệnh như sau:

```
EXEC sp_bindrule rule_Dvtinh_Hople , "VATTU.Dvtinh"
```



Bây giờ khi thực hiện thêm một vật tư mới vào bảng VATTU quy tắc kiểm tra miền giá trị dữ liệu của cột đơn vị tính sẽ được áp dụng, có nghĩa là giá trị dữ liệu tại cột đơn vị tính chỉ có thể là: **cái, bộ, kg, m2, m3**.

Khi đó nếu chúng ta thêm vật tư mới như sau:

```
INSERT INTO VATTU
```

```
VALUES ("TG01", "Bia lon Tiger", "Thùng", 60)
```

Thì giá trị dữ liệu tại cột đơn vị tính (**Thùng**) đã vi phạm quy tắc kiểm tra mà chúng ta đã đặt ra trước đó và sau cùng dữ liệu sẽ không được lưu trữ vào bảng VATTU.



#### Ví dụ:

Để áp dụng quy tắc kiểm tra có tên là rule\_Soluong\_Duong cho kiểu dữ liệu mới có tên là uddt\_Soluong. Chúng ta thực hiện câu lệnh như sau:

```
EXEC sp_bindrule rule_Soluong_Duong, "Uddt_Soluong"
```

Bây giờ khi thực hiện cập nhật dữ liệu tại các cột số lượng đầu kỳ, số lượng nhập, số lượng xuất và số lượng cuối kỳ trong bảng TONKHO thì giá trị dữ liệu tại các cột này phải đảm bảo là dương mới không vi phạm quy tắc kiểm tra miền giá trị dữ liệu.

Sau cùng khi chúng ta không còn muốn áp dụng các quy tắc kiểm tra miền giá trị dữ liệu cho các cột trong bảng hoặc các kiểu dữ liệu do người dùng định nghĩa thì bắt buộc phải sử dụng thủ tục nội tại hệ thống có tên là **sp\_unbindrule** để làm điều đó.

Cú pháp:

```
EXEC sp_unbindrule Tên_đối_tượng
```

Trong đó:

- ✓ Tên đối tượng: tên cột của bảng hoặc tên kiểu dữ liệu do người dùng định nghĩa. Thông thường nếu tên của cột bên trong bảng thì bắt buộc phải theo định dạng là: **tên\_bảng.tên\_cột**.



#### Ví dụ:

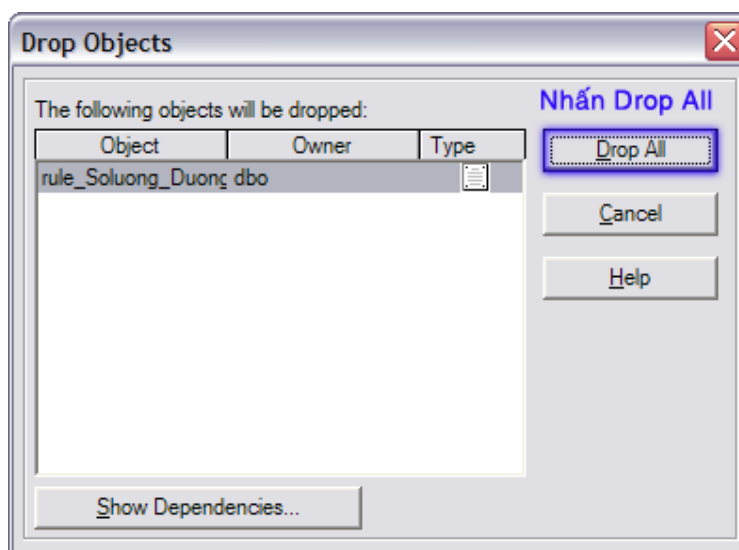
Để bỏ đi hết tất cả các quy tắc kiểm tra đã áp dụng cho kiểu dữ liệu mới do người dùng định nghĩa có tên là uddt\_Soluong. Chúng ta thực hiện câu lệnh như bên dưới:

```
EXEC sp_unbindrule "uddt_Soluong"
```

### III.4. Xóa quy tắc kiểm tra miền giá trị dữ liệu

Khi một quy tắc kiểm tra miền giá trị dữ liệu trong cơ sở dữ liệu không còn sử dụng để áp dụng cho các kiểu dữ liệu do người dùng định nghĩa hoặc các cột trong bảng, chúng ta có thể hủy bỏ nó đi. Tuy nhiên nếu quy tắc kiểm tra miền giá trị dữ liệu còn được nối kết vào ít nhất một cột bên trong một bảng nào đó hoặc một kiểu dữ liệu do người dùng định nghĩa thì các bạn sẽ không thể nào hủy được quy tắc kiểm tra miền giá trị đó.

Để hủy một quy tắc kiểm tra miền giá trị dữ liệu, chúng ta sẽ chọn chức năng **Delete** trên thực đơn tắt sau khi nhấn chuột phải vào tên quy tắc kiểm tra miền giá trị dữ liệu muốn hủy bỏ trong ứng dụng Enterprise Manager và xác nhận đồng ý hủy bằng cách chọn nút **Drop All** trong màn hình hủy bỏ các đối tượng bên trong cơ sở dữ liệu của Microsoft SQL Server.



**Hình 3-15.** Màn hình xác nhận hủy bỏ quy tắc kiểm tra dữ liệu.

Ngoài ra chúng ta cũng có thể sử dụng lệnh có tên **DROP RULE** để hủy bỏ quy tắc kiểm tra miền giá trị dữ liệu.

Cú pháp:

```
DROP RULE Tên_qui_tắc [, ...]
```

Trong đó:

- ✓ Tên quy tắc: tên các quy tắc muốn hủy bỏ. Các quy tắc này phải tồn tại trong cơ sở dữ liệu và không còn nối kết với bất kỳ cột nào trong bảng hoặc các kiểu dữ liệu do người dùng định nghĩa.



**Ví dụ:**

Hủy bỏ quy tắc kiểm tra miền giá trị dữ liệu *rule\_Dvtinh\_Hople* trong cơ sở dữ liệu *QLBanHang*.

```
DROP RULE rule_Dvtinh_Hople
```

Tuy nhiên hệ thống sẽ xuất hiện thông báo lỗi bởi vì quy tắc này vẫn còn áp dụng cho cột đơn vị tính trong bảng *VATTU*.

```
Server: Msg 3716, Level 16, State 1, Line 1
```

```
The rule 'rule_Dvtinh_Hople' cannot be dropped because it is bound to one or more column.
```

## IV. Giá trị mặc định (Default)

### IV.1. Khái niệm

Giống như khái niệm **DEFAULT constraint**, đối tượng giá trị mặc định cho phép chúng ta tạo ra **giá trị mặc định** cho các **cột dữ liệu** hoặc các **kiểu dữ liệu do người dùng định nghĩa** trong trường hợp khi thêm mới mẫu tin vào bảng mà giá trị tại các cột đó người dùng không cung cấp.

Cũng giống như kiểu dữ liệu do người dùng định nghĩa hoặc quy tắc kiểm tra miền giá trị dữ

liệu, giá trị mặc định thường được sử dụng cho các **cơ sở dữ liệu lớn** nhằm đảm bảo tính nhất quán về dữ liệu của các cột trong các bảng.



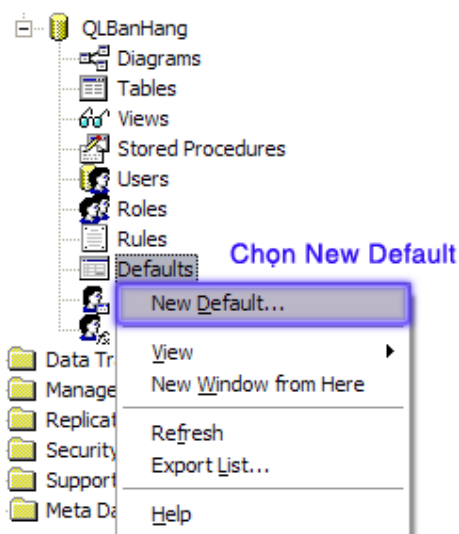
**Ví dụ:**

Trong một cơ sở dữ liệu gồm có các bảng: khách hàng, nhà cung cấp, nhân viên. Trong mỗi bảng lại có cột website (công ty hoặc cá nhân) dùng để lưu địa chỉ trang web của các khách hàng, nhà cung cấp, nhân viên. Tuy nhiên dữ liệu tại cột này là không bắt buộc phải có bởi vì không phải khách hàng nào, nhà cung cấp nào hoặc nhân viên nào cũng đều có trang web riêng. Thay vì để dữ liệu hiển thị tại các cột này là chữ “NULL” khi người sử dụng không đưa vào giá trị tại cột này thì chúng ta sẽ tạo giá trị mặc định cho các cột này là chuỗi “Chưa có”.

## IV.2. Tạo mới giá trị mặc định

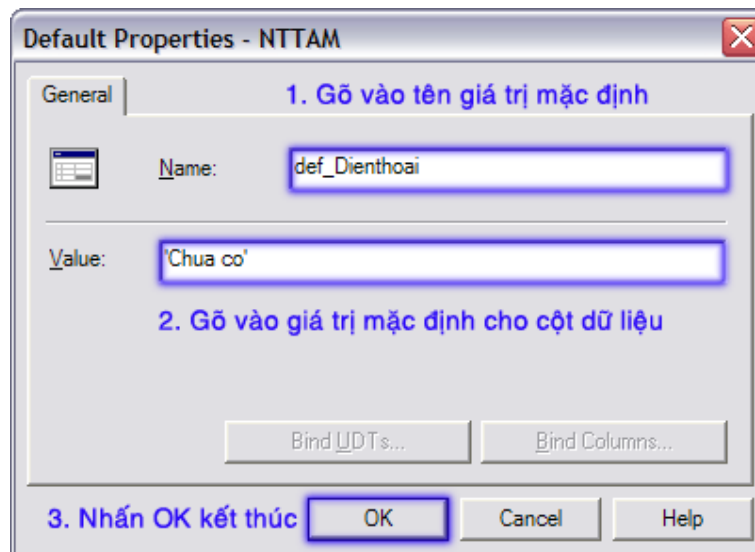
Giống như các đối tượng khác trong Microsoft SQL Server, chúng ta có hai (2) cách để có thể tạo mới đối tượng giá trị mặc định. Các bước bên dưới mà chúng tôi trình bày sẽ hướng dẫn các bạn cách thức tạo ra giá trị mặc định bằng tiện ích Enterprise Manager.

- ✓ Bước 1: Khởi động ứng dụng Enterprise Manager. Chọn chức năng **New Default...** trong thực đơn tắt sau khi nhấn chuột phải trên đối tượng **Defaults**.



**Hình 3-16.** Chọn New Default để tạo giá trị mặc định.\_

- ✓ Bước 2: Trong màn hình định nghĩa giá trị mặc định lần lượt chỉ định các thuộc tính liên quan đến giá trị mặc định bao gồm: tên của giá trị mặc định, giá trị cụ thể hoặc tên hàm hoặc biểu thức tính toán làm giá trị mặc định.



**Hình 3-17.** Các thuộc tính liên quan đến giá trị mặc định.

Sau cùng nhấn OK để lưu lại giá trị mặc định mới vừa định nghĩa. Lúc bây giờ trong cơ sở dữ liệu Quản lý bán hàng của chúng ta sẽ có thêm một đối tượng giá trị mặc định mới vừa tạo.

Ngoài ra chúng ta có cũng có thể tạo mới giá trị mặc định bằng lệnh **CREATE DEFAULT** có cú pháp mô tả như bên dưới.

Cú pháp:

```
CREATE DEFAULT Tên_giá_trị_mặc_định
AS
    Biểu_thức
```

Trong đó:

- ✓ Tên giá trị mặc định: tên giá trị mặc định được tạo mới, tên giá trị mặc định này phải là **duy nhất** trong một cơ sở dữ liệu.
- ✓ Biểu thức: là một giá trị cụ thể nào đó như số, ngày, chuỗi hoặc một hàm, biểu thức tính toán.



**Ví dụ:**

Để tạo giá trị mặc định cho các cột điện thoại là chuỗi "Chưa có". Chúng ta thực hiện câu lệnh **CREATE DEFAULT** như sau:

```
CREATE DEFAULT Def_Dienthoai
AS
    "Chưa có"
```

### IV.3. Liên kết giá trị mặc định vào cột dữ liệu

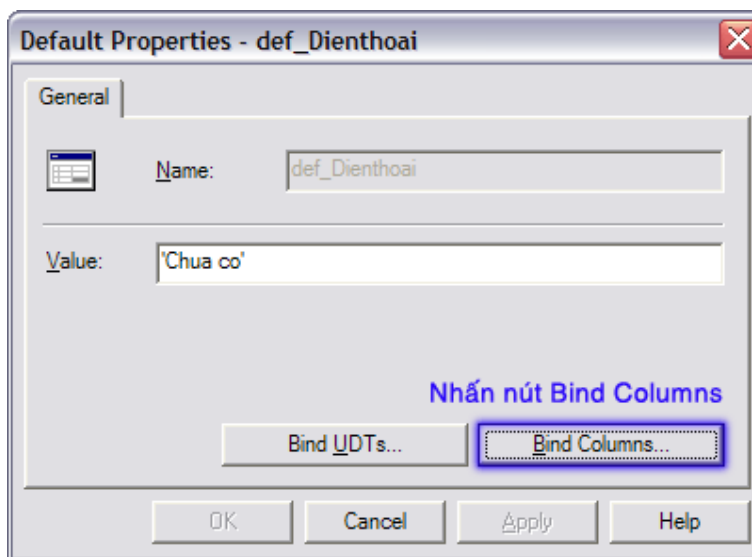
Sau khi tạo ra các giá trị mặc định bằng tiện ích Enterprise Manager hoặc câu lệnh **CREATE DEFAULT**, các giá trị mặc định này vẫn chưa hoạt động cho đến khi nào chúng ta chỉ định rõ ràng là nó sẽ **liên kết** các **giá trị mặc định** vào những **cột** nào trong bảng hoặc những **kiểu dữ liệu do người dùng định nghĩa**. Ngay sau khi liên kết, các giá trị mặc định sẽ được áp dụng



cho các cột khi thêm mới dữ liệu mà người sử dụng không cung cấp thông tin tại các cột đó.

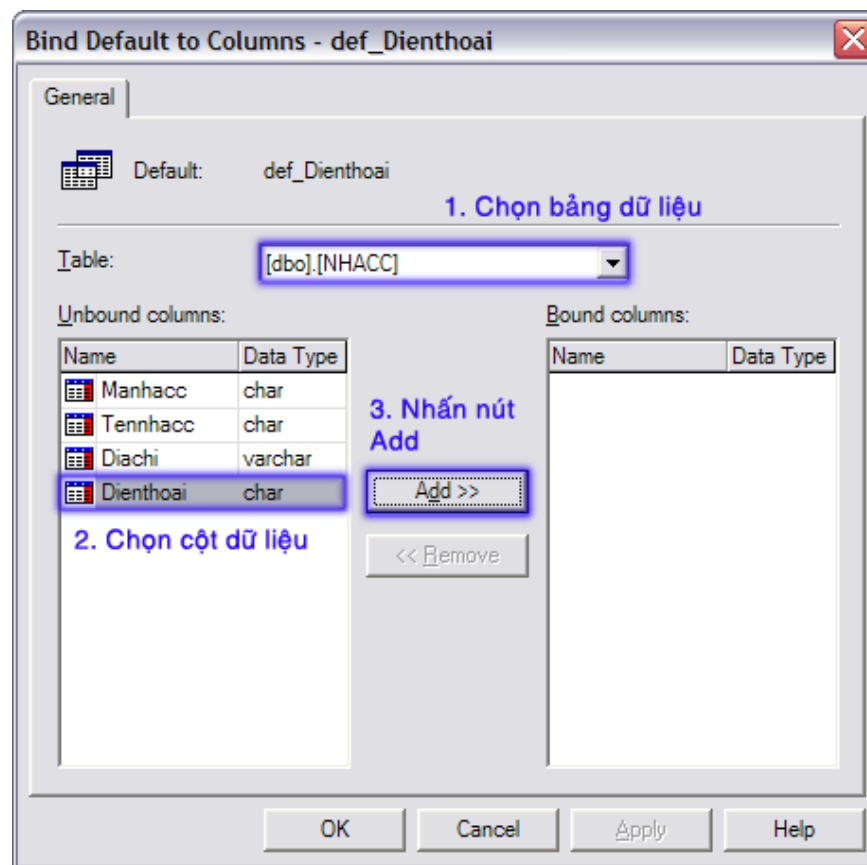
Các bước bên dưới mà chúng tôi trình bày sẽ hướng dẫn các bạn cách thức để **liên kết** giá trị mặc định vào một cột trên bảng dữ liệu trong tiện ích Enterprise Manager.

- ✓ Bước 1: Khởi động ứng dụng Enterprise Manager. Chọn chức năng **Properties** trong thực đơn tắt sau khi nhấn chuột phải trên **tên của giá trị mặc định** muốn liên kết vào các **cột** trong bảng hoặc **kiểu dữ liệu do người dùng định nghĩa**. Trong màn hình các thuộc tính của giá trị mặc định, nhấn vào nút **Bind Columns** để chọn ra các cột trong bảng mà giá trị mặc định sẽ liên kết vào.



**Hình 3-18.** Chọn giá trị mặc định liên kết cho các cột.

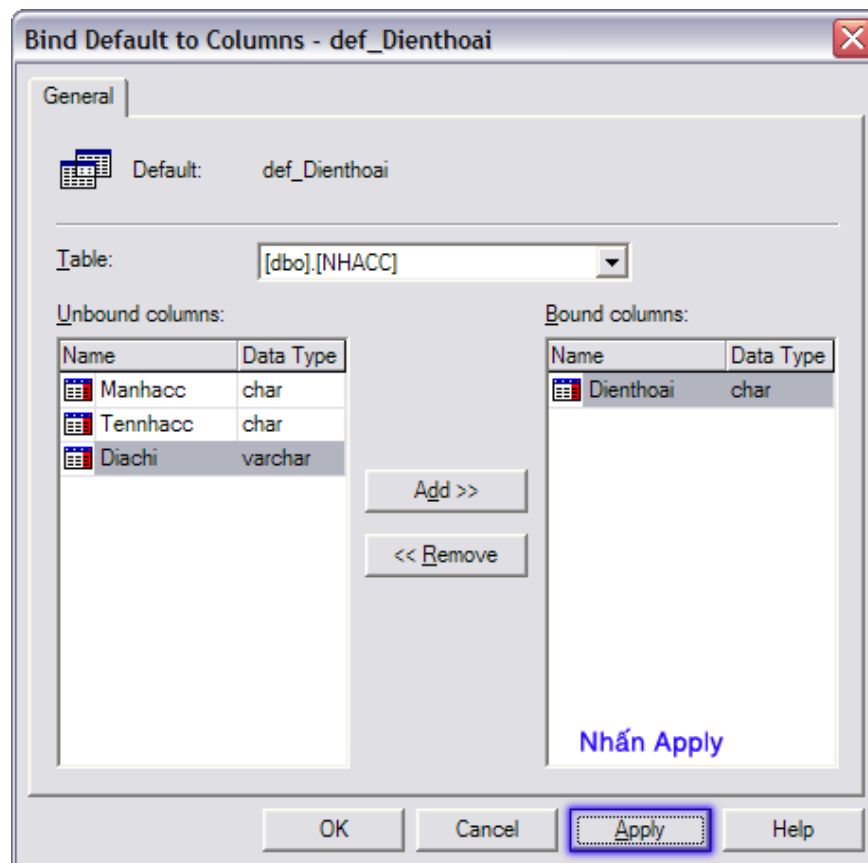
- ✓ Bước 2: Trong màn hình chọn các cột, chúng ta lần lượt chọn bảng dữ liệu, các cột trong bảng đó để liên kết giá trị mặc định vào các cột. Sau đó nhấn nút Add.



**Hình 3-19.** Chọn các cột để liên kết với giá trị mặc định.



- ✓ Bước 3: Trong màn hình chọn các cột, chúng ta nhấn nút Apply để liên kết giá trị mặc định cho các cột đã chọn trước đó. Chúng ta có thể quay lại thực hiện lại bước 2 và bước 3 trước đó cho các cột nằm bên các bảng khác.



**Hình 3-20.** Nhấn Apply để liên kết giá trị mặc định cho cột đã chọn.

- ✓ Bước 4: Sau cùng nhấn **OK** để đóng màn hình chọn các cột. Quay lại màn hình các thuộc tính của giá trị mặc định, tiếp tục lại nhấn OK lần nữa để kết thúc quá trình liên kết các giá trị mặc định vào các cột.



**Chú ý:**

Đối với các cột đã có định nghĩa giá trị mặc định trước đó bằng mệnh đề **DEFAULT** hoặc **DEFAULT constraint** thì chúng ta **không được phép** liên kết đối tượng giá trị mặc định vào các cột này. Bởi vì nó đã có rồi!

Ngoài ra chúng ta có cũng có liên kết giá trị mặc định cho các cột trong bảng hoặc các kiểu dữ liệu do người dùng định nghĩa bằng thủ tục nội tại hệ thống **sp\_bindefault** (chỉ có **một** chữ d)

Cú pháp:

```
EXEC sp_bindefault Tên_mặc_định, Tên_đối_tượng
```

Trong đó:

- ✓ Tên mặc định: tên giá trị mặc định đã được tạo ra trước đó.
- ✓ Tên đối tượng: tên cột của bảng hoặc tên kiểu dữ liệu do người dùng định nghĩa. Thông thường nếu sử dụng tên của cột trong bảng thì bắt buộc phải theo định dạng là: **tên\_bảng.tên\_cột**.

**Ví dụ:**

Để liên kết giá trị mặc định vừa tạo `def_Dienthoai` vào cột `Dienthoai` có trong bảng `NHACC`. Chúng ta thực hiện câu lệnh như sau:

```
EXEC sp_bindefault def_Dienthoai ,
    "NHACC.Dienthoai"
```

Bây giờ khi thực hiện thêm một nhà cung cấp mới vào bảng `NHACC` thì giá trị mặc định của cột điện thoại sẽ là chữ “Chưa có” khi người sử dụng bỏ trống thông tin này.

Khi đó nếu chúng ta thêm nhà cung cấp `C07` với các thông tin như sau:

```
INSERT INTO NHACC (Manhacc, Tennhacc, Diachi)
VALUES("C07" , "Cao Minh Trung" , "11 LôA Cx Phú Lâm")
```

Thì giá trị tại cột điện thoại của nhà cung cấp `C07` sẽ là chữ “Chưa có”

Cuối cùng khi chúng ta không còn muốn **liên kết** giá trị mặc định vào các **cột** trong bảng hoặc các **kiểu dữ liệu do người dùng định nghĩa** thì bắt buộc phải sử dụng thủ tục nội tại hệ thống có tên là **`sp_unbindefault`** để làm điều đó (chỉ có **một** chữ d).

Cú pháp:

```
EXEC sp_unbindefault Tên_đối_tượng
```

Trong đó:

- ✓ Tên đối tượng: tên cột của bảng hoặc tên kiểu dữ liệu do người dùng định nghĩa. Thông thường tên của cột bắt buộc phải theo định dạng là: **tên\_bảng.tên\_cột**.

**Ví dụ:**

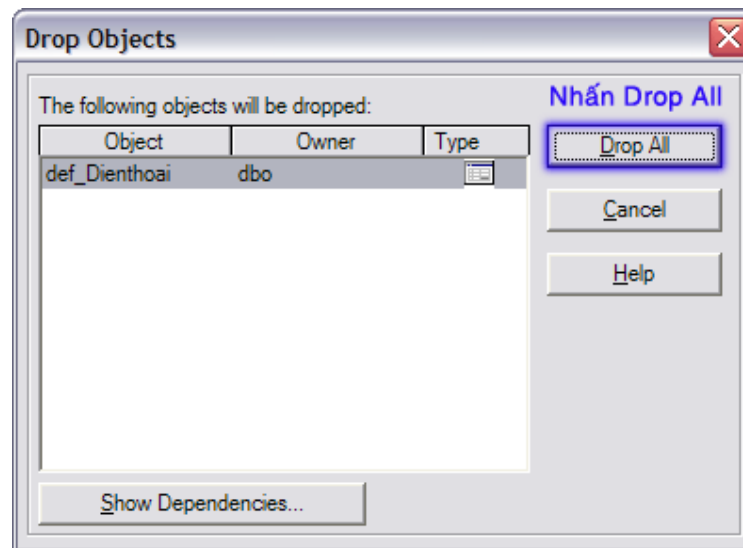
Để hủy bỏ giá trị mặc định đã liên kết cho cột điện thoại trong bảng `NHACC` trước đây. Chúng ta thực hiện câu lệnh như sau:

```
EXEC sp_unbindefault "NHACC.Dienthoai"
```

## V. Xóa giá trị mặc định

Khi giá trị mặc định trong cơ sở dữ liệu **không còn sử dụng** cho các kiểu dữ liệu do người dùng định nghĩa hoặc các cột trong bảng, chúng ta có thể hủy bỏ nó đi. Tuy nhiên nếu giá trị mặc định còn được liên kết vào **ít nhất một** cột bên trong một bảng nào đó hoặc một kiểu dữ liệu do người dùng định nghĩa thì các bạn sẽ **không thể** nào hủy được nó.

Để hủy một giá trị mặc định, chúng ta sẽ chọn chức năng **Delete** trên thực đơn tắt sau khi nhấn chuột phải vào tên giá trị mặc định muốn hủy bỏ trong ứng dụng Enterprise Manager và xác nhận đồng ý hủy bằng cách chọn nút **Drop All** trong màn hình hủy bỏ các đối tượng bên trong cơ sở dữ liệu của Microsoft SQL Server.



**Hình 3-21.** Màn hình xác nhận hủy bỏ giá trị mặc định.

Ngoài ra chúng ta cũng có thể sử dụng lệnh có tên **DROP DEFAULT** để hủy bỏ giá trị mặc định khi không còn dùng nữa.

Cú pháp:

```
DROP DEFAULT Tên_gt_mặc_định [, ...]
```

Trong đó:

- ✓ Tên giá trị mặc định: tên các giá trị mặc định muốn hủy bỏ. Các giá trị mặc định này phải **tồn tại** trong cơ sở dữ liệu và không còn liên kết với bất kỳ cột nào trong bảng hoặc các kiểu dữ liệu do người dùng định nghĩa.



**Ví dụ:**

Hủy giá trị mặc định có tên *def\_Dienthoai* trong cơ sở dữ liệu *QLBanHang*.

```
DROP DEFAULT def_Dienthoai
```

Tuy nhiên hệ thống sẽ xuất hiện thông báo lỗi nếu như giá trị mặc định vẫn còn liên kết với cột điện thoại trong bảng *NHACC*.

```
Server: Msg 3716, Level 16, State 3, Line 1
```

*The default 'def\_Dienthoai' cannot be dropped because it is bound to one or more column.*

Tóm lại trong chương hai chúng tôi đã giúp các bạn hiểu và sử dụng các đối tượng dữ liệu cơ sở trong Microsoft SQL Server một cách đúng đắn. Việc tạo ra các đối tượng này phần lớn có thể được thực hiện bằng **hai** tiện ích: hoặc là thực hiện các thao tác dễ dùng trong tiện ích **Enterprise Manager** hoặc là thực hiện các câu lệnh khó nhớ trong **Query Analyzer**. Tuy nhiên chúng tôi vẫn khuyến cáo các bạn nên sử dụng các lệnh trong Query Analyzer bởi vì chúng sẽ giúp các bạn làm quen với các lệnh trong **Transaction-SQL** là ngôn ngữ được sử dụng khi lập trình trong môi trường Microsoft SQL Server.



## Bài 4

# LẬP TRÌNH VỚI CƠ SỞ DỮ LIỆU

### Tóm tắt

Lý thuyết 6 tiết - Thực hành 8 tiết

Mục tiêu	Các mục chính	Bài tập bắt buộc	Bài tập làm thêm
<ul style="list-style-type: none"> <li>✓ Giúp cho học viên nắm vững các kiến thức và thao tác thành thạo với ngôn ngữ lập trình T-SQL.</li> <li>✓ Biến và các thao tác với biến</li> <li>✓ Các toán tử trong SQL</li> <li>✓ Nhắc lại các câu truy vấn dữ liệu</li> <li>✓ Giới thiệu các cấu trúc điều khiển và một số hàm thường dùng trong SQL</li> <li>✓ Khai báo và sử dụng biến kiểu CURSOR</li> </ul>	<ul style="list-style-type: none"> <li>I. Biến cục bộ</li> <li>II. Biến hệ thống</li> <li>III. Các toán tử</li> <li>IV. Các câu lệnh truy vấn dữ liệu</li> <li>V. Cấu trúc điều khiển</li> <li>VI. Sử dụng biến kiểu dữ liệu cursor</li> <li>VII. Các hàm thường dùng</li> </ul>	4.1 4.2 4.3 4.4	

# I. Biến cục bộ

Biến trong chương trình dùng để **lưu trữ các giá trị tạm thời** trong quá trình tính toán các xử lý bởi vì sau khi thoát khỏi chương trình hoặc tắt máy tính thì giá trị của các biến này sẽ không còn trong bộ nhớ nữa.

Thông thường mỗi một biến dùng để lưu trữ duy nhất một loại dữ liệu. Do đó khi một biến đã xác định để lưu trữ dữ liệu dạng ngày thì chúng ta **không thể** ép buộc biến lưu trữ các dữ liệu dạng khác như là: số hoặc chuỗi. Mỗi biến cần phải có một tên biến rõ ràng và **duy nhất trong một phạm vi** để khi cần tham chiếu, chúng ta sẽ chỉ định đến tên của biến đó. Trong Transaction–SQL có hai loại biến khác nhau: biến cục bộ và biến hệ thống.

## I.1. Khai báo biến cục bộ

Khai báo biến cục bộ là việc chỉ định cho hệ thống máy tính cấp phát một **vùng nhớ** bên trong bộ nhớ RAM (random access memory) của máy tính để chương trình có thể lưu trữ các giá trị tạm thời trong quá trình tính toán.

Trong Transaction–SQL việc sử dụng biến cần phải được khai báo **tường minh rõ ràng**, có nghĩa là bắt buộc các bạn cần phải khai báo biến cục bộ trước rồi sau đó mới được phép sử dụng. Điều này sẽ giúp các bạn hiểu rất rõ về việc khai báo và sử dụng các biến bên trong chương trình của mình.

Để khai biến cục bộ trong Transaction–SQL, chúng ta sẽ sử dụng lệnh **DECLARE** với cú pháp đầy đủ được mô tả như bên dưới.

Cú pháp:

```
DECLARE @Tên_biến Kiểu_dữ_liệu [, ...]
```

Trong đó:

- ✓ Tên biến: tên của biến được khai báo, tên biến luôn luôn bắt đầu bằng ký tự a–vòng (@). Thông thường tên biến phải duy nhất trong một phạm vi hoạt động.
- ✓ Kiểu dữ liệu: là các kiểu dữ liệu cơ bản của Microsoft SQL Server hoặc các kiểu dữ liệu do người dùng định nghĩa, dùng để chỉ định loại dữ liệu mà biến sẽ lưu trữ. Các kiểu dữ liệu text, ntext hoặc image không được chấp nhận trong việc khai báo biến.



### Ví dụ:

Để khai báo các biến dùng để lưu trữ giá trị tổng số lượng đặt hàng, họ tên nhà cung cấp, ngày xuất hàng. Chúng ta sử dụng lệnh **DECLARE** như sau:

```
DECLARE @Tongslat INT, @Hotenncc CHAR(50)  
DECLARE @Ngayxh DATETIME
```

Khác với một vài ngôn ngữ lập trình, chúng ta **không thể** gán giá trị khởi tạo cho biến lúc khai báo chúng. Do đó hành động kế tiếp mà chúng tôi muốn trình bày là việc gán giá trị cần lưu trữ vào bên trong biến.

## 1.2. Gán giá trị cho biến

Như chúng tôi đã giới thiệu ý nghĩa của biến cục bộ trong chương trình dùng để **lưu trữ các giá trị tạm thời** trong quá trình tính toán các xử lý. Do đó sau khi đã khai báo biến, việc kế tiếp là chúng ta có thể gán giá trị cần lưu trữ vào các biến này.

Để gán trị cần lưu trữ vào các biến, chúng ta sẽ sử dụng lệnh **SET** hoặc lệnh **SELECT** cùng với phép gán (=). Thông thường lệnh **SET** chỉ để gán các **giá trị cụ thể** hoặc các **biểu thức tính toán** hoặc **giá trị tính toán** từ các biến khác.



### Ví dụ:

Để gán giá trị là ngày 25/03/2002 vào biến ngày xuất hàng đã được khai báo theo thí dụ ở phần trên. Chúng ta sử dụng lệnh **SET** như sau:

```
DECLARE @Ngayxh DATETIME
SET @Ngayxh='2002-03-25'
```



### Chú ý:

Đối với kiểu dữ liệu dạng ngày trong Microsoft SQL Server thông thường chúng tôi sử dụng theo **định dạng yyyy-mm-dd** để gán giá trị vào biến hoặc vào trong cơ sở dữ liệu.

Ngược lại với lệnh **SET**, lệnh **SELECT** dùng để gán các giá trị được lấy ra hoặc tính toán từ dữ liệu của các cột bên trong các bảng dữ liệu. Ngoài ra trong cùng một lệnh **SELECT** cho phép cùng lúc đồng thời các bạn có thể gán các giá trị khác nhau từ các cột dữ liệu vào bên trong các biến khác nhau.



### Ví dụ:

Để tính ra tổng số lượng đặt hàng mà dữ liệu của nó được lấy từ cột **SLDAT** (số lượng đặt) trong bảng **CTDONDH**. Chúng ta sử dụng lệnh **SELECT** như sau:

```
DECLARE @Tongslat INT
SELECT @Tongslat = SUM(SLDAT)
FROM CTDONDH
```



### Ví dụ:

Để tính ra đồng thời giá trị số lượng đặt hàng thấp nhất và cao nhất. Chúng ta chỉ cần sử dụng duy nhất một lệnh **SELECT** như sau:

```
DECLARE @MinSlat INT, @MaxSlat INT
SELECT @MinSlat=MIN(SLDAT), @MaxSlat=MAX(SLDAT)
FROM CTDONDH
```

Cẩn thận khi sử dụng câu lệnh **SELECT** để gán giá trị cần lưu trữ vào các biến bởi vì trong trường hợp nếu câu lệnh **SELECT** trả về nhiều dòng dữ liệu thì chỉ có giá trị của **dòng dữ liệu sau cùng** mới được lưu trữ vào biến. Do đó muốn tránh những trường hợp này cách tốt nhất là chúng ta luôn đảm bảo dữ liệu của mình chỉ trả về duy nhất một dòng trong câu lệnh **SELECT**, thông thường các bạn có thể sử dụng mệnh đề **WHERE** để lọc dữ liệu theo đúng điều kiện mình cần hoặc kết hợp các hàm thống kê **MIN**, **MAX**, **SUM** để giới hạn số dòng trả về.

**Ví dụ:**

Để tính ra số lượng đặt hàng cao nhất của mặt hàng “Đầu DVD Hitachi 1 đĩa” có mã vật tư là “DD01”. Chúng ta sử dụng lệnh **SELECT** như sau:

```
DECLARE @MaxSlDat INT
SELECT @MaxSlDat=MAX(SLDAT)
FROM CTDONDH
WHERE MAVTU="DD01"
```

Hoặc:

```
DECLARE @MaxSlDat INT
SELECT TOP 1 @MaxSlDat=SLDAT
FROM CTDONDH
WHERE MAVTU="DD01"
ORDER BY SLDAT DESC
```

Câu lệnh **SELECT** thứ hai kết hợp các mệnh đề **ORDER BY DESC** dùng để sắp xếp dữ liệu giảm dần theo cột số lượng đặt và sau đó kết hợp thêm mệnh đề **TOP 1** để lấy ra thông tin của dòng đầu tiên. Khi đó dữ liệu của lệnh **SELECT** sẽ trả về một dòng có giá trị số lượng đặt hàng là lớn nhất.

Mặc dù Microsoft SQL Server có cơ chế **tự động** chuyển đổi kiểu nhưng chúng tôi không khuyến cáo các bạn sử dụng cách này. Do vậy khi khai báo biến chúng ta cần **chỉ đúng kiểu dữ liệu** mà biến sẽ lưu trữ.

**Ví dụ:**

Biến @TongSoDH bên dưới có kiểu không hợp lệ, tuy nhiên hệ thống vẫn thực hiện bình thường không thông báo lỗi. Biến @TongSoDH nên có kiểu là số nguyên INT.

```
DECLARE @TongSoDH VARCHAR(5)
SELECT @TongSoDH=COUNT(SODH)
FROM DONDH
PRINT @TongSoDH
```

### 1.3. Xem giá trị hiện hành của biến

Sau khi đưa các giá trị cần lưu trữ vào bên trong các biến, các bạn cũng có thể cần xem giá trị hiện hành mà biến đang lưu trữ là bao nhiêu. Để làm điều này chúng ta có thể sử dụng lệnh **PRINT** để in nội dung mà biến hiện đang lưu trữ ra màn hình. Cú pháp của lệnh **PRINT** được mô tả đầy đủ như sau:

Cú pháp:

```
PRINT @Tên_biến | Biểu_thức_chuỗi
```

Trong đó:

- ✓ Tên biến: tên của biến đã được khai báo trước đó mà chúng ta muốn xem giá trị hiện hành đang lưu trữ. Thông thường nếu kiểu dữ liệu của biến không phải là kiểu chuỗi (char hoặc varchar) thì chúng ta nên sử dụng các hàm CAST hoặc hàm CONVERT để chuyển về kiểu



dữ liệu chuỗi tương ứng. (xem ý nghĩa của các hàm này ở phần các hàm thường gần cuối bài này)

- ✓ Biểu thức chuỗi: là một biểu thức chuỗi văn bản cần in ra. Độ dài tối đa của chuỗi là 8.000 ký tự.



**Ví dụ:**

Để tính đồng thời giá trị số lượng đặt hàng thấp nhất và cao nhất, sau đó hiển thị kết quả ra màn hình cho chúng ta biết. Chúng ta sử dụng lệnh các *SELECT* và *PRINT* như sau:

```
DECLARE @MinSlDat INT, @MaxSlDat INT
SELECT @MinSlDat=MIN(SLDAT), @MaXSlDat=MAX(SLDAT)
FROM CTDONDH
PRINT "Số lượng thấp nhất là : "
PRINT @MinSlDat
PRINT "Số lượng cao nhất là : " + CONVERT(VARCHAR(10), @MaxSlDat)
```

Trong thí dụ này chúng tôi có sử dụng hàm *CONVERT* dùng để chuyển đổi kiểu dữ liệu của biến *@MaxSlDat* từ kiểu số nguyên (*INT*) sang kiểu chuỗi (*VARCHAR*) và toán tử cộng (+) dùng để nối các chuỗi lại với nhau trong câu lệnh *PRINT* thứ hai. Cú pháp đầy đủ của hàm *CONVERT* xin các bạn xem ở phần các hàm thường dùng ở cuối bài này.

## 1.4. Phạm vi hoạt động của biến

Trong Transaction–SQL phạm vi hoạt động của biến chỉ nằm bên trong một **thủ tục nội tại** (stored procedure) hoặc một **lô** (batch) chứa các câu lệnh mà biến đã được khai báo bên trong đó.

Khái niệm về lô được xem như là một **nhóm tập hợp** của một hoặc nhiều câu lệnh Transaction–SQL sẽ được biên dịch đồng thời cùng lúc tại Microsoft SQL Server và sau đó hệ thống sẽ thực thi các lệnh này ngay sau khi đã biên dịch thành công. Để chỉ định một lô chúng ta sử dụng từ khóa **GO**. Nên nhớ rằng đây chỉ là một **từ khóa** để chỉ định kết thúc một lô.



**Ví dụ:**

Chúng ta có một lô trong Transaction–SQL như sau:

```
SELECT * FROM NHACC
ORDER BY Tennhacc
SELECT * FROM VATTU
ORDER BY Tenvtu DESC
GO
```

Do các câu lệnh trong một lô sẽ được biên dịch tại Microsoft SQL Server vì thế khi có ít nhất một lệnh bên trong lô có lỗi về cú pháp lúc biên dịch thì sẽ **không** có câu lệnh nào được thực thi bên trong lô đó.



**Ví dụ:**

Như thí dụ ở trên nhưng nếu chúng ta có bổ sung thêm lệnh INSERT mà cú pháp lệnh này bị sai (thiếu từ khóa VALUES) thì các lệnh SELECT bên trong lô này sẽ không được thực hiện.

```
SELECT * FROM NHACC
ORDER BY Tennhacc
INSERT INTO NHACC
("C01", "Trường Đức Cường", "12 Lê Lợi", "0918.123456")
SELECT * FROM VATTU
ORDER BY Tenvtu DESC
GO
```

Hệ thống xuất hiện thông báo lỗi

```
Server: Msg 170, Level 15, State 1, Line 4
Line 4: Incorrect syntax near 'C01'.
```

Tuy nhiên đối với các lỗi khi thực hiện (run-time) bên trong một lô nếu trường hợp các lỗi vi phạm ràng buộc toàn vẹn dữ liệu thì hệ thống Microsoft SQL Server chỉ ngưng lại câu lệnh gây ra lỗi và thực hiện tiếp các lệnh kế tiếp bên trong lô.

**Ví dụ:**

Vẫn tiếp thí dụ trên nhưng nếu chúng ta có bổ sung từ khóa VALUES trong lệnh INSERT thì lần này mặc dù chỉ có mỗi lệnh INSERT vi phạm ràng buộc toàn vẹn (giả sử trùng khóa chính ở cột mã nhà cung cấp) nhưng các lệnh SELECT bên trong lô này vẫn được thực hiện bình thường.

```
SELECT * FROM NHACC ORDER BY Tennhacc

INSERT INTO NHACC
VALUES ("C01", "Trường Đức Cường", "12 Lê Lợi", "0918.123456")

SELECT * FROM VATTU ORDER BY Tenvtu DESC
GO
```

Qua các thí dụ ở trên, chúng tôi hy vọng rằng các bạn đã hiểu rõ khái niệm về lô các lệnh trong Transaction-SQL. Ghi nhớ rằng phạm vi của biến chỉ nằm bên trong một lô mà biến đã được khai báo trước đó.

**Ví dụ:**

Khai báo một biến cục bộ kiểu ngày dùng để lưu ngày đặt hàng gần đây nhất dựa trên dữ liệu của bảng DONDH tại cột Ngaydh. Sau cùng in giá trị đó ra màn hình.

```
DECLARE @Ngaydhgn DATETIME
SELECT @Ngaydhgn=MAX(NGAYDH)
FROM DONDH
GO
PRINT "Ngày đặt hàng gần nhất: " + CONVERT(CHAR(12), @Ngaydhgn)
GO
```



Hệ thống sẽ hiển thị thông báo lỗi nếu chúng ta chèn thêm từ khóa **GO** ở giữa hai lệnh **SELECT** và **PRINT**. Bởi vì khi đó các lệnh này được chia ra làm hai lô và lô thứ hai sẽ không hiểu biến @Ngaydthgn đã được khai báo trong lô thứ nhất.



**Chú ý:**

Đối với các lệnh **CREATE** như là: **CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER, CREATE VIEW** không được phép kết hợp với các lệnh khác trong cùng một lô.

## II. Biến hệ thống

Các biến hệ thống bên trong Microsoft SQL Server luôn bắt đầu bằng **hai** chữ a-vòng (@@) và giá trị mà chúng đang lưu trữ là do **hệ thống Microsoft SQL Server cung cấp**, người lập trình không thể can thiệp trực tiếp để gán giá trị vào các biến hệ thống.

### II.1. Ý nghĩa sử dụng

Khi lập trình trong Transaction-SQL chúng ta vẫn thường xuyên sử dụng giá trị của các biến hệ thống để giúp kiểm tra kết quả thực hiện các lệnh trước đó. Thật ra các biến hệ thống chính là các hàm đã được Microsoft SQL Server xây dựng bên trong và người lập trình chỉ việc tra cứu các giá trị hiện thời của nó để thực hiện các xử lý tiếp sau đó.

Một vài biến hệ thống có giá trị thay đổi thường xuyên tùy thuộc vào các câu lệnh thực hiện trong Transaction-SQL, tuy nhiên bên cạnh đó cũng có một vài biến hệ thống có giá trị rất ít khi thay đổi.



**Ví dụ:**

Biến @@ROWCOUNT trả về tổng số mẫu tin được thực hiện trong câu lệnh gần nhất. Biến hệ thống này có giá trị thay đổi rất thường xuyên.

```
SELECT * FROM NHACC
SELECT @@ROWCOUNT
UPDATE VATTU
    SET PHANTRAM = PHANTRAM + 5
    WHERE MAVTU LIKE "TV%"
SELECT @@ROWCOUNT
```

Giá trị của biến hệ thống @@ROWCOUNT ở lần thứ nhất sẽ trả về tổng số mẫu tin hiện đang có trong bảng NHACC, và giá trị của biến hệ thống @@ROWCOUNT ở lần thứ hai sẽ trả về tổng số mẫu tin có các mã vật tư bắt đầu bằng chữ "TV" trong bảng VATTU.



**Ví dụ:**

Biến @@VERSION trả về phiên bản, ngày của sản phẩm Microsoft SQL Server và loại CPU, hệ điều hành của máy chủ cài đặt Microsoft SQL Server. Biến hệ thống này có giá trị rất ít thay đổi, giá trị của nó chỉ thay đổi khi nào các bạn nâng cấp hệ thống Microsoft SQL Server.

```
PRINT @@VERSION
```

## II.2. Một vài biến hệ thống thường dùng

Trong phần này chúng tôi muốn giới thiệu cho các bạn biết ý nghĩa của **một vài biến hệ thống** thường dùng trong Microsoft SQL Server. Việc sử dụng giá trị của các biến này trong khi lập trình Transaction-SQL sẽ được trình bày trong các thí dụ ở các phần kế tiếp.

Tên biến	Kiểu trả về	Dùng để trả về
<i>Connections</i>	Số nguyên	Tổng số các kết nối vào Microsoft SQL Server từ khi nó được khởi động.
<i>Error</i>	Số nguyên	Số mã lỗi của câu lệnh thực hiện gần nhất. Khi một câu lệnh thực hiện thành công thì biến này có giá trị là 0.
<i>Fetch_Status</i>	Số nguyên	Trạng thái của việc đọc dữ liệu trong bảng theo cơ chế từng mẫu tin (cursor). Khi đọc dữ liệu của mẫu tin thành công thì biến này có giá trị là 0.
<i>Language</i>	Chuỗi	Tên ngôn ngữ mà hệ thống Microsoft SQL Server đang sử dụng. Mặc định là US_English
<i>RowCount</i>	Số nguyên	Tổng số mẫu tin được tác động vào câu lệnh truy vấn gần nhất.
<i>ServerName</i>	Chuỗi	Tên của máy tính cục bộ được cài đặt Microsoft SQL Server.
<i>ServiceName</i>	Chuỗi	Tên dịch vụ chạy kèm theo bên dưới Microsoft SQL Server.
<i>Version</i>	Chuỗi	Phiên bản, ngày của sản phẩm Microsoft SQL Server và loại CPU, hệ điều hành của máy chủ cài Microsoft SQL Server.

## III. Các toán tử

Việc sử dụng các toán tử được kết hợp vào bên trong các mệnh đề **WHERE**, **HAVING** trong các câu lệnh truy vấn hoặc bên trong các cấu trúc điều khiển **IF**, **WHILE** mà chúng tôi sẽ giới thiệu trong bài này.

### III.1. Toán tử số học

Toán tử số học được phép sử dụng trong việc **tính toán** các phép tính: **cộng, trừ, nhân, chia và chia lấy phần dư**. Các biểu thức, giá trị tính toán phải có kiểu dữ liệu là số khi thực hiện các toán tử này. Thứ tự ưu tiên của các toán tử số học là: nhân, chia và chia lấy phần dư trước và sau đó là đến cộng và trừ sau. Tuy nhiên theo chúng tôi thì các bạn nên chỉ định tường minh thứ tự thực hiện các toán tử số học bằng cách đưa vào các dấu ngoặc đơn trong một biểu thức tính toán.



#### Ví dụ:

Hai câu lệnh bên dưới sẽ trả về kết quả như nhau:

```
SELECT 10 + 15*70/100
```

Và

```
SELECT 10 + ( (15*70)/100 )
```

*Tuy nhiên với câu lệnh thứ hai chúng ta thấy rõ ràng hơn bởi vì phép nhân bên trong cùng của dấu ngoặc đơn sẽ thực hiện trước, sau đó là tới phép chia và cuối cùng mới là phép cộng.*



Toán tử số học được phép sử dụng cho các kiểu dữ liệu số như là: int, smallint, tinyint, numeric, decimal, float, real, money và smallmoney. Tuy nhiên đối với phép tính chia lấy phần dư chỉ được sử dụng cho các kiểu dữ liệu số nguyên như là: int, smallint và tinyint.

Bảng bên dưới mô tả ký hiệu và ý nghĩa của các toán tử số học.

Ký hiệu	Ý nghĩa
+	Thực hiện phép cộng hai số.
-	Thực hiện phép trừ hai số.
*	Thực hiện phép nhân hai số.
/	Thực hiện phép chia hai số.
%	Thực hiện phép chia lấy phần dư.



**Ví dụ:**

Thể hiện phép chia lấy phần dư của hai số nguyên 15 và 6. Chúng ta sử dụng lệnh như sau:

```
SELECT 15%6
```

Kết quả trả về là: 3.

## III.2. Toán tử nối chuỗi

Toán tử nối chuỗi được phép sử dụng trong việc **kết nối các chuỗi** rời rạc thành một chuỗi liên tục. Các biểu thức, giá trị khi nối chuỗi **phải** có kiểu dữ liệu là chuỗi hoặc phải được chuyển đổi thành kiểu dữ liệu chuỗi trước khi thực hiện toán tử này. Ký hiệu của toán tử nối các chuỗi lại với nhau là dấu cộng (+).



**Ví dụ:**

Để nối hai chuỗi "Hello" và "The World!" lại thành chuỗi "Hello The World!". Chúng ta sử dụng lệnh như sau:

```
SELECT "Hello" + " " + "The World!"
```



**Ví dụ:**

Cho biết ngày đặt hàng của đơn đặt hàng số D007 là bao nhiêu. Chúng ta sử dụng lệnh như sau:

```
SELECT "Ngày đặt hàng D007 là: "
      + CAST(NGAYDH AS CHAR(11))
FROM DONDH
WHERE SODH="D007"
```

Trong thí dụ này chúng ta bắt buộc phải chuyển đổi giá trị dữ liệu của cột ngày đặt hàng trong bảng DONDH từ kiểu dữ liệu ngày sang kiểu dữ liệu chuỗi trước khi thực hiện toán tử nối chuỗi.



### III.3. Toán tử so sánh

Toán tử so sánh được phép sử dụng để thực hiện các **phép so sánh** như là: bằng, khác, lớn hơn, nhỏ hơn... cho các biểu thức cần được so sánh. Giá trị trả về của việc so sánh sẽ là đúng hoặc sai tùy thuộc vào biểu thức điều kiện mà chúng ta đưa ra để so sánh.

Các toán tử so sánh có thể được sử dụng trên nhiều kiểu dữ liệu khác nhau như là: kiểu số, kiểu chuỗi hoặc kiểu ngày. Thông thường các biểu thức so sánh sẽ được lồng vào các mệnh đề **WHERE** hoặc **HAVING** của các câu lệnh truy vấn.

Bảng bên dưới mô tả ký hiệu và ý nghĩa của các toán tử so sánh.

Ký hiệu	Ý nghĩa
=	Thực hiện phép so sánh bằng.
>	Thực hiện phép so sánh lớn hơn.
<	Thực hiện phép so sánh nhỏ hơn.
>=	Thực hiện phép so sánh lớn hơn hoặc bằng.
<=	Thực hiện phép so sánh nhỏ hơn hoặc bằng.
<>	Thực hiện phép so sánh khác.
!=	Thực hiện phép so sánh khác.
/>	Thực hiện phép so sánh không lớn hơn.
/<	Thực hiện phép so sánh không nhỏ hơn.



#### Ví dụ:

Sử dụng toán tử so sánh bằng để lọc ra các vật tư hiện đang có trong bảng VATTU với đơn vị tính là "Bộ". Chúng ta sử dụng lệnh như sau:

```
SELECT * FROM VATTU
WHERE DVTINH="Bộ"
```

Nhận xét thấy rằng trong thí dụ này toán tử so sánh bằng được thực hiện trên kiểu **dữ liệu chuỗi** và kết hợp trong mệnh đề **WHERE**.



#### Ví dụ:

Sử dụng toán tử so sánh lớn hơn để lọc ra các phiếu xuất hàng có tổng trị giá lớn hơn 3,000,000 VNĐ. Chúng ta sử dụng lệnh như sau:

```
SELECT SOPX, SUM(SLXUAT*DGXUAT) AS TONGTG
FROM CTPXUAT
GROUP BY SOPX
HAVING SUM(SLXUAT*DGXUAT)>3000000
```

Nhận xét thấy rằng trong thí dụ này toán tử so sánh lớn hơn được thực hiện trên kiểu **dữ liệu số** và kết hợp trong mệnh đề **HAVING**.

### III.4. Toán tử luận lý

Toán tử luận lý được phép sử dụng để thực hiện việc kết hợp nhiều biểu thức so sánh đơn lẻ thành một biểu thức so sánh chung. Có ba toán tử luận lý rất quan trọng thường dùng là: **AND**, **OR** và **NOT**. Theo chúng tôi khi sử dụng toán tử luận lý các bạn nên **thêm vào các dấu ngoặc**

**đơn** cần thiết để giúp cho người đọc dễ hiểu bởi vì nó làm cho câu lệnh rõ ràng hơn.



**Ví dụ:**

Hiển thị danh sách các vật tư trong bảng VATTU thỏa điều kiện như sau:

Đơn vị tính là “Bộ” và phần trăm lớn hơn 10.

Hoặc đơn vị tính là “Cái” và phần trăm lớn hơn 20.

Chúng ta sử dụng lệnh như sau:

```
SELECT * FROM VATTU
WHERE (DVTINH="Bộ" AND PHANTRAM>10)
      OR (DVTINH="Cái" AND PHANTRAM>20)
```

Trong thí dụ này cả hai toán tử **AND** và **OR** được sử dụng kết hợp trong các biểu thức so sánh.

## IV. Các câu lệnh truy vấn dữ liệu

Lập trình trong Transaction–SQL chủ yếu là các bạn sẽ **sử dụng các câu lệnh truy vấn** và kết hợp với **cấu trúc điều khiển** thích hợp cùng các biến đã được khai báo để thực hiện các hành động thích hợp cho việc cập nhật dữ liệu vào bên trong cơ sở dữ liệu.

Trong phần này chúng tôi sẽ trình bày các lệnh truy vấn thường dùng như là: thêm dòng dữ liệu mới vào bảng, xóa các dòng dữ liệu đang có trong bảng, thay đổi giá trị các cột dữ liệu bên trong bảng, chọn lựa các dòng dữ liệu từ các bảng cần thiết. Tuy nhiên đối với cú pháp đầy đủ của lệnh **SELECT** rất phức tạp và khó nhớ, vì thế chúng tôi sẽ trình bày riêng rẽ theo từng thành phần khác nhau nhằm giúp các bạn dễ hiểu, dễ nhớ.

### IV.1. Lệnh SELECT FROM

Ý nghĩa hoạt động của câu lệnh **SELECT FROM** dùng để cho phép chúng ta có thể **chọn lựa các dữ liệu** cần thiết từ một hoặc nhiều bảng có quan hệ bên trong một cơ sở dữ liệu.

Câu lệnh này thường được dùng rất nhiều bên trong Transaction–SQL. Tuy nhiên cũng giống như phần trình bày cú pháp của lệnh **CREATE TABLE** trước đây, cuối cùng các bạn vẫn có thể sử dụng cùng lúc đồng thời đầy đủ các mệnh đề của lệnh **SELECT FROM**.

#### IV.1.1. Lệnh SELECT FROM đơn giản

Với cú pháp **SELECT FROM** bên dưới cho phép các bạn có thể chọn ra **dữ liệu của các cột** hiện có bên trong một bảng. Với cú pháp này tên các cột phải được chỉ định rõ ràng.

Cú pháp:

```
SELECT Danh_sách_các_cột
FROM Tên_bảng
```

Trong đó:

- ✓ Danh sách các cột: là tên của các cột hiện đang có bên trong bảng mà chúng ta cần lấy dữ liệu.

- ✓ Tên bảng: tên bảng cần hiển thị dữ liệu.

**Ví dụ:**

Để hiển thị thông tin của các vật tư trong bảng VATTU gồm những cột như sau: mã vật tư, tên vật tư. Chúng ta thực hiện câu lệnh **SELECT FROM** như sau:

```
SELECT MAVTU, TENVTU
FROM VATTU
```

Kết quả truy vấn trả về

```
MAVTU TENVTU
-----
TG01  Bia lon Tiger
DD01  Đầu DVD Hitachi 1 đĩa
DD02  Đầu DVD Hitachi 3 đĩa
VD01  Đầu VCD Sony 1 đĩa
VD02  Đầu VCD Sony 3 đĩa
TV14  Tivi Sony 14 inches
TV21  Tivi Sony 21 inches
TV29  Tivi Sony 29 inches
TL15  Tủ lạnh Sanyo 150 lit
TL90  Tủ lạnh Sanyo 90 lit

(10 row(s) affected)
```

**IV.1.2. Mệnh đề sắp xếp dữ liệu**

Với cú pháp **SELECT FROM** bên dưới kết hợp mệnh đề **ORDER BY** cho phép các bạn có thể lấy dữ liệu của các cột bên trong một bảng, sau đó **sắp xếp lại dữ liệu** theo thứ tự chỉ định là tăng hoặc giảm.

Cú pháp:

```
SELECT Danh_sách_các_cột
FROM Tên_bảng
ORDER BY Tên_cột_sx [DESC] [, ...]
```

Trong đó:

- ✓ Tên cột sắp xếp: là tên cột được sắp xếp dữ liệu. Thứ tự ưu tiên sắp xếp các cột dữ liệu từ trái sang phải và mặc định theo thứ tự tăng dần.
- ✓ Từ khóa DESC: dùng chỉ thay đổi thứ tự sắp xếp là giảm dần. Mặc định thứ tự sắp xếp là tăng dần.

**Ví dụ:**

Để hiển thị thông tin của các vật tư trong bảng VATTU gồm những cột: mã vật tư, tên vật tư, phần trăm có sắp xếp dữ liệu theo cột tỷ lệ phần trăm tăng dần. Chúng ta thực hiện câu lệnh **SELECT FROM** như sau:

```
SELECT MAVTU, TENVTU, PHANTRAM
FROM VATTU
ORDER BY PHANTRAM
```

Kết quả truy vấn trả về

MAVTU	TENVTU	PHANTRAM
TV21	Tivi Sony 21 inches	15
TV29	Tivi Sony 29 inches	15
TL90	Tủ lạnh Sanyo 90 lit	20
TV14	Tivi Sony 14 inches	20
TL15	Tủ lạnh Sanyo 150 lit	25
VD01	Đầu VCD Sony 1 đĩa	30
VD02	Đầu VCD Sony 3 đĩa	30
DD01	Đầu DVD Hitachi 1 đĩa	40
DD02	Đầu DVD Hitachi 3 đĩa	40
TG01	Bia lon Tiger	60

(10 row(s) affected)

**IV.1.3. Mệnh đề chọn các dòng dữ liệu**

Với cú pháp **SELECT FROM** bên dưới kết hợp mệnh đề **WHERE** cho phép các bạn có thể **lọc các dòng dữ liệu** bên trong một bảng phải thỏa điều kiện đưa ra trong mệnh đề **WHERE**.

Cú pháp:

```
SELECT [DISTINCT] [TOP Số [PERCENT]]
Danh_sách_các_cột
FROM Tên_bảng
WHERE Điều_kiện_lọc
[ ORDER BY Tên_cột [DESC] [, ...] ]
```

Trong đó:

- ✓ Từ khóa **DISTINCT**: dùng để chỉ định truy vấn chỉ chọn ra các dòng dữ liệu duy nhất, không trùng lặp dữ liệu.
- ✓ Từ khóa **TOP**: dùng để chỉ định truy vấn chỉ chọn ra chính xác bao nhiêu dòng dữ liệu đầu tiên. Nếu có thêm từ khóa **PERCENT** đi kèm theo thì truy vấn chỉ chọn ra bao nhiêu phần trăm mẫu tin đầu tiên, lúc bấy giờ con số mà các bạn chỉ định phải nằm trong phạm vi từ 0 đến 100. Thông thường khi sử dụng từ khóa **TOP** thì chúng ta sẽ kết hợp mệnh đề **ORDER**



BY để sắp xếp lại dữ liệu theo một thứ tự nào đó.

- ✓ Điều kiện lọc: là điều kiện chỉ định việc lọc ra các mẫu tin bên trong bảng. Thông thường là một biểu thức luận lý.



**Ví dụ:**

Để hiển thị toàn bộ thông tin của các vật tư trong bảng VATTU sao cho chỉ chọn ra các vật tư có đơn vị tính là "Cái". Chúng ta thực hiện câu lệnh **SELECT FROM** như sau:

```
SELECT *
FROM VATTU
WHERE DVTINH="Cái"
```

Ký hiệu \* trong thí dụ này là đại diện cho tất cả các cột có bên trong bảng. Kết quả truy vấn trả về

Mavtu	Tenvtu	DvTinh	Phantram
TL15	Tủ lạnh Sanyo 150 lit	Cái	25
TL90	Tủ lạnh Sanyo 90 lit	Cái	20
TV14	Tivi Sony 14 inches	Cái	20
TV21	Tivi Sony 21 inches	Cái	15
TV29	Tivi Sony 29 inches	Cái	15

(5 row(s) affected)



**Ví dụ:**

Giống như thí dụ trên nhưng chúng ta chỉ muốn chọn ra vật tư đầu tiên có tỷ lệ phần trăm cao nhất. Chúng ta thực hiện câu lệnh **SELECT FROM** có kết hợp mệnh đề **TOP** như sau:

```
SELECT TOP 1 *
FROM VATTU
WHERE DVTINH="Cái"
ORDER BY PHANTRAM DESC
```

Kết quả truy vấn trả về

Mavtu	Tenvtu	DvTinh	Phantram
TL15	Tủ lạnh Sanyo 150 lit	Cái	25

(1 row(s) affected)

Đối với các người sử dụng ngôn ngữ SQL cũ trước đây, mệnh đề **WHERE** còn giúp họ có thể liên kết dữ liệu của nhiều bảng có quan hệ trong các truy vấn lấy dữ liệu từ nhiều bảng khác nhau.

Cú pháp:

```
SELECT Danh_sách_các_cột
FROM Tên_bảng1, Tên_bảng2
WHERE Mệnh_đề_liên_kết
```

Trong đó:

- ✓ Mệnh đề liên kết: thông thường dùng để chỉ định các cột có quan hệ chung của giữa hai bảng tham chiếu trong truy vấn, có dạng như sau:

**Tên\_bảng1.Tên\_cột = Tên\_bảng1.Tên\_cột**



**Ví dụ:**

Để hiển thị thông tin của các đơn đặt hàng trong bảng *DONDH* kèm theo cột họ tên của nhà cung cấp tương ứng trong bảng *NHACC* và sắp xếp dữ liệu hiển thị theo thứ tự mã nhà cung cấp tăng dần. Chúng ta thực hiện câu lệnh **SELECT FROM** như sau:

```
SELECT NCC.MANHACC, TENNHACC, SODH
      FROM DONDH DH, NHACC NCC
      WHERE DH.MANHACC=NCC.MANHACC
      ORDER BY NCC.MANHACC
```

Kết quả truy vấn trả về

	MANHACC	TENNHACC	SODH
C01	Lê Minh Trí		D002
C02	Trần Minh Thạch		D003
C02	Trần Minh Thạch		D005
C02	Trần Minh Thạch		D007
C03	Nguyễn Hồng Phương		D001
C05	Lưu Nguyệt Quế		D004
C05	Lưu Nguyệt Quế		D006

(7 row(s) affected)

Nhận xét rằng trong thí dụ trên hai bảng *DONDH* và *NHACC* có chung cột quan hệ là *MANHACC* sẽ được sử dụng trong mệnh đề **WHERE**. Do cột *MANHACC* nằm trong hai bảng *DONDH* và *NHACC* vì thế chúng ta cần phải chỉ định rõ ràng là lấy *MANHACC* trong bảng *NHACC* bằng cách ghi **NCC.MANHACC** sau mệnh đề **SELECT**.

#### IV.1.4. Mệnh đề nhóm dữ liệu

Với cú pháp **SELECT FROM** bên dưới kết hợp mệnh đề **GROUP BY** cho phép các bạn có thể **nhóm dữ liệu** của các dòng bên trong một bảng và được phép sử dụng các hàm thống kê đi kèm theo để tính toán các dữ liệu có tính chất thống kê tổng hợp. Thông thường sau khi nhóm dữ liệu, chúng ta nên sắp xếp lại dữ liệu để hiển thị theo một thứ tự nào đó. Do vậy chúng ta sẽ sử dụng mệnh đề **ORDER BY** sau mệnh đề **GROUP BY**.

Cú pháp:

```
SELECT Danh_sách_các_cột | Hàm_thống_kê AS Bí_danh
      FROM Tên_bảng
      [ WHERE Điều_kiện_lọc ]
      GROUP BY Danh_sách_cột_nhóm
      [ ORDER BY Tên_cột [DESC] [, ...] ]
```

Trong đó:



- ✓ Hàm thống kê: là tên của các hàm thống kê và các tham số tương ứng dùng để tính tổng (SUM), tính giá trị thấp nhất (MIN), tính giá trị cao nhất (MAX), đếm các mẫu tin (COUNT), tính giá trị trung bình (AVG) của các dữ liệu bên trong bảng.
- ✓ Bí danh: là tiêu đề mới của các cột tính toán. Các tiêu đề này chỉ có hiệu lực lúc hiển thị dữ liệu trong câu lệnh truy vấn mà không làm ảnh hưởng đến cấu trúc bên trong của bảng.
- ✓ Danh sách cột nhóm dữ liệu: là danh sách tên các cột được nhóm dữ liệu để tính toán.

**Ví dụ:**

Để thống kê tổng số đơn đặt hàng mà công ty đã đặt hàng theo từng nhà cung cấp và sắp xếp dữ liệu hiển thị theo thứ tự tổng số đơn đặt hàng tăng dần. Chúng ta thực hiện câu lệnh **SELECT FROM** như sau:

```
SELECT MANHACC, COUNT(*) AS TONG_SODH
FROM DONDH
GROUP BY MANHACC
ORDER BY TONG_SODH
```

Kết quả truy vấn trả về

```
MANHACC TONG_SODH
-----
C01      1
C03      1
C05      2
C02      3
(4 row(s) affected)
```

**IV.1.5. Mệnh đề lọc dữ liệu sau khi đã nhóm**

Với cú pháp **SELECT FROM** bên dưới kết hợp mệnh đề **HAVING** cho phép các bạn có thể **lọc lại dữ liệu sau khi đã nhóm dữ liệu** của các dòng bên trong một bảng. Khác với mệnh đề **WHERE** dùng để lọc các dòng dữ liệu hiện đang có bên trong bảng, mệnh đề **HAVING** chỉ được phép sử dụng đi kèm theo mệnh đề **GROUP BY** dùng để lọc lại dữ liệu sau khi đã nhóm. Điều này có nghĩa là mệnh đề **HAVING** chỉ được dùng kèm với mệnh đề **GROUP BY**.

Cú pháp:

```
SELECT Danh_sách_các_cột | Hàm_thống_kê AS Bí_danh
FROM Tên_bảng
[ WHERE Điều_kiện_lọc ]
GROUP BY Danh_sách_cột_nhóm
HAVING Điều_kiện_lọc_nhóm
[ ORDER BY Tên_cột [DESC] [, ...] ]
```

Trong đó:

- ✓ Điều kiện lọc nhóm: là điều kiện dùng để lọc lại dữ liệu sau khi đã nhóm dữ liệu. Thông thường là các biểu thức luận lý.

**Ví dụ:**

Theo thí dụ trên nhưng chúng ta chỉ cần lọc ra những nhà cung cấp có mã bắt đầu bằng chữ "C" và tổng số các đơn đặt hàng lớn hơn 1 sau khi đã tính toán dữ liệu theo nhóm. Chúng ta thực hiện câu lệnh **SELECT FROM** như sau:

```
SELECT MANHACC, COUNT(*) AS TONG_SODH
FROM DONDH
WHERE MANHACC LIKE "C%"
GROUP BY MANHACC
HAVING COUNT(*)>1
ORDER BY TONG_SODH
```

Kết quả truy vấn trả về

```
MANHACC TONG_SODH
-----
C05      2
C02      3
(2 row(s) affected)
```

Trong thí dụ này các bạn thấy rằng việc lọc dữ liệu được chia ra ở hai mệnh đề khác nhau. Thứ nhất mệnh đề **WHERE MANHACC LIKE "C%"** dùng để lọc ra các mẫu tin trong bảng DONDH sao cho mã nhà cung cấp phải bắt đầu bằng chữ "C", thứ hai mệnh đề **HAVING COUNT(\*)>1** dùng để lọc lại các nhà cung cấp nào có tổng số các đơn đặt hàng lớn hơn 1 sau khi đã nhóm để tính ra tổng số các đơn đặt hàng theo từng nhà nhà cung cấp.

Mệnh đề liên kết dữ liệu trong hai bảng

Với cú pháp **SELECT FROM** bên dưới kết hợp mệnh đề **JOIN** cho phép các bạn liên kết **hai bảng có quan hệ với nhau** để lấy ra các dữ liệu chung. Điểm quan trọng giữa những bảng này phải có các cột quan hệ chung nhau và thứ tự quan hệ khi chúng ta chỉ định giữa các bảng cũng sẽ làm ảnh hưởng đến kết quả của truy vấn.

Cú pháp:

```
SELECT Danh_sách_các_cột
FROM Tên_bảng
INNER {LEFT | RIGHT | FULL [OUTER]}
JOIN Tên_bảng_quan_hệ
ON Điều_kiện_quan_hệ
```

Trong đó:

- ✓ Từ khóa **INNER JOIN**: dùng để chỉ định việc so sánh giá trị trong các cột của các bảng là tương đương (dữ liệu đều có ở cả hai bảng). Hệ thống sẽ trả về các mẫu tin thỏa điều kiện quan hệ ở cả hai bảng.
- ✓ Từ khóa **LEFT | RIGHT | FULL**: dùng để chỉ định việc so sánh giá trị các cột của bảng được ưu tiên cho mỗi quan hệ bên nhánh trái, phải hoặc cả hai bên. Việc thay đổi thứ tự ưu tiên này sẽ làm ảnh hưởng đến kết quả truy vấn.



- ✓ Từ khóa OUTER: được dùng kết hợp cho các quan hệ ưu tiên dữ liệu. Tuy nhiên chúng ta được phép bỏ đi khi sử dụng loại quan hệ ưu tiên LEFT | RIGHT | FULL.
- ✓ Điều kiện quan hệ: là một biểu thức so sánh bằng, chỉ ra tên các cột quan hệ giữa hai bảng gần giống như biểu thức sau mệnh đề WHERE dùng để liên kết hai bảng.

**Ví dụ:**

Để hiển thị thông tin của các đơn đặt hàng trong bảng DONDH kèm theo cột họ tên nhà cung cấp tương ứng trong bảng NHACC và sắp xếp dữ liệu theo cột mã nhà cung cấp tăng dần.

Chúng ta thực hiện câu lệnh **SELECT FROM** như sau:

```
SELECT NCC.MANHACC, TENNHACC, SODH
FROM DONDH DH
INNER JOIN NHACC NCC
ON DH.MANHACC=NCC.MANHACC
ORDER BY NCC.MANHACC
```

Kết quả truy vấn trả về

MANHACC	TENNHACC	SODH
C01	Lê Minh Trí	D002
C02	Trần Minh Thạch	D003
C02	Trần Minh Thạch	D005
C02	Trần Minh Thạch	D007
C03	Nguyễn Hồng Phương	D001
C05	Lưu Nguyệt Quế	D004
C05	Lưu Nguyệt Quế	D006

(7 row(s) affected)

**Chú ý:**

Trong các truy vấn lấy dữ liệu từ nhiều bảng có quan hệ, chúng ta phải bắt buộc sử dụng định dạng: **tên\_bảng.tên\_cột** cho các **cột trùng tên** giữa các bảng. Theo thí dụ trên thì cột MANHACC xuất hiện ở cả hai bảng DONDH và NHACC do vậy chúng ta phải ghi là: NCC.MANHACC và DH.MANHACC.

Ngoài ra chúng ta cũng có thể sử dụng khái niệm **bí danh** cho các bảng nhằm để làm ngắn gọn câu lệnh mỗi khi tham chiếu đến tên các bảng. Theo thí dụ trên bảng DONDH có bí danh là DH, bảng NHACC có bí danh là NCC. Các bí danh này các bạn có thể đặt tên tùy thích, tuy nhiên chúng ta nên đặt ngắn gọn, gợi nhớ và không được phép trùng nhau bên trong một câu lệnh truy vấn.

**Ví dụ:**

Giống như thí dụ trên nhưng chúng tôi muốn rằng hiển thị ra tất cả các nhà cung cấp hiện có trong bảng NHACC. Để làm được điều này chúng ta thấy rằng thứ tự quan hệ phải ưu tiên dữ liệu bên bảng NHACC. Chúng ta thực hiện câu lệnh **SELECT FROM** như sau:

```
SELECT NCC.MANHACC, TENNHACC, SODH
      FROM DONDH DH
      RIGHT OUTER JOIN NHACC NCC
      ON DH.MANHACC=NCC.MANHACC
      ORDER BY NCC.MANHACC
```

Kết quả truy vấn trả về

	MANHACC	TENNHACC	SODH
C01	Lê Minh Trí		D002
C02	Trần Minh Thạch		D003
C02	Trần Minh Thạch		D005
C02	Trần Minh Thạch		D007
C03	Nguyễn Hồng Phương		D001
C04	Trương Nhật Thắng		NULL
C05	Lưu Nguyệt Quế		D004
C05	Lưu Nguyệt Quế		D006
C07	Cao Minh Trung		NULL

(9 row(s) affected)

Nhận xét thấy rằng sẽ có thêm hai nhà cung cấp mới trong kết quả truy vấn sau khi thay đổi thứ tự ưu tiên quan hệ dữ liệu cho bảng NHACC (**RIGHT JOIN** bởi vì bảng NHACC nằm bên phải trong quan hệ của bảng DONDH và NHACC). Tuy nhiên giá trị dữ liệu tại cột số đơn đặt hàng của hai nhà cung cấp này là **NULL** bởi vì công ty chưa bao giờ đặt hàng các nhà cung cấp này!

**Ví dụ:**

Hoàn toàn giống như thí dụ trên nhưng lần này chúng ta sẽ sử dụng thứ tự ưu tiên quan hệ dữ liệu bên trái. Chúng ta thực hiện câu lệnh **SELECT FROM** như sau:

```
SELECT NCC.MANHACC, TENNHACC, SODH
FROM NHACC NCC
LEFT JOIN DONDH DH
ON DH.MANHACC=NCC.MANHACC
ORDER BY NCC.MANHACC
```

Theo các bạn thì kết quả truy vấn trả về có gì khác so với thí dụ ở trên khi thay đổi thứ tự ưu tiên quan hệ dữ liệu bên phải (**RIGHT JOIN**) hay không?

Câu trả lời là kết quả của hai câu lệnh truy vấn **hoàn toàn như nhau**, tuy nhiên khi sử dụng thứ tự ưu tiên quan hệ dữ liệu bên trái trong thí dụ này chúng tôi đã cố tình thay đổi tên bảng **NHACC** ngay phía sau mệnh đề **FROM** để muốn chỉ định thứ tự ưu tiên lấy dữ liệu bên bảng **NHACC**.

Trong thực tế việc chọn lựa để sử dụng mệnh đề **LEFT JOIN** hoặc **RIGHT JOIN** là không quan trọng mà thay vào đó các bạn phải hiểu rằng dữ liệu mà chúng ta cần chọn ra phải ưu tiên nằm bên trong bảng nào. Thông thường chúng tôi có một quy định là bảng nào ưu tiên dữ liệu sẽ được ghi ngay sau mệnh đề **FROM**, kế tiếp sử dụng mệnh đề **LEFT JOIN** chỉ định tên của bảng quan hệ cần lấy thông tin.

Mệnh đề liên kết dữ liệu nhiều bảng

Với cú pháp **SELECT FROM** kết hợp mệnh đề **JOIN** của phần trình bày ở trên, chúng ta có **liên kết tối đa 256** bảng dữ liệu trong một câu truy vấn. Một bảng có thể liên kết với nhiều bảng khác nhau trong một câu truy vấn. Các liên kết có thể được định nghĩa dựa trên các cột giống nhau của các bảng.

**Ví dụ:**

Để biết được danh sách tên các vật tư đã đặt hàng trong tháng 01/2002, các bạn phải lấy thông tin từ các bảng: **VATTU** (lấy cột tên vật tư), **CTDONDH** (lấy cột mã vật tư), **DONDH** (lấy cột ngày đặt hàng so sánh trong tháng 01/2002 và tạo quan hệ trung gian cho hai bảng **VATTU** và **DONDH**). Nhận xét thấy rằng trong truy vấn này dữ liệu cần lấy ra từ 3 bảng khác nhau nhưng có quan hệ.

Chúng ta thực hiện câu lệnh **SELECT FROM** như sau:

```
SELECT DISTINCT VT.MAVTU, TENVTU
FROM DONDH DH
INNER JOIN CTDONDH CTDH
ON DH.SODH=CTDH.SODH
INNER JOIN VATTU VT
ON VT.MAVTU=CTDH.MAVTU
WHERE MONTH(NGAYDH)=01
AND YEAR(NGAYDH)=2002
```

Kết quả truy vấn trả về



```
MAVTU TENVTU
```

```
-----
```

```
DD01  Đầu DVD Hitachi 1 đĩa
```

```
DD02  Đầu DVD Hitachi 3 đĩa
```

```
VD02  Đầu VCD Sony 3 đĩa
```

```
(3 row(s) affected)
```

Mệnh đề nối dữ liệu từ hai truy vấn

Việc kết hợp dữ liệu của hai truy vấn **SELECT FROM** bằng mệnh đề **UNION** cho phép các bạn có thể tạo ra **một tập hợp** các mẫu tin từ các mẫu tin có trong câu lệnh **SELECT FROM** thứ nhất và các mẫu tin có trong câu lệnh **SELECT FROM** thứ hai. Khác với việc liên kết dữ liệu bằng mệnh đề **JOIN**, mệnh đề **UNION** thực ra chỉ thực hiện việc thêm vào các dòng dữ liệu trong câu lệnh **SELECT FROM** thứ nhất vào cuối các dòng dữ liệu trong câu lệnh **SELECT FROM** thứ hai.

Thông thường chúng ta sử dụng mệnh đề **UNION** dùng để nối dữ liệu từ các bảng khác nhau trong cơ sở dữ liệu thành một bộ các mẫu tin liên tục nhau. Các cột chỉ định trong hai câu lệnh **SELECT FROM** phải có cùng kiểu dữ liệu tương thích thứ tự như nhau, tổng số các cột phải bằng nhau. Việc định dạng tiêu đề của các cột tính toán chỉ cần thực hiện trong câu lệnh truy vấn đầu tiên.

Cú pháp:

```
SELECT Danh_sách_các_cột1
      FROM Tên_bảng1

UNION

SELECT Danh_sách_các_cột2
      FROM Tên_bảng2

[ORDER BY Danh_sách_cộtsx]
```

Trong đó:

✓ Mệnh đề **UNION**: dùng để nối dữ liệu của hai truy vấn.



**Ví dụ:**

Để tính ra đồng thời tổng số lượng nhập, tổng số lượng xuất của các vật tư trong tháng 01/2002. Chúng ta thực hiện các câu lệnh **SELECT FROM** như sau:

```
SELECT MAVTU, "X" AS LOAI, SUM(SLXUAT) AS TONGSL
FROM CTPXUAT CTPX INNER JOIN PXUAT PX ON PX.SOPX=CTPX.SOPX
WHERE CONVERT(CHAR(7), NGAYXUAT, 21)="2002-01"
GROUP BY MAVTU
UNION
SELECT MAVTU, "N" , SUM(SLNHAP)
FROM CTPNHAP CTPN INNER JOIN PNHAP PN ON PN.SOPN=CTPN.SOPN
```





```
WHERE CONVERT(CHAR(7), NGAYNHAP, 21)="2002-01"
GROUP BY MAVTU
```

*Kết quả truy vấn trả về*

```
MAVTU LOAI TONGSL
```

```
-----
DD01 N      10
DD02 N      15
VD02 N      30
DD01 X       6
VD02 X      10
DD02 X       7
```

(6 row(s) affected)

Nhận xét thấy rằng tổng số cột mà các truy vấn về sẽ là 3 cột, thứ tự kiểu dữ liệu của các cột phải tương thích nhau (chuỗi, chuỗi, số), việc định dạng tiêu đề cột chỉ thực hiện tại truy vấn **SELECT FROM** thứ nhất.

Chúng ta thấy rằng các dòng dữ liệu trong truy vấn thứ nhất sẽ được thêm vào cuối các dòng dữ liệu của truy vấn thứ hai. Tuy nhiên các mẫu tin hiển thị chưa được sắp xếp theo một thứ tự nào cả. Do đó nếu muốn các mẫu tin được sắp xếp lại theo một thứ tự nào đó thì chúng ta sẽ kết hợp mệnh đề **ORDER BY** vào cuối.



#### Ví dụ:

Thực hiện lại truy vấn trên nhưng có kết hợp thêm mệnh đề **ORDER BY** để thấy rõ số lượng nhập, số lượng xuất của từng vật tư trong tháng 01/2002.

```
SELECT MAVTU, "X" AS LOAI, SUM(SLXUAT) AS TONGSL
FROM CTPXUAT CTPX INNER JOIN PXUAT PX ON PX.SOPX=CTPX.SOPX
WHERE CONVERT(CHAR(7), NGAYXUAT, 21)="2002-01"
GROUP BY MAVTU
UNION
SELECT MAVTU, "N" , SUM(SLNHAP)
FROM CTPNHAP CTPN INNER JOIN PNHAP PN ON PN.SOPN=CTPN.SOPN
WHERE CONVERT(CHAR(7), NGAYNHAP, 21)="2002-01"
GROUP BY MAVTU
ORDER BY MAVTU, LOAI
```

*Kết quả truy vấn trả về*

```
MAVTU LOAI TONGSL
```

```
-----
DD01 N      10
DD01 X       6
DD02 N      15
DD02 X       7
VD02 N      30
```



VD02 X 10

(6 row(s) affected)

#### IV.1.6. Mệnh đề chép dữ liệu ra bảng mới

Với cú pháp **SELECT FROM** bên dưới kết hợp mệnh đề **INTO** cho phép chúng ta **sao chép dữ liệu và cấu trúc** từ kết quả của một truy vấn cho ra một bảng dữ liệu mới bên trong cơ sở dữ liệu hiện hành hoặc các **bảng dữ liệu tạm thời** dùng để tính toán các xử lý phức tạp. Trong trường hợp nếu chúng ta muốn tạo ra bảng dữ liệu mới thì bắt buộc tên của bảng phải duy nhất trong cơ sở dữ liệu.

Chúng ta có thể chỉ định các ký tự dấu thăng (#) hoặc hai ký tự dấu thăng (##) phía trước tên bảng được tạo trong câu lệnh **SELECT INTO** dùng để tạo ra các bảng tạm **cục bộ** (local) hoặc các bảng tạm **toàn cục** (global). Bảng tạm cục bộ chỉ được sử dụng bởi người tạo ra nó và hệ thống sẽ tự động hủy bỏ bảng tạm cục bộ khi người tạo ra bảng ngưng nối kết vào Microsoft SQL Server. Ngược lại bảng tạm toàn cục được sử dụng cho nhiều người khác nhau và hệ thống tự động hủy bỏ bảng tạm toàn cục khi không còn người sử dụng nào nối kết vào Microsoft SQL Server.

Cú pháp:

```
SELECT Danh_sách_các_cột
      INTO Tên_bảng_mới
      FROM Tên_bảng_dl
```

Trong đó:

- ✓ Tên bảng mới: là tên của bảng mới sẽ được tạo lập có cấu trúc và dữ liệu từ truy vấn.
- ✓ Tên bảng dữ liệu: là tên của bảng chứa dữ liệu nguồn cho việc sao chép.



#### Ví dụ:

Để tạo ra bảng tạm cục bộ chứa thông tin thuế trị giá gia tăng (VAT) là 10% thành tiền của các phiếu nhập hàng trong tháng 01/2002. Chúng ta thực hiện các câu lệnh **SELECT INTO** như sau:

```
SELECT PN.SOPN, THANH_TIEN=SUM(SLNNHAP*DGNHAP) ,
      SUM(SLNNHAP*DGNHAP)*0.1 AS THUE_VAT INTO #THUE_PNHAP_200201
FROM CTPNHAP CTPN INNER JOIN PNHAP PN
      ON PN.SOPN=CTPN.SOPN
WHERE CONVERT(CHAR(7), NGAYNHAP, 21)="2002-01"
GROUP BY PN.SOPN
```

Sau đó thực hiện lệnh kế tiếp để xem dữ liệu hiện đang được lưu trữ trong bảng tạm #THUE\_PNHAP\_200201

```
SELECT *
      FROM #THUE_PNHAP_200201
```



Kết quả truy vấn trả về

SOPN THANH_TIEN	THUE_VAT
N001 55000000.0000	5500000.00000
N002 22500000.0000	2250000.00000
N003 75000000.0000	7500000.00000
(3 row(s) affected)	

Các bạn tạm thời ngắt nối kết vào Microsoft SQL Server và sau đó thực hiện nối kết lại vào Microsoft SQL Server, thực hiện lại truy vấn xem dữ liệu của bảng tạm cục bộ #THUE\_PNHAP\_200201 đã tạo trước đó. Kết quả hệ thống trả về như thế nào? Tại sao lại như thế?

Bảng tạm #THUE\_PNHAP\_200201 đã không còn vì hệ thống đã **tự động hủy bỏ** bảng tạm cục bộ khi bạn ngưng nối kết vào Microsoft SQL Server.

#### IV.1.7. Mệnh đề thống kê dữ liệu

Với cú pháp **SELECT FROM** bên dưới kết hợp mệnh đề **COMPUTE** cho phép các bạn có thể tạo ra **dòng thống kê dữ liệu** ở bên cuối kết quả truy vấn. Tuy nhiên nếu các bạn sử dụng thêm mệnh đề **COMPUTE BY** tiếp theo thì hệ thống sẽ thống kê dữ liệu theo từng nhóm dữ liệu.

Cú pháp:

```
SELECT Danh_sách_các_cột
FROM Tên_bảng
COMPUTE COUNT|MIN|MAX|SUM|AVG (Tên_cột)
[ BY Tên_cột_nhóm ]
```

Trong đó:

- ✓ Count, Min, Max, Sum, Avg: là các hàm thống kê tính toán dữ liệu mà kết quả sẽ xuất hiện ở cuối kết quả truy vấn hoặc từng nhóm dữ liệu.
- ✓ Tên cột: tên các cột hoặc biểu thức được tính toán kèm với các hàm thống kê chỉ định trước đó.



#### Ví dụ:

Để hiển thị thông tin chi tiết các vật tư đã đặt hàng cho các nhà cung cấp có mã là "C02" hoặc "C03". Có thống kê tổng số lượng đặt, số lượng nhiều nhất, số lượng đặt ít nhất trên kết quả truy vấn. Chúng ta thực hiện các câu lệnh **SELECT FROM** như sau:

```
SELECT DH.SODH, VT.MAVTU, TENVTU, SLDAT, MANHACC
FROM CTDONDH CTDH INNER JOIN VATTU VT ON CTDH.MAVTU=VT.MAVTU
JOIN DONDH DH ON DH.SODH = CTDH.SODH
WHERE MANHACC IN ("C02", "C03")
COMPUTE SUM(SLDAT), MAX(SLDAT), MIN(SLDAT)
```



Kết quả truy vấn trả về

```

SODH MAVTU TENVTU          SLDAT  MANHACC
-----
10      C03
D001 DD02  Đầu DVD Hitachi 3 đĩa  15    C03
D003 TV14  Tivi Sony 14 inches     10    C02
D003 TV29  Tivi Sony 29 inches     20    C02
D005 TV14  Tivi Sony 14 inches     10    C02
D005 TV29  Tivi Sony 29 inches     20    C02

                                sum
                                =====
                                85

                                max
                                =====
                                20

                                min
                                =====
                                10

(7 row(s) affected)

```



**Ví dụ:**

Theo thí dụ trên nhưng chúng ta muốn thống kê theo từng nhà cung cấp. Lúc này các bạn bắt buộc sử dụng mệnh đề **COMPUTE BY** tuy nhiên cần phải kết hợp với mệnh đề **ORDER BY**. Chúng ta thực hiện các câu lệnh **SELECT FROM** như sau:

```

SELECT DH.SODH, VT.MAVTU, TENVTU, SLDAT, MANHACC
FROM CTDONDH CTDH
INNER JOIN  VATTU VT
ON CTDH.MAVTU=VT.MAVTU
JOIN DONDH DH
ON DH.SODH = CTDH.SODH
WHERE MANHACC IN ("C02", "C03")
ORDER BY MANHACC
COMPUTE SUM(SLDAT) BY MANHACC

```

Kết quả truy vấn trả về

```

SODH MAVTU TENVTU          SLDAT  MANHACC
-----
D003 TV14  Tivi Sony 14 inches     10    C02
D003 TV29  Tivi Sony 29 inches     20    C02

```



```
D005 TV14 Tivi Sony 14 inches      10    C02
D005 TV29 Tivi Sony 29 inches      20    C02
```

sum

=====

60

```
D001 DD01 Đầu DVD Hitachi 1 đĩa    10    C03
D001 DD02 Đầu DVD Hitachi 3 đĩa    15    C03
```

sum

=====

25

(8 row(s) affected)

## IV.2. Truy vấn con

Trong khi lập trình bên trong Transaction–SQL, có đôi lúc chúng ta sẽ sử dụng đến truy vấn con để tính toán dữ liệu. Truy vấn con chỉ là một câu lệnh truy vấn chọn lựa (SELECT) được lồng vào các câu lệnh truy vấn khác nhằm thực hiện các truy vấn tính toán phức tạp. Khi sử dụng đến truy vấn con chúng ta cần **lưu tâm** đến một vài yếu tố như sau:

- ✓ Cần mở và đóng ngoặc đơn cho câu lệnh truy vấn con.
- ✓ Chúng ta chỉ được phép tham chiếu đến tên một cột hoặc một biểu thức sẽ trả về giá trị trong truy vấn con.
- ✓ Kết quả của truy vấn con có thể trả về là một giá trị đơn lẻ hoặc một danh sách các giá trị.
- ✓ Cấp độ lồng nhau của các truy vấn con bên trong Microsoft SQL Server là không giới hạn.

Truy vấn con trả về một giá trị đơn

Là các truy vấn mà kết quả trả về của nó luôn luôn đảm bảo chỉ là một giá trị đơn. Thông thường các truy vấn dạng này sẽ sử dụng các hàm tính toán thống kê dữ liệu.



### Ví dụ:

Để biết được danh sách các đơn đặt hàng gần đây nhất. Trước tiên chúng ta phải tính ra được ngày đặt hàng gần đây là bao nhiêu bằng câu lệnh truy vấn như sau:

```
SELECT MAX(NGAYDH)
FROM DONDH
```

Kết quả truy vấn trả về

```
-----
2002-03-15 00:00:00.000
```

(1 row(s) affected)

Sau đó chúng ta sẽ lọc ra danh sách các đơn đặt hàng có ngày đặt hàng là ngày “2002-03-15”. Thực hiện truy vấn như sau:

```
SELECT *
```



```
FROM DONDH
WHERE NGAYDH = "2002-03-15"
```

Tuy nhiên không thể nào chắc rằng ngày đặt hàng gần nhất trong bảng DONDH luôn là “2002-03-15”.

Do đó câu lệnh truy vấn ở trên chỉ đúng tại thời điểm này mà thôi. Để đảm bảo rằng chúng ta luôn có được danh sách các đơn đặt hàng gần đây nhất, các bạn sẽ kết hợp cả hai câu truy vấn đã thực hiện ở trên như sau:

```
SELECT *
FROM DONDH
WHERE NGAYDH = (SELECT MAX(NGAYDH)
FROM DONDH)
```

Nhận xét thấy rằng câu lệnh truy vấn con chỉ sử dụng một hàm tính toán thống kê là **MAX** nên kết quả luôn luôn trả về một giá trị đơn.



#### Ví dụ:

Muốn biết tổng số lượng đã đặt hàng của từng vật tư. Chúng ta thực hiện câu truy vấn như sau:

```
SELECT TENVТУ, SUM(SLDAT) AS TONGSLDAT
FROM VATTU VT INNER JOIN CTDONDH CTDH ON CTDH.MAVТУ=VT.MAVТУ
GROUP BY TENVТУ
```

Kết quả truy vấn trả về

TENVТУ	TONGSLDAT
Đầu DVD Hitachi 1 đĩa	10
Đầu DVD Hitachi 3 đĩa	15
Đầu VCD Sony 1 đĩa	20
Đầu VCD Sony 3 đĩa	30
Tivi Sony 14 inches	30
Tivi Sony 29 inches	60
Tủ lạnh Sanyo 90 lit	10

(7 row(s) affected)



#### Ví dụ:

Với câu lệnh truy vấn bên dưới sẽ trả về tổng cộng số lượng đặt hàng của tất cả các vật tư có trong bảng CTDONDH.

```
SELECT SUM(SLDAT) AS TONGCONG
FROM CTDONDH
```

Chúng ta kết hợp hai câu truy vấn trên để biết được tỷ lệ phần trăm số lượng đặt hàng của từng vật tư trên tổng cộng các số lượng đặt hàng của toàn bộ các vật tư.

```
SELECT TENVТУ, SUM(SLDAT) AS TONGSLDAT,
(SELECT SUM(SLDAT) FROM CTDONDH) AS TONGCONG ,
( ( CONVERT(MONEY, SUM(SLDAT)) /
(SELECT SUM(SLDAT) FROM CTDONDH) ) * 100 ) AS PHTRAM
```

```
FROM VATTU VT INNER JOIN CTDONDH CTDH ON CTDH.MAVTU=VT.MAVTU
GROUP BY TENVTU
```

Kết quả truy vấn trả về

TENVTU	TONGSLDAT	TONGCONG	PHTRAM
Đầu DVD Hitachi 1 đĩa	10	175	5.7100
Đầu DVD Hitachi 3 đĩa	15	175	8.5700
Đầu VCD Sony 1 đĩa	20	175	11.4200
Đầu VCD Sony 3 đĩa	30	175	17.1400
Tivi Sony 14 inches	30	175	17.1400
Tivi Sony 29 inches	60	175	34.2800
Tủ lạnh Sanyo 90 lit	10	175	5.7100

(7 row(s) affected)

#### IV.2.1. Truy vấn con trả về danh sách các giá trị

Là các truy vấn mà kết quả trả về của nó là một **danh sách** các giá trị hay còn gọi là một tập hợp các phần tử. Thông thường các truy vấn con dạng này sẽ lấy dữ liệu của một hoặc nhiều bảng khác thực hiện việc so sánh trong mệnh đề **WHERE** của truy vấn cha. Toán tử **IN** sẽ được sử dụng để so sánh trong truy vấn con dạng này bởi vì nó dùng chỉ định việc so sánh một phần tử có thuộc trong một tập hợp các phần tử hay không.



##### Ví dụ:

Để biết các nhà cung cấp nào mà công ty đã đặt hàng trong tháng 01/2002. Chúng ta có thể thực hiện câu truy vấn như sau:

```
SELECT MANHACC
FROM DONDH
WHERE CONVERT(CHAR(7), NGAYDH, 21)="2002-01"
```

Kết quả truy vấn trả về

```
MANHACC
-----
C03
C01
(2 row(s) affected)
```

Tuy nhiên thông tin mà chúng ta muốn hiển thị ra đầy đủ sẽ là họ tên các nhà cung cấp chứ không phải là mã nhà cung cấp. Do thế chúng ta sẽ sử dụng truy vấn như sau:

```
SELECT TENNHACC, DIENTHOAI
FROM NHACC
WHERE MANHACC IN ("C01", "C03")
```

Đầu đảm bảo rằng trong tháng 01/2002 công ty chỉ đặt hàng cho hai nhà cung cấp C01 và C03. Do thế để luôn luôn có được danh sách họ tên các nhà cung cấp mà công ty đã đặt hàng trong



tháng 01/2002, chúng ta thực hiện truy vấn con như sau:

```
SELECT TENNHACC, DIENTHOAI
FROM NHACC
WHERE MANHACC IN
      (SELECT MANHACC
       FROM DONDH
       WHERE CONVERT(CHAR(7), NGAYDH, 21)="2002-01")
```

Kết quả truy vấn trả về

TENNHACC	DIENTHOAI
-----	
Lê Minh Trí	8781024
Nguyễn Hồng Phương	9600125
(2 row(s) affected)	

Các bạn cũng có thể sử dụng từ khóa **EXISTS** hoặc mệnh đề **JOIN** để thực hiện việc so sánh các dòng dữ liệu trong các truy vấn con trả về danh sách các giá trị. Từ khóa **EXISTS** dùng để kiểm tra tính tồn tại của dữ liệu, ngay sau **EXISTS** là một câu lệnh **SELECT** mà kết quả trả về của nó là một tập hợp trống hoặc có chứa nhiều phần tử.



**Ví dụ:**

Hai câu lệnh truy vấn bên dưới đều có kết quả trả về như câu lệnh truy vấn ở thí dụ bên trên.

```
SELECT TENNHACC, DIENTHOAI
FROM NHACC NCC
WHERE EXISTS (SELECT *
              FROM DONDH DH
              WHERE CONVERT(CHAR(7), NGAYDH, 21)="2002-01"
                    AND DH.MANHACC=NCC.MANHACC)
```

Hoặc

```
SELECT TENNHACC, DIENTHOAI
FROM NHACC NCC INNER JOIN DONDH DH ON DH.MANHACC=NCC.MANHACC
WHERE CONVERT(CHAR(7), NGAYDH, 21)="2002-01"
```

Nếu muốn sử dụng các toán tử so sánh bình thường (=, >, <, <> ...) trong truy vấn con trả về danh sách các giá trị thì bắt buộc chúng ta phải kết hợp các từ khóa **ANY**, **ALL** phía trước câu lệnh truy vấn con. Chúng tôi muốn giới thiệu một quy tắc mà các bạn nên nhớ là:

**“IN sẽ tương đương = ANY và NOT IN sẽ tương đương <> ALL”**



**Ví dụ:**

Để biết danh sách các nhà cung cấp nào mà công ty chưa bao giờ đặt hàng. Chúng ta có thể thực hiện câu truy vấn như sau:

```
SELECT TENNHACC, DIENTHOAI
FROM NHACC
WHERE MANHACC NOT IN (SELECT DISTINCT MANHACC
                      FROM DONDH)
```

Hoặc

```
SELECT TENNHACC, DIENTHOAI
FROM NHACC
WHERE MANHACC <> ALL (SELECT DISTINCT MANHACC
                     FROM DONDH)
```

Kết quả truy vấn trả về

TENNHACC	DIENTHOAI
Trương Nhật Thăng	8757757
Cao Minh Trung	Chưa có

(2 row(s) affected)

Tuy nhiên nếu các bạn hiểu sai các qui tắc trên “**NOT IN**” sẽ tương đương “**<> ANY**”. Khi đó với câu truy vấn bên dưới sẽ trả về kết quả sai hoàn toàn!

```
SELECT TENNHACC, DIENTHOAI
FROM NHACC
WHERE MANHACC <> ANY (SELECT DISTINCT MANHACC
                     FROM DONDH)
```

Kết quả truy vấn trả về

TENNHACC	DIENTHOAI
Lê Minh Trí	8781024
Trần Minh Thạch	7698154
Nguyễn Hồng Phương	9600125
Trương Nhật Thăng	8757757
Lưu Nguyệt Quế	7964251
Cao Minh Trung	Chưa có

(6 row(s) affected)

Việc chúng tôi trình bày thêm các từ khóa **ALL**, **ANY** trong các truy vấn con nhằm giúp các bạn hiểu thêm trong thực hiện việc so sánh dữ liệu của các truy vấn con với truy vấn cha, tuy nhiên theo quan điểm cá nhân của chúng tôi thì các bạn chỉ cần nhớ các toán tử **IN** hoặc **NOT IN** và sử dụng cho đúng trong các trường hợp so sánh cần thiết đối với dạng truy vấn con trả về danh sách các giá trị.

### IV.3. Lệnh INSERT INTO

Lệnh truy vấn kế tiếp mà chúng tôi muốn trình bày là lệnh cho phép các bạn **thêm mới** một hoặc nhiều dòng dữ liệu vào bên trong một bảng. Trước tiên chúng ta sẽ làm quen với lệnh **INSERT INTO** là lệnh chỉ cho phép các bạn thêm mới **một** dòng dữ liệu vào bên trong bảng.

Cú pháp:

```
INSERT INTO Tên_bảng [ (Danh_sách_cột) ]  
VALUES (Danh_sách_giá_trị)
```

Trong đó:

- ✓ Tên bảng: tên bảng được thêm mới dòng dữ liệu.
- ✓ Danh sách cột: danh sách tên các cột hiện có trong bảng. Các bạn có thể không cần chỉ định ra tên của các cột, tuy nhiên khi đó danh sách các giá trị mà chúng ta đưa vào phải theo đúng thứ tự vật lý của các cột bên trong bảng khi tạo cấu trúc bảng trước đó.



**Ví dụ:**

Để thêm một vật tư mới vào bảng VATTU. Chúng ta sử dụng lệnh **INSERT INTO** như sau:

```
INSERT INTO VATTU (MAVTU, TENVTU, DVTINH, PHANTRAM)  
VALUES ("LO01", "Loa Panasonic 1000W", "Bộ", 10)
```

Hoặc chúng ta cũng có thể thực hiện nhanh lệnh như sau:

```
INSERT INTO VATTU  
VALUES ("LO01", "Loa Panasonic 1000W", "Bộ", 10)
```

Tuy nhiên lúc bấy giờ câu lệnh thứ hai chỉ đúng khi thứ tự của các cột trong bảng VATTU phải là: mã vật tư, tên vật tư, đơn vị tính và tỷ lệ phần trăm.

Trong trường hợp khi chúng ta có một **danh sách** các dữ liệu hiện đang có bên trong một hoặc nhiều bảng khác nhau, theo yêu cầu muốn các dữ liệu này sẽ được thêm mới vào bên trong một bảng. Nếu bảng này chưa có trong cơ sở dữ liệu thì các bạn có thể sử dụng lệnh **SELECT INTO** mà chúng tôi đã trình bày trước đây để sao chép dữ liệu và tạo cấu trúc cho bảng mới. Tuy nhiên nếu theo yêu cầu muốn chúng ta phải chèn thêm (append) các dòng dữ liệu này vào bảng dữ liệu hiện có trong cơ sở dữ liệu thì bắt buộc các bạn phải sử dụng lệnh **INSERT...SELECT**.

Cú pháp:

```
INSERT [INTO] Tên_bảng [ (Danh_sách_cột) ]  
SELECT Danh_sách_cột  
FROM Tên_bảng_dị_nguồn  
WHERE Điều_kiện_lọc
```

Trong đó:

- ✓ Tên bảng: tên bảng được thêm các dòng dữ liệu mới.
- ✓ Các mệnh đề của lệnh SELECT bên trên hoàn toàn giống như lệnh SELECT FROM mà chúng ta đã biết đến trong phần trên.

**Ví dụ:**

Để tính ra tổng số lượng nhập, tổng số lượng xuất của các vật tư trong tháng 01/2002, sau đó lấy các dữ liệu này thêm mới vào bảng **TONKHO** để cập nhật lại tình hình nhập xuất hàng hóa trong tháng 01/2002. Chúng ta sử dụng lệnh các lệnh như sau:

Trước tiên tạo ra bảng ảo dùng để tính tổng số lượng nhập của các vật tư trong tháng 01/2002.

```
CREATE VIEW vw_TONGN_200201
AS
SELECT MAVTU, SUM(SLNHAP) AS TONGNHAP
FROM CTPNHAP CTPN INNER JOIN PNHAP PN ON PN.SOPN=CTPN.SOPN
WHERE CONVERT(CHAR(7), NGAYNHAP, 21)="2002-01"
GROUP BY MAVTU
GO
```

Kế tiếp tạo ra bảng ảo dùng để tính tổng số lượng xuất của các vật tư trong tháng 01/2002.

```
CREATE VIEW vw_TONGX_200201
AS
SELECT MAVTU, SUM(SLXUAT) AS TONGXUAT
FROM CTPXUAT CTPX INNER JOIN PXUAT PX ON PX.SOPX=CTPX.SOPX
WHERE CONVERT(CHAR(7), NGAYXUAT, 21)="2002-01"
GROUP BY MAVTU
GO
```

Sử dụng lệnh **INSERT...SELECT** để thêm dữ liệu vào bảng **TONKHO**.

```
INSERT TONKHO
SELECT "2002-01", TN.MAVTU, 0, TONGNHAP, TONGXUAT,
TONGNHAP- TONGXUAT
FROM vw_TONGN_200201 TN LEFT JOIN vw_TONGX_200201 TX
ON TN.MAVTU=TX.MAVTU
```

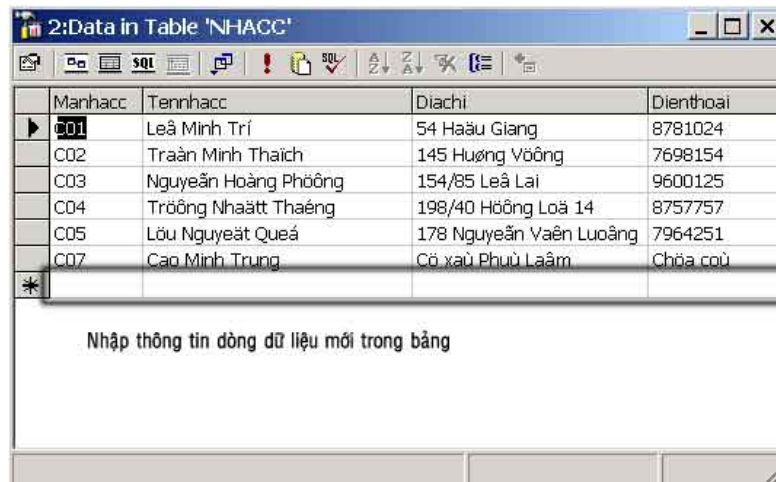
Trên đây chỉ là một thí dụ minh họa về lệnh **INSERT...SELECT**, trong thực tế việc cập nhật dữ liệu trong bảng **TONKHO** sẽ hoàn toàn được tự động thực hiện thông qua các các hành động nhập xuất vật tư. Để làm được điều này chúng ta phải cần biết đến khái niệm **trigger** bên trong bảng. Đối tượng này sẽ được chúng tôi trình bày ở bài 7.

Thông thường khi thêm mới dữ liệu vào bảng nếu chúng ta vô tình có vi phạm các ràng buộc toàn vẹn dữ liệu đã định nghĩa trước đó thì tuyệt đối dữ liệu sẽ không được lưu vào bên trong bảng. Một vài ràng buộc toàn vẹn dữ liệu thường vi phạm là: khóa ngoại, miền giá trị, không được phép bỏ trống dữ liệu khi thêm mới.

Tóm lại việc thêm mới dữ liệu vào bên trong bảng bằng lệnh **INSERT** có thể được thêm từng dòng một (**INSERT INTO**) hoặc cùng lúc nhiều dòng (**INSERT...SELECT**).

Ngoài ra các bạn cũng có thể sử dụng tiện ích Enterprise Manager để nhập mới dữ liệu trực tiếp vào bên trong bảng. Chọn chức năng **Open Table ► Return all rows** trong thực đơn tắt sau khi nhấn chuột phải trên tên bảng muốn nhập mới dữ liệu. Trong màn hình hiển thị các dòng dữ

liệu trong bảng, nhập thông tin dữ liệu mới tại dòng dữ liệu trống bên dưới cùng.



Manhacc	Tennhacc	Diachi	Dienthoai
C01	Leã Minh Trí	54 Haàu Giang	8781024
C02	Traàn Minh Thaich	145 Hường Vông	7698154
C03	Nguyeãn Hoång Phông	154/85 Leã Lai	9600125
C04	Trông Nhaatt Thaeng	198/40 Hông Loà 14	8757757
C05	Lôu Nguyeatt Queá	178 Nguyeãn Vaãn Luoang	7964251
C07	Cao Minh Trung	Cô xâu Phuù Laãm	Chôa còu
*			

Nhập thông tin dòng dữ liệu mới trong bảng

**Hình 4-1.** Màn hình hiển thị các dòng dữ liệu trong bảng.

#### IV.4. Lệnh DELETE FROM

Trái ngược với hành động thêm mới dữ liệu vào bảng, chúng ta có thể **hủy bỏ các dòng dữ liệu** hiện đang có trong bảng khi không còn sử dụng nữa. Cần thận với hành động này bởi vì chúng ta không thể khôi phục lại các dữ liệu sau khi đã ra lệnh hủy bỏ nó. Lệnh **DELETE FROM** bên dưới cho phép chúng ta hủy bỏ các dòng dữ liệu hiện đang có bên trong một bảng.

Cú pháp:

```
DELETE [FROM] Tên_bảng
      [FROM Tên_bảng1
      INNER|LEFT|RIGHT JOIN Tên_bảng2
      ON Biểu_thức_liên_kết]
      [WHERE Điều_kiện_xóa]
```

Trong đó:

- ✓ Tên bảng: tên bảng có các dòng dữ liệu muốn hủy bỏ.
- ✓ Tên bảng1, tên bảng2: Tên các bảng có quan hệ dữ liệu, được dùng để kết nối các quan hệ nhằm tra cứu các thông tin trong khi xóa dữ liệu.
- ✓ Điều kiện xóa dữ liệu: là biểu thức luận lý chỉ định các dòng dữ liệu phải thỏa điều kiện đưa ra thì mới bị hủy bỏ.

Lưu ý:

Trong lệnh DELETE nếu quên **không sử dụng mệnh đề WHERE** thì tất cả các dòng dữ liệu hiện đang có bên trong bảng dữ liệu sẽ bị hủy tất cả!

**Ví dụ:**

Để hủy bỏ các nhà cung cấp mà công ty chưa bao giờ đặt hàng. Chúng ta sử dụng lệnh **DELETE** như sau:

```
DELETE NHACC
FROM NHACC NCC LEFT JOIN DONDH DH ON DH.MANHACC=NCC.MANHACC
WHERE DH.SODH IS NULL
```

Nhận xét thấy rằng trong thí dụ này chúng tôi sử dụng mệnh đề **LEFT JOIN** để thay đổi ưu tiên quan hệ dữ liệu bên bảng NHACC, kế tiếp trong mệnh đề **WHERE DH.SODH IS NULL** dùng để chỉ định việc xóa đi những nhà cung nào chưa có số đặt hàng (số đặt hàng đang là trống).

**Ví dụ:**

Chúng ta cũng có thể sử dụng truy vấn con để thực hiện hành động hủy bỏ các nhà cung cấp trong bảng NHACC mà công ty chưa bao giờ đặt hàng bằng câu lệnh như sau:

```
DELETE NHACC
WHERE MANHACC NOT IN (SELECT DISTINCT MANHACC
FROM DONDH)
```

Nhận xét thấy rằng kết quả của truy vấn con sẽ trả về tập hợp các nhà cung cấp mà công ty đã đặt hàng, sau đó trong mệnh đề **WHERE MANHACC NOT IN** dùng để xác định các nhà cung cấp **không** nằm trong tập hợp các nhà cung cấp đã được đặt hàng.

Để hủy bỏ các đơn đặt hàng trong tháng 01/2002. Chúng ta sử dụng lệnh **DELETE** như sau:

```
DELETE DONDH
WHERE CONVERT(CHAR(7), NGAYDH, 21)="2002-01"
```

Hệ thống Microsoft SQL Server sẽ xuất hiện thông báo lỗi vì việc xóa dữ liệu trong bảng DONDH sẽ **vi phạm ràng buộc khóa ngoại** bên bảng CTDONDH.

```
Server: Msg 547, Level 16, State 1, Line 1
DELETE statement conflicted with COLUMN REFERENCE constraint 'FRK_CTDONDH_SODH'.
The conflict occurred in database 'QLBanHang', table 'CTDONDH', column 'Sodh'.
The statement has been terminated.
```

Thông thường khi hủy bỏ các dòng dữ liệu hiện có bên trong một bảng nào đó nếu chúng ta vô tình vi phạm các ràng buộc toàn vẹn dữ liệu về khóa ngoại đã được định nghĩa trước đó thì tuyệt đối dữ liệu sẽ không bị hủy ra khỏi bảng. Muốn tránh những sai sót này, trước tiên chúng ta cần phải hủy bỏ dữ liệu bên nhánh quan hệ nhiều (nhánh quan hệ con) trước. Do đó trở lại thí dụ trên chúng ta thấy rằng cần phải xóa đi chi tiết các đơn đặt hàng có liên quan trong tháng 01/2002 bên bảng CTDONDH.

Thực hiện truy vấn như sau:

```
DELETE CTDONDH
FROM CTDONDH CTDH INNER JOIN DONDH DH ON DH.SODH=CTDH.SODH
WHERE CONVERT(CHAR(7), NGAYDH, 21)="2002-01"
```

Trong câu lệnh truy vấn này chúng ta muốn xóa các dòng dữ liệu trong bảng CTDONDH, tuy nhiên cần phải **liên kết** thêm bảng DONDH vào để có thời gian chỉ lọc ra các đơn đặt hàng

trong tháng 01/2002.

Sau đó tiếp tục cho thực hiện lại truy vấn hủy bỏ các đơn đặt hàng trong tháng 01/2002 ở thí dụ trước đó. Lần này theo các bạn thì câu lệnh truy vấn sẽ trả về kết quả là thành công hay lại tiếp tục thất bại nữa?

Kết quả truy vấn chỉ trả về thành công khi nào các đơn đặt hàng trong tháng 01/2002 **chưa được nhập hàng về** (dữ liệu liên quan của các đơn đặt hàng chưa có trong bảng PNHAP), ngược lại khi các đơn đặt hàng này đã có nhập hàng về rồi thì hệ thống sẽ xuất hiện thông báo. Bởi vì bảng PNHAP đã định nghĩa có quan hệ khóa ngoại với bảng DONDH theo cột số đặt hàng trước đó.

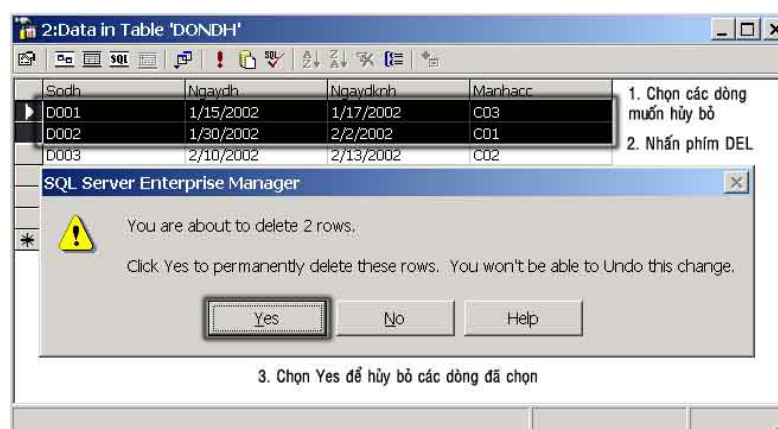
Server: Msg 547, Level 16, State 1, Line 1

DELETE statement conflicted with COLUMN REFERENCE constraint 'FRK\_PNHAP\_SODH'. The conflict occurred in database 'QLBanHang', table 'PNHAP', column 'Sodh'.

The statement has been terminated.

Tóm lại việc hủy bỏ các dữ liệu hiện có bên trong một bảng bằng lệnh **DELETE** có thể cùng lúc sẽ hủy bỏ được nhiều dòng thỏa điều kiện đưa ra. Kết quả của lệnh truy vấn này đôi khi có thể gây ra lỗi nếu chúng có **vi phạm** các ràng buộc toàn vẹn **khóa ngoại** bên nhánh quan hệ dữ liệu nhiều.

Ngoài ra các bạn cũng có thể sử dụng tiện ích Enterprise Manager để hủy bỏ các dòng dữ liệu hiện có bên trong bảng. Chọn chức năng **Open Table ► Return all rows** trong thực đơn tắt sau khi nhấn chuột phải trên tên bảng muốn hủy bỏ các dòng dữ liệu. Trong màn hình hiển thị các dòng dữ liệu trong bảng, đánh dấu chọn ra các dòng dữ liệu liên tiếp nhau muốn hủy bỏ, nhấn phím **Del** và chọn **Yes** để đồng ý việc hủy bỏ các dòng đã chọn.



**Hình 4-2.** Cách thức hủy bỏ dữ liệu trong tiện ích Enterprise Manager.

## IV.5. Lệnh UPDATE SET

Các dữ liệu sau khi được lưu trữ vào bên trong bảng đều đảm bảo rằng các giá trị đó sẽ đúng mãi mãi. Đôi khi các giá trị này cũng cần phải được thay đổi lại cho đúng theo một yêu cầu nào đó. Bằng tiện ích Enterprise Manager, các bạn có thể thay đổi trực tiếp các giá trị tại các dòng dữ liệu hiện có bên trong bảng trên màn hình hiển thị các dòng dữ liệu.



Bên cạnh đó trong các trường hợp cần **sửa đổi** đồng thời nhiều dòng dữ liệu bên trong một bảng chúng ta sẽ sử dụng lệnh **UPDATE SET**. Tuy nhiên cần thận khi sử dụng lệnh này bởi vì chúng ta sẽ **không có cơ hội** khôi phục lại các giá trị sau khi thay đổi. Cú pháp đầy đủ của lệnh **UPDATE SET** được mô tả như bên dưới.

Cú pháp:

```
UPDATE Tên_bảng
SET Tên_cột = Biểu_thức [ , ...]
[FROM Tên_bảng1
INNER|LEFT|RIGHT JOIN Tên_bảng2
ON Biểu_thức_liên_kết]
[WHERE Điều_kiện_sửa_đổi]
```

Trong đó:

- ✓ Tên bảng: tên bảng có chứa các dòng dữ liệu muốn sửa đổi.
- ✓ Tên cột: tên cột muốn sửa đổi giá trị dữ liệu. Chúng ta có thể thay đổi giá trị của nhiều cột bên trong một bảng trong cùng một câu lệnh UPDATE SET.
- ✓ Biểu thức: là một giá trị cụ thể hoặc một hàm tính toán mà giá trị trả về của nó sẽ được cập nhật vào tên cột trong bảng chỉ định trước đó.
- ✓ Tên bảng1, tên bảng2: Tên các bảng có quan hệ dữ liệu, được dùng để kết nối quan hệ trong khi sửa đổi dữ liệu.
- ✓ Điều kiện sửa đổi: là biểu thức luận lý chỉ định các dòng dữ liệu phải thỏa điều kiện đưa ra thì mới bị sửa đổi.



#### Chú ý:

Trong lệnh UPDATE nếu **không** có sử dụng mệnh đề WHERE thì tất cả các mẫu tin trong bảng sẽ bị sửa đổi.



#### Ví dụ:

Để có được tổng trị giá của các phiếu nhập hàng. Chúng ta có thể thêm một cột mới có tên TGNHAP (trị giá nhập) trong bảng PNHAP. Sử dụng lệnh **ALTER TABLE** để thêm vào một cột như sau:

```
ALTER TABLE PNHAP
ADD TGNHAP MONEY
```

Sau đó sử dụng lệnh **UPDATE SET** có kết hợp **truy vấn con** để tính ra tổng trị giá nhập dựa vào giá trị dữ liệu của các cột SLNHAP và TGNHAP trong bảng CTPNHAP theo từng số phiếu nhập hàng.

```
UPDATE PNHAP
SET TGNHAP=0
GO
UPDATE PNHAP
SET TGNHAP = ( SELECT SUM(SLNHAP*DGNHAP)
```



```
FROM CTPNHAP CTPN
WHERE PN.SOPN=CTPN.SOPN )
FROM PNHAP PN
GO
```

Cuối cùng kiểm tra lại kết quả sau khi đã thực hiện cập nhật tính giá trị cho cột trị giá nhập (TGNHAP).

```
SELECT *
FROM PNHAP
```

Kết quả truy vấn trả về

Sopn	Sodh	Ngaynhap	TGNHAP
N001	D001	2002-01-17 00:00:00.000	55000000.0000
N002	D001	2002-01-20 00:00:00.000	22500000.0000
N003	D002	2002-01-31 00:00:00.000	75000000.0000

(3 row(s) affected)



#### Ví dụ:

Để giảm giá 10% cho tất cả các phiếu bán hàng trong ngày cuối cùng của tháng 01/2002. Chúng ta sử dụng lệnh **UPDATE SET** như sau:

```
UPDATE CTPXUAT
SET DGXUAT = DGXUAT*0.9
FROM PXUAT PX INNER JOIN CTPXUAT CTPX ON PX.SOPX=CTPX.SOPX
WHERE NGAYXUAT="2002-01-31"
```

Tóm lại việc cập nhật các dữ liệu trong bảng bao gồm các hành động thêm, hủy và sửa đổi dữ liệu. Các hành động này chỉ tác động đến dữ liệu bên trong của **một và chỉ một bảng** mà thôi. Tuy nhiên bên trong các lệnh **INSERT...SELECT**, **DELETE**, **UPDATE SET** các bạn có thể tham chiếu đến một hoặc nhiều bảng khác để tạo ra các kết nối quan hệ nhằm lấy ra thông tin của các bảng khác dùng trong các điều kiện so sánh bên trong mệnh đề **WHERE**.

Các bạn phải nhớ là luôn luôn thận trọng khi sử dụng các lệnh **DELETE** và **UPDATE SET** vì khi đó các bạn sẽ **không có hội để phục hồi** lại dữ liệu cũ trước đó. Theo kinh nghiệm cá nhân của chúng tôi, trước khi thực hiện các lệnh này các bạn **nên** tạo ra bản dự phòng (backup) bằng lệnh **SELECT INTO** hoặc chèn thêm các cột tạm vào bên trong bảng để chứa giá trị trước khi thay đổi. Ngoài ra nếu hiểu rõ chế độ giao tác (transaction) là gì thì các bạn nên thực hiện các hành động cập nhật dữ liệu bên trong các giao tác bởi vì trong chế độ này chúng ta có thể phục hồi lại các giá trị dữ liệu đã bị cập nhật bên trong bảng.

## IV.6. Biểu thức CASE

Biểu thức **CASE** trong Transaction–SQL vô cùng hữu ích. Hoạt động của biểu thức **CASE** rất đơn giản chỉ là thực hiện việc so sánh một biểu thức bất kỳ với hàng loạt các giá trị chỉ định





trước đó, nếu bạn là người lập trình trong môi trường Visual Basic thì biểu thức **CASE** của Transaction–SQL **gần giống** như cấu trúc điều khiển Select Case.

Tuy nhiên biểu thức **CASE** hoàn toàn **không phải là một cấu trúc điều khiển**, điều này có nghĩa là nó chỉ được sử dụng lồng vào các câu lệnh khác mà không thể thực hiện đơn lẻ như các cấu trúc điều khiển khác. Biểu thức **CASE** có thể sử dụng ở hai dạng khác nhau.

Cú pháp CASE dạng đơn giản:

```
CASE Biểu_thức
    WHEN Giá_trị_1 THEN Biểu_thức_kết_quả_1
    [WHEN Giá_trị_2 THEN Biểu_thức_kết_quả_2
    ...]
    [ ELSE Biểu_thức_kết_quả_N]
END
```

Trong đó:

- ✓ Biểu thức: biểu thức tính toán hoặc tên cột dữ liệu của bảng được dùng để so sánh.
- ✓ Giá trị 1, giá trị 2: là các giá trị cụ thể để so sánh bằng (=) với biểu thức.
- ✓ Biểu thức kết quả 1, biểu thức kết quả 2: là các biểu thức sẽ được trả về khi việc so sánh của biểu thức bằng với các giá trị so sánh tương ứng.
- ✓ Biểu thức kết quả N: là biểu thức sẽ được trả về khi tất cả các trường hợp so sánh đều không bằng với các giá trị đưa ra.



**Ví dụ:**

Để hiển thị danh sách các vật tư có trong bảng *VATTU* theo từng loại hàng, có đếm tổng số các vật tư theo từng loại hàng. Chúng ta sử dụng lệnh **SELECT FROM** có kết hợp biểu thức **CASE** đơn giản như sau:

```
SELECT LOAI=
    CASE LEFT(MAVTU, 2)
        WHEN "DD" THEN "Đầu DVD"
        WHEN "VD" THEN "Đầu VCD"
        WHEN "TV" THEN "Tivi"
        WHEN "TL" THEN "Tủ lạnh"
        WHEN "BI" THEN "Bia lon"
        WHEN "LO" THEN "Loa thùng"
        ELSE "Chưa phân loại"
    END,
    MAVTU, TENVTU, DVTINH
FROM VATTU
ORDER BY LEFT(MAVTU, 2)
COMPUTE COUNT(MAVTU) BY LEFT(MAVTU, 2)
```

Kết quả truy vấn trả về

LOAI	MAVTU	TENVTU	DVTINH
-----			



```

Bia lon   BI01  Bia lon Tiger           Thùng
cnt
=====
1

Đầu DVD   DD01  Đầu DVD Hitachi 1 đĩa           Bộ
Đầu DVD   DD02  Đầu DVD Hitachi 3 đĩa           Bộ
cnt
=====
2

Loa thùng LO01  Loa Panasonic 1000W           Bộ
cnt
=====
1

Tủ lạnh   TL15  Tủ lạnh Sanyo 150 lit           Cái
Tủ lạnh   TL90  Tủ lạnh Sanyo 90 lit           Cái
cnt
=====
2

Tivi      TV14  Tivi Sony 14 inches             Cái
Tivi      TV21  Tivi Sony 21 inches             Cái
Tivi      TV29  Tivi Sony 29 inches             Cái
cnt
=====
3

Đầu VCD   VD01  Đầu VCD Sony 1 đĩa              Bộ
Đầu VCD   VD02  Đầu VCD Sony 3 đĩa              Bộ
cnt
=====
2
(17 row(s) affected)

```

Cú pháp CASE dạng tìm kiếm:

```

CASE
    WHEN Bt_logic_1 THEN Biểu_thức_kết_quả_1
    [WHEN Bt_logic_2 THEN Biểu_thức_kết_quả_2
      ...]
    [ ELSE Biểu_thức_kết_quả_N]
END

```

Trong đó:

- ✓ Biểu thức logic1, biểu thức logic2: là các biểu thức luận lý dùng để thực hiện các phép so sánh trong biểu thức CASE.
- ✓ Biểu thức kết quả 1, biểu thức kết quả 2: là các biểu thức sẽ được trả về khi một trong các biểu thức luận lý so sánh có kết quả là đúng.
- ✓ Biểu thức kết quả N: là biểu thức sẽ được trả về khi tất cả các biểu thức logic so sánh đưa ra đều sai.



**Ví dụ:**

Để hiển thị danh sách các vật tư có trong bảng VATTU, thông tin bổ sung thêm chuỗi ghi chú, tùy thuộc vào giá trị của cột tỷ lệ phần trăm giá bán. Chúng ta sử dụng lệnh **SELECT FROM** có kết hợp biểu thức **CASE** tìm kiếm như sau:

```
SELECT GHICHU=
CASE
  WHEN PHANTRAM <20 THEN "Lời ít"
  WHEN PHANTRAM BETWEEN 20 AND 40 THEN "Lời nhiều"
  ELSE "Rất lời"
END,
TENVTU, DVTINH, PHANTRAM
FROM VATTU
ORDER BY PHANTRAM
```

Kết quả truy vấn trả về

GHICHU	TENVTU	DVTINH	PHANTRAM
Lời ít	Loa Panasonic 1000W	Bộ	10
Lời ít	Tivi Sony 21 inches	Cái	15
Lời ít	Tivi Sony 29 inches	Cái	15
Lời nhiều	Tủ lạnh Sanyo 90 lít	Cái	20
Lời nhiều	Tivi Sony 14 inches	Cái	20
Lời nhiều	Tủ lạnh Sanyo 150 lít	Cái	25
Lời nhiều	Đầu VCD Sony 1 đĩa	Bộ	30
Lời nhiều	Đầu VCD Sony 3 đĩa	Bộ	30
Lời nhiều	Đầu DVD Hitachi 1 đĩa	Bộ	40
Lời nhiều	Đầu DVD Hitachi 3 đĩa	Bộ	40
Rất lời	Bia lon Tiger	Thùng	60

(11 row(s) affected)

Thực tế thì những người lập trình trong môi trường Transaction–SQL thường sử dụng biểu thức **CASE** tìm kiếm bởi vì khi đó các biểu thức luận lý mà chúng ta dùng để so sánh được phép chứa nhiều toán tử so sánh khác nhau, trong khi đó biểu thức **CASE** đơn giản ở phần trên chỉ cho phép chúng ta thực hiện phép so sánh bằng (=) trên một biểu thức đơn giản mà thôi.

**Ví dụ:**

Để giảm giá bán hàng trong tháng 02/2002 theo qui tắc :

Nếu số lượng hàng  $\leq 2$  thì không giảm giá.

Nếu số lượng hàng từ 3 đến 10 thì giảm 10%.

Nếu số lượng hàng  $> 10$  thì giảm 20%.

Chúng ta sử dụng lệnh **UPDATE SET** có kết hợp biểu thức **CASE** tìm kiếm như sau:

```
UPDATE CTPXUAT
SET DGXUAT =
CASE
    WHEN SLXUAT<=2 THEN DGXUAT
    WHEN SLXUAT BETWEEN 3 AND 10 THEN DGXUAT*0.9
    ELSE DGXUAT*0.8
END
FROM CTPXUAT CTPX INNER JOIN PXUAT PX ON PX.SOPX=CTPX.SOPX
WHERE CONVERT(CHAR(7), NGAYXUAT, 21)="2002-02"
```

Tóm lại biểu thức **CASE** có thể được phép kết hợp sử dụng trong các câu lệnh **SELECT**, **UPDATE SET**, **DELETE** dùng để biện luận các trường hợp khác nhau của các giá trị dữ liệu bên trong một câu lệnh truy vấn.

## V. Cấu trúc điều khiển

Trong phần này chúng tôi giới thiệu cho các bạn hai cấu trúc điều khiển cơ bản nhất trong Transaction–SQL là: cấu trúc rẽ nhánh và cấu trúc lặp.

### V.1. Cấu trúc rẽ nhánh IF...ELSE

Với cấu trúc rẽ nhánh, người lập trình có thể chỉ định một hoặc nhiều câu lệnh sẽ được thực hiện khi giá trị của một biểu thức luận lý là đúng hoặc là sai. Cấu trúc rẽ nhánh được phép sử dụng bên trong một lô (batch) các lệnh hoặc bên trong một thủ tục nội tại. Cấu trúc rẽ nhánh **được phép lồng** nhiều cấp bên trong và cấp độ lồng nhau của các cấu trúc rẽ nhánh là **không** có giới hạn.

Cú pháp:

```
IF Biểu_thức_luận_lý
    Câu_lệnh1 | Khối_lệnh1
[ ELSE
    Câu_lệnh2 | Khối_lệnh2 ]
```

Trong đó:

- ✓ Biểu thức luận lý: thông thường là một biểu thức so sánh dùng để chỉ ra một điều kiện so sánh nào đó.
- ✓ Câu lệnh 1 | Khối lệnh 1: các lệnh sẽ được thực hiện khi biểu thức luận lý so sánh có giá trị là đúng (True).



- ✓ Câu lệnh 2 | Khối lệnh 2: các lệnh sẽ được thực hiện khi biểu thức luận lý so sánh có giá trị là sai (False).

### Lưu ý:

Khối lệnh mà chúng tôi đề cập trong đây là một tập hợp từ hai câu lệnh trở lên, trong các trường hợp này bắt buộc các bạn phải sử dụng các từ khóa **BEGIN** và **END** để hình thành một nhóm các câu lệnh trong một khối lệnh.

Cú pháp:

```
BEGIN
```

```
    Các_câu_lệnh_trong_khối_lệnh
```

```
END
```



### Ví dụ:

Tính xem có vật tư nào đã bán ra với số lượng nhiều hơn 4 không? Nếu có thì in ra danh sách các vật tư đó, ngược lại thì thông báo chưa bán được vật tư nào với số lượng nhiều hơn 4. Chúng ta sử dụng cú pháp **IF...ELSE** như sau:

```
IF (SELECT COUNT(*) FROM CTPXUAT
    WHERE SLXUAT>4) > 0
BEGIN
    PRINT "Danh sách các hàng hóa bán với số lượng > 4"
    SELECT CTPX.MAVTU, TENVTU, SLXUAT
    FROM CTPXUAT CTPX
    INNER JOIN VATTU VT
    ON VT.MAVTU=CTPX.MAVTU
    WHERE SLXUAT>4
END
ELSE
    PRINT "Chưa bán hàng hóa nào với số lượng >4"
```

Kết quả trả về

```
Danh sách các hàng hóa bán với số lượng > 4
MAVTU TENVTU                SLXUAT
-----
DD02  Đầu DVD Hitachi 3 đĩa    5
VD02  Đầu VCD Sony 3 đĩa      10

(2 row(s) affected)
```

Nhận xét thấy rằng trong thí dụ này chúng tôi sử dụng biểu thức luận lý là một biểu thức so sánh có sử dụng truy vấn **SELECT COUNT WHERE** dùng để đếm các hàng hóa bán ra có số lượng lớn hơn 4. Nếu kết quả đếm > 0 thì chúng ta in danh sách các vật tư đó ra. Ngược lại thông báo chưa bán hàng hóa với số lượng nhiều hơn 4.

Khi lập trình trong Transaction–SQL, thông thường chúng ta cần phải kiểm tra dữ liệu có tồn tại bên trong các bảng trước khi thực hiện tiếp các hành động liên quan đến các dòng dữ liệu đó.



Cú pháp **IF** có kết hợp từ khóa **EXISTS** dùng để kiểm tra sự tồn tại của các dòng dữ liệu bên trong bảng rất hữu hiệu.

Cú pháp:

```
IF EXISTS (Câu_lệnh_SELECT)
    Câu_lệnh1 | Khối_lệnh1
[ ELSE
    Câu_lệnh2 | Khối_lệnh2 ]
```

Trong đó:

- ✓ Từ khóa **EXISTS**: dùng để kiểm tra sự tồn tại các dòng dữ liệu trong câu lệnh truy vấn **SELECT** sau đó. Kết quả **IF** trả về đúng (True) khi câu lệnh truy vấn **SELECT** trả về ít nhất một dòng dữ liệu, ngược lại thì trả về sai (False)



**Ví dụ:**

Thực hiện lại thí dụ trên nhưng sử dụng cú pháp **IF EXISTS** dùng để kiểm tra xem đã có hàng hóa nào bán ra với số lượng nhiều hơn 4 chưa.

```
IF EXISTS (SELECT * FROM CTPXUAT WHERE SLXUAT>4)
BEGIN
    PRINT "Danh sách các hàng hóa bán với số lượng > 4"
    SELECT CTPX.MAVTU, TENVTU, SLXUAT
    FROM CTPXUAT CTPX INNER JOIN VATTU VT ON VT.MAVTU=CTPX.MAVTU
    WHERE SLXUAT>4
END
ELSE
    PRINT "Chưa bán hàng hóa nào với số lượng >4"
```



**Ví dụ:**

Để kiểm tra xem có phiếu nhập hàng nào đã lập vào ngày chủ nhật không? Nếu có thì in ra danh sách các phiếu nhập hàng đó. Chúng ta sử dụng cú pháp **IF** như sau:

```
IF EXISTS (SELECT * FROM PNHAP
    WHERE DATENAME(dw, NGAYNHAP)="Sunday")
BEGIN
    PRINT "Danh sách các phiếu nhập vào ngày chủ nhật"
    SELECT *
    FROM PNHAP
    WHERE DATENAME(dw, NGAYNHAP)="Sunday"
END
```

Nhận xét thấy rằng trong thí dụ này chúng tôi sử dụng hàm **DATENAME** với tham số **dw** (viết tắt của từ day of week – ngày trong tuần) dùng để trả về ngày trong tuần của ngày nhập hàng.



## V.2. Cấu trúc lặp WHILE

Với cấu trúc lặp, người lập trình có thể chỉ định một hoặc nhiều câu lệnh sẽ được **thực hiện lặp lại** nhiều lần **trong khi** mà khi giá trị của biểu thức luận lý so sánh vẫn còn **đúng**. Giống như cấu trúc rẽ nhánh, cấu trúc lặp được phép sử dụng bên trong một lô (batch) các lệnh hoặc bên trong một thủ tục nội tại. Giữa cấu trúc rẽ nhánh và cấu trúc lặp không có thứ tự ưu tiên khi chúng lồng vào nhau và cấp độ lồng nhau là **không** có giới hạn.

Thực tế việc sử dụng cấu trúc lặp **WHILE** bị giới hạn trong nhiều trường hợp. Bởi vì bản thân các lệnh truy vấn cập nhật dữ liệu như là: **SELECT, UPDATE SET, DELETE** trong Transaction-SQL đã tự động thực hiện việc lặp từ dòng dữ liệu đầu tiên đến dòng dữ liệu cuối cùng bên trong bảng.

Cấu trúc lặp **WHILE** thông thường được dùng với các biến có kiểu dữ liệu **cursor**, cách thức sử dụng biến kiểu dữ liệu cursor sẽ được trình bày trong phần kế tiếp của bài này.

Cú pháp:

```
WHILE Biểu_thức_luận_lý
BEGIN
    Các_lệnh_lặp
END
```

Trong đó:

- ✓ Biểu thức luận lý: thông thường là một biểu thức so sánh để chỉ các lệnh sẽ được lặp lại trong khi mà giá trị của biểu thức vẫn còn đúng.
- ✓ Các lệnh lặp: các câu lệnh được thực hiện bên trong vòng lặp.



### Ví dụ:

Để in ra 10 số nguyên dương bắt đầu từ 100. Chúng ta sử dụng cấu trúc lặp **WHILE** như sau:

```
DECLARE @Songuyen INT
SET @Songuyen=100
WHILE (@Songuyen<110)
BEGIN
    PRINT "Số nguyên : " + CONVERT(CHAR(3), @Songuyen)
    SET @Songuyen = @Songuyen + 1
END
```

Kết quả trả về

```
Số nguyên : 100
Số nguyên : 101
Số nguyên : 102
Số nguyên : 103
Số nguyên : 104
Số nguyên : 105
```



Số nguyên : 106  
Số nguyên : 107  
Số nguyên : 108  
Số nguyên : 109

Chúng ta có thể sử dụng từ khóa **BREAK** lồng vào cấu trúc lặp **WHILE** để có thể kết thúc việc lặp của các lệnh bên trong vòng lặp mà **không cần** xét đến giá trị trả về của biểu thức luận lý dùng để so sánh phía sau từ khóa **WHILE** phải là **sai**. Tuy nhiên từ khóa **BREAK** thường được sử dụng kèm theo với một biểu thức luận lý khác.



**Ví dụ:**

Thực hiện việc lặp giống thí dụ trên, tuy nhiên muốn rằng vòng lặp sẽ bị kết thúc khi mới in tới số nguyên 105. Chúng ta sử dụng cấu trúc lặp **WHILE** như sau:

```
DECLARE @Songuyen INT
SET @Songuyen=100
WHILE (@Songuyen<110)
BEGIN
    PRINT "Số nguyên : " + CONVERT(CHAR(3), @Songuyen)
    IF @Songuyen=105
        BREAK
    SET @Songuyen = @Songuyen + 1
END
PRINT "Kết thúc vòng lặp"
```

**Kết quả trả về**

Số nguyên : 100  
Số nguyên : 101  
Số nguyên : 102  
Số nguyên : 103  
Số nguyên : 104  
Số nguyên : 105  
Kết thúc vòng lặp

Hơn thế nữa chúng ta có thể sử dụng từ khóa **CONTINUE** lồng vào cấu trúc lặp **WHILE** để chỉ định các lệnh bên trong vòng lặp ở **phía dưới** từ khóa **CONTINUE** tạm thời không thực hiện tiếp, khi đó con trỏ vòng lặp sẽ nhảy về đầu vòng lặp để kiểm tra giá trị của biểu thức luận lý so sánh là vẫn còn đúng hay không. Tuy nhiên từ khóa **CONTINUE** thông thường được dùng kèm theo với một biểu thức luận lý khác.



**Ví dụ:**

Thực hiện việc lặp giống các thí dụ trên, tuy nhiên muốn rằng vòng lặp sẽ in xốt số nguyên 105. Chúng ta sử dụng cấu trúc lặp **WHILE** như sau:

```
DECLARE @Songuyen INT
SET @Songuyen=99
WHILE (@Songuyen<110)
```



```

BEGIN
    SET @Songuyen = @Songuyen + 1
    IF @Songuyen=105
        CONTINUE
    PRINT "Số nguyên : " + CONVERT(CHAR(3), @Songuyen)
END
PRINT "Kết thúc vòng lặp"

```

*Kết quả trả về*

```

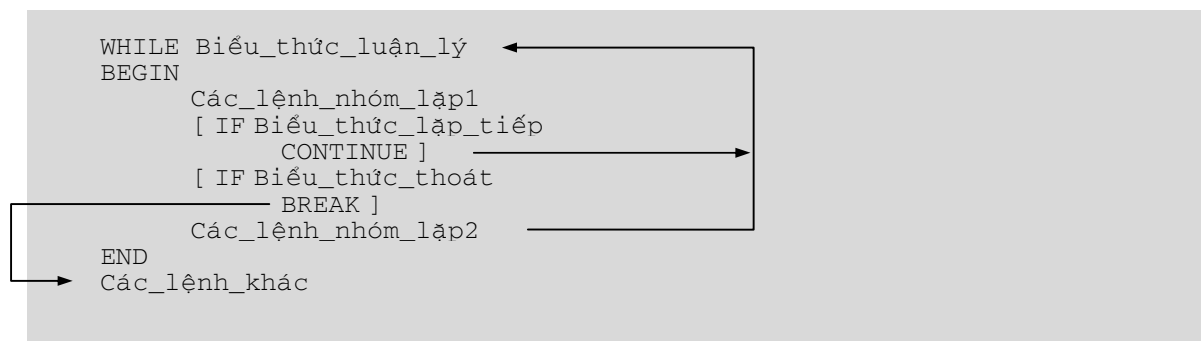
Số nguyên : 100
Số nguyên : 101
Số nguyên : 102
Số nguyên : 103
Số nguyên : 104
Số nguyên : 106
Số nguyên : 107
Số nguyên : 108
Số nguyên : 109
Số nguyên : 110
Kết thúc vòng lặp

```

Kiểm chứng kết quả trả về chúng ta suy luận khi vòng lặp thực hiện đến giá trị của biến @Songuyen = 105 thì khi đó lệnh **PRINT “Số nguyên : ”**... không được thực hiện và con trỏ chương trình được quay lên đầu vòng lặp để kiểm tra tiếp biểu thức luận lý so sánh.

Để giúp các bạn hiểu rõ ý nghĩa cú pháp đầy đủ của cấu trúc lặp **WHILE** kèm với các từ khóa **CONTINUE** hoặc **BREAK**, chúng tôi xin minh họa sơ đồ tóm tắt bên dưới.

Cú pháp:



#### Ví dụ:

Để tăng tự động tỷ lệ phần trăm cho các vật tư trong bảng VATTU theo qui tắc sau:

Mỗi lần chỉ tăng lên 5% cho các vật tư có giá trị tại cột tỷ lệ nhỏ hơn 30%.

Lặp lại hành động tăng trong khi mà giá trị trung bình tỷ lệ phần trăm của các vật tư vẫn còn thấp hơn 40%.

Chúng ta sử dụng các lệnh như sau:

```

--In giá trị trung bình tỷ lệ trước khi tăng
PRINT "Trung bình tỷ lệ phần trăm trước khi tăng"

```



```

SELECT AVG(PHANTRAM) FROM VATTU
--Bắt đầu thực hiện việc lặp tăng tự động
DECLARE @Lantang INT
SET @Lantang=0
WHILE (SELECT AVG(PHANTRAM) FROM VATTU)<40
BEGIN
    UPDATE VATTU
    SET PHANTRAM = PHANTRAM + 5
    WHERE PHANTRAM<30
    SET @Lantang = @Lantang+1
    IF NOT EXISTS (SELECT * FROM VATTU
                    WHERE PHANTRAM<30)
        BREAK
END
--In giá trị trung bình tỷ lệ sau khi tăng
PRINT "Đã tăng :" + CONVERT(VARCHAR(7), @Lantang) + " lần"
PRINT "Trung bình tỷ lệ phần trăm sau khi tăng"
SELECT AVG(PHANTRAM) FROM VATTU

```

Nhận xét thấy rằng trong thí dụ này khá phức tạp, do thế chúng tôi phải chèn thêm vào các ghi chú (comment) để giúp các bạn thấy được từng xử lý rời rạc nhằm **để xem, để hiểu**. Trong Transaction–SQL chúng ta có thể chèn các ghi chú trong các câu lệnh bằng hai dấu trừ liên tiếp nhau hoặc muốn che lại một khối các lệnh liên tiếp nhau thì các bạn sẽ sử dụng cặp ký tự như bên dưới.

```

/*
Các lệnh được che lại
*/

```

Ngoài ra trong vòng lặp chúng tôi có sử dụng lệnh **IF EXISTS** dùng để kiểm tra trường hợp sau khi **đã tăng hết tất cả** tỷ lệ phần trăm các vật tư với **đều lớn hơn 30** mà trung bình tỷ lệ phần trăm của các vật tư vẫn **chưa lớn hơn 40** thì bắt buộc vòng lặp phải được thoát ra ngoài, bởi vì nếu không thì vòng lặp sẽ bị lặp vô tận không bao giờ thoát ra được. Hy vọng qua thí dụ trên các bạn đã hiểu rõ về các thành phần bên trong cấu trúc lặp **WHILE**.

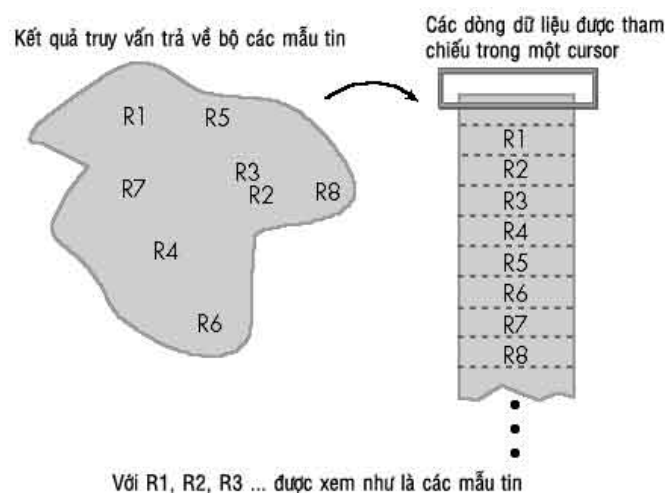
## VI. Sử dụng biến kiểu dữ liệu cursor

Phần lớn các cơ sở dữ liệu quan hệ thường làm việc trên dữ liệu của nhiều dòng mẫu tin – còn gọi là một bộ các mẫu tin, thí dụ lệnh SELECT kết quả luôn trả về nhiều dòng dữ liệu hơn là một dòng dữ liệu. Ngược lại, đối với một số ngôn ngữ lập trình hoặc bên trong các ứng dụng thì người lập trình vẫn còn các thói quen xử lý và tính toán dữ liệu trên từng dòng riêng lẻ. Để đáp ứng được yêu cầu này của các người lập trình – muốn làm việc chỉ trên từng dòng dữ liệu tại thời điểm hiện hành, Microsoft SQL Server tạo ra một kiểu dữ liệu đó chính là kiểu cursor.

## VI.1. Khái niệm về cursor

Các bạn có thể hình dung kiểu dữ liệu cursor (chúng tôi tạm dịch là kiểu con trỏ) giống như một cuốn sổ danh bạ chứa thông tin liên lạc của các khách hàng giao dịch trong một công ty. Bằng cách dò tìm thủ công chúng ta sẽ phải sử dụng đến **mắt và tay** để tham chiếu đến tên của các khách hàng bất kỳ trong sổ danh bạ đó. Chúng ta có thể di chuyển lên, xuống hoặc qua trang để tìm ra các khách hàng mong muốn, nhưng tại thời điểm hiện hành tay và mắt của chúng ta chỉ đứng tại một khách hàng mà thôi.

Hoạt động của kiểu dữ liệu cursor trong Transaction–SQL hoàn toàn giống như thí dụ minh họa ở trên. Tuy nhiên cursor có nhiều kiểu khác nhau cho phép các bạn có thể chọn lựa để định nghĩa theo đúng yêu cầu mà mình mong muốn. Tùy thuộc vào kiểu cursor đã định nghĩa mà việc đọc và cập nhật dữ liệu sẽ có hiệu lực như thế nào.



**Hình 4-3.** Hình ảnh minh họa so sánh cơ chế cursor và bộ các mẫu tin.

## VI.2. Các bước sử dụng kiểu dữ liệu cursor

Đối với các kiểu dữ liệu thông thường sau khi khai báo biến cùng với kiểu dữ liệu thích hợp, chúng ta sẽ được phép gán giá trị cần lưu trữ vào bên trong biến. Hoạt động của biến kiểu dữ liệu cursor hoàn toàn **không đơn giản** như thế, để sử dụng được biến kiểu dữ liệu cursor các bạn phải thực hiện một cách thủ tục qua nhiều bước khác nhau.

Trong phần này chúng tôi sẽ trình bày chi tiết các bước thực hiện khi sử dụng biến kiểu dữ liệu cursor trong Transaction–SQL. Để làm việc với biến có kiểu cursor các bạn phải thực hiện từng bước như sau:

- ✓ Định nghĩa biến kiểu cursor bằng lệnh DECLARE.
- ✓ Sử dụng lệnh OPEN để mở ra cursor đã định nghĩa trước đó.
- ✓ Đọc và xử lý trên từng dòng dữ liệu bên trong cursor.
- ✓ Đóng cursor lại bằng lệnh CLOSE và DEALLOCATE.

Định nghĩa biến có kiểu cursor



Việc định nghĩa biến có kiểu cursor trong Transaction–SQL là việc chỉ định đến những dòng dữ liệu có bên trong các bảng dữ liệu nào mà biến sẽ tham chiếu đến. Thông thường trong lệnh định nghĩa biến có kiểu cursor bên dưới sẽ chỉ định ra **loại** của cursor cho các mục đích sử dụng về sau này thuận tiện hơn.

Cú pháp:

```
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_cộtcn]]
```

Trong đó:

- ✓ Tên cursor: tên của biến kiểu cursor.
- ✓ Từ khóa LOCAL | GLOBAL: dùng chỉ định phạm vi hoạt động của biến cursor hoặc là cục bộ (local) bên trong một thủ tục, lô (batch) các lệnh, một trigger hoặc là toàn cục (global) bên trong một kết nối. Một biến cursor có tính toàn cục sẽ được phép tham chiếu trong bất kỳ thủ tục nào của kết nối tạo ra biến cursor đó.
- ✓ Từ khóa FORWARD\_ONLY: dùng chỉ định việc đọc dữ liệu trong cursor chỉ theo chiều đi tới mà thôi (duyệt từ mẫu tin đầu tiên đến mẫu tin cuối cùng)
- ✓ Từ khóa SCROLL: dùng chỉ định việc đọc dữ liệu trong cursor được phép di chuyển tới lui, qua lại các dòng mẫu tin bên trong cursor tùy thích.
- ✓ Từ khóa STATIC: dùng chỉ định dữ liệu đọc bên trong cursor là tĩnh. Khi đó nếu những người dùng khác có các thay đổi ở bên dưới dữ liệu gốc (base table) thì các thay đổi đó sẽ không được cập nhật tự động trong dữ liệu của cursor. Bởi vì khi đó dữ liệu trong cursor chính là dữ liệu của một bảng tạm đã được hệ thống sao chép và lưu trữ trong cơ sở dữ liệu tempdb của hệ thống khi định nghĩa cursor.
- ✓ Từ khóa DYNAMIC: dùng chỉ định dữ liệu bên trong cursor là động. Khi đó việc cập nhật dữ liệu trong bảng cơ sở (base table) bởi những người dùng khác sẽ được cập nhật tự động trong dữ liệu cursor có kiểu là DYNAMIC.
- ✓ Từ khóa KEYSET: có hoạt động gần giống với kiểu DYNAMIC, các thay đổi dữ liệu trên các cột không là khóa chính trong bảng cơ sở bởi những người dùng khác sẽ được cập nhật trong dữ liệu cursor. Tuy nhiên đối với các mẫu tin vừa thêm mới hoặc các mẫu tin đã bị hủy bỏ bởi những người dùng khác sẽ không được hiển thị trong dữ liệu cursor có kiểu là KEYSET.
- ✓ Từ khóa READ\_ONLY: dùng chỉ định dữ liệu bên trong cursor là chỉ đọc nhằm hạn chế việc sửa đổi dữ liệu bên trong cursor. Khi khai báo cursor với kiểu dữ liệu là tĩnh (STATIC) thì dữ liệu trong cursor xem như là chỉ đọc.
- ✓ Từ khóa SCROLL\_LOCK: dùng chỉ định hệ thống Microsoft SQL Server tự động khóa các dòng mẫu tin cần phải thay đổi giá trị hoặc bị hủy bỏ bên trong bảng nhằm đảm bảo các hành động cập nhật luôn luôn thành công.



- ✓ Câu lệnh SELECT: dùng để chỉ đến các cột có bên trong bảng mà chúng ta cần đọc dữ liệu. Câu lệnh SELECT trong cursor không thể chứa các mệnh đề: INTO, COMPUTE, COMPUTE BY.
- ✓ Danh sách các cột cập nhật: chỉ định danh sách tên các cột sẽ được phép thay đổi giá trị trong cursor. Mặc định tất cả các cột trong mệnh đề SELECT sẽ được phép thay đổi giá trị nếu dữ liệu cursor không phải là chỉ đọc.

**Ví dụ:**

Để định nghĩa một biến cursor chứa toàn bộ các dòng dữ liệu bên trong bảng VATTU, các dòng dữ liệu trong cursor cho phép được cập nhật. Chúng ta sử dụng lệnh khai báo biến cursor như sau:

```
DECLARE cur_Vattu CURSOR
DYNAMIC
FOR
SELECT * FROM VATTU
```

**Ví dụ:**

Để định nghĩa một biến cursor chứa toàn bộ các dòng dữ liệu bên trong bảng NHACC, các dữ liệu trong cursor chỉ được phép đọc và việc đọc dữ liệu trong cursor chỉ theo một chiều tới. Chúng ta sử dụng lệnh khai báo biến cursor như sau:

```
DECLARE cur_Nhacc CURSOR
FORWARD_ONLY
STATIC
READ_ONLY
FOR
SELECT * FROM NHACC
```

Nhận xét thấy rằng ở thí dụ trên chúng ta có thể bỏ đi từ khóa READ\_ONLY bởi vì bản thân từ khóa STATIC đã định nghĩa dữ liệu của cursor là chỉ đọc. Tuy nhiên chúng tôi vẫn khuyến cáo các bạn nên ghi nhớ ý nghĩa của từng từ khóa riêng lẻ để định nghĩa ra dữ liệu của các cursor đúng với yêu cầu mà mình cần sử dụng.

**VI.2.1. Mở cursor**

Để có thể đọc được các dòng dữ liệu bên trong cursor trước tiên các bạn cần phải mở cursor ra bằng lệnh **OPEN**. Hoạt động bên trong của lệnh này thực ra là hệ thống sẽ thực hiện câu lệnh truy vấn **SELECT** đã được chỉ định trong lệnh định nghĩa biến cursor trước đó.

Trong trường hợp nếu chúng ta định nghĩa sử dụng cursor với kiểu **STATIC** hoặc **KEYSET** thì hệ thống sẽ tạo ra bảng tạm chứa các dữ liệu kết quả của lệnh **SELECT** nằm trong cơ sở dữ liệu **tempdb**. Cú pháp lệnh **OPEN** khá đơn giản được mô tả như bên dưới.

Cú pháp:

```
OPEN Tên_cursor
```

Trong đó:



- ✓ Tên cursor: tên của biến kiểu cursor đã định nghĩa trước đó bằng lệnh DECLARE.

**Ví dụ:**

Để mở các cursor đã định nghĩa ở các thí dụ trên. Chúng ta sử dụng lệnh **OPEN** như sau:

```
OPEN cur_Vattu
```

Hoặc

```
OPEN cur_Nhacc
```

**VI.2.2. Đọc và xử lý dữ liệu trong cursor**

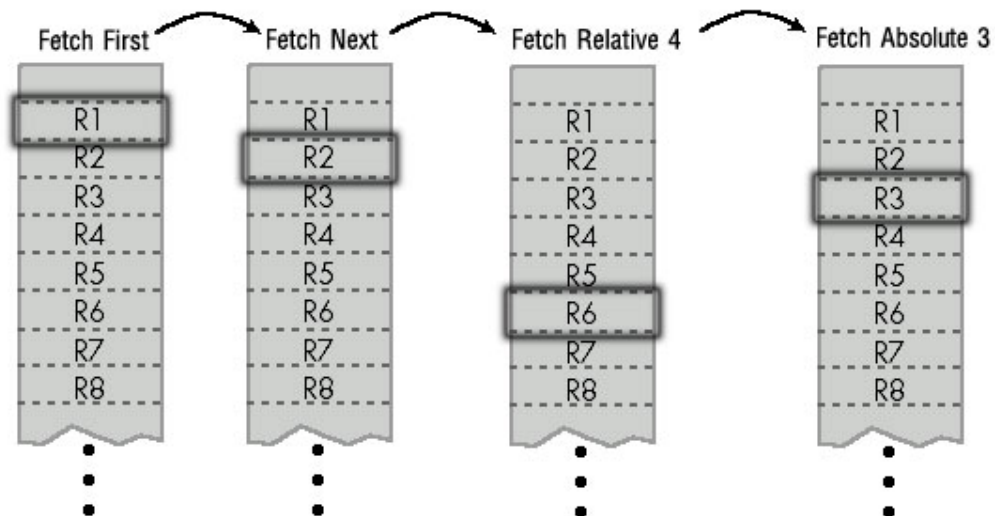
Sau khi định nghĩa và mở biến cursor, hành động kế tiếp mà chúng ta thường thực hiện là việc đọc và xử lý dữ liệu trong cursor. Chúng ta sử dụng lệnh **FETCH** cùng với các từ khóa tương ứng để chỉ định việc đọc các dòng dữ liệu bên trong cursor theo một thứ tự nào. Cú pháp lệnh **FETCH** được mô tả chi tiết bên dưới.

Cú pháp:

```
FETCH [NEXT | PRIOR | FIRST | LAST  
      | ABSOLUTE n | RELATIVE n]  
FROM Tên_cursor  
[INTO Danh_sách_biến]
```

Trong đó:

- ✓ Các từ khóa NEXT, PRIOR, FIRST, LAST: dùng để đọc dữ liệu của dòng dữ liệu kế tiếp (next), dòng dữ liệu phía trước (prior), dòng dữ liệu đầu tiên (first), dòng dữ liệu cuối cùng (last) so với dòng dữ liệu hiện hành bên trong cursor. Sau khi đọc dữ liệu thành công, dòng dữ liệu hiện hành sẽ bị thay đổi chính là dòng vừa mới được đọc.
- ✓ Từ khóa ABSOLUTE: dùng để chỉ định việc đọc dòng dữ liệu chính xác thứ n bên trong cursor. Với n là số nguyên dương dùng chỉ định việc đọc dữ liệu tại dòng thứ n được đếm từ dòng đầu tiên, với n là số nguyên âm dùng chỉ định việc đọc dữ liệu tại dòng thứ n được đếm ngược từ dòng cuối cùng trở lên.
- ✓ Từ khóa RELATIVE: dùng để chỉ định việc đọc dữ liệu tại một dòng tương đối so với dòng dữ liệu hiện hành. Với n là một số nguyên có thể dương hoặc âm để chỉ định việc đọc theo chiều tới hoặc lui so với dòng dữ liệu hiện hành.
- ✓ Tên cursor: tên của biến kiểu cursor đã định nghĩa trước đó bằng lệnh DECLARE.
- ✓ Danh sách biến: danh sách tên các biến cục bộ đã được định nghĩa trước đó. Các biến này sẽ lưu trữ các giá trị dữ liệu được đọc từ lệnh FETCH.



**Hình 4-4.** Hình ảnh minh hoạt việc đọc dữ liệu theo các thứ tự khác nhau.

Trong quá trình đọc và xử lý các dòng dữ liệu không đảm bảo luôn luôn là thành công bởi vì có sự truy cập bởi những người dùng khác hoặc một lý do khác. Do đó hệ thống Microsoft SQL Server cung cấp cho các bạn một biến hệ thống có tên là **@@FETCH\_STATUS** dùng để kiểm tra tình trạng đọc dữ liệu là thành công hay thất bại. Giá trị của biến trả về 0 khi việc đọc dữ liệu là thành công.



**Ví dụ:**

Để đọc dữ liệu cursor bảng VATTU chỉ lọc các vật tư là Tivi. Chúng ta sử dụng các lệnh như sau:

```
--1. Khai báo biến cursor
DECLARE cur_Vattu CURSOR KEYSET
FOR
    SELECT * FROM VATTU
    WHERE MAVTU LIKE "TV%"
    ORDER BY MAVTU

--2. Mở cursor
OPEN cur_Vattu

--3. Đọc dữ liệu
FETCH NEXT FROM cur_Vattu
WHILE @@FETCH_STATUS = 0
BEGIN
    -- Đọc tiếp các dòng kế
    FETCH NEXT FROM cur_Vattu
    -- Thực hiện đọc tiếp các dòng kế
    -- trong khi việc đọc dữ liệu thành công
END
```



```
--4. Đóng cursor
CLOSE cur_Vattu
DEALLOCATE cur_Vattu
```

*Kết quả trả về*

MAVTU	TENVTU	DVTINH	PHANTRAM
TV14	Tivi Sony 14 inches	Cái	20
(1 row(s) affected)			
MAVTU	TENVTU	DVTINH	PHANTRAM
TV21	Tivi Sony 21 inches	Cái	15
(1 row(s) affected)			
MAVTU	TENVTU	DVTINH	PHANTRAM
TV29	Tivi Sony 29 inches	Cái	15
(1 row(s) affected)			

Nhận xét thấy rằng trong thí dụ trên chúng tôi có ghi chú các bước thực hiện khi sử dụng biến cursor nhằm giúp các bạn dễ đọc, dễ hiểu. Việc đọc dữ liệu được thực hiện trong vòng lặp **WHILE** nhằm tìm ra tất cả các vật tư là Tivi, sử dụng điều kiện lặp **@@FETCH\_STATUS=0** để nói rằng nếu việc đọc dữ liệu của lệnh **FETCH NEXT** thành công thì sẽ tiếp tục lặp. Hai lệnh cuối cùng **CLOSE** và **DEALLOCATE** dùng để đóng lại cursor sau khi đã đọc và xử lý dữ liệu xong.



**Ví dụ:**

Để cập nhật giá trị dữ liệu cho cột **TGNHAP** (trị giá nhập) trong bảng **PNHAP** bằng cách duyệt qua từng phiếu nhập, tính ra trị giá nhập của từng phiếu căn cứ vào số lượng nhập và đơn giá nhập của từng vật tư trong bảng **CTPNHAP**, sau cùng cập nhật vào cột **TGNHAP**. Chúng ta sử dụng các lệnh như sau để thực hiện các hành động mô tả như trên:

```
--1. Khai báo biến cursor, các biến cục bộ
DECLARE @sSopn CHAR(4), @nTongtg MONEY
DECLARE cur_Pnhap CURSOR
FORWARD_ONLY
FOR
    SELECT SOPN
    FROM PNHAP

--2. Mở cursor
OPEN cur_Pnhap

--3. Đọc dữ liệu và cập nhật giá trị
WHILE 0=0
```





```

BEGIN
    FETCH NEXT FROM cur_Pnhap
    INTO @sSopn
    IF @@FETCH_STATUS<>0
        BREAK

    SELECT @nTongtg = SUM(SLNNHAP*DGNHAP)
    FROM CTPNHAP
    WHERE SOPN=@sSopn

    PRINT "Đang cập nhật phiếu nhập : " + @sSopn + " ..."
    UPDATE PNHAP
    SET TGNHAP = @nTongtg
    WHERE CURRENT OF cur_Pnhap
END

--4. Đóng cursor
CLOSE cur_Pnhap
DEALLOCATE cur_Pnhap

```

Nhận xét thấy rằng trong thí dụ này chúng tôi sử dụng vòng lặp **WHILE** mà điều kiện lặp là **0=0** để chỉ định điều kiện so sánh vòng lặp là **luôn luôn đúng**, do đó bên trong vòng lặp này chúng ta bắt buộc phải thoát khỏi vòng lặp bằng lệnh **BREAK** và điều kiện thoát là khi việc đọc dữ liệu bị lỗi (**@@FETCH\_STATUS<>0**). Ngoài ra việc cập nhật dữ liệu bằng lệnh **UPDATE** mà trong mệnh đề **WHERE** chúng tôi có sử dụng từ khóa **CURRENT OF <tên cursor>** dùng để chỉ định việc cập nhật dữ liệu trên dòng dữ liệu hiện hành của cursor.

Đóng cursor

Đây là công việc **sau cùng** cần phải thực hiện khi sử dụng biến có kiểu cursor. Thông thường để đóng cursor chúng ta phối hợp cả hai lệnh mô tả như bên dưới.

Cú pháp:

```

CLOSE    Tên_cursor
DEALLOCATE    Tên_cursor

```

Trong đó:

✓ Tên cursor: tên của biến kiểu cursor đã được định nghĩa và mở ra trước đó.

Lệnh **CLOSE** chỉ là thực hiện hành động giải phóng các dòng dữ liệu tham chiếu bên trong biến cursor, chúng ta có thể mở lại cursor mà **không** cần thiết phải định nghĩa lại biến cursor bằng lệnh **DECLARE**. Tuy nhiên việc đọc dữ liệu sẽ không còn là hợp lệ nữa sau khi chúng ta đã ra lệnh **CLOSE** để đóng cursor.

Lệnh **DEALLOCATE** để giải phóng thật sự biến cursor ra khỏi bộ nhớ. Sau khi thực hiện lệnh



này, nếu có lệnh nào tham chiếu đến tên cursor đều sẽ gây ra lỗi. Để giúp các bạn hiểu rõ các bước thực hiện khi sử dụng biến có kiểu dữ liệu cursor, chúng tôi xin minh họa sơ đồ tóm tắt bên dưới.

```
--1. Khai báo biến cursor
DECLARE Tên_cursor CURSOR
[Kiểu_đọc | cập_nhật_dữ_liệu]
FOR
    Câu_lệnh_SELECT

--2. Mở cursor
OPEN Tên_cursor

--3. Đọc dữ liệu và cập nhật giá trị
WHILE 0=0
BEGIN
    FETCH NEXT FROM <Tên_cursor>
    [ INTO Danh_sách_biến ]
    IF @@FETCH_STATUS<>0
        BREAK
    --Cập nhật dữ liệu trong cursor
END

--4. Đóng cursor
CLOSE Tên_cursor
DEALLOCATE Tên_cursor
```

Các bạn thấy rằng tiện ích duy nhất khi làm việc với cursor là chúng cho phép các bạn có thể **duyet từng tự trên các dòng dữ liệu** (giống như đối tượng recordset của ADO hoặc DAO trong ngôn ngữ lập trình Visual Basic), tuy nhiên Microsoft SQL Server vẫn khuyến cáo chúng ta **không nên** lạm dụng quá nhiều các hành động cập nhật dữ liệu bằng cursor bởi vì nó sẽ làm cho các xử lý này chậm. Bằng chứng là chúng ta có thể tính toán cho giá trị cho cột trị giá nhập (TGNHAP) trong bảng PNHAP bằng một lệnh UPDATE duy nhất thay vì phải sử dụng quá nhiều lệnh cho các xử lý trong cursor (xem thí dụ ở trên).



**Ví dụ:**

```
UPDATE PNHAP
SET TGNHAP = (SELECT SUM(SLNHAP*DGNHAP)
              FROM CTPNHAP CTPN
              WHERE PN.SOPN=CTPN.SOPN )
FROM PNHAP PN
```

Tóm lại vậy thì khi nào chúng ta sẽ sử dụng kiểu dữ liệu cursor trong Transaction–SQL để giải quyết các vấn đề? Câu trả lời tốt nhất cho các bạn là:

- ✓ Đầu tiên xin nhớ rằng Microsoft SQL Server là một hệ quản trị cơ sở dữ liệu quan hệ (Relational DataBase Management System) do đó chúng luôn chọn các giải pháp làm việc trên bộ các mẫu tin.
- ✓ Kế tiếp khi cần giải quyết các vấn đề cập nhật dữ liệu thì các bạn luôn luôn ưu tiên chọn ra các hướng giải quyết trên bộ các mẫu tin bởi vì khi đó sẽ làm cho các xử lý được nhanh hơn.
- ✓ Sau cùng các hướng giải quyết theo kiểu cursor chỉ là giải pháp sau cùng nhất để chọn lựa khi không còn giải pháp nào tốt hơn.



## VII. Các hàm thường dùng

### VII.1. Các hàm chuyển đổi kiểu dữ liệu

Công dụng của các hàm này dùng để chuyển đổi qua lại các kiểu dữ liệu tương thích nhau bên trong Microsoft SQL Server. Thông thường trong các xử lý chúng ta thường chuyển đổi các kiểu dữ liệu số hoặc kiểu dữ liệu ngày giờ về kiểu dữ liệu chuỗi để hiển thị ra màn hình.

#### VII.1.1. Hàm CAST

Với cú pháp hàm **CAST** bên dưới cho phép các bạn có thể chuyển đổi một biểu thức nào đó sang một kiểu dữ liệu bất kỳ mong muốn. Thông thường đối với các kiểu dữ liệu **image**, **text**, **ntext** rất hạn chế trong việc chuyển đổi qua lại các kiểu dữ liệu khác.

Cú pháp:

```
CAST (Biểu_thức AS Kiểu_dữ_liệu)
```

Trong đó:

- ✓ Biểu thức: là tên của một cột trong bảng hoặc một biểu thức tính toán cần chuyển sang kiểu dữ liệu mới.
- ✓ Kiểu dữ liệu: tên kiểu dữ liệu mới mà biểu thức sẽ được chuyển đổi sang.



#### Ví dụ:

Để hiển thị danh sách các vật tư có trong bảng VATTU, trong đó cột tỷ lệ phần trăm được hiển thị theo dạng xxx%. Chúng ta sử dụng hàm **CAST** để chuyển đổi giá trị cột phần trăm từ kiểu dữ liệu số sang kiểu dữ liệu chuỗi và sử dụng toán tử cộng chuỗi (+) để nối thêm ký tự %.

```
SELECT MAVTU, TENVTU,
       TYLE=CAST(PHANTHAM AS VARCHAR(3))+ "%"
FROM VATTU
```

Kết quả truy vấn trả về

MAVTU	TENVTU	TYLE
BI01	Bia lon Tiger	60%
DD01	Đầu DVD Hitachi 1 đĩa	40%
DD02	Đầu DVD Hitachi 3 đĩa	40%
LO01	Loa Panasonic 1000W	10%
TL15	Tủ lạnh Sanyo 150 lit	25%
TL90	Tủ lạnh Sanyo 90 lit	20%
TV14	Tivi Sony 14 inches	20%
TV21	Tivi Sony 21 inches	15%
TV29	Tivi Sony 29 inches	15%
VD01	Đầu VCD Sony 1 đĩa	30%
VD02	Đầu VCD Sony 3 đĩa	30%

(11 row(s) affected)



### VII.1.2. Hàm CONVERT

Với cú pháp hàm **CONVERT** bên dưới cho phép các bạn có thể chuyển đổi một biểu thức nào đó sang một kiểu dữ liệu bất kỳ mong muốn nhưng có thể theo một **định dạng** nào đó (đặc biệt đối với kiểu dữ liệu ngày).

Cú pháp:

CONVERT (Kiểu\_dữ\_liệu, Biểu\_thức [, Định\_dạng])

Trong đó:

- ✓ Kiểu dữ liệu: tên kiểu dữ liệu mà biểu thức sẽ được chuyển đổi sang.
- ✓ Biểu thức: là tên của cột bên trong bảng hoặc một biểu thức tính toán muốn chuyển sang kiểu dữ liệu mới.
- ✓ Định dạng: là một con số chỉ định việc định dạng cho việc chuyển đổi dữ liệu từ dạng ngày sang dạng chuỗi. Bảng bên dưới mô tả một số định dạng thường dùng trong hàm CONVERT.

Định dạng năm (yy)	Định dạng năm (yyyy)	Hiển thị dữ liệu
1	101	mm/dd/yy
2	102	yy.mm.dd
3	103	dd/mm/yy
4	104	dd.mm.yy
5	105	dd-mm-yy
6	106	dd mon yy
7	107	mon dd,yy
8	108	hh:mm:ss
9	109	mon dd yyyy hh:mm:ss
10	110	mm-dd-yy
11	111	yy/mm/dd
12	112	yymmdd
13	113	dd mon yyyy hh:mm:ss
14	114	hh:mm:ss:mmm
	21 hoặc 121	yyyy-mm-dd hh:mi:ss.mmm
	20 hoặc 120	yyyy-mm-dd hh:mi:ss

**Ví dụ:**

Để hiển thị chi tiết và đếm tổng số các đơn đặt hàng theo từng **năm tháng**. Chúng ta sử dụng hàm **CONVERT** để chuyển đổi giá trị cột ngày đặt hàng từ kiểu dữ liệu ngày sang chuỗi.

```
SELECT NAMTHANG=CONVERT(CHAR(6), NGAYDH, 112),
       SODH, CONVERT(CHAR(10), NGAYDH, 103) AS NGAYDH
FROM DONDH
ORDER BY NAMTHANG
COMPUTE COUNT(SODH) BY NAMTHANG
```

Kết quả truy vấn trả về

```
NAMTHANG SODH NGAYDH
-----
200201   D001 15/01/2002
200201   D002 30/01/2002
          cnt
          =====
          2
200202   D003 10/02/2002
200202   D004 17/02/2002
          cnt
          =====
          2
200203   D005 01/03/2002
200203   D006 12/03/2002
          cnt
          =====
          2
(9 row(s) affected)
```

**VII.1.3. Hàm STR**

Với cú pháp hàm **STR** bên dưới cho phép các bạn có thể chuyển đổi kiểu dữ liệu số sang kiểu dữ liệu chuỗi. Phải đảm bảo đủ vùng trắng để chứa các ký số khi chuyển đổi sang kiểu dữ liệu chuỗi.

Cú pháp:

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

Trong đó:

- ✓ Số thực: là một biểu thức có kiểu dữ liệu số thực.
- ✓ Số ký tự: số vùng trắng dành để chứa các ký số sau khi chuyển sang kiểu dữ liệu chuỗi.
- ✓ Số lẻ: chỉ định số thập phân.
- ✓ Chuỗi: là chuỗi ký tự kết quả chứa các ký số sau khi đã chuyển đổi kiểu dữ liệu số thành

kiểu dữ liệu chuỗi.



**Ví dụ:**

Để liệt kê danh sách các vật tư đã nhập trong tháng 01/2002 có hiển thị thêm cột đơn vị tính. Chúng ta sử dụng hàm **STR** để chuyển đổi giá trị cột số lượng nhập từ kiểu dữ liệu số sang kiểu dữ liệu chuỗi và kết hợp toán tử cộng chuỗi (+) để nối thêm giá trị dữ liệu cột đơn vị tính.

```
SELECT PN.SOPN, SLNHAP=STR(SLNHAP, 5)+" "+DVTINH, TENVTU
FROM CTPNHAP CTPN INNER JOIN VATTU VT ON VT.MAVTU=CTPN.MAVTU
INNER JOIN PNHAP PN ON PN.SOPN=CTPN.SOPN
WHERE CONVERT(CHAR(6), NGAYNHAP, 112)="200201"
```

Kết quả truy vấn trả về

SOPN	SLNHAP	TENVTU
N001	8 Bộ	Đầu DVD Hitachi 1 đĩa
N001	10 Bộ	Đầu DVD Hitachi 3 đĩa
N002	2 Bộ	Đầu DVD Hitachi 1 đĩa
N002	5 Bộ	Đầu DVD Hitachi 3 đĩa

(5 row(s) affected)

## VII.2. Các hàm ngày giờ

Các hàm này thường có tham số vào là kiểu dữ liệu ngày giờ và giá trị trả về của chúng có thể là kiểu dữ liệu số, chuỗi hoặc ngày giờ. Bảng bên dưới mô tả từ viết tắt của các **đơn vị thời gian** được dùng cho các tham số trong một số các hàm ngày giờ.

Từ viết tắt	Ý nghĩa chỉ định	Miền giá trị
<i>yy</i>	Năm	1900-9999
<i>qq</i>	Quý	1-4
<i>mm</i>	Tháng	1-12
<i>dy</i>	Ngày trong năm	1-366
<i>dd</i>	Ngày trong tháng	1-31
<i>wk</i>	Tuần	1-53
<i>dw</i>	Ngày trong tuần	1-7 (Chủ nhật-thứ bảy)
<i>hh</i>	Giờ trong ngày	0-23
<i>mi</i>	Phút trong giờ	0-59
<i>ss</i>	Giây trong phút	0-59
<i>ms</i>	Phần trăm mili giây	0-999

### VII.2.1. Hàm DATEADD

Với cú pháp hàm **DATEADD** bên dưới có kết quả trả về là một ngày mới sau khi đã **cộng thêm** hoặc **trừ đi** theo một đơn vị thời gian bất kỳ cho một ngày chỉ định.

Cú pháp:

```
DATEADD (Đơn_vị, Con_số, Ngày_chỉ_định) → Ngày_mới
```



Trong đó:

- ✓ Đơn vị: là đơn vị thời gian dùng cho việc giảm hoặc tăng ngày, có thể là ngày (dd), tháng (mm), năm (yy)...
- ✓ Con số: là một số nguyên có thể âm hoặc dương chỉ định việc giảm hoặc tăng theo đơn vị thời gian chỉ định trước đó.
- ✓ Ngày chỉ định: là một biểu thức, tên cột dữ liệu, giá trị cụ thể có kiểu dữ liệu ngày.
- ✓ Ngày mới: là một giá trị ngày mới sau khi đã tăng hoặc giảm.



**Ví dụ:**

Để hiển thị thông tin danh sách đơn đặt hàng có kèm theo ngày hết hạn nhận hàng. Biết rằng ngày hết hạn nhận hàng được tính là 20 ngày sau ngày đặt hàng. Chúng ta sử dụng hàm **DATEADD** như sau:

```
SELECT SODH, D1=CONVERT(CHAR(10), NGAYDH, 103),
       D2=CONVERT(CHAR(10), DATEADD(DD,20,NGAYDH),103)
FROM DONDH
```

Kết quả truy vấn trả về

```
SODH  D1          D2
-----
D001  15/01/2002  04/02/2002
D002  30/01/2002  19/02/2002
D003  10/02/2002  02/03/2002
D004  17/02/2002  09/03/2002
D005  01/03/2002  21/03/2002
D006  12/03/2002  01/04/2002
(6 row(s) affected)
```

### VII.2.2. Hàm DATEDIFF

Với cú pháp hàm **DATEDIFF** bên dưới có kết quả trả về là một số nguyên, nói lên **khoảng cách đại số** của hai ngày theo một đơn vị thời gian bất kỳ.

Cú pháp:

```
DATEDIFF (Đơn_vị, Ngày1, Ngày2) → Số_nguyên
```

Trong đó:

- ✓ Đơn vị: là đơn vị thời gian dùng để chỉ định việc so sánh hai ngày, có thể là ngày (dd), tháng (mm), năm (yy) ...
- ✓ Ngày1, Ngày2: là các biểu thức, tên cột dữ liệu, giá trị cụ thể có kiểu dữ liệu ngày.
- ✓ Số nguyên: là một số nguyên có thể âm hoặc dương trả về khoảng cách đại số giữa ngày1 và ngày2.

**Ví dụ:**

Để hiển thị thông tin danh sách đơn đặt hàng có kèm theo số ngày chênh lệch giữa ngày đặt hàng và ngày nhận hàng dự kiến. Chúng ta sử dụng hàm **DATEDIFF** như sau:

```
SELECT SODH,
       D1=CONVERT(CHAR(10), NGAYDH, 103),
       D1=CONVERT(CHAR(10), NGAYDKNH, 103),
       SONGAY=DATEDIFF(DD, NGAYDH, NGAYDKNH)
FROM DONDH
```

Kết quả truy vấn trả về

SODH	D1	D1	SONGAY
D001	15/01/2002	17/01/2002	2
D002	30/01/2002	02/02/2002	3
D003	10/02/2002	13/02/2002	3
D004	17/02/2002	20/02/2002	3
D005	01/03/2002	05/03/2002	4
D006	12/03/2002	14/03/2002	2

(6 row(s) affected)

**VII.2.3. Hàm DATENAME**

Với cú pháp hàm **DATENAME** bên dưới có kết quả trả về là chuỗi **thời gian đại diện** của một ngày chỉ định theo một đơn vị thời gian bất kỳ.

Cú pháp:

DATENAME (Đơn\_vị, Ngày) → Chuỗi

Trong đó:

- ✓ Đơn vị: là đơn vị thời gian dùng để chỉ định sẽ lấy ra chuỗi thời gian đại diện, có thể là ngày (dd), tháng (mm), năm (yy)...
- ✓ Ngày: là một biểu thức, tên cột dữ liệu, giá trị cụ thể có kiểu dữ liệu ngày.
- ✓ Chuỗi: trả về chuỗi thời gian đại diện.

**Ví dụ:**

Để hiển thị thông tin danh sách đơn đặt hàng có kèm cột thứ trong tuần của ngày đặt hàng. Chúng ta sử dụng hàm **DATENAME** như sau:

```
SELECT SODH,
       D1=CONVERT(CHAR(10), NGAYDH, 103),
       THU=DATENAME(DW, NGAYDH)
FROM DONDH
```





Kết quả truy vấn trả về

```
SODH D1 THU
-----
D001 15/01/2002 Tuesday
D002 30/01/2002 Wednesday
D003 10/02/2002 Sunday
D004 17/02/2002 Sunday
D005 01/03/2002 Friday
D006 12/03/2002 Tuesday
(6 row(s) affected)
```

#### VII.2.4. Hàm GETDATE

Với cú pháp đơn giản của hàm **GETDATE** bên dưới có kết quả trả về là **ngày giờ hiện hành** của hệ thống Microsoft SQL Server.

Cú pháp:

```
GETDATE() → Ngày_giờ
```



**Ví dụ:**

Để hiển thị ngày giờ hiện hành. Chúng ta sử dụng hàm **GETDATE** như sau:

```
SELECT CONVERT(CHAR(20), GETDATE(), 13)
```

Kết quả truy vấn trả về

```
-----
30 Mar 2002 03:24:45
(1 row(s) affected)
```

#### VII.2.5. Hàm DATEPART

Với cú pháp hàm **DATEPART** bên dưới có kết quả trả về là một **số nguyên** chỉ định thời gian đại diện của một ngày theo một đơn vị thời gian bất kỳ

Cú pháp:

```
DATEPART (Đơn_vị, Ngày) → Số_nguyên
```

Trong đó:

- ✓ Đơn vị: là đơn vị thời gian dùng để chỉ định sẽ lấy ra một con số, có thể là ngày (dd), tháng (mm), năm (yy)...
- ✓ Ngày: là một biểu thức, tên cột, giá trị cụ thể có kiểu dữ liệu ngày.
- ✓ Số nguyên: trả về số thời gian đại diện. Đối với các ngày trong tuần nếu kết quả là 1 thì xem như là chủ nhật, 2 là thứ hai... và 7 là thứ bảy.

**Ví dụ:**

Để hiển thị các đơn đặt hàng theo từng tháng trong năm 2002. Chúng ta sử dụng hàm **DATEPART** như sau:

```
SELECT SODH, THANG=DATEPART(MM, NGAYDH),
       D1=CONVERT(CHAR(10), NGAYDH, 103)
FROM DONDH
WHERE YEAR(NGAYDH)=2002
ORDER BY THANG
```

Kết quả truy vấn trả về

SODH	THANG	D1
-----		
D001	1	15/01/2002
D002	1	30/01/2002
D003	2	10/02/2002
D004	2	17/02/2002
D005	3	01/03/2002
D006	3	12/03/2002

(6 row(s) affected)

**VII.2.6. Các hàm DAY, MONTH, YEAR**

Với cú pháp chung bên dưới của các hàm **DAY**, **MONTH**, **YEAR** có kết quả trả về là một số nguyên chỉ định ngày (day), tháng (month), năm (year) của một ngày bất kỳ. Các bạn có thể sử dụng các hàm này hoặc hàm **DATEPART** tương ứng với các đơn vị thời gian là: dd, mm, yy.

Cú pháp:

```
DAY   (Ngày) → Số_nguyên
MONTH (Ngày) → Số_nguyên
YEAR  (Ngày) → Số_nguyên
```

**Ví dụ:**

Mình họa như thí dụ trên nhưng chúng ta sử dụng hàm **MONTH** như sau:

```
SELECT SODH, THANG=MONTH(NGAYDH),
       D1=CONVERT(CHAR(10), NGAYDH, 103)
FROM DONDH
WHERE YEAR(NGAYDH)=2002
ORDER BY THANG
```

Kết quả truy vấn trả về

SODH	THANG	D1
-----		
D001	1	15/01/2002



```
D002 1      30/01/2002
D003 2      10/02/2002
D004 2      17/02/2002
D005 3      01/03/2002
D006 3      12/03/2002
```

(6 row(s) affected)

### VII.3. Các hàm toán học

Các hàm này thường có tham số vào là kiểu dữ liệu số và giá trị trả về của chúng cũng là kiểu dữ liệu số. Thông thường khi lập trình trong Transaction-SQL chúng tôi rất ít khi sử dụng các hàm toán học.

#### VII.3.1. Hàm ABS

Với cú pháp hàm **ABS** bên dưới có kết quả trả về là **trị tuyệt đối** (absolute) của một số bất kỳ. Kết quả trả về luôn là một số dương.

Cú pháp:

```
ABS (Biểu_thức_số) → Số
```

Trong đó:

- ✓ Biểu thức số: là một biểu thức có kiểu dữ liệu là số mà chúng ta muốn tính trị tuyệt đối.



#### Ví dụ:

Thực hiện lệnh bên dưới để lấy trị tuyệt đối của hai số: 2002.02 và -1972.

```
SELECT ABS(2002.02), ABS(-1972)
```

Kết quả trả về

```
-----
2002.02  1972
```

(1 row(s) affected)

#### VII.3.2. Hàm PI

Với cú pháp đơn giản của hàm **PI** bên dưới có kết quả trả về là giá trị của hằng số pi trong toán học.

Cú pháp:

```
PI() → Số
```

**Ví dụ:**

Thực hiện lệnh bên dưới để in ra giá trị của hằng số pi.

```
SELECT PI()
```

Kết quả trả về

```
-----  
3.1415926535897931
```

```
(1 row(s) affected)
```

**VII.3.3. Hàm POWER**

Với cú pháp hàm **POWER** bên dưới có kết quả trả về là phép tính lũy thừa của một số bất kỳ nào đó theo một số mũ chỉ định.

Cú pháp:

```
POWER (Biểu_thức_số, Số_mũ) → Số
```

Trong đó:

- ✓ Biểu thức số: là một biểu thức có giá trị kiểu dữ liệu số.
- ✓ Số mũ: là một số dương thực hiện phép lũy thừa.

**Ví dụ:**

Thực hiện lệnh bên dưới để tính ra giá trị  $2^5$  (2 lũy thừa 5)

```
SELECT POWER(2, 5)
```

Kết quả trả về

```
-----  
32
```

```
(1 row(s) affected)
```

**VII.3.4. Hàm RAND**

Với cú pháp hàm **RAND** bên dưới có kết quả trả về là một số thực ngẫu nhiên mà hệ thống Microsoft SQL Server tự động tạo ra đảm bảo không trùng lặp.

Cú pháp:

```
RAND ([Số_nguồn]) → Số_ngẫu_nhiên
```

Trong đó:

- ✓ Số nguồn: là một giá trị số nguyên có phạm vi không vượt quá phạm vi của kiểu dữ liệu int làm giá trị nguồn cho hệ thống tạo ra số ngẫu nhiên.
- ✓ Số ngẫu nhiên: là một thực dương có miền giá trị từ 0 đến 1.

**Ví dụ:**

Thực hiện vòng lặp tạo ra liên tục 5 số ngẫu nhiên.

```
SET NOCOUNT ON
DECLARE @counter smallint
SET @counter = 1
WHILE @counter < 5
BEGIN
    SELECT RAND(@counter)
    SET @counter = @counter + 1
END
SET NOCOUNT OFF
GO
```

Kết quả trả về

```
-----
0.71359199321292355
-----
0.7136106261841817
-----
0.71362925915543995
-----
0.7136478921266981
```

**VII.3.5. Hàm ROUND**

Với cú pháp hàm **ROUND** bên dưới có kết quả trả về là một số đã được làm tròn.

Cú pháp:

**ROUND** (Biểu\_thức\_số, Vtrí\_làm\_tròn) → Số

Trong đó:

- ✓ Biểu thức số: là một biểu thức có kiểu dữ liệu là số thực.
- ✓ Vị trí làm tròn: là một số nguyên âm hoặc dương dùng để chỉ định vị trí muốn làm tròn, được tính từ vị trí dấu chấm thập phân.

**Ví dụ:**

Thực hiện các câu lệnh **SELECT** có sử dụng các hàm **ROUND** bên dưới.

Thực hiện lệnh	Kết quả trả về
SELECT ROUND(153.5341, 2)	153.5300
SELECT ROUND(153.5341, 0)	154.0000
SELECT ROUND(153.5341, -1)	150.0000



### VII.3.6. Hàm SIGN

Với cú pháp hàm **SIGN** bên dưới có kết quả trả về là một con số qui định dấu của biểu thức số. Kết quả trả về là 1 nếu biểu thức số dương, là -1 nếu biểu thức số âm, là 0 nếu biểu thức số bằng không.

Cú pháp:

SIGN (Biểu\_thức\_số) → Số\_nguyên

Trong đó:

- ✓ Biểu thức số: là một biểu thức có kiểu dữ liệu là số.



#### Ví dụ:

Thực hiện câu lệnh **SELECT** có sử dụng các hàm **SIGN** bên dưới.

```
SELECT SIGN(-2002.02), SIGN(1999), SIGN(0)
```

Kết quả truy vấn trả về

```
-----
-1.00    1        0
(1 row(s) affected)
```

### VII.3.7. Hàm SQRT

Với cú pháp hàm **SQRT** bên dưới dùng để thực hiện việc tính căn bậc hai của một số dương bất kỳ. Kết quả trả về là một số dương.

Cú pháp:

SQRT (Biểu\_thức\_số) → Số

Trong đó:

- ✓ Biểu thức số: là một biểu thức số có giá trị luôn dương.



#### Ví dụ:

Thực hiện lệnh bên dưới để lấy căn bậc hai của các số: 9 và 2.

```
SELECT SQRT(9), SQRT(2)
```

Kết quả trả về

```
-----
3.0          1.4142135623730951
(1 row(s) affected)
```

## VII.4. Các hàm xử lý chuỗi

Các hàm này thường có tham số vào là kiểu dữ liệu chuỗi và giá trị trả về của chúng cũng là kiểu dữ liệu chuỗi hoặc kiểu dữ liệu số. Thông thường khi lập trình trong Transaction-SQL chúng tôi thường kết hợp sử dụng các hàm này trong các xử lý chuỗi phức tạp.



#### VII.4.1. Các hàm UPPER, LOWER

Với cú pháp chung bên dưới của các hàm **UPPER**, **LOWER** có kết quả trả về là một chuỗi sau khi đã được chuyển đổi các ký tự bên trong chuỗi thành chữ in (upper), hoặc chữ thường (lower).

Cú pháp:

```
UPPER (Chuỗi) → Chuỗi_chữ_in
LOWER (Chuỗi) → Chuỗi_chữ_thường
```



##### Ví dụ:

Để hiển thị thông tin trong bảng NHACC theo yêu cầu: dữ liệu cột họ tên nhà cung cấp được đổi thành chữ in, dữ liệu cột địa chỉ được đổi thành chữ thường. Chúng ta sử dụng lệnh sau:

```
SELECT UPPER(TENNHACC) AS "Chữ IN",
       LOWER(DIACHI) AS "Chữ thường"
FROM NHACC
```

Kết quả truy vấn trả về

Chữ IN	Chữ thường
LÊ MINH TRÍ	54 hậu giang
TRẦN MINH THẠCH	145 hùng vương
NGUYỄN HỒNG PHƯƠNG	154/85 lê lai
TRƯƠNG NHẬT THẮNG	198/40 hương lộ 14
LƯU NGUYỆT QUẾ	178 nguyễn văn luông
CAO MINH TRUNG	cư xá phú lâm

(6 row(s) affected)

#### VII.4.2. Các hàm LEFT, RIGHT, SUBSTRING

Với cú pháp chung bên dưới các hàm **LEFT**, **RIGHT**, **SUBSTRING** có kết quả trả về là một chuỗi con được trích ra từ chuỗi nguồn. Chuỗi con được trích ra tại vị bắt đầu từ bên trái (left), bên phải (right) hoặc tại bất kỳ vị trí nào (substring) và lấy ra bao nhiêu ký tự.

Cú pháp:

```
LEFT (Chuỗi nguồn, Số_ký_tự) → Chuỗi_con
RIGHT (Chuỗi nguồn, Số_ký_tự) → Chuỗi_con
SUBSTRING (Chuỗi nguồn, Vị_trí, Số_ký_tự) → Chuỗi_con
```

Trong đó:

- ✓ Chuỗi nguồn: là chuỗi ký tự nguồn chứa các ký tự muốn được chọn lựa để trích ra.
- ✓ Số ký tự: là một số nguyên dương chỉ định số ký tự bên trong chuỗi nguồn sẽ được trích ra.
- ✓ Vị trí: là số nguyên dương chỉ định tại vị trí bắt đầu trích được áp dụng cho hàm

**SUBSTRING.**

- ✓ Chuỗi con: là chuỗi kết quả trả về sau khi thực hiện việc trích các ký tự đã chỉ định trong các tham số trên.

**Ví dụ:**

Thực hiện các câu lệnh **SELECT** có sử dụng các hàm **LEFT**, **RIGHT**, **SUBSTRING** bên dưới để lấy ra các từ mong muốn.

Thực hiện lệnh	Kết quả trả về
SELECT LEFT("Hello The World", 5)	Hello
SELECT SUBSTRING("Hello The World", 7, 3)	The
SELECT RIGHT("Hello The World", 5)	World

**VII.4.3. Các hàm LTRIM và RTRIM**

Với cú pháp chung bên dưới của các hàm **LTRIM**, **RTRIM** có kết quả trả về là một chuỗi đã được cắt bỏ các khoảng trắng ở đầu chuỗi (ltrim), hoặc các khoảng trắng ở cuối chuỗi ( rtrim).

Cú pháp:

LTRIM (Chuỗi) → Chuỗi\_mới  
RTRIM (Chuỗi) → Chuỗi\_mới

Trong đó:

- ✓ Chuỗi: là một chuỗi có ký tự sẽ được cắt bỏ các khoảng trắng ở đầu chuỗi hoặc cuối chuỗi.
- ✓ Chuỗi mới: là chuỗi kết quả đã được cắt bỏ các khoảng trắng ở đầu chuỗi hoặc cuối chuỗi.

**Ví dụ:**

Thực hiện các câu lệnh **SELECT** có sử dụng các hàm **LTRIM**, **RTRIM** bên dưới để cắt bỏ các khoảng trắng ở đầu và cuối chuỗi.

Thực hiện lệnh	Kết quả trả về
SELECT "Năm : 20" + LTRIM(" 02")	Năm : 2002
SELECT RTRIM("Năm : 20 ") + "02"	Năm : 2002

**VII.4.4. Hàm SPACE**

Với cú pháp hàm **SPACE** đơn giản bên dưới có kết quả trả về là một chuỗi chứa N ký tự trắng.

Cú pháp:

SPACE (N) → Chuỗi\_trắng

Trong đó:

- ✓ N: là một số nguyên dương dùng để chỉ định chuỗi chứa bao nhiêu ký tự trắng.



**Ví dụ:**

Thực hiện câu lệnh **SELECT** có sử dụng hàm **SPACE** bên dưới dùng để tạo ra 10 ký tự trắng phía trước chữ "World".

```
SELECT "Hello The" + SPACE(10) + "World"
```

Kết quả trả về

```
-----
Hello The      World
```

```
(1 row(s) affected)
```

**VII.4.5. Hàm REPLICATE**

Với cú pháp hàm **REPLICATE** bên dưới có kết quả trả về là một chuỗi chứa các ký tự được **lặp** lại N lần.

Cú pháp:

```
REPLICATE (Chuỗi_lặp, N) → Chuỗi_mới
```

Trong đó:

- ✓ Chuỗi lặp: là một chuỗi có các ký tự sẽ được sao chép.
- ✓ N: là một số nguyên dương chỉ định số lần sao chép.

**Ví dụ:**

Thực hiện câu lệnh **SELECT** có sử dụng hàm **REPLICATE** bên dưới dùng để sao chép 5 lần chữ "Wo" sau chuỗi "Hello The World".

```
SELECT "Hello The World" + REPLICATE ( " Wo !", 5)
```

Kết quả trả về

```
-----
Hello The World Wo ! Wo ! Wo ! Wo ! Wo !
```

```
(1 row(s) affected)
```

**VII.4.6. Hàm LEN**

Với cú pháp đơn giản của hàm **LEN** bên dưới có kết quả trả về là một số nguyên dương dùng để chỉ định chiều dài của một chuỗi chứa bao nhiêu ký tự.

Cú pháp:

```
LEN (Chuỗi) → Số_nguyên
```

Trong đó:

- ✓ Chuỗi: là chuỗi cần tính ra có bao nhiêu ký tự.
- ✓ Số nguyên: trả về chiều dài của chuỗi.

**Ví dụ:**

Thực hiện câu lệnh **SELECT** có sử dụng hàm **LEN** bên dưới dùng để trả về chiều dài chuỗi “Trung tam tin học”.

```
SELECT LEN("Trung tam tin hoc")
```

Kết quả trả về

```
-----
17
(1 row(s) affected)
```

**VII.4.7. Hàm REVERSE**

Với cú pháp đơn giản hàm **REVERSE** bên dưới có kết quả trả về là một chuỗi đảo ngược.

Cú pháp:

```
REVERSE (Chuỗi) → Chuỗi_đảo
```

**Ví dụ:**

Thực hiện câu lệnh **SELECT** có sử dụng hàm **REVERSE** bên dưới dùng để trả về chuỗi ngược của chuỗi “TTTH-DHKHTN”.

```
SELECT REVERSE("TTTH-DHKHTN")
```

Kết quả trả về

```
-----
NTHKHD-HTTT
(1 row(s) affected)
```

**VII.4.8. Hàm STUFF**

Với cú pháp hàm **STUFF** bên dưới có kết quả trả về là một chuỗi mới sau khi đã hủy bỏ một số ký tự hiện có và thêm vào một chuỗi con khác tại vị trí vừa hủy bỏ.

Cú pháp:

```
STUFF(Chuỗi nguồn, Vị_trí, Chiều_dài, Chuỗi_con) → Chuỗi_mới
```

Trong đó:

- ✓ Chuỗi nguồn: là một chuỗi nguồn chứa các ký tự.
- ✓ Vị trí: là một số nguyên chỉ định vị trí bắt đầu hủy bỏ các ký tự bên trong chuỗi nguồn.
- ✓ Chiều dài: là một số nguyên chỉ định bao nhiêu ký tự sẽ bị hủy bỏ trong chuỗi nguồn đếm từ bên trái vị trí chỉ định.
- ✓ Chuỗi con: là một chuỗi sẽ được thêm vào chuỗi nguồn tại vị trí hủy bỏ ở trên.

**Ví dụ:**

Thực hiện câu lệnh **SELECT** có sử dụng hàm **STUFF** bên dưới dùng để hủy bỏ vài ký tự trong một chuỗi và thêm vào một chuỗi khác tại vị trí đã hủy bỏ.

```
SELECT STUFF("1234567890", 4, 3, "ABCDEF")
```

Kết quả trả về

```
-----  
123ABCDEF7890
```

```
(1 row(s) affected)
```

**VII.4.9. Hàm REPLACE**

Với cú pháp hàm **REPLACE** bên dưới có kết quả trả về là một chuỗi mới sau khi đã được tìm và thay thế (nếu có) một chuỗi nhỏ vào trong một chuỗi nguồn.

Cú pháp:

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

Trong đó:

- ✓ Chuỗi nguồn: là một chuỗi nguồn chứa các ký tự.
- ✓ Chuỗi tìm: là chuỗi con cần tìm xem có xuất hiện bên trong chuỗi nguồn hay không.
- ✓ Chuỗi thay thế: là chuỗi sẽ được thay thế khi tìm thấy chuỗi tìm trong chuỗi nguồn.

**Ví dụ:**

Thực hiện câu lệnh **SELECT** có sử dụng hàm **REPLACE** bên dưới dùng để tìm và thay thế từ "CSC" thành từ "TTTH" trong chuỗi nguồn "Chào mừng đến CSC-HCMUNS" (chữ này viết tắt bởi các từ Computer Science Center–Ho Chi Minh University Natural Science)

```
SELECT REPLACE("Chào mừng đến CSC-HCMUNS", "CSC", "TTTH")
```

Kết quả trả về

```
-----  
Chào mừng đến TTTH-HCMUNS
```

```
(1 row(s) affected)
```

**VII.4.10. Hàm CHAR**

Với cú pháp hàm **CHAR** bên dưới có kết quả trả về là một ký tự tương ứng trong bảng mã ASCII, trong bảng mã này qui định mỗi một ký tự trong máy tính sẽ có tương ứng một con số. Phạm vi của các số từ 0 đến 255.

Cú pháp:

```
CHAR (Số) → Ký_tự
```

**Ví dụ:**

Sử dụng hàm **CHAR** để in ra danh sách các nhà cung cấp dạng bì thư cho các thư mời, mỗi khách hàng sẽ gồm: họ tên, địa chỉ. Biết rằng các ký tự 10 và 13 dùng để xuống dòng và về đầu dòng.

```
SELECT "Kính gửi: " + TENNHACC+
      CHAR(13)+ DIACHI + CHAR(10)
FROM NHACC
```

***Kết quả trả về***

```
-----
Kính gửi: Lê Minh Trí
54 Hậu Giang
```

```
Kính gửi: Trần Minh Thạch
145 Hùng Vương
```

```
Kính gửi: Nguyễn Hồng Phương
154/85 Lê Lai
```

```
Kính gửi: Trương Nhật Thắng
198/40 Hương Lộ 14
```

```
Kính gửi: Lưu Nguyệt Quế
178 Nguyễn Văn Lương
```

```
Kính gửi: Cao Minh Trung
Cư xá Phú Lâm
```

```
(6 row(s) affected)
```

**VII.4.11. Hàm ASCII**

Với cú pháp hàm **ASCII** bên dưới có kết quả trả về là một con số nguyên có phạm vi từ 0 đến 255, tương ứng trong bảng mã ASCII của ký tự đã gửi vào. Chức năng của hàm này hoàn toàn ngược lại với hàm **CHAR**.

Cú pháp:

```
ASCII(Ký_tự) → Số
```

**Ví dụ:**

Sử dụng hàm **ASCII** để biết được mã ASCII của các ký tự từ A đến P.

```
DECLARE @KytuD CHAR(1), @KytuC CHAR(1)
SET @KytuD= "A"
SET @KytuC= "Q"
WHILE (@KytuD<>@KytuC)
BEGIN
    SELECT "Ký tự : " + @KytuD ,
           "Mã Ascii: " + CAST(ASCII(@KytuD) AS CHAR(5))
    SET @KytuD = CHAR(ASCII(@KytuD)+1)
END
```

*Kết quả trả về*

```
-----
Ký tự :A Mã Ascii: 65

(1 row(s) affected)

-----

Ký tự :B Mã Ascii: 66

(1 row(s) affected)

...

-----

Ký tự :O Mã Ascii: 79

(1 row(s) affected)

-----

Ký tự :P Mã Ascii: 80

(1 row(s) affected)
```

Tóm lại trong bài ba chúng tôi đã giới thiệu cho các bạn hầu hết các khái niệm căn bản lập trình trong Transaction–SQL. Cách sử dụng và ý nghĩa của **biến cục bộ, cấu trúc điều khiển** chương trình, **hàm tính toán sẵn có** của Microsoft SQL Server vào chi tiết bên trong các thủ tục nội tại hoặc các trigger là hoàn toàn do chính các bạn chủ động lựa chọn. Trong các thí dụ minh họa ở các bài kế tiếp chúng ta sẽ sử dụng lại các phần trong bài này một cách nhuần nhuyễn hơn.



# Bài 5

## THỦ TỤC NỘI TẠI

### Tóm tắt

Lý thuyết 8 tiết - Thực hành 12 tiết

Mục tiêu	Các mục chính	Bài tập bắt buộc	Bài tập làm thêm
<ul style="list-style-type: none"> <li>✓ Trình bày một số khái niệm về thủ tục nội tại</li> <li>✓ Giới thiệu các thao tác đối với thủ tục nội tại (thêm, xóa, sửa và thực thi)</li> <li>✓ Trình bày một số khái niệm về tham số trong thủ tục nội tại</li> <li>✓ Trình bày một số khái niệm về giao tác</li> </ul>	<ul style="list-style-type: none"> <li>I. Khái niệm về thủ tục nội tại</li> <li>II. Các hành động cơ bản với thủ tục nội tại</li> <li>III. Tham số bên trong thủ tục nội tại</li> <li>IV. Một số vấn đề khác trong thủ tục nội tại</li> <li>V. Giao tác (Transaction)</li> </ul>	5.1 5.2 5.3 5.4	5.5



# I. Khái niệm về thủ tục nội tại

## I.1. Thủ tục nội tại là gì?

Thủ tục nội tại là một tập hợp chứa **các dòng lệnh**, **các biến** và **các cấu trúc điều khiển** viết bằng ngôn ngữ Transaction–SQL, dùng để thực hiện một hành động nào đó, tất cả nội dung của một thủ tục nội tại sẽ được **lưu trữ tại cơ sở dữ liệu** của Microsoft SQL Server.

Các nét **đặc trưng** của một thủ tục nội tại cũng hoàn toàn giống các thủ tục trong các ngôn ngữ lập trình khác: **tên thủ tục** nội tại, **tham số truyền giá trị** vào và **tham số nhận giá trị** trả ra. Ngoài ra bên trong một thủ tục nội tại chúng ta cũng được phép **gọi thực thi một thủ tục nội tại khác**. Phạm vi hoạt động của các thủ tục nội tại do người dùng tạo ra chỉ có tính **cục bộ** bên trong một cơ sở dữ liệu lưu trữ thủ tục đó.

Một **nét riêng biệt** của thủ tục nội tại là nó có thể **được gọi thực hiện** trong môi trường **không** phải là Microsoft SQL Server. Khi xây dựng giao diện màn hình trên các ngôn ngữ lập trình khác nhau chúng ta vẫn có thể gọi thực hiện các thủ tục nội tại một cách dễ dàng.

Ngoài ra do thủ tục nội tại được **lưu trữ vật lý** trong cơ sở dữ liệu của Microsoft SQL Server, nên các thủ tục nội tại sẽ được **thực thi khá nhanh** bởi vì nội dung bên trong thủ tục nội tại đã **được phân tích cú pháp** các lệnh khi chúng được tạo mới. **Lần đầu tiên** khi thủ tục nội tại được gọi thực hiện thì nội dung các lệnh bên trong thủ tục nội tại sẽ **được biên dịch** và **lưu lại trên bộ nhớ**, kể từ các **lần kế tiếp** thì thủ tục nội tại sẽ được thực thi **nhANH hơn** (vì các mã lệnh đã được lưu lại trên bộ nhớ). Đây cũng là một trong những lý do mà tại sao chúng ta nên sử dụng thủ tục nội tại trong Microsoft SQL Server.

## I.2. Các thủ tục nội tại hệ thống

Trong Microsoft SQL Server cung cấp cho chúng ta một số lớn các thủ tục nội tại hệ thống dùng để thực hiện các xử lý trong việc **quản trị cơ sở dữ liệu**. Một thủ tục nội tại hệ thống luôn luôn được bắt đầu bằng chữ **sp\_** và hầu hết tất cả các thủ tục nội tại hệ thống được lưu trữ bên trong cơ sở dữ liệu **Master**.

Việc làm quen và sử dụng nhuần nhuyễn các thủ tục hệ thống sẽ giúp chúng ta hiểu rõ thêm về **cách tổ chức cấu trúc** các bảng hệ thống bên dưới Microsoft SQL Server. Ngoài ra còn giúp chúng ta biết được các **xử lý bí ẩn** bên dưới tiện ích Enterprise Manager sau khi chúng ta thực hiện các hành động chọn lựa trong tiện ích đồ họa này.



### Ví dụ:

Chúng ta có thể tạo mới một người dùng có tên là “TTTH”, mật khẩu “T3HNVC” cho phép truy cập vào cơ sở dữ liệu NorthWind của Microsoft SQL Server bằng thủ tục hệ thống `sp_addlogin` như sau:

```
EXEC sp_addlogin "TTTH", "T3HNVC", "NorthWind"
```

Trong phần giáo trình này chúng tôi chỉ có ý định hướng dẫn các bạn thực hiện việc **triển khai**



một cơ sở dữ liệu trong Microsoft SQL Server. Do đó trong chương này chúng tôi sẽ **không đề cập** đến các thủ tục nội tại hệ thống của Microsoft SQL Server mà chỉ trình bày cách thức tạo ra một **thủ tục do người dùng định nghĩa** để thực hiện một hành động riêng cho ứng dụng của chúng ta. Trong những phần kế tiếp khi đề cập đến cụm từ thủ tục nội tại thì xem như chúng ta nói đến các thủ tục nội tại do người dùng định nghĩa.

### 1.3. Các lợi ích khi sử dụng thủ tục nội tại

Các nét **đặc trưng** mà chúng tôi muốn giới thiệu bên dưới nhằm giúp các bạn hiểu rõ về các lợi ích khi sử dụng thủ tục nội tại trong Microsoft SQL Server. Các bạn **nên đọc và suy xét** những điều được mô tả bên dưới thật **rõ ràng** để có thể áp dụng vào đúng các trường hợp thực tế trong khi xây dựng các ứng dụng theo mô hình khách chủ.

- ✓ **Tốc độ xử lý** của các thủ tục nội tại sẽ **rất nhanh** bởi vì bản thân nội dung của các thủ tục nội tại được lưu trữ và thực hiện ngay tại máy chủ. Hơn thế nữa, các dữ liệu cũng được lưu trữ trên cùng máy chủ nên không cần mất thời gian để truyền dữ liệu qua hệ thống mạng cho các xử lý bên trong thủ tục nội tại bởi vì **các xử lý và dữ liệu** cùng được **lưu trữ** trên cùng **một vị trí vật lý** là máy chủ.
- ✓ Việc **tổ chức và phân chia** các xử lý thành **hai nơi** khác nhau: tại máy chủ hoặc tại máy trạm sẽ giúp các bạn có thể làm giảm thời gian xây dựng ứng dụng. Bởi vì bản thân một thủ tục nội tại trong Microsoft SQL Server có thể được **gọi thực thi** nhiều lần trên **các màn hình khác nhau** trong các **ngôn ngữ lập trình khác nhau**: Visual Basic, C++, Delphi...
- ✓ Giống như các lợi ích khi sử dụng đối tượng bảng ảo (view), các bạn có thể sử dụng thủ tục nội tại để **phân cấp quyền hạn** (security) cho những người sử dụng với các hành động cập nhật dữ liệu như là: **thêm, sửa, xóa** hoặc xem **nội dung của các báo cáo**.

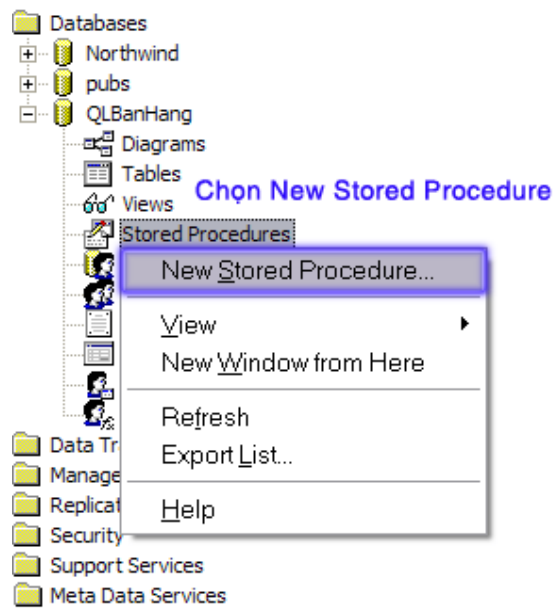
## II. Các hành động cơ bản với thủ tục nội tại

### II.1. Tạo mới một thủ tục nội tại

Giống như các đối tượng khác trong Microsoft SQL Server mà chúng tôi đã trình bày trước đây trong chương hai, các bạn vẫn có hai cách để có thể tạo mới một thủ tục nội tại. Các bước bên dưới sẽ hướng dẫn các bạn cách thức **tạo mới một thủ tục nội tại** bằng tiện ích **Enterprise Manager**.

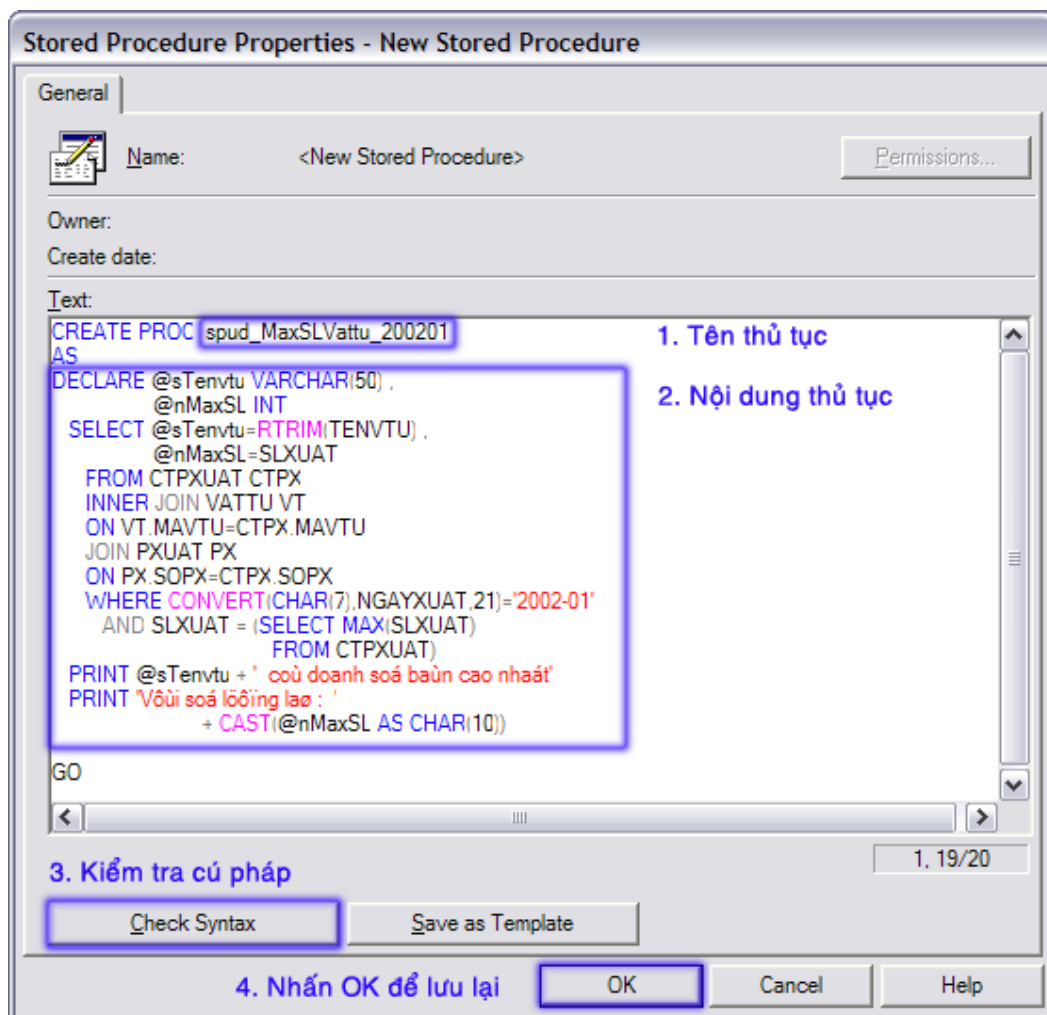
- ✓ Bước 1: Khởi động tiện ích Enterprise Manager. Chọn chức năng **New Stored Procedure...** trong thực đơn tắt sau khi nhấn chuột phải trên đối tượng **Stored Procedures** để tạo mới một thủ tục nội tại đơn giản.





**Hình 5-1.** Chọn New Stored Procedure để tạo thủ tục nội tại mới.

- ✓ Bước 2: Trong màn hình định nghĩa thủ tục nội tại mới lần lượt chỉ định **tên của thủ tục nội tại** và các **câu lệnh cần thiết** bên trong nội dung của thủ tục nội tại dùng để xử lý, tính toán theo một yêu cầu cụ thể nào đó. Nhấn vào nút **Check Syntax** để hệ thống **kiểm tra cú pháp** các lệnh bên trong thủ tục nội tại có hợp lệ hay không? Sau cùng nhấn vào nút **OK** để lưu lại nội dung của thủ tục nội tại vừa mới tạo.



**Hình 5-2. Màn hình chỉ định các thuộc tính liên quan đến thủ tục nội tại.**

- ✓ Nhận xét thấy rằng trong màn hình minh họa trên **tên thủ tục** và **nội dung thủ tục** được ngăn cách nhau bởi từ khóa **AS**. Xử lý bên trong thủ tục này dùng để tính ra tên của vật tư nào kèm với số lượng với doanh số bán ra là cao nhất.

Ngoài ra chúng ta có cũng có thể tạo mới thủ tục nội tại bằng lệnh **CREATE PROCEDURE** trong tiên ích Query Analyzer. Cú pháp của lệnh này được mô tả như bên dưới.

Cú pháp:

```
CREATE PROC[EDURE] Tên_thủ_tục
AS
[DECLARE Biến_cục_bộ]
    Các_lệnh
```

Trong đó:

- ✓ **Tên thủ tục:** tên thủ tục nội tại được tạo mới, tên thủ tục nội tại này phải là **duy nhất** trong một cơ sở dữ liệu.
- ✓ **Biến cục bộ:** là những biến cục bộ được sử dụng tính toán tạm thời bên trong thủ tục, những biến này chỉ có phạm vi cục bộ bên trong thủ tục nội tại.



- ✓ **Các lệnh:** các lệnh bên trong thủ tục nội tại dùng để xử lý tính toán theo một yêu cầu nào đó.

**Ví dụ:**

Để tính ra vật tư nào có doanh số bán cao nhất trong tháng 01/2002. Chúng ta thực hiện lệnh **CREATE PROCEDURE** như sau:

```
CREATE PROC spud_MaxSLVattu_200201
AS
    DECLARE @sTenvtu VARCHAR(50), @nMaxSL INT
    SELECT @sTenvtu=RTRIM(TENVTU), @nMaxSL=SLXUAT
    FROM CTPXUAT CTPX INNER JOIN VATTU VT ON VT.MAVTU=CTPX.MAVTU
        JOIN PXUAT PX ON PX.SOPX=CTPX.SOPX
    WHERE CONVERT(CHAR(7),NGAYXUAT,21)="2002-01"
        AND SLXUAT = (SELECT MAX(SLXUAT)
        FROM CTPXUAT INNER JOIN PXUAT ON
        PXUAT.SOPX=CTPXUAT.SOPX WHERE CONVERT(CHAR(7),NGAYXUAT,21)="2002-01")

    PRINT @sTenvtu + " có doanh số bán cao nhất"
    PRINT "Với số lượng là : " + CAST(@nMaxSL AS CHAR(10))
GO
```

Nhận xét thấy rằng trong thí dụ này chúng ta có sử dụng **hai biến cục bộ** dùng để lưu trữ tên của vật tư có số lượng bán ra nhiều nhất trong tháng 01/2002 sau đó dùng nó để hiển thị ở cuối thủ tục.

## II.2. Gọi thực hiện thủ tục nội tại

Chúng ta chỉ có thể **gọi thực hiện thủ tục nội tại** bằng lệnh **EXECUTE** trong tiện ích Query Analyzer. Cú pháp bên dưới mô tả việc gọi thực hiện một thủ tục nội tại đơn giản – không có tham số gọi vào hoặc đón nhận giá trị trả về cả!

Cú pháp:

```
EXEC[UTE] Tên_thủ_tục
```

Trong đó:

- ✓ **Tên thủ tục:** tên thủ tục nội tại đã được tạo trước đó mà chúng ta muốn gọi thực thi.

**Ví dụ:**

Để thực hiện thủ tục **spud\_MaxSLVattu\_200201** đã được tạo ra trong thí dụ trước đó. Chúng ta thực hiện lệnh **EXECUTE** như sau:

```
EXECUTE spud_MaxSLVattu_200201
```

Kết quả trả về

```
Đầu VCD Sony 3 đĩa có doanh số bán cao nhất
Với số lượng là : 10
```



### II.3. Hủy bỏ thủ tục nội tại

Khi một thủ tục nội tại **không còn cần sử dụng** nữa thì chúng ta có thể hủy bỏ nó ra khỏi cơ sở dữ liệu. **Cẩn thận** khi sử dụng hành động này bởi vì các bạn sẽ **không** có cơ hội **phục hồi** lại nội dung của thủ tục sau khi đã xóa. Cú pháp lệnh **DROP PROCEDURE** bên dưới cho phép chúng ta có thể hủy bỏ một thủ tục nội tại.

Cú pháp:

```
DROP PROC[EDURE] Tên_thủ_tục
```

Trong đó:

- ✓ **Tên thủ tục:** tên thủ tục nội tại đã được tạo trước đó mà chúng ta muốn hủy bỏ khi không còn sử dụng nữa.



**Ví dụ:**

Để xóa bỏ thủ tục *spud\_MaxSLVattu\_200201* đã được tạo ra trong thí dụ trước đó. Chúng ta thực hiện lệnh **DROP PROCEDURE** như sau:

```
DROP PROC spud_MaxSLVattu_200201
```

### II.4. Thay đổi nội dung của thủ tục nội tại

Đôi khi nội dung của các thủ tục cần phải **thay đổi** lại để cho các hành động bên trong thủ tục nội tại thực hiện được đúng đắn **theo các yêu cầu mới**. Chúng ta có **hai** cách để thực hiện việc thay đổi nội dung bên trong của thủ tục nội tại:

- ✓ Hoặc là dùng lệnh **xóa bỏ thủ tục nội tại cũ** và **tạo lại thủ tục nội tại mới** với nội dung đã được thay đổi phù hợp theo yêu cầu mới.
- ✓ Hoặc là dùng lệnh **ALTER PROCEDURE** để thay đổi nội dung. Cú pháp của lệnh ALTER PROCEDURE bên dưới **hoàn toàn giống** cú pháp của lệnh CREATE PROCEDURE mà chúng tôi đã trình bày trước đây.

Cú pháp:

```
ALTER PROC[EDURE] Tên_thủ_tục
AS
[DECLARE Biến_cục_bộ]
Các_lệnh
```



**Ví dụ:**

Cần bổ sung thêm phần kiểm tra dữ liệu đã có bán hàng trong tháng 01/2002 trong thủ tục nội tại *spud\_MaxSLVattu\_200201* đã được tạo ra ở thí dụ trước đó. Chúng ta thực hiện lệnh **ALTER PROCEDURE** như sau:

```
ALTER PROC spud_MaxSLVattu_200201 AS
DECLARE @sTenvtu VARCHAR(50), @nMaxSL INT
IF NOT EXISTS (SELECT MAVTU
FROM CTPXUAT CTPX INNER JOIN PXUAT PX ON
PX.SOPX=CTPX.SOPX
```



```

WHERE CONVERT(CHAR(7),NGAYXUAT,21)="2002-01")

BEGIN
    PRINT "Tháng 01/2002 chưa có bán vật tư nào cả!"
    RETURN
END

SELECT @sTenvtu=RTRIM(TENVTU), @nMaxSL=SLXUAT
FROM CTPXUAT CTPX INNER JOIN VATTU VT ON VT.MAVTU=CTPX.MAVTU
        JOIN PXUAT PX ON PX.SOPX=CTPX.SOPX
WHERE CONVERT(CHAR(7), NGAYXUAT, 21)="2002-01"
        AND SLXUAT = (SELECT MAX(SLXUAT)
                        FROM CTPXUAT INNER JOIN PXUAT ON
PXUAT.SOPX=CTPXUAT.SOPX WHERE CONVERT(CHAR(7),NGAYXUAT,21)="2002-01")
PRINT @sTenvtu + " có doanh số bán cao nhất "
PRINT "Với số lượng là : " + CAST(@nMaxSL AS CHAR(10))
GO

```

Nhận xét thấy rằng trong thí dụ này chúng tôi sử dụng lệnh **RETURN** dùng để **thoát ra khỏi thủ tục nội tại** trong trường hợp khi không có vật tư nào bán ra trong tháng 01/2002. Ý nghĩa của lệnh **RETURN** dùng để thoát khỏi một thủ tục nội tại, các dòng lệnh phía sau lệnh **RETURN** sẽ không được thực hiện sau khi thủ tục thực hiện lệnh **RETURN**.

### III. Tham số bên trong thủ tục nội tại

Thủ tục nội tại **rất hữu ích** trong mô hình khách chủ, nhưng nó sẽ trở nên **hiệu quả hơn** khi chúng ta biết cách **sử dụng các tham số** đi kèm với nó. Với những tham số này chúng ta có thể **truyền vào các giá trị** cần thiết cho các xử lý bên trong của thủ tục nội tại. Bên dưới là một vài hướng dẫn mà các bạn **cần ghi nhớ** khi sử dụng tham số trong thủ tục nội tại:

- ✓ Chúng ta có thể định nghĩa ra **một hoặc nhiều tham số** bên trong một thủ tục nội tại.
- ✓ **Tên của các tham số** mà chúng ta định nghĩa chỉ có **phạm vi cục bộ** bên trong thủ tục nội tại. Chúng tôi gọi chúng là các **tham số hình thức**.
- ✓ Giống như **tên của các tham số** trong các ngôn ngữ lập trình khác, các bạn có thể đặt tên tham số trong thủ tục nội tại một cách **gợi nhớ và duy nhất**.
- ✓ Một thủ tục được phép có **tối đa 1024** tham số.

#### III.1. Tham số đầu vào

**Tham số đầu vào** là loại tham số cho phép chúng ta có thể **truyền vào các giá trị** cho những xử lý bên trong một thủ tục nội tại. Các giá trị này **thật cần thiết** cho các hành động tính toán bên trong thủ tục nội tại.

Cú pháp:



```
CREATE PROC[EDURE] Tên_thủ_tục
@Tên_tham_số Kiểu_dữ_liệu [=Giá_trị] [, ...]
AS
[DECLARE Biến_cục_bộ]
    Các_lệnh
```

Trong đó:

- ✓ **Tên tham số:** tên tham số của thủ tục phải **duy nhất** trong thủ tục và nên đặt tên tham số một cách gợi nhớ.
- ✓ **Kiểu dữ liệu:** kiểu dữ liệu của tham số qui định **loại dữ liệu** tương ứng được truyền vào cho thủ tục.
- ✓ **Giá trị:** giá trị mặc định được gán vào tham số khi tham số **không** được nhận giá trị **khi thủ tục được gọi thực hiện**.



**Ví dụ:**

Để xây dựng thủ tục nội tại tính tổng trị giá của một phiếu xuất vật tư cần có **một tham số vào** là số phiếu xuất với kiểu dữ liệu là chuỗi. Chúng ta thực hiện lệnh **CREATE PROCEDURE** có một tham số vào như sau:

```
CREATE PROC spud_TongTGXuat
@sSopx CHAR(4)
AS
DECLARE @nTongTG MONEY
SELECT @nTongTG=SUM(SLXUAT*DGXUAT)
FROM CTPXUAT
WHERE @sSopx=SOPX
PRINT "Trị giá phiếu xuất "
+ CAST(@sSopx AS CHAR(4))
PRINT "Là : "
+ CAST(@nTongTG AS VARCHAR(15))
GO
```

Gọi thực hiện thủ tục trên và truyền vào giá trị cho tham số là phiếu xuất "X001" để tính tổng trị giá của phiếu xuất "X001".

```
EXEC spud_TongTGXuat "X001"
```

Hoặc có thể **chỉ định tường minh** tên tham số và giá trị lúc gọi thủ tục.

```
EXEC spud_TongTGXuat @sSopx="X001"
```

Kết quả trả về

```
Trị giá phiếu nhập X001
Là : 6300000.00
```



**Ví dụ:**

Để xây dựng thủ tục nội tại tính ra số lượng đặt hàng của một vật tư bên trong một đơn đặt hàng có **hai tham số vào** là số đặt hàng và mã vật tư đều có kiểu dữ liệu là chuỗi. Chúng ta thực hiện lệnh **CREATE PROCEDURE** có hai tham số vào như sau:



```

CREATE PROC spud_TinhSLDat
@sSodh CHAR(4),
@sMavtu CHAR(4)
AS
    DECLARE @nSldat INT
    IF NOT EXISTS(SELECT SODH FROM CTDONDH WHERE SODH=@sSodh
                                                           AND MAVTU=@sMavtu)
    BEGIN
        PRINT "Xin xem lại số đặt hàng, mã vật tư!"
        RETURN
    END
    SELECT @nSldat=SLDAT
    FROM CTDONDH
    WHERE SODH=@sSodh AND MAVTU=@sMavtu
    PRINT "Đơn đặt hàng " + @sSodh
    PRINT "Với mã vật tư " + @sMavtu
    PRINT "Có số lượng đặt là ." + CAST(@nSldat AS VARCHAR(10))
GO

```

Gọi thực hiện thủ tục để tính ra số lượng đặt hàng của vật tư “DD02” trong đơn đặt hàng “D001”.

```
EXEC spud_TinhSLDat "D001", "DD02"
```

Hoặc

```
EXEC spud_TinhSLDat @sSodh="D001", @sMavtu="DD02"
```

Hoặc

```
EXEC spud_TinhSLDat @sMavtu="DD02", @sSodh="D001"
```

Kết quả đều trả về như nhau:

```

Đơn đặt hàng D001
Với mã vật tư DD02
Có số lượng đặt là :15

```

Nhận xét thấy rằng trong thí dụ này chúng tôi trình bày ba (3) cách gọi thực hiện thủ tục **hoàn toàn khác nhau**.

- ✓ Với cách gọi thứ nhất thì **thứ tự các giá trị** của các tham số mà chúng ta truyền vào cho thủ tục **bắt buộc phải đúng** với thứ tự các tham số hình thức đã được định nghĩa trong lệnh tạo thủ tục.
- ✓ Với cách gọi thứ hai và thứ ba thì chúng ta **chỉ định tường minh** tên tham số hình thức định nghĩa trong lệnh tạo thủ tục để truyền vào các giá trị mong muốn và **thứ tự các tham số là không còn quan trọng** nữa.

### III.2. Tham số đầu ra

Trong những thí dụ trên chúng ta thấy rõ ý nghĩa sử dụng của các tham số đầu vào, tuy nhiên trong thực tế chúng ta cũng mong muốn **nhận các giá trị trả về** từ những xử lý bên trong thủ



tục nội tại thông qua các **tham số**. Khái niệm này còn gọi là tham số đầu ra – là những tham số mà **giá trị** của nó sẽ **được tính toán** từ bên trong thủ tục nội tại và các giá trị đó sẽ được giữ nguyên sau khi thoát ra khỏi thủ tục.

Khái niệm này hoàn toàn giống như khái niệm tham số **truyền theo địa chỉ (by Reference)** trong một số các ngôn ngữ lập trình như là: Pascal, Visual Basic.

Cú pháp:

```
CREATE PROC[EDURE] Tên_thủ_tục
@Tên_tham_số Kiểu_dữ_liệu OUTPUT [, ...]
AS
[DECLARE Biến_cục_bộ]
    Các_lệnh
```

Trong đó:

✓ **Từ khóa OUTPUT:** dùng để chỉ định loại tham số là tham số đầu ra.



**Ví dụ:**

Sửa đổi lại thủ tục nội tại ở trên có **hai tham số vào** là số đặt hàng và mã vật tư đều có kiểu dữ liệu là chuỗi, trả ra số lượng đặt hàng của một vật tư tương ứng bên trong một đơn đặt hàng thông qua **một tham số ra**. Chúng ta thực hiện lệnh *ALTER PROCEDURE* có hai tham số vào và một tham số ra như sau:

```
ALTER PROC spud_TinhSLDat
@sSodh CHAR(4),
@sMavtu CHAR(4),
@nSLdat INT OUTPUT
AS
    IF NOT EXISTS(SELECT SODH FROM CTDONDH
                    WHERE SODH=@sSodh AND MAVTU=@sMavtu)
    BEGIN
        PRINT "Xin xem lại số đặt hàng, mã vật tư!"
        RETURN
    END

    SELECT @nSLdat=SLDAT
    FROM CTDONDH
    WHERE SODH=@sSodh AND MAVTU=@sMavtu
GO
```

Gọi thực hiện thủ tục

```
DECLARE @nSLdathang INT
EXEC spud_TinhSLDat @sSodh="D001",@sMavtu="DD02",
    @nSLdat=@nSLdathang OUTPUT
PRINT "Đơn đặt hàng D001 với vật tư DD02"
PRINT "Có số lượng đặt là: " + CAST(@nSLdathang AS VARCHAR(10))
```



*Kết quả trả về*

Đơn đặt hàng D001 với vật tư DD02

Có số lượng đặt là: 15

Nhận xét thấy rằng trong thí dụ này chúng ta **bắt buộc** phải khai báo thêm **một biến cục bộ** **@nSLdathang** bên ngoài thủ tục dùng để **đón nhận giá trả về** của tham số ra **@nSLdat**. Từ khóa **OUTPUT bắt buộc** phải được sử dụng tại **hai nơi** khi sử dụng các loại tham số đầu ra (lúc định nghĩa tham số và lúc gọi thực hiện thủ tục nội tại). **Tên của biến cục bộ** bên ngoài thủ tục và **tên của tham số hình thức** lúc định nghĩa thủ tục có thể được phép **trùng tên nhau**, tuy nhiên trong thí dụ trên chúng tôi **cố tình đặt tên khác nhau** để giúp cho các bạn dễ dàng phân biệt: biến cục bộ là **@nSLdathang** và tham số hình thức **@nSLdat**.

## IV. Một số vấn đề khác trong thủ tục nội tại

Trong những phần trên chúng ta đã tìm hiểu về ý nghĩa của việc sử dụng thủ tục nội tại, cách thức khai báo và sử dụng tham số bên trong thủ tục, các loại tham số của thủ tục. Trong phần này chúng tôi muốn nói thêm về một số vấn đề khác trong thủ tục nội tại như là: mã hóa nội dung của thủ tục nội tại, gọi thực hiện các thủ tục nội tại lồng nhau, các trường hợp cần thiết phải sử dụng lệnh **RETURN** bên trong thủ tục.

### IV.1. Mã hóa nội dung thủ tục

Giống như việc mã hóa nội dung của câu lệnh truy vấn trong các bảng ảo (view) mà trước đây chúng tôi đã trình bày trong chương 2, trong những trường hợp **cần phải mã hóa** các câu lệnh bên trong nội dung của thủ tục nội tại thì chúng ta sử dụng thành phần **WITH ENCRYPTION** kèm với cú pháp của lệnh tạo thủ tục nội tại hoặc lệnh sửa đổi nội dung thủ tục nội tại. Vị trí của mệnh đề này được **đặt trước** từ khóa **AS** và **phía sau** của tên tham số cuối cùng trong thủ tục (nếu có tham số).

Cú pháp:

```
CREATE|ALTER PROC[EDURE] Tên_thủ_tục
[@Tên_tham_số Kiểu_dữ_liệu [OUTPUT] [, ...] ]
WITH ENCRYPTION
AS
[DECLARE Biến_cục_bộ]
Các_lệnh
```

**Ví dụ:**

Xây dựng một thủ tục nội tại mà xử lý của nó dùng để chuyển đổi một chuỗi chứa các **ký tự** bất kỳ thành một chuỗi các **ký số** theo một **qui tắc nào đó** do chúng ta qui định trước. Tuy nhiên các xử lý trong thủ tục này cần phải được mã hóa để hạn chế người sử dụng phát hiện ra qui tắc chuyển đổi.

Mục đích của thủ tục này dùng để **mã hóa mật khẩu** của các người sử dụng chương trình và lưu



vào cột mật khẩu trong bảng người dùng (NGUOIDUNG). Nội dung của thủ tục này như sau:

```
CREATE PROC spud_Mahoa_Kytu
    @sChuoivao VARCHAR(10),
    @sChuoirra VARCHAR(10) OUTPUT
WITH ENCRYPTION
AS
    DECLARE @nI INT, @nSotong INT
    SET @nI=1
    SET @nSotong = 0
    WHILE @nI<=LEN(@sChuoivao)
    BEGIN
        SET @nSotong = @nSotong +
            ASCII(SUBSTRING(@sChuoivao, @nI, 1))*10*@nI
        SET @nI = @nI + 1
    END
    SET @sChuoirra = LTRIM(STR(@nSotong))
GO
```

Gọi thực hiện thủ tục

```
DECLARE @sChuoi_mahoa VARCHAR(10)
EXEC spud_Mahoa_Kytu "IALMW1972AMB", @sChuoi_mahoa OUTPUT
PRINT 'Chuỗi được chuyển đổi: ' + @sChuoi_mahoa
```

Kết quả trả về

```
Chuỗi được chuyển đổi: 34070
```

Sau khi đã mã hóa nội dung thủ tục bằng chức năng **WITH ENCRYPTION**, các bạn **hoàn toàn không** có cách nào để xem lại nội dung của các lệnh bên trong thủ tục. Tuy nhiên cách gọi thực hiện thủ tục nội tại đã được mã hóa vẫn bình thường như các thủ tục khác.

Chúng ta có thể **xem nội dung** của các lệnh bên trong một thủ tục nội tại **chưa được mã hóa** bằng thủ tục nội tại hệ thống **sp\_helptext**. Theo thí dụ trên nếu chúng ta thực hiện lệnh:

```
EXEC sp_HelpText spud_Mahoa_Kytu
```

Khi đó hệ thống sẽ thông báo không thể hiển thị nội dung của thủ tục vì đã bị mã hóa.

```
The object comments have been encrypted.
```

## IV.2. Biên dịch thủ tục

Chúng ta có sử dụng thành phần **WITH RECOMPILE** đi kèm với **lệnh tạo mới** thủ tục nội tại hoặc **lệnh gọi thực hiện** thủ tục nội tại dùng để chỉ định **tiến trình thực hiện** thủ tục nội tại trên máy chủ như thế nào.

Chúng ta có thể tạo thủ tục nội tại kèm theo thành phần **WITH RECOMPILE** dùng để chỉ ra thủ tục nội tại sẽ được **tự động biên dịch** lại trong **mỗi lần nó được gọi thực hiện**. Khi đó hệ



thống **sẽ không** lưu lại trong vùng nhớ của phiên bản thủ tục đã được biên dịch trước đó khi thủ tục được gọi thực hiện lần đầu tiên. Việc sử dụng chức năng này **không thông dụng** lắm vì nó sẽ làm **tốc độ** của các thủ tục nội tại **chậm đi**. Tuy nhiên, đối với các thủ tục nội tại có sử dụng nhiều câu truy vấn SELECT tham chiếu đến các bảng mà dữ liệu của chúng bị cập nhật thường xuyên thì việc biên dịch và lưu trữ phương án thực hiện các câu truy vấn như cách thông thường lại không hiệu quả bằng việc để SQL Server biên dịch thủ tục lại mỗi lần thực hiện.

Cú pháp:

```
CREATE|ALTER PROC[EDURE] Tên_thủ_tục
[@Tên_tham_số Kiểu_dữ_liệu OUTPUT [, ...] ]
WITH RECOMPILE [ ENCRYPTION ]
AS
[DECLARE Biến_cục_bộ]
Các_lệnh
```



**Ví dụ:**

Trở lại thí dụ tính tổng trị giá của từng phiếu xuất vật tư trước đây, bổ sung thêm thành phần **WITH RECOMPILE** để chỉ định hệ thống luôn luôn biên dịch lại thủ tục nội tại trong mỗi lần nó được gọi thực hiện.

```
ALTER PROC spud_TongTGXuat
@sSopx CHAR(4)
WITH RECOMPILE
AS
DECLARE @nTongTG MONEY
SELECT @nTongTG=SUM(SLXUAT*DGXUAT)
FROM CTPXUAT
WHERE @sSopx=SOPX
PRINT "Trị giá phiếu nhập " + CAST(@sSopx AS CHAR(4))
PRINT "Là : " + CAST(@nTongTG AS VARCHAR(15))
GO
```

Ngoài ra trong một trường hợp khác nữa, nếu **chỉ muốn biên dịch lại** thủ tục lúc gọi thực hiện thì chúng ta có thể sử dụng thành phần **WITH RECOMPILE** ngay sau câu lệnh gọi thực hiện thủ tục.

Cú pháp:

```
EXEC Tên_thủ_tục [Các_tham_số] WITH RECOMPILE
```

### IV.3. Thủ tục lồng nhau

Bên trong một thủ tục nội tại chúng ta có thể **gọi thực hiện** các thủ tục nội tại khác. **Cấp độ lồng** tối đa của các thủ tục là **32 cấp**. Thủ tục ngoài cùng nhất gọi là thủ tục cấp 1, thủ tục được gọi thực hiện kế tiếp là cấp 2... chúng ta có thể sử dụng biến hệ thống @@NESTLEVEL để biết được cấp độ lồng hiện hành của thủ tục nội tại là bao nhiêu. Tuy nhiên bên trong một



thủ tục chúng ta có thể gọi thực hiện nhiều thủ tục khác nhau và số lần gọi thực hiện các thủ tục khác nhau là không giới hạn



**Ví dụ:**

Để xây dựng thủ tục tính số lượng đặt hàng, tổng số lượng nhập hàng của một vật tư liên quan cho một số đơn đặt hàng chỉ định. Chúng ta có thể gọi thực hiện lại thủ tục **spud\_TinhSLDat** đã tạo ra trong thí dụ trước đây bên trong nội dung thủ tục mới.

```
CREATE PROC spud_SosanhDat_Nhap
    @sSodh CHAR(4) ,
    @sMavtu CHAR(4) ,
    @nSLdat INT OUTPUT ,
    @nSLnhap INT OUTPUT
AS
    --Gọi thực hiện lại tính số lượng đặt
    EXEC spud_TinhSLDat @sSodh, @sMavtu ,@nSLdat OUTPUT
    IF @nSLdat IS NULL
    BEGIN
        RETURN
    END
    SELECT @nSLnhap=SUM(SLNHAP)
    FROM CTPNHAP CTPN INNER JOIN PNHAP PN ON PN.SOPN=CTPN.SOPN
    WHERE SODH=@sSodh AND MAVTU=@sMavtu

    IF @nSLnhap IS NULL
        SET @nSLnhap=0
GO
```

*Gọi thực hiện thủ tục*

```
DECLARE @nSLdathang INT , @nSLnhaphang INT
EXEC spud_SosanhDat_Nhap "D006", "TV29",
    @nSLdathang OUTPUT ,
    @nSLnhaphang OUTPUT

PRINT "Đơn đặt hàng D006 với vật tư TV29"
PRINT "Có số lượng đặt là: " + CAST(@nSLdathang AS VARCHAR(10))
PRINT "Đã giao là: " + CAST(@nSLnhaphang AS VARCHAR(10))
```

*Kết quả trả về*

```
Đơn đặt hàng D006 với vật tư TV29
Có số lượng đặt là: 20
Đã giao là: 0
```

#### IV.4. Sử dụng lệnh RETURN trong thủ tục

Thông thường chúng ta sẽ sử dụng lệnh **RETURN** dùng để **thoát ra khỏi thủ tục** trong các trường hợp dữ liệu không hợp lệ. Ngoài ra lệnh **RETURN** cũng cho phép chúng ta trả về **một**



**số nguyên** tại nơi đã gọi thực hiện thủ tục. Khi đó thủ tục sẽ trả về giá trị là một số nguyên, khái niệm này giúp cho **thủ tục trở thành một hàm**. (có trả về giá trị)

Mặc định lệnh **RETURN** không có giá trị chỉ định thì thủ tục sẽ **trả về giá trị là không** (0). Phần lớn các thủ tục hệ thống hoặc các biến hệ thống sẽ **trả về giá trị 0** khi thực hiện **thành công**, trả về giá trị **khác không** dùng để chỉ định **có lỗi xảy ra** khi thực hiện thủ tục.

Cú pháp:

```
RETURN [Số_nguyên]
```

Khi bên trong một thủ tục nội tại **có sử dụng lệnh RETURN** thì thông thường chúng ta sẽ phải **sử dụng một biến cục bộ** để đón nhận **giá trị trả về** của thủ tục đó. Khi đó cú pháp của lệnh gọi thực hiện thủ tục phải được bổ sung như sau:

```
EXEC @Biến=Tên_thủ_tục [Các_tham_số]
```

Trong đó:

- ✓ **Biến**: biến cục bộ được định nghĩa để đón nhận giá trị trả về của thủ tục có sử dụng lệnh RETURN.



**Ví dụ:**

Để xây dựng một thủ tục nội tại tính ra tổng số lượng đặt hàng của một vật tư đối với một nhà cung cấp chỉ định, chúng ta cần phải kiểm tra xem giá trị của mã vật tư và mã nhà cung cấp mà người dùng truyền vào thủ tục có đúng hay không? Qui định rằng thủ tục trả về 1 khi mã vật tư không tồn tại, trả về 2 khi mã nhà cung cấp không tồn tại.

```
CREATE PROC spud_TinhTongSLdat
    @sManhacc CHAR(3) ,
    @sMavtu CHAR(4) ,
    @nTongSLdat INT OUTPUT
AS
    IF NOT EXISTS(SELECT * FROM VATTU WHERE MAVTU=@sMavtu)
        RETURN 1
    IF NOT EXISTS(SELECT * FROM NHACC WHERE MANHACC=@sManhacc)
        RETURN 2
    SELECT @nTongSLdat=SUM(SLDAT)
    FROM DONDH DH INNER JOIN CTDONDH CTDH ON DH.SODH=CTDH.SODH
    WHERE MANHACC=@sManhacc AND MAVTU=@sMavtu

    IF @nTongSLdat IS NULL
        SET @nTongSLdat=0
    RETURN
GO
```

Gọi thực hiện thủ tục

```
DECLARE @nTongSLdat INT, @nKetqua INT
EXEC @nKetqua=spud_TinhTongSLdat "C02", "TV14",
```



@nTongSLdat OUTPUT

```

IF @nKetqua=1
    PRINT "Mã vật tư không đúng"
ELSE
    IF @nKetqua=2
        PRINT "Mã nhà cung cấp không đúng"
    ELSE
        PRINT "Tổng số lượng đặt là: " +
            CAST(@nTongSLdat AS VARCHAR(10))
GO

```

*Kết quả trả về*

Tổng số lượng đã đặt là: 20

Nhận xét thấy rằng trong thí dụ này chúng ta định nghĩa ra một biến cục bộ tên **@nKetqua** có kiểu dữ liệu là số nguyên (INT) dùng để đón nhận giá trị trả về của thủ tục, sau đó căn cứ vào giá trị của biến cục bộ này để biện luận các trường hợp dữ liệu truyền vào không hợp lệ hoặc hợp lệ.

## IV.5. Sử dụng bảng tạm trong thủ tục

Đôi khi việc tính toán bên trong thủ tục **vô cùng phức tạp** mà nếu chỉ **căn cứ** vào các **dữ liệu hiện có** bên trong các bảng chúng ta **không thể tính ra** được, vì thế thông thường các bạn có thể tạo ra các **bảng ảo** (view) trước đó hoặc các **bảng tạm trung gian** ngay bên trong thủ tục để có được các dữ liệu mong muốn cho các tính toán phức tạp. Để tạo ra bảng tạm ngay bên trong thủ tục nội tại, các bạn sử dụng lệnh **SELECT INTO** với cú pháp mô tả như bên dưới.

Cú pháp:

```

SELECT Danh_sách_các_cột
      INTO #Tên_bảng_tạm
      FROM Tên_bảng_dl

```

Trong đó:

- ✓ **Tên bảng tạm:** là tên của bảng tạm sẽ được tạo lập có cấu trúc và dữ liệu từ kết quả của truy vấn chọn lựa sau đó.
- ✓ **Tên bảng dữ liệu:** là tên của bảng chứa dữ liệu nguồn.

Chúng ta có thể chỉ định các ký tự **một dấu thăng** (#) hoặc ký tự **hai dấu thăng** (##) phía trước tên bảng tạm được tạo ra trong câu lệnh **SELECT INTO** dùng để chỉ định việc tạo ra các bảng tạm **cục bộ** (#) hoặc các bảng tạm **toàn cục** (##). Thông thường chúng ta chỉ cần sử dụng bảng tạm cục bộ bên trong thủ tục và **xin nhớ rằng nên xóa đi** các bảng tạm cục bộ đã tạo ra bằng lệnh **DROP TABLE** trước khi kết thúc thủ tục.

Ví dụ:



Để tính ra vật tư nào có doanh thu bán ra cao nhất trong một năm tháng bất kỳ, chúng ta sẽ tạo một bảng tạm để tính ra tổng thành tiền bán của các vật tư, sau đó sắp xếp dữ liệu trong bảng tạm theo thứ tự giảm dần của cột tổng thành tiền. Vật tư đầu tiên trong bảng tạm chính là vật tư có doanh thu bán ra cao nhất.

```
CREATE PROC spud_TinhDThuCaonhat
    @sNamthang CHAR(7) ,
    @sTenvtu CHAR(50) OUTPUT ,
    @nTongttien MONEY OUTPUT
AS
    --Tính toán dữ liệu ra bảng tạm gồm 3 cột:
    --Mavtu, Tenvtu, Thanhvien. Sắp xếp giảm dần
    --theo cột thành tiền.
    SELECT VT.MAVTU, TENVTU, SUM(SLXUAT*DGXUAT) AS THANHVIEN
        INTO #tab_DoanhThu
    FROM PXUAT PX INNER JOIN CTPXUAT CTPX ON PX.SOPX=CTPX.SOPX
        INNER JOIN VATTU VT ON VT.MAVTU=CTPX.MAVTU
    WHERE CONVERT(CHAR(7), NGAYXUAT, 21)=@sNamthang
    GROUP BY VT.MAVTU, TENVTU
    ORDER BY 3 DESC

    --Lấy ra vật tư đầu tiên trong dữ liệu bảng tạm
    --là vật tư có doanh thu cao nhất (nhờ sắp xếp)
    SELECT TOP 1 @sTenvtu=TENVTU, @nTongttien=THANHVIEN
    FROM #tab_DoanhThu

    --Xóa dữ liệu bảng tạm trước khi thoát thủ tục
    DROP TABLE #tab_DoanhThu
GO
```

Gọi thực hiện thủ tục

```
DECLARE @sTenvtu CHAR(50), @nTongttien MONEY
EXEC spud_TinhDThuCaonhat "2002-01",
    @sTenvtu OUTPUT ,
    @nTongttien OUTPUT

IF @sTenvtu IS NULL
    PRINT "Không có dữ liệu tính toán"
ELSE
BEGIN
    PRINT RTRIM(@sTenvtu) + " có doanh thu cao nhất"
    PRINT "Là : " +CAST(@nTongttien AS VARCHAR(20)) + " VND"
END
```

Kết quả trả về

```
Đầu DVD Hitachi 3 đĩa có doanh thu cao nhất
Là : 28665000.00 VND
```

## IV.6. Tham số kiểu cursor bên trong thủ tục

Trong các trường hợp cần khai báo **tham số kiểu cursor** trong các thủ tục nội tại chúng ta cần ghi nhớ các điều như sau:

- ✓ Tham số kiểu cursor thường là loại tham số **trả về giá trị danh sách** các dòng dữ liệu theo một điều kiện chọn lọc nào đó.
- ✓ Các **hành động liên quan** khi sử dụng biến có **kiểu cursor** được **chia ra làm 2 phần**: bên trong thủ tục và bên ngoài thủ tục.

Các hành động bên trong thủ tục sẽ gồm những việc: **định nghĩa dữ liệu** cho biến kiểu cursor và **mở cursor**. Hai hành động này chính là bước 1 và bước 2 trong 4 bước cần thực hiện khi sử dụng biến có kiểu dữ liệu cursor. (xem lại phần sử dụng biến kiểu cursor trong chương 3)

Các hành động bên ngoài thủ tục sẽ gồm những việc: **đọc từng dòng** dữ liệu bên trong cursor và sau cùng là **đóng cursor** lại. Hai hành động này chính là bước 3 và bước 4 trong 4 bước cần thực hiện khi sử dụng biến có kiểu dữ liệu cursor. Tuy nhiên **không đảm bảo** dữ liệu bên trong cursor lúc được định nghĩa bên trong thủ tục **luôn luôn có**, vì thế chúng ta sẽ **chủ động qui định giá trị trả về** của thủ tục có sử dụng kiểu cursor là **0 hoặc 1** để chỉ định biến cursor có dữ liệu hoặc là không.



### Ví dụ:

*Xây dựng một thủ tục trả về danh sách các mã vật tư đã bán ra nhiều nhất trong năm tháng nào đó. Chúng ta tạo một thủ tục có tham số **kiểu dữ liệu cursor** chứa danh sách các vật tư đã bán ra nhiều nhất.*

```
CREATE PROC spud_TinhDsoban
    @sNamThang CHAR(6),
    @cur_Dsvtu CURSOR VARYING OUTPUT
AS

--Tạo bảng tạm tính ra tổng số lượng bán
SELECT CTX.MAVTU, SUM(SLXUAT) AS TONGSLBAN
                                INTO #tab_TongSLBan
FROM CTPXUAT CTX
    INNER JOIN VATTU VT ON VT.MAVTU = CTX.MAVTU
    INNER JOIN PXUAT PX ON PX.SOPX = CTX.SOPX
WHERE CONVERT(CHAR(6), NGAYXUAT, 112)= @sNamThang
GROUP BY CTX.MAVTU

--Kiểm tra dữ liệu có phát sinh?
IF EXISTS (SELECT MAVTU FROM #tab_TongSLBan)
BEGIN
    --B1: Khởi tạo giá trị biến CURSOR
    SET @cur_Dsvtu = CURSOR FORWARD_ONLY
```





```

FOR
    SELECT MAVTU, TONGSLBAN
    FROM #tab_TongSLBan
    WHERE TONGSLBAN = ( SELECT MAX(TONGSLBAN)
                        FROM #tab_TongSLBan)

--B2: Mở cursor ra
OPEN @cur_Dsvtu
DROP TABLE #tab_TongSLBan
RETURN 0
END
ELSE
    -- Khi không có dữ liệu phát sinh
    DROP TABLE #tab_TongSLBan
    RETURN 1
GO

```

Sau đó gọi thực hiện thủ tục này để đón nhận danh sách các mã vật tư đã bán ra nhiều nhất trong năm tháng 200201.

```

--Khai báo biến cục bộ
DECLARE @cur_Dsvt CURSOR, @nGttv INT, @sMavtu CHAR(4),
                                              @nTongslBan INT

--Gọi thực hiện thủ tục
EXEC @nGttv = spud_TinhDsoban '200203', @cur_Dsvt OUTPUT
--Xử lý tiếp sau đó
IF @nGttv =0
BEGIN
    PRINT 'Danh sách các vật tư'
    WHILE (0=0)
    BEGIN
        FETCH NEXT FROM @cur_Dsvt
        INTO @sMavtu, @nTongslBan
        IF @@FETCH_STATUS<>0
            BREAK
        PRINT 'Mã vật tư: ' + @sMavtu
        PRINT 'Tổng số lượng : ' + CAST(@nTongslBan AS VARCHAR(10))
        PRINT REPLICATE('-', 50)
    END
END
ELSE
    PRINT 'Không có bán hàng trong năm tháng chỉ định'

```

#### IV.7. Thủ tục cập nhật bảng dữ liệu

Trong các thí dụ mà chúng tôi trình bày ở những phần trên hầu như chức năng chính của thủ



tục chỉ dùng để **tính toán dữ liệu**, tuy nhiên chúng ta có thể sử dụng thủ tục để **cập nhật dữ liệu**. Các thủ tục dạng này sẽ được gọi thực hiện và truyền vào các giá trị cho tham số từ bên trong một môi trường khác (màn hình của Visual Basic, Visual Fox) hoặc Query Analyzer.



**Ví dụ:**

Xây dựng thủ tục thêm mới dữ liệu vào bảng VATTU với tên **spud\_VATTU\_Them** gồm có 4 tham số vào chính là các giá trị thêm mới cho các cột trong bảng VATTU: mã vật tư, tên vật tư, đơn vị tính và phần trăm. Trong đó cần kiểm tra các ràng buộc dữ liệu **phải hợp lệ** trước khi thực hiện lệnh **INSERT INTO** để thêm dữ liệu vào bảng VATTU.

Mã vật tư phải duy nhất.

Tỷ lệ phần trăm phải nằm trong miền giá trị từ 0 đến 100.

Chúng ta thực hiện lệnh **CREATE PROCEDURE** như sau:

```
CREATE PROC SPUD_VATTU_THEM
    @sMavtu CHAR(4) ,
    @sTenvtu VARCHAR(50) ,
    @sDvtinh VARCHAR(50) ,
    @nPhanTram INT
AS
-- Định nghĩa chuỗi lỗi
DECLARE @sErrMsg VARCHAR(200)
-- Kiểm tra có mã vật tư chưa?
IF EXISTS(SELECT Mavtu
          FROM VATTU
          WHERE Mavtu=@sMavtu)
BEGIN
    SET @sErrMsg = 'Mã vật tư [' + @sMavtu + '] đã có ,
                                                            xin cấp một mã khác'
    RAISERROR(@sErrMsg, 16, 1)
    RETURN
END

-- Kiểm tra tỷ lệ phần trăm nằm ngoài 0..100?
IF @nPhanTram NOT BETWEEN 0 AND 100
BEGIN
    SET @sErrMsg = 'Tỷ lệ phần trăm phải nằm trong
                    đoạn [0, 100]'
    RAISERROR(@sErrMsg, 16, 1)
    RETURN
END

-- Khi các RBTV hợp lệ thì thêm dữ liệu vào bảng VATTU
INSERT INTO VATTU (Mavtu, Tenvtu, Dvtinh, PhanTram)
VALUES (@sMavtu, @sTenvtu, @sDvtinh, @nPhanTram)
GO
```



Giả sử rằng dữ liệu trong bảng VATTU hiện tại như sau:

MAVTU TENVTU

```
-----
DD01  Đầu DVD Hitachi 1 đĩa
DD02  Đầu DVD Hitachi 3 đĩa
VD01  Đầu VCD Sony 1 đĩa
VD02  Đầu VCD Sony 3 đĩa
TV14  Tivi Sony 14 inches
TV29  Tivi Sony 29 inches
TL15  Tủ lạnh Sanyo 150 lit
TL90  Tủ lạnh Sanyo 90 lit
```

(8 row(s) affected)

Lần lượt gọi thực hiện thủ tục trong môi trường Query Analyzer để thêm dữ liệu vào bảng VATTU.

```
EXEC SPUD_VATTU_THEM 'TV29', 'Ti vi 29 inches SamSung', 'Bộ', 15
```

Khi đó hệ thống sẽ xuất hiện thông báo lỗi (mã vật tư đã có) mà chúng ta đã biện luận bên trong thủ tục.

```
Server: Msg 50000, Level 16, State 1, Line 0
```

```
Mã vật tư [TV29] đã có rồi, xin cấp một mã khác
```

Hoặc thực hiện

```
EXEC SPUD_VATTU_THEM 'SS29', 'Ti vi 29 inches SamSung', 'Bộ', -15
```

Khi đó hệ thống sẽ xuất hiện thông báo lỗi (tỷ lệ phần trăm không đúng) mà chúng ta đã biện luận bên trong thủ tục.

```
Server: Msg 50000, Level 16, State 1, Line 0
```

```
Tỷ lệ phần trăm phải nằm trong đoạn [0, 100]
```

Cuối cùng thực hiện

```
EXEC SPUD_VATTU_THEM 'SS29', 'Ti vi 29 inches SamSung', 'Bộ', 15
```

Khi đó thủ tục sẽ **không** thông báo lỗi và lưu lại dữ liệu vào bảng VATTU theo như ý chúng ta.

Qua thí dụ này các bạn có thể hiểu thêm một tính năng của thủ tục nữa là **cập nhật dữ liệu**, tương tự như thế chúng ta có thể xây dựng các thủ tục **sửa đổi** hoặc **hủy bỏ dữ liệu** trên bảng dữ liệu thông qua các thủ tục nội tại.

## IV.8. Thủ tục hiển thị dữ liệu

Ngoài ra thủ tục còn có thêm một tính năng khác nữa, đó là **hiển thị dữ liệu** nguồn cho các báo cáo. Đối với các báo cáo chúng ta có thể lấy dữ liệu từ nhiều nguồn khác nhau:

- ✓ Đối tượng bảng dữ liệu hoặc bảng ảo – view.
- ✓ Câu lệnh SELECT trực tiếp.
- ✓ Đối tượng thủ tục nội tại.

Theo chúng tôi khi lấy dữ liệu nguồn cho báo cáo thì các bạn nên chọn đối tượng thủ tục nội



tại. Bởi vì bên trong thủ tục được phép chứa đựng các cấu trúc điều khiển, các biến cục bộ để tính toán tạm thời dữ liệu và sắp xếp dữ liệu nguồn cho các báo cáo hiển thị các dữ liệu phức tạp.



#### Ví dụ:

*Xây dựng thủ tục hiển thị dữ liệu cho báo cáo đơn đặt hàng. Các thông tin trên báo cáo gồm các cột trong các bảng dữ liệu DONDH, CTDONDH và VATTU, NHACC. Thủ tục có **một tham số vào** là số đặt hàng dùng để lọc dữ liệu báo cáo theo đúng số đặt hàng gửi vào, tuy nhiên nếu khi gọi thủ tục mà không truyền vào thì xem như hiển thị tất cả các số đặt hàng có trong bảng DONDH.*

*Chúng ta thực hiện lệnh CREATE PROCEDURE như sau:*

```
CREATE PROC SPUD_DONDH_BaocaoDondh
    @sSodh CHAR(4)=NULL
AS
    --Lệnh để bỏ lệnh đếm dòng dữ liệu
    SET NOCOUNT ON
    IF @sSodh IS NULL
        --Khi không có SODH gửi vào
        SELECT DH.*, CT.MAVTU, TENVTU, SLDAT, TENNHACC
        FROM DONDH DH
            INNER JOIN CTDONDH CT ON CT.SODH = DH.SODH
            INNER JOIN NHACC NCC ON NCC.MANHACC = DH.MANHACC
            INNER JOIN VATTU VT ON VT.MAVTU = CT.MAVTU
        ORDER BY CT.SODH, CT.MAVTU
    ELSE
        --Khi có SODH
        SELECT DH.*, CT.MAVTU, TENVTU, SLDAT, TENNHACC
        FROM DONDH DH
            INNER JOIN CTDONDH CT ON CT.SODH = DH.SODH
            INNER JOIN NHACC NCC ON NCC.MANHACC = DH.MANHACC
            INNER JOIN VATTU VT ON VT.MAVTU = CT.MAVTU
        WHERE CT.SODH = @sSodh
        ORDER BY CT.MAVTU
GO
```

*Lần lượt gọi thực hiện thủ tục trong môi trường Query Analyzer để kiểm tra tính đúng đắn của thủ tục đã tạo*

```
EXEC SPUD_DONDH_BaocaoDondh 'D002'
```

*Hoặc*

```
EXEC SPUD_DONDH_BaocaoDondh
```

*Khi đó tùy vào số đặt hàng có truyền vào thủ tục hay không mà kết quả của thủ tục sẽ trả về là nhiều hoặc ít dòng dữ liệu có trong bảng DONDH và các bảng liên quan.*

## V. Giao tác (Transaction)

### V.1. Khái niệm về giao tác

Giao tác trong các cơ sở dữ liệu quan hệ lớn **được sử dụng** trong những trường hợp mà các hành động **cập nhật dữ liệu** trên **nhiều bảng khác nhau** được thực hiện **trong cùng một đơn vị** (unit). Nói một cách khác thì **các hành động** cập nhật dữ liệu trong một đơn vị sẽ **được ghi nhận** lại khi tất cả các **hành động con bên trong đó thực hiện thành công**, ngược lại nếu có **ít nhất một hành động** nào đó thực hiện **thất bại** thì tất cả các hành động bên trong đơn vị sẽ **bị hủy bỏ** để đảm bảo tính toàn vẹn của dữ liệu.



**Ví dụ:**

Chúng ta hình dung một khách hàng có cùng lúc hai loại tài khoản trong ngân hàng. **Một là tài khoản thanh toán** dùng để thực hiện các giao dịch thu chi qua lại của khách hàng với các công ty khác. **Hai là tài khoản tiết kiệm** cá nhân của khách hàng cho phép khách hàng gửi tiền tiết kiệm để lấy tiền lãi theo định kỳ 3 tháng.

Giả sử sau thời gian 3 tháng, khách hàng đến ngân hàng để nhận số tiền lãi từ tài khoản tiết kiệm cá nhân. Tuy nhiên khách hàng này muốn bộ phận giao dịch tài khoản **thực hiện tự động** chuyển số tiền lãi từ tài khoản tiết kiệm sang tài khoản thanh toán của mình.

Nhận xét thấy rằng trong hệ thống chương trình tại ngân hàng phải thực hiện hai hành động cập nhật dữ liệu: **một** là lấy ra số tiền lãi trong tài khoản tiết kiệm, **hai** là nạp số tiền lãi vào tài khoản thanh toán. Chuyện gì sẽ xảy ra nếu một trong hai hành động thực hiện không thành công mà hành động kia vẫn được ghi lại nhận vào cơ sở dữ liệu?

- ✓ **Trường hợp 1:** nếu hành động **rút số tiền lãi** trong tài khoản tiết kiệm thực hiện **thành công** và hành động **nạp số tiền lãi** đó vào tài khoản thanh toán thực hiện **thất bại** thì xem như khách hàng đã mất đi số tiền lãi của tài khoản tiết kiệm (**khách hàng mất tiền**).
- ✓ **Trường hợp 2:** nếu hành động **rút số tiền lãi** trong tài khoản tiết kiệm thực hiện **thất bại** và hành động **nạp số tiền lãi** đó vào tài khoản thanh toán thực hiện **thành công** thì xem như khách hàng có thêm số tiền lãi ở **cả hai** tài khoản (**ngân hàng mất tiền**).

Nhận xét thấy rằng cả hai trường hợp trên đều làm cho **hệ thống vi phạm tính toàn vẹn dữ liệu** và có ảnh hưởng **uy tín chất lượng** của ngân hàng. Nhưng nếu nhờ vào khái niệm của giao tác, chúng ta có thể quy định cả **hai hành động** trên sẽ nằm bên **trong một đơn vị giao tác** nhằm nói rằng chúng sẽ được **ghi nhận lại khi cả hai hành động con** bên trong đó thực hiện **thành công**, ngược lại nếu **trường hợp 1** hoặc **trường hợp 2** mô tả ở phần trên có **xảy ra** thì tất cả các hành động bên trong giao tác sẽ bị **hủy bỏ** (không ghi lại các thay đổi dữ liệu). Điều này sẽ làm cho hệ thống không vi phạm tính toàn vẹn dữ liệu.

### V.2. Giao tác không tường minh

Có **hai loại giao tác** được sử dụng trong Transaction–SQL: tường minh và không tường minh. **Mặc định** các lệnh bên trong một lô (batch) chứa các câu lệnh sẽ có loại giao tác là **không**

**tường minh**, điều này có nghĩa là nếu có ít nhất một câu lệnh thực hiện không thành công bên trong lô thì tất cả các lệnh còn lại sẽ không được ghi nhận lại. Chúng tôi hoàn toàn **không khuyến khích** các bạn sử dụng loại giao tác này.



**Ví dụ:**

Chúng ta cho thực hiện cùng lúc ba lệnh để cập nhật dữ liệu trên ba bảng khác nhau trong cùng một lô. Tuy nhiên ở câu **lệnh cuối cùng** thực hiện thất bại do vi phạm tính toàn vẹn dữ liệu khóa ngoại (vì đơn đặt hàng đã được nhận hàng rồi nên không thể xóa được), nên các lệnh trước đó trong cùng một lô sẽ không được ghi nhận lại.

```
--Thêm vật tư mới
INSERT INTO VATTU
VALUES ("BU01", "Bàn ủi PhiLip", "Cái", 17)

--Sửa đổi tên nhà cung cấp C01
UPDATE NHACC
SET TENNHACC="Lê Hoài Trí"
WHERE MANHACC="C01"

--Xóa đơn đặt hàng D001
DELETE DONDH
WHERE SODH ="D001"
GO
```

Để kiểm chứng lại các lệnh thêm vật tư mới, sửa đổi tên nhà cung cấp có được ghi nhận lại hay không? Chúng ta thực hiện các lệnh **SELECT FROM** để xem lại dữ liệu các bảng VATTU và NHACC.

```
SELECT * FROM NHACC
SELECT * FROM VATTU
```

Nhận xét thấy rằng trong thí dụ này các lệnh thêm vật tư mới, sửa đổi tên nhà cung cấp hoàn toàn không được ghi nhận lại trong lô khi câu lệnh cuối cùng thực hiện bị lỗi (vì vật tư mới không được thêm vào bảng VATTU)

### V.3. Giao tác tường minh

Thông thường giao tác tường minh được sử dụng trong các trường hợp cập nhật dữ liệu trên nhiều bảng khác nhau và phải đảm bảo các hành động này nằm trong cùng một đơn vị xử lý.

Để bắt đầu một giao tác tường minh, chúng ta phải sử dụng câu **BEGIN TRAN** trong dòng lệnh đầu tiên của một đơn vị xử lý. Để chỉ định cho Microsoft SQL Server **kết thúc giao tác** và **ghi nhận** lại các hành động cập nhật dữ liệu thì chúng ta sử dụng lệnh **COMMIT TRAN** và ngược lại sử dụng lệnh **ROLLBACK TRAN** để chỉ định cho Microsoft SQL Server **kết thúc giao tác** mà **không ghi nhận** lại các hành động cập nhật dữ liệu trong giao tác.

Lệnh chỉ định bắt đầu một giao tác



Như phần trên chúng tôi đã trình bày lệnh **BEGIN TRAN** dùng để sử dụng trong các giao tác tường minh. Mỗi giao tác có thể được phép **lồng các giao tác con** bên trong đó, chúng ta có thể chỉ định tên cho từng giao tác lồng nhau nhằm thực hiện dễ dàng việc kết thúc của mỗi giao tác. Biến hệ thống @@TRANCOUNT trả về cấp độ lồng hiện hành bên trong các giao tác. Cú pháp lệnh chỉ định bắt đầu một giao tác được mô tả như bên dưới.

Cú pháp:

```
BEGIN TRAN[SACTION] [Tên_giao_tác]
```

Trong đó:

- ✓ **Tên giao tác:** tên của giao tác được chỉ định rõ ràng, chỉ nên sử dụng tên giao tác khi **cấp độ lồng nhau** của các giao tác nhiều hơn **hai (2)** cấp.



**Ví dụ:**

Sử dụng lệnh *BEGIN TRAN* để chỉ định bắt đầu thực hiện giao tác: thêm vật tư mới vào bảng *VATTU*, tuy nhiên khi kết thúc giao tác không lưu lại vật tư này.

```
SET ANSI_WARNINGS OFF
GO
SELECT COUNT(*) AS "Tổng vật tư trước khi thêm"
FROM VATTU
BEGIN TRAN
INSERT INTO VATTU (Mavtu, Tenvtu, Dvtinh, Phantram)
VALUES ("BU01", "Bàn ủi PhiLip", "Cái", 17)
SELECT COUNT(*) AS "Tổng vật tư sau khi thêm trong gt"
FROM VATTU
ROLLBACK TRAN
SELECT COUNT(*) AS "Tổng vật tư hiện tại"
FROM VATTU
SET ANSI_WARNINGS ON
```

*Kết quả trả về*

```
Tổng vật tư trước khi thêm
-----
11

Tổng vật tư sau khi thêm trong gt
-----
12

Tổng vật tư hiện tại
-----
11
```

Nhận xét thấy rằng trong thí dụ này trước khi thực hiện giao tác chúng ta có 11 vật tư, sau đó



thêm vào một vật tư mới trong giao tác. Tuy nhiên cuối cùng khi kết thúc giao tác chúng ta không ghi lại hành động thêm vật tư bằng lệnh **ROLLBACK TRAN**, do đó tổng số vật tư vẫn là 11 vật tư khi kết thúc giao tác.

Các lệnh chỉ định kết thúc một giao tác

Theo thí dụ trên chúng ta có thể hiểu ý nghĩa của lệnh **ROLLBACK TRAN** dùng để chỉ định **kết thúc giao tác** nhưng **không ghi nhận** lại các hành động cập nhật dữ liệu bên trong giao tác.

Ngoài ra chúng ta có thể sử dụng lệnh **COMMIT TRAN** dùng để chỉ định **kết thúc giao tác** nhưng **đồng ý ghi nhận** lại các hành động cập nhật dữ liệu bên trong giao tác. Cú pháp của cả hai lệnh này được mô tả như bên dưới.

Cú pháp:

```
ROLLBACK TRAN[SACTION] [Tên_giao_tác]
```

Hoặc

```
COMMIT TRAN[SACTION] [Tên_giao_tác]
```

Trong đó:

- ✓ **Tên giao tác:** tên của giao tác được định nghĩa trước đó trong câu lệnh BEGIN TRAN.



**Ví dụ:**

Tạo một bảng tạm dùng để minh họa việc sử dụng các giao tác lồng nhau. Kết thúc giao tác ngoài cùng bằng lệnh **ROLLBACK TRAN** và không ghi nhận lại các hành động cập nhật dữ liệu của các giao tác con trước đó. Điều này có nghĩa là dữ liệu của bảng tạm #TestTran là hoàn toàn trống.

```
CREATE TABLE #TestTran (CotA INT PRIMARY KEY,
                        CotB CHAR(3))
GO
BEGIN TRAN Cap1 -- @@TRANCOUNT=1.
    INSERT INTO #TestTran VALUES (1, 'aaa')
    BEGIN TRAN Cap2 -- @@TRANCOUNT=2.
        INSERT INTO #TestTran VALUES (2, 'bbb')
        BEGIN TRAN Cap3 -- @@TRANCOUNT=3.
            INSERT INTO #TestTran VALUES (3, 'ccc')
        COMMIT TRAN Cap3 -- @@TRANCOUNT=2.
    GO
    COMMIT TRAN Cap2 -- @@TRANCOUNT=1.
GO
ROLLBACK TRAN Cap1 -- @@TRANCOUNT=0.
GO
SELECT * FROM #TestTran
GO
DROP TABLE #TestTran
GO
```





Nhận xét thấy rằng trong thí dụ này **tên của các giao tác** được sử dụng trong các lệnh **ROLLBACK TRAN** hoặc **COMMIT TRAN** chỉ để giúp cho chúng ta dễ đọc và dễ thấy được cấp độ hiện hành của các giao tác lồng nhau, nó hoàn toàn không có một mối liên hệ gì giữa tên giao tác trong các lệnh BEGIN TRAN trước đó.

Phân vùng trong giao tác

Chúng ta có thể **chỉ định** việc **đồng ý ghi nhận** hoặc **không ghi nhận** lại các hành động **cập nhật dữ liệu riêng lẻ** bên trong **một** giao tác bằng cách **phân chia** thành **nhiều vùng nhỏ** cho các câu lệnh bên trong một giao tác.

Bằng cách này chúng ta **chia nhỏ** các hành động bên trong giao tác ra **thành nhiều phần**, tương ứng **từng phần nhỏ** chúng ta có thể dễ dàng **chủ động đồng ý ghi nhận** hoặc **không ghi nhận** lại việc cập nhật dữ liệu. Cú pháp của lệnh **SAVE TRANSACTION** cho phép chúng ta có thể làm được những điều như đã mô tả ở trên.

Cú pháp:

```
SAVE TRAN[SACTION] [Tên_vùng]
    Các_lệnh
```

Trong đó:

- ✓ **Tên vùng:** dùng để chỉ định vùng chứa các lệnh cập nhật dữ liệu và tên vùng nên duy nhất trong một giao tác.
- ✓ **Các lệnh:** các lệnh được phân chia theo vùng bên trong giao tác.



**Ví dụ:**

Như thí dụ trên, tuy nhiên chúng ta muốn **phân chia** lệnh thêm mới mẫu tin thứ nhất và thứ hai trong vùng thứ nhất, lệnh thêm mới mẫu tin thứ 3 trong một vùng thứ hai trong cùng một giao tác. **Kết thúc giao tác** thực hiện **ghi nhận lại các lệnh trong vùng thứ nhất** nhưng **không ghi nhận lại các lệnh trong vùng thứ hai** (chỉ có mẫu tin thứ nhất và thứ hai được ghi lại).

```
CREATE TABLE #TestTran (CotA INT PRIMARY KEY,
    CotB CHAR(3))
GO
BEGIN TRAN
--Vùng thứ 1
SAVE TRAN Dong_1_2
    INSERT INTO #TestTran VALUES (1, 'aaa')
    INSERT INTO #TestTran VALUES (2, 'bbb')
--Vùng thứ 2
SAVE TRAN Dong_3
    INSERT INTO #TestTran VALUES (3, 'ccc')
ROLLBACK TRAN Dong_3
COMMIT TRAN Dong_1_2
GO
SELECT * FROM #TestTran
```



```
GO
DROP TABLE #TestTran
GO
```

Kiểm lỗi bên trong giao tác

Thông thường khi làm việc bên trong giao tác chúng ta sẽ **không** bao giờ chỉ định rõ ràng việc kết thúc một giao tác bằng **các lệnh cụ thể** COMMIT TRAN hoặc ROLLBACK TRAN, mà thay vào đó chúng ta sẽ **kiểm tra** theo một **điều kiện qui định trước**. Nếu **điều kiện** này bị **sai** thì bắt buộc chúng ta sẽ **không ghi nhận** các hành động cập nhật dữ liệu trong giao tác, **ngược lại** sẽ **đồng ý ghi nhận** các hành động đó.

Để làm được điều này **thông thường** chúng ta sử dụng giá trị của biến hệ thống @@ERROR trong việc kiểm tra xem kết quả của câu lệnh thực hiện gần nhất là **thành công** hoặc **thất bại**.

Giá trị của biến hệ thống @@ERROR trả về là **không** khi câu lệnh gần nhất thực hiện **thành công**, ngược lại thì trả về giá trị **khác không** khi câu lệnh gần nhất thực hiện **thất bại**.



**Ví dụ:**

*Thực hiện công việc cấp phát số chứng từ tự động cho các bảng DONDH, PNHAP, PXUAT đảm bảo rằng các số này không bị trùng lặp khi cùng lúc có nhiều người sử dụng cùng lập các chứng từ liên quan. Thực hiện từng bước như sau:*

*Đầu tiên **xây dựng bảng CAP\_SOCTU** dùng lưu trữ số chứng từ được cấp kế tiếp cho các bảng, gồm có các cột: tên bảng (tenbang), số chứng từ (soctu), ký tự đầu (kytu). Trong đó cột tên bảng tham gia làm khóa chính.*

```
CREATE TABLE CAP_SOCTU
(
    TENBANG CHAR(20) ,
    SOCTU INT ,
    KYTU CHAR(1)
    PRIMARY KEY (TENBANG)
)
```

*Kế tiếp lần lượt thêm các dòng dữ liệu vào bảng CAP\_SOCTU.*

```
INSERT INTO CAP_SOCTU VALUES ("DONDH", 100, "D")
INSERT INTO CAP_SOCTU VALUES ("PNHAP", 100, "N")
INSERT INTO CAP_SOCTU VALUES ("PXUAT", 100, "X")
```

*Sau cùng **xây dựng thủ tục cấp số chứng từ tự động đảm bảo không trùng lặp**. Có sử dụng việc kiểm tra lỗi khi thực hiện các lệnh trong giao tác.*

```
CREATE PROCEDURE spud_Cap_Soctu_ke
    @sTenbang CHAR(20),
    @sSoctuke CHAR(4) OUTPUT
AS
DECLARE @nError INT, @NRowCount INT,
        @nSoctuke INT, @sChuoiTam CHAR(4),
        @sKytu CHAR(1)
```



```
BEGIN TRAN
--Tăng số chứng từ kế tiếp
UPDATE CAP_SOCTU
SET    SOCTU = SOCTU+1
WHERE TENBANG = @sTenbang

--Kiểm tra việc tăng có thành công không?
SELECT @nError = @@ERROR ,
        @nRowCount = @@ROWCOUNT

--Nếu có lỗi hoặc không cập nhật được mẫu tin nào
IF @nError<>0 OR @nRowCount<>1
BEGIN
    ROLLBACK TRAN
    RETURN -999
END

--Lấy ra số chứng từ mà chúng ta đã tăng thành công
SELECT @nSoctuke=SOCTU,
        @sKytu=KYTU
FROM   CAP_SOCTU
WHERE  TENBANG = @sTenbang

--Kiểm tra việc lấy dữ liệu có thành công không?
SELECT @nError = @@ERROR ,
        @nRowCount = @@ROWCOUNT

--Nếu có lỗi hoặc không lấy được mẫu tin nào
IF @nError<>0 OR @nRowCount<>1
BEGIN
    ROLLBACK TRAN
    RETURN -998
END

--Tính số chứng từ khi không có lỗi nào hết
SET @sChuo iTam = LTRIM(STR(@nSoctuke))
SET @sSoctuke = @sKytu +
    REPLICATE("0", 3-LEN(@sChuo iTam)) +
    @sChuo iTam

COMMIT TRAN
RETURN 0
GO
```

Gọi thực hiện thủ tục trên để có được số chứng từ kế tiếp cho bảng PXUAT.



```
DECLARE @sSoctu CHAR(4), @nGttv INT
EXEC  @nGttv=spud_Cap_Soctu_ke "PXUAT" ,
      @sSoctu OUTPUT
IF @nGttv<>0
    PRINT "Có lỗi khi cấp số chứng từ, xem lại..."
ELSE
    PRINT "Số chứng từ mới là: "+@sSoctu
```

*Kết quả trả về*

```
Số chứng từ mới là: X101
```

Tóm lại trong chương này chúng tôi đã trình bày về đối tượng **thủ tục nội tại** (stored procedure) của Microsoft SQL Server. Việc sử dụng đối tượng này để **cung cấp các dữ liệu, các tính toán** trên các **màn hình** nhập liệu, **báo cáo** bên trong ứng dụng sẽ làm cho tốc độ các xử lý tại nhánh máy chủ được nhanh hơn trong các ứng dụng mô hình khách chủ.



## Bài 6

# HÀM DO NGƯỜI DÙNG ĐỊNH NGHĨA

### Tóm tắt

Lý thuyết 4 tiết - Thực hành 4 tiết

Mục tiêu	Các mục chính	Bài tập bắt buộc	Bài tập làm thêm
Hướng dẫn học viên cách sử dụng đối tượng mới trong SQL Server 2000 là User Define Function.	I. Khái quát về hàm do người dùng định nghĩa II. Làm việc với UDF III. Các thao tác trên UDF	6.1 6.2 6.3	



# I. Khái quát về hàm do người dùng định nghĩa

Hàm do người dùng định nghĩa (user defined functions - UDF) là một đối tượng mới được bổ sung của phiên bản SQL Server 2000. Trước hết, bạn cần biết rằng UDF mang đầy đủ các ý nghĩa và tính chất của một hàm như đa số các ngôn ngữ lập trình khác, có nghĩa là một UDF là một chương trình con đảm trách một xử lý nào đó với đặc tính là sẽ nhận các tham số đầu vào và trả về một giá trị kết quả xử lý tính toán được dựa trên các tham số đầu vào đã nhận.

Không chỉ UDF mà tất cả các hàm nói chung (Các phiên bản SQL Server trước đây cung cấp các hàm được cài đặt sẵn như `getdate()`, `object_name()`,...) có thể phân ra thành hai nhóm : hàm xác định (deterministic) và hàm không xác định (non-deterministic).

Các hàm xác định sẽ luôn luôn trả về cùng giá trị nếu giá trị các tham số được cung cấp (truyền vào) là như nhau.

Các hàm không xác định có thể tạo ra các kết quả khác biệt tại mỗi thời điểm chúng được gọi thực hiện, ngay cả khi giá trị các tham số được cung cấp vẫn không thay đổi.

Trong một số tài liệu bạn thấy có xuất hiện thuật ngữ “scalar functions – các hàm vô hướng hay hàm đơn trị” dùng để chỉ các hàm trả về một giá trị, tương phản với nó là các hàm trả về một table.

Hàm `getdate()` là một ví dụ điển hình cho cả hai : hàm vô hướng (scalar) và hàm không xác định (non-deterministic). Nếu bạn gọi thực hiện nó hai lần với cùng một dòng lệnh gọi, bạn sẽ nhận được hai kết quả khác nhau và chỉ có duy nhất một giá trị cho mỗi lần gọi.

Như chúng tôi đã trình bày, UDF mang đầy đủ các tính chất của một hàm, ngoài ra, bạn cũng có thể hình dung UDF là sự kết hợp của hai đối tượng View và Stored Procedure. Hơn thế nữa, nó còn có những tính năng sử dụng mà View và Stored Procedure không có được, hay nói cách khác nó khắc phục một số các hạn chế của View và Stored Procedure. Ví dụ : Stored Procedure không thể là một phần của câu lệnh SELECT nhưng UDF , View không hỗ trợ sự đệ quy trong khi với UDF thì bạn có thể làm được điều này.

## II. Làm việc với UDF

### II.1. Tạo mới UDF

Giống như các đối tượng khác trong Microsoft SQL Server mà chúng tôi đã trình bày trước đây ở các chương trước, các bạn có hai (2) cách để có thể tạo mới một UDF: sử dụng các câu lệnh T-SQL hoặc dùng tiện ích Enterprise Manager.

#### II.1.1. Sử dụng câu lệnh T-SQL

Bạn có thể sử dụng câu lệnh `CREATE FUNCTION` để tạo một UDF bằng cách sử dụng tiện ích SQL Query Analyzer hoặc sử dụng công cụ dạng dấu nhắc lệnh (command-prompt) điển hình như công cụ `osql`.

Cú pháp:

```
CREATE FUNCTION [Tên_FUNCTION] (Khai báo các tham số )
RETURNS Kiểu_dữ Liệu_trả_về AS
BEGIN
    --Các câu lệnh bên trong FUNCTION--
    RETURN
END
```

### Mệnh đề CREATE FUNCTION

Việc tạo UDF bắt đầu với mệnh đề CREATE FUNCTION theo sau là tên của UDF. Ngay sau tên của UDF là phần khai báo các tham số của UDF. Chú ý là các tham số phải tuân thủ quy tắc của một định danh và phải bắt đầu bằng ký hiệu @. Về nguyên tắc, một UDF có thể không có tham số.



**Ví dụ:**

UDF không có tham số

```
CREATE FUNCTION [Test_Func] ()
```



**Ví dụ:**

UDF có tham số

```
CREATE FUNCTION [Chuan_Ho_Ten] (@ChuoHoTen varchar(50))
```

### Mệnh đề RETURNS

Dùng để thiết lập kiểu dữ liệu trả về của UDF. Có hai cách thiết lập chính:

- ✓ Trả về giá trị kiểu vô hướng, còn gọi là hàm đơn trị. Khi chỉ định kiểu này, UDF sẽ trả về giá trị kết quả đơn lẻ như là: một chuỗi, một giá trị logic, hoặc một giá trị kiểu số.
- ✓ Trả về một table. Trên thực tế bạn có thể chỉ định trả về hai loại table:
  - **Inline table** (hàm đọc bảng): Với loại này, bạn chỉ có thể thực hiện bên trong nó câu lệnh SELECT rõ ràng không quá phức tạp. UDF loại này thực chất được dùng để thay thế cho các đối tượng VIEW, bởi nó khắc phục được nhược điểm không có tham số của VIEW. Bạn có thể hình dung một UDF loại inline table giống như là một VIEW có tham số.
  - **Multistatement table** (hàm tạo bảng): Khi bạn chỉ định kiểu này, UDF trở nên rất giống các Stored Procedure. Nó cho phép thực hiện các câu lệnh SELECT phức tạp, hơn nữa nó còn cho phép thực hiện các câu lệnh logic khác như UPDATE, INSERT INTO,... Đồng thời bạn có thể thiết lập cấu trúc trong cặp dấu ngoặc đơn ngay sau câu lệnh RETURNS. Loại này sẽ luôn trả về một biến table (chỉ duy nhất một biến table).



**Ví dụ:**

```
Create View V_DSHangHoa
As
    Select * From DM_HANG_HOA where MaLoai_HH='TiVi'
Go
```

**Ví dụ:**

Việc tạo view ở ví dụ trên thật khó làm cho ta thỏa mãn bởi trong trường hợp ta muốn danh sách gồm các mặt hàng không phải là loại TiVi thì sao? Bạn cũng đã biết là VIEW không hỗ trợ tham số. Để giải quyết vấn đề này, bạn có thể tạo một UDF với tham số cho phép truyền vào giá trị loại hàng hoá.

```
CREATE FUNCTION F_DSHangHoa(@LoaiHH varchar(50))
RETURNS Table AS
    Return (Select * From DM_HANG_HOA where MaLoai_HH=@LoaiHH)
Go
```

Khi sử dụng UDF loại hàm đọc bảng bạn cần quan tâm đến các quy tắc sau:

- ✓ Mệnh đề RETURNS chứa duy nhất từ khoá table. Bạn không phải định nghĩa kiểu của biến trả về, bởi vì nó là tập được định dạng bởi tập kết quả truy vấn của câu lệnh SELECT trong mệnh đề RETURN.
- ✓ Không có phần thân của hàm với việc bắt đầu bởi BEGIN và kết thúc bởi END.
- ✓ Mệnh đề RETURN chứa một câu lệnh SELECT đơn giản nằm trong cặp dấu ngoặc đơn. Câu lệnh SELECT được sử dụng trong hàm có các hạn chế giống như câu lệnh SELECT được sử dụng trong đối tượng VIEW.

**Ví dụ:**

```
CREATE FUNCTION F_DSHangHoa(@LoaiHH varchar(50),
                             @PhanTram numeric)
RETURNS @DSHangHoa Table
(
    Ma_HH varchar(50),
    Ten_HH varchar(50),
    DonGiaKhuyenMai numeric
)
As
Begin
    Insert Into @DSHangHoa(Ma_HH,Ten_HH,DonGiaKhuyenMai)
    Select ID_HH,Ten_HH,DonGiaHienHanH
    From DM_HANG_HOA where IDLoai_HH=@LoaiHH

    Update @DSHangHoa
    Set DonGiaKhuyenMai=DonGiaKhuyenMai-
        (DonGiaKhuyenMai*@PhanTram)/100

    Return
End
```

Khi sử dụng UDF loại Multistatement table bạn cần quan tâm đến các hạn chế sau:

- ✓ Không thể gọi một stored procedure từ các câu lệnh bên trong nó.



- ✓ Không thể sử dụng các hàm loại không xác định được xây dựng sẵn trong SQL Server, ví dụ: Getdate, Rand, ...
- ✓ Việc sử dụng RAISERROR và @@ERROR là hoàn toàn không hợp lệ.
- ✓ UDF không thể được sử dụng để sửa đổi thông tin trên table cơ sở.

### Mệnh đề AS

Mệnh AS được khai báo trước các câu lệnh bên trong UDF.

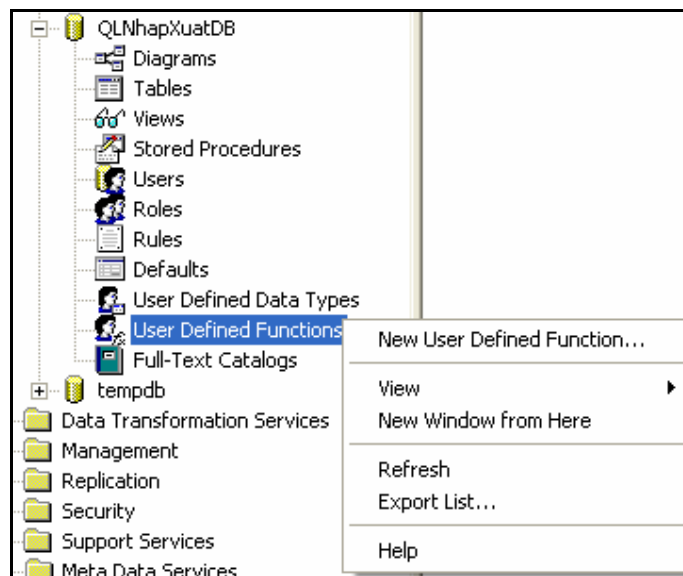
### Khối lệnh BEGIN ... END

Phần thân của UDF được bắt đầu sau từ khoá BEGIN và kết thúc bởi từ khóa END. Chú ý rằng khối lệnh này không sử dụng cho trường hợp UDF loại đọc bảng.

### II.1.2. Sử dụng tiện ích Enterprise Manager

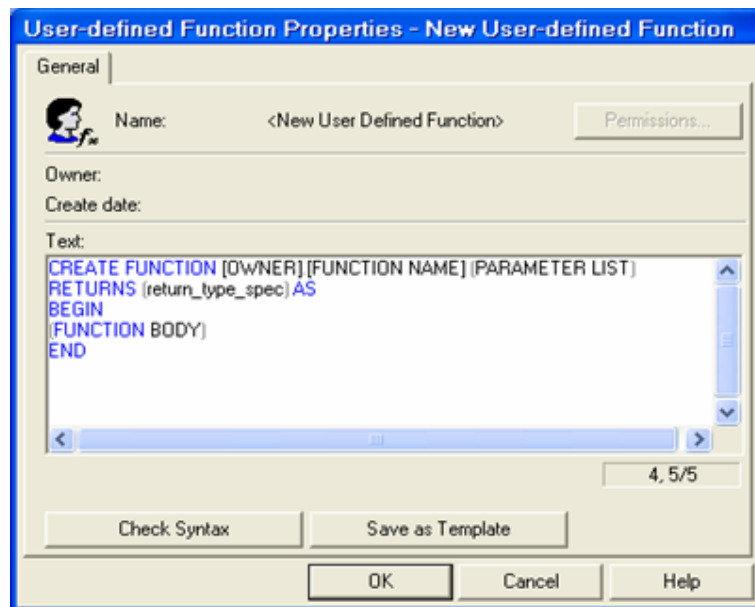
Tiện ích Enterprise Manager cho phép bạn có tạo các UDF một cách trực quan hơn. Bạn có thể thực hiện theo các bước sau :

- ✓ Bước 1. Khởi động tiện ích Enterprise Manager. Chọn mục User Defined Functions và chọn tiếp New User Defined Function.



Hình 6-1. Chọn New User Defined Function định nghĩa UDF mới.

- ✓ Bước 2. Xuất hiện màn hình chỉ định các thuộc tính liên quan đến UDF đồng thời cho phép gõ vào các câu lệnh trong phần thân của UDF.



Hình 6-2. Chọn New User Defined Function định nghĩa UDF mới.

## II.2. Quản lý UDF

### II.2.1. Thay đổi tên của UDF

Trên thực tế đôi khi bạn cần sửa đổi tên của UDF, trong trường hợp này, bạn có thể sử dụng một thủ tục do SQL Server cung cấp sẵn có tên là `sp_rename`.

Cú pháp:

```
sp_rename[@objname='tên_đối_tượng',
          [@newname='tên_mới'
          [, [ @objtype = ] 'loại_đối_tượng']
```



#### Ví dụ:

```
sp_rename @objname='F_DSHangHoa',
          @newname='F_DSHangHoa_Test',
          @objtype = 'OBJECT'
```

Nếu thực hiện thành công `sp_rename` sẽ trả về giá trị 0, nếu bị lỗi sẽ trả về giá trị khác 0.

### II.2.2. Thay đổi nội dung các câu lệnh chứa bên trong UDF.

Để sửa đổi các câu lệnh bên trong UDF, bạn có thể xóa và tạo lại UDF. Thay vì vậy, bạn có thể bỏ qua bước xóa bằng cách sử dụng câu lệnh `ALTER FUNCTION`.

Cú pháp của câu lệnh `ALTER FUNCTION` cũng giống như `CREATE FUNCTION`, nhưng `ALTER FUNCTION` không gỡ bỏ UDF ra khỏi các table hệ thống `SysComments` và `SysObjects`.

**Ví dụ:**

```
ALTER FUNCTION [dbo].[Test_function] (@b int, @c int)
    RETURNS int
AS
BEGIN
    Declare @kq int
    If @b>2
        set @kq= @b+@c
    Else
        set @kq=@b+@c+1
    Return @kq
END
```

**II.2.3. Xem các UDF**

Khi UDF được tạo, tên và thông tin nhận dạng khác liên quan UDF được lưu trữ trong table hệ thống có tên SysObjects.

Bạn có thể sử dụng câu lệnh SELECT để hiển thị các UDF đang có.

**Ví dụ:**

```
SELECT * FROM QLNhapXuatDB ..SysObjects WHERE type = 'IF' or type='FN'
```

**Chú ý:**

Type='IF' tương ứng với UFDs loại hàm tạo bảng và hàm đọc bảng ; type='FN' tương ứng với UFDs loại hàm đơn trị.

Bạn có thể truy vấn cột TEXT của table hệ thống SysComments để thấy các câu lệnh chứa bên trong một UDF. Điều này thật hữu ích nếu bạn muốn sửa lại nội dung của một UFDs khi phải đang làm việc trên một tiện ích không phải là Enterprise Manager.

**Ví dụ:**

```
sp_helptext @objname = InLineF_DSHangHoa
```

**II.2.4. Xoá UFDs**

Xoá một UFDs có nghĩa là gỡ bỏ nó ra khỏi các table hệ thống SysComments và SysObjects. Bạn có thể sử dụng câu lệnh DROP FUNTION để xoá UDF.

**Ví dụ:**

```
DROP FUNTION Test_function
```

### III. Các thao tác trên UDF

#### III.1. Gọi thực hiện các UDF thuộc loại hàm đơn trị

Đối với các UDF thuộc loại hàm đơn trị, bạn có thể gọi thực hiện chúng tại bất kỳ nơi nào mà một biểu thức đơn trị có kiểu dữ liệu tương đương được chấp nhận trong các câu lệnh T-SQL.

##### III.1.1. Sử dụng trong các câu truy vấn (Queries):

Sử dụng như là một biểu thức trong danh sách chọn (Select\_list) của một câu lệnh SELECT:



**Ví dụ:**

Xây dựng một UDF có tên ChuanChuoi với tham số truyền vào là một chuỗi, trả về một chuỗi được chuẩn theo qui ước cắt bỏ các khoảng trắng đầu và cuối chuỗi, cắt bỏ các khoảng trắng thừa sao cho giữa hai từ chỉ cách nhau bởi duy nhất một khoảng trắng, ký tự đầu tiên của từ là ký tự hoa, các ký tự còn lại là chữ thường. UDF này có thể được xây dựng như sau:

```
CREATE FUNCTION ChuanChuoi(@Chuoi varchar(50))
RETURNS varchar(50) AS
BEGIN
    declare @i int
    while CharIndex(' ',@Chuoi)>0
        set @Chuoi=replace(@Chuoi,' ',' ')
    set @Chuoi=LTrim(RTrim(@Chuoi))
    set @Chuoi=Lower(@Chuoi)
    set @Chuoi=' '+ @Chuoi
    set @i=CharIndex(' ',@Chuoi)
    while @i<>0
    Begin
        set @Chuoi=STUFF( @Chuoi,@i+1,1,Upper(substring(@Chuoi,@i+1,1)))
        set @i=CharIndex(' ',@Chuoi,@i+1)
    End
    set @Chuoi=STUFF( @Chuoi,1,1,null)
    Return(@Chuoi)
END
```

Trong câu lệnh SELECT bạn có thể gọi thực hiện UDF trên như sau :

```
SELECT ID_KHG, dbo.ChuanChuoi(Ten_KHG) as TenChuan FROM DM_KHACH_HANG
```

Sử dụng như là một biểu thức trong một mệnh đề WHERE hoặc HAVING



**Ví dụ:**

```
SELECT * FROM DM_KHACH_HANG
WHERE dbo.ChuanChuoi(Ten_KHG)='Tran Ti'
```

**Sử dụng như là một biểu thức trong một mệnh đề GROUP BY****Ví dụ:**

```
SELECT  dbo.ChuanChuoi(DM_KHACH_HANG.Ten_KHG), COUNT(PHIEU_XUAT.ID_PX) AS
[So lan mua] FROM      DM_KHACH_HANG INNER JOIN
PHIEU_XUAT ON DM_KHACH_HANG.ID_KHG = PHIEU_XUAT.ID_KHG
GROUP BY  dbo.ChuanChuoi(DM_KHACH_HANG.Ten_KHG)
```

**Sử dụng như là một biểu thức trong một mệnh đề ORDER BY****Ví dụ:**

```
SELECT * FROM PHIEU_XUAT
ORDER BY dbo.ChuanNgay(Ngay_PX))
```

**Sử dụng như là một biểu thức trong mệnh đề SET bên trong một câu lệnh UPDATE****Ví dụ:**

```
UPDATE DM_KHACH_HANG
SET Ten_KHG=dbo.ChuanChuoi(Ten_KHG)
```

**Sử dụng như là một biểu thức trong một mệnh đề VALUES của câu lệnh INSERT****Ví dụ:**

```
Declare @HoTen varchar(50)
SET @HoTen=' tran thi mit'
INSERT DM_KHACH_HANG
VALUES('KHG05',dbo.ChuanChuoi(@HoTen))
```

**III.1.2. Sử dụng trong kiểm tra ràng buộc dạng CHECK constraint****Ví dụ:**

*Xây dựng UDF với mục tiêu hàm này sẽ nhận tham số đầu vào mà mã hàng hoá, hàm sẽ tìm và trả về giá trị đơn giá hiện hành tương ứng với mã hàng hoá được lưu trữ trong table DM\_HANG\_HOA.*

```
CREATE FUNCTION DonGiaHienHanh(@MaHH varchar(50))
RETURNS numeric AS
BEGIN
    Return(Select  DonGiaHienHanh
              From DM_HANG_HOA Where ID_HH=@MaHH)
END
```

*Bạn có thể sử dụng hàm đã xây dựng ở trên cho việc kiểm tra ràng buộc trên table CT\_PHIEU\_XUAT với yêu cầu giá trị tại cột DonGia không được vượt quá 10 phần trăm đơn giá hiện hành của mặt hàng tương ứng đang được lưu trữ trong table DM\_HANG\_HOA:*

```
ALTER TABLE CT_PHIEU_XUAT
ADD CONSTRAINT LayDonGia
CHECK (DonGia<=dbo.DonGiaHienHanh(ID_HH) +
      ((dbo.DonGiaHienHanh(ID_HH)*10)/100))
```

### III.1.3. Sử dụng để định nghĩa giá trị mặc định của cột trong table bằng từ khoá DEFAULT



**Ví dụ:**

Xây dựng UDF với mục tiêu hàm này sẽ trả về giá trị đơn giá hiện hành nhỏ nhất đang được lưu trữ trong table DM\_HANG\_HOA.

```
CREATE FUNCTION DonGiaMin()
RETURNS numeric AS
BEGIN
    Return(Select  Min(DonGiaHienHanH) From DM_HANG_HOA )
END
```

Bạn có thể sử dụng hàm đã xây dựng ở trên cho việc định nghĩa giá trị mặc định của cột DonGia trên table CT\_PHIEU\_XUAT như sau :

```
ALTER TABLE CT_PHIEU_XUAT
ADD
CONSTRAINT Def_DonGia DEFAULT (dbo.DonGiaMin()) FOR DonGia
```

### III.1.4. Sử dụng cho các cột có dạng cột tính toán trong table (Computed columns)

Trong trường hợp việc tính toán cho cột dạng tính toán trên table có tính phức tạp, bạn có thể xây dựng một UDF để thực hiện điều này.



**Ví dụ:**

```
CREATE FUNCTION [dbo].[Test_function] (@b int, @c int)
RETURNS int AS
BEGIN
    Declare @kq int
    If @b>2
        Set @kq= @b+@c
    Else
        Set @kq=@b+@c+1
    Return @kq
END
```

Bạn có thể sử dụng UDF đã xây dựng ở trên để gán giá trị tính toán cho cột trên table bằng cách gọi thực hiện chúng theo cách sau :

```
CREATE TABLE TableTestFunction
(
    a int primary key,
    b int,
    c int,
    d as [dbo].[Test_function](b,c)
)
```



### III.1.5. Sử dụng như là một biểu thức trong câu lệnh CASE

Trong một số trường hợp mà chương trình có sử dụng CASE, nếu biểu thức điều kiện của CASE quá phức tạp, hoặc để phân nhỏ xử lý nhằm làm giảm bớt độ phức tạp của các câu lệnh T-SQL, bạn có thể xây dựng một UDF với nhiệm vụ xử lý trả về giá trị thích hợp trường hợp và bạn sử dụng chúng như là một biểu thức trong CASE.



#### Ví dụ:

*Yêu cầu liệt kê danh sách các khách hàng, nếu là khách hàng có doanh số mua lớn hơn tổng số tiền X thì cột loại khách hàng sẽ có giá trị “Khách hàng thân thiện”, ngược lại sẽ là “Khách hàng bình thường”. Ta sẽ xây dựng UDF có hai tham số đầu vào là mã khách hàng và số tiền xác định loại khách hàng. UDF sẽ được xây dựng như sau:*

```
CREATE FUNCTION TestFuncCase(@MaKH varchar(50),
                             @SoTien numeric)
RETURNS int AS
BEGIN
    Declare @TongSoTien numeric, @KetQua int
    Select @TongSoTien=Sum(ThanhTien)
    From CT_PHIEU_XUAT
    Where ID_PX in(Select ID_PX
                  From PHIEU_XUAT
                  Where ID_KHG=@MaKH)
    If @TongSoTien>@SoTien
    Set @KetQua=1
    Else
    Set @KetQua=0
    Return @KetQua
END
```

*Bạn có thể sử dụng UDF đã xây dựng ở trên như là một biểu thức của CASE bằng cách gõ vào các câu lệnh như sau:*

```
SELECT ID_KHG, Ten_KHG,
       Case dbo.TestFuncCase(ID_KHG,500)
       When 1 Then 'Khach hang than thiet'
       Else 'Khach hang binh thuong'
       End as 'Loai KHG'
FROM DM_KHACH_HANG
```

### III.1.6. Tính đệ quy của UDF

Một UDF có khả năng gọi lại chính nó hay còn gọi là “đệ quy”. Lập trình đệ quy nói chung là cách lập trình nên tránh do nó thiếu tường minh và đôi khi chúng dường như nằm ngoài tầm kiểm soát chắc chắn của bạn, thông thường gặp những bài toán như vậy người ta sẽ tìm cách xây dựng thuật toán theo hướng khử đệ quy. Tuy nhiên, trong một số bài toán chúng tỏ ra đặc biệt hữu dụng, đó cũng là lý do khiến chúng vẫn tồn tại trong lập trình. Chúng tôi sẽ minh họa

khả năng đệ quy của UDF qua ví dụ tính giai thừa, một ví dụ đặc trưng về tính đệ quy :



**Ví dụ:**

```
CREATE FUNCTION GiaiThua (@x bigint)
RETURNS bigint
AS
BEGIN
    Declare @i bigint
    If @x > 20 OR @x IS NULL
        Set @i = NULL
    Else
        If @x < 2
            Set @i = @x
        Else
            Set @i = @x * dbo.GiaiThua(@x - 1)
    Return @i
END
```

Bạn có thể gõ vào dòng lệnh sau để gọi UDF tính giai thừa trên:

```
SELECT dbo.GiaiThua (3)
```

### III.2. Sử dụng các UDF thuộc loại hàm đọc bảng

Ý nghĩa và cách tạo một UDF thuộc loại đọc bảng chúng tôi đã đề cập đến ở mục (II.1.1) của chương này. Ở đây, chúng tôi muốn đề cập đến cách sử dụng các UDF loại này sau khi chúng được tạo.



**Ví dụ:**

Xây dựng UDF với yêu cầu nhận tham số đầu vào là giá trị kiểu ký tự tương ứng với ký tự đầu tiên của họ tên cần tìm. Câu lệnh SELECT bên trong UDF sẽ thực hiện việc tạo danh sách khách hàng có ký tự đầu tiên của họ tên giống với ký tự tham số đầu vào. Kết quả trả về của UDF là một bảng chứa danh sách khách hàng tìm được.

```
CREATE FUNCTION dbo.LocDSKhachHang(@KyTuDau char(1))
RETURNS TABLE
AS
RETURN SELECT *
FROM DM_KHACH_HANG
WHERE LEFT(Ten_KHG, 1) = @KyTuDau
```

Việc sử dụng UDF đã xây dựng ở trên có thể thực hiện như sau:

```
SELECT * FROM dbo.LocDSKhachHang('T')
```

### III.3. Sử dụng các UDF thuộc loại hàm tạo bảng

Tương tự như đối với loại đọc bảng, ý nghĩa và cách tạo một UDF thuộc loại hàm tạo bảng chúng tôi đã đề cập đến ở mục (II.1.1) của chương này. Ở đây, chúng tôi muốn đề cập đến cách sử dụng chúng sau khi được tạo.



**Ví dụ:**

Xây dựng UDF với yêu cầu nhận tham số đầu vào là giá trị loại hàng hoá và phần trăm giảm giá khuyến mãi. UDF sẽ xử lý tính toán và tạo bảng trả về chứa thông tin mã hàng hoá, tên hàng hoá, đơn giá khuyến mãi. UDF được viết như sau:

```
CREATE FUNCTION F_DSHangHoa(@LoaiHH varchar(50),
                             @PhanTram numeric)

RETURNS @DSHangHoa Table
(
    Ma_HH varchar(50),
    Ten_HH varchar(50),
    DonGiaKhuyenMai numeric
)
AS
Begin
    Insert Into @DSHangHoa(Ma_HH,Ten_HH,DonGiaKhuyenMai)
    Select ID_HH,Ten_HH,DonGiaHienHanH
    From DM_HANG_HOA where IDLoai_HH=@LoaiHH

    Update @DSHangHoa
    Set DonGiaKhuyenMai=DonGiaKhuyenMai-
        (DonGiaKhuyenMai*@PhanTram)/100

    Return
End
```

Để sử dụng UDF đã xây dựng ở trên, bạn gõ vào các câu lệnh sau:

```
SELECT * FROM dbo.F_DSHangHoa('TiVi',10)
```

Cách sử dụng khiến ta có thể hình dung UDF loại này vừa giống như một VIEW đồng thời cũng giống như một STORED PROCEDURE. Thực vậy, UDF loại hàm tạo bảng là sự kết hợp giữa VIEW và STORED PROCEDURE. Nó lấy các tính năng của một STORED PROCEDURE để khắc phục các nhược điểm của một VIEW.



# Bài 7

## TRIGGER

### Tóm tắt

Lý thuyết 6 tiết - Thực hành 14 tiết

Mục tiêu	Các mục chính	Bài tập bắt buộc	Bài tập làm thêm
<ul style="list-style-type: none"> <li>✓ Trình bày các khái niệm về trigger</li> <li>✓ Giới thiệu các thao tác đối với trigger (thêm mới, xóa, chỉnh sửa nội dung)</li> <li>✓ Giới thiệu một số vấn đề trong quá trình xây dựng trigger để kiểm tra ràng buộc dữ liệu</li> <li>✓ Giới thiệu một số vấn đề trong quá trình xây dựng trigger để cập nhật tự động giá trị</li> </ul>	I. Khái quát về trigger II. Làm việc với trigger III. Trigger kiểm tra ràng buộc dữ liệu IV. Trigger cập nhật giá trị tự động	7.1 7.2 7.3 7.4 7.5	7.6



# I. Khái quát về trigger

## I.1. Trigger là gì?

Trigger còn được xem là một dạng đặc biệt của thủ tục nội tại, bởi vì bên trong nội dung của trigger lưu trữ các câu lệnh dùng để thực hiện một số hành động nào đó mà người lập trình sẽ chỉ ra. Tuy nhiên khác với thủ tục nội tại, trigger **hoàn toàn không có tham số và được liên kết với một bảng trong CSDL**. Ngoài ra chúng ta không thể gọi thực hiện trực tiếp trigger bằng lệnh EXECUTE như thủ tục nội tại hoặc bằng bất kỳ một lệnh nào khác, thay vào đó trigger sẽ được thực hiện tự động khi dữ liệu của bảng có liên quan đến trigger được cập nhật: thêm, xóa hay sửa.

Chính nhờ vào tính năng đặc biệt là **tự động thực hiện** mà phần lớn nội dung các lệnh bên trong trigger chỉ dùng để kiểm tra các ràng buộc toàn vẹn dữ liệu phức tạp. **Mặc định** các xử lý bên trong trigger sẽ hoạt động theo **cơ chế giao tác** (transaction) bắt đầu bằng lệnh INSERT, DELETE hay UPDATE tác động lên bảng của trigger đó, do vậy trong trường hợp nếu các dữ liệu cập nhật vào bên trong bảng được kiểm tra **có vi phạm** các ràng buộc toàn vẹn dữ liệu đã định nghĩa trước đó thì chúng ta sẽ **không cho phép** lưu lại các cập nhật trước đó bằng cách sử dụng lệnh **ROLLBACK TRANSACTION** bên trong trigger. Ngược lại khi các dữ liệu **không vi phạm** các ràng buộc toàn vẹn thì chúng ta **không cần làm gì cả** và lúc đó mặc định dữ liệu sẽ được ghi nhận xuống bên dưới bảng.

## I.2. Các xử lý bên trong trigger

Phần lớn các **ràng buộc toàn vẹn dữ liệu đơn giản** như là: **khóa nội, khóa ngoại, miễn giá trị...** sẽ được tổ chức ngay bên trong câu lệnh **CREATE TABLE** thông qua các đối tượng constraint. Tuy nhiên đối với các ràng buộc toàn vẹn dữ liệu phức tạp khác – là những qui tắc được định nghĩa dùng để kiểm tra tính toàn vẹn của dữ liệu trên **nhiều cột** hoặc **nhiều dòng** của **các bảng khác nhau**. Khi đó bắt buộc chúng ta phải xây dựng các hành động kiểm tra ràng buộc toàn vẹn dữ liệu phức tạp bên trong các trigger liên quan đến các bảng dữ liệu tương ứng.

Về **thứ tự** bên trong của hệ thống Microsoft SQL Server **khi thực hiện kiểm tra các ràng buộc toàn vẹn** dữ liệu trong một bảng dữ liệu sẽ là:

- ✓ Các ràng buộc toàn vẹn dữ liệu trong **đối tượng constraint** sẽ được **thực hiện trước**.
- ✓ Kế tiếp mới đến các ràng buộc toàn vẹn dữ liệu trong đối tượng trigger (nếu có) sẽ được thực hiện.

Do vậy khi sử dụng trigger chúng ta phải tạm thời tắt bỏ các kiểm tra ràng buộc toàn vẹn dữ liệu của đối tượng constraint trên bảng lên quan bằng câu lệnh **ALTER TABLE NOCHECK CONSTRAINT** để cho các hành động bên trong đối tượng trigger được thực hiện.

Phần lớn các xử lý bên trong trigger mà chúng ta sẽ phải cần thực hiện bao gồm:



- ✓ Kiểm tra các ràng buộc toàn vẹn dữ liệu phức tạp.
- ✓ Cập nhật tự động dữ liệu các bảng liên quan khi dữ liệu của một bảng nào đó được cập nhật.
- ✓ Chỉ định các chuỗi lỗi tiếng Việt để giúp cho người sử dụng chương trình dễ hiểu và sửa sai.

### 1.2.1. Kiểm tra các ràng buộc dữ liệu phức tạp

Khác với **CHECK constraint** mà chúng tôi đã trình bày trước đây chỉ dùng để kiểm tra các ràng buộc toàn vẹn dữ liệu **miền giá trị** của các cột bên trong một bảng dữ liệu, các hành động trong một trigger có thể tham chiếu đến giá trị của một hoặc nhiều cột trong các bảng khác nhau nhằm kiểm tra, **so sánh** theo các **điều kiện** nào đó.



#### Ví dụ:

Khi thực hiện **thêm mới** một phiếu nhập hàng cho một số đặt hàng trước đó thì chúng ta cần phải kiểm tra ràng buộc toàn vẹn dữ liệu phức tạp là **giá trị của cột ngày nhập hàng** trong bảng PNHAP **phải sau giá trị của cột ngày đặt hàng** trong bảng DONDH.



#### Ví dụ:

Khi thực hiện **sửa đổi** số lượng đặt hàng của một số đặt hàng trong bảng CTDONDH thì chúng ta phải kiểm tra **giá trị số lượng đặt hàng** sau khi sửa **không được phép nhỏ hơn tổng số lượng đã nhập hàng** (nếu có) trong bảng CTPNHAP tương ứng.

### 1.2.2. Cập nhật tự động dữ liệu các bảng liên quan

Các hành động này thông thường được viết trong những trigger **thêm, sửa và xóa dữ liệu** của bảng nhằm đảm bảo các **thay đổi dữ liệu** giữa các **bảng có liên quan được nhất quán, đồng bộ** khi dữ liệu của các bảng khác bị cập nhật bên trong một cơ sở dữ liệu.



#### Ví dụ:

Khi thực hiện việc **hủy bỏ** một phiếu xuất trong bảng PXUAT thì chúng ta sẽ ra lệnh **hủy bỏ tự động** chỉ tiết các phiếu xuất liên quan trong bảng CTPXUAT.



#### Ví dụ:

Khi **thêm mới** một dòng dữ liệu trong bảng CTPNHAP thì chúng ta sẽ ra lệnh **cập nhật tự động** lại giá trị của cột TSLNHAP trong bảng TONKHO tương ứng với mã vật tư và tháng năm đã lập trong bảng PNHAP.

### 1.2.3. Chỉ định các chuỗi lỗi dễ hiểu

Thông thường khi dữ liệu **có vi phạm các ràng buộc** toàn vẹn dữ liệu đã định nghĩa thông qua **đối tượng constraint** hay các qui tắc kiểm tra miền giá trị dữ liệu (rule) thì hệ thống sẽ **hiển thị các thông báo lỗi** của Microsoft SQL Server đã được định nghĩa sẵn, điều này thường là **một khó khăn** cho những người sử dụng. Trong các trường hợp này nếu chúng ta muốn **hiển thị**



các **thông báo lỗi thật rõ ràng, dễ hiểu** và có thể đã được **Việt hóa hoàn toàn** thì bắt buộc chúng ta phải sử dụng hàm **RAISERROR** bên trong các xử lý của trigger để hiển thị các thông báo lỗi dạng này.



**Ví dụ:**

Trở lại hai thí dụ trên sau khi kiểm tra dữ liệu **đã vi phạm** các ràng buộc toàn vẹn dữ liệu đã định nghĩa thì chúng ta sẽ hiển thị những **thông báo lỗi rõ ràng** cho người dùng dễ hiểu như là: “Ngày nhập hàng phải sau ngày đặt hàng tương ứng” hoặc “Không được phép sửa đổi số lượng đặt hàng nhỏ hơn tổng số lượng đã nhập hàng hiện tại”

### 1.3. Các hạn chế trên trigger

Microsoft SQL Server có **hạn chế** một vài câu lệnh mà chúng ta **không thể thực hiện** bên trong các xử lý của trigger. Phần lớn các câu lệnh này là những câu lệnh làm **thay đổi đến cấu trúc cơ sở dữ liệu** như là: các lệnh **CREATE, ALTER hoặc DROP**; các lệnh liên quan đến **việc cấp phát quyền hạn** trên các đối tượng trong cơ sở dữ liệu như là: **GRANT, REVOKE hoặc DENY**.

Ngoài ra bên trong trigger chúng ta **không thể tạo ra** hoặc **tham chiếu** đến các **bảng tạm** (bắt đầu bằng ký tự # hoặc ##) hoặc các **bảng ảo** (view) trong quá trình thực hiện các xử lý tính toán.

### 1.4. Các loại trigger

Kể từ phiên bản 2000, SQL Server cung cấp cho bạn hai loại trigger là AFTER Trigger và INSTEAD OF Trigger.

After trigger đã có từ những phiên bản trước. Khi có một hành động cập nhật dữ liệu (thêm/xoá/sửa) xảy ra trên bảng, After trigger sẽ được thực hiện sau cùng tức là sau khi SQL Server thực hiện các hành động kiểm tra kiểu dữ liệu, và các ràng buộc toàn vẹn như khoá chính, khoá ngoại, miền giá trị,... Do đó khi After trigger thực hiện thì dữ liệu đã được cập nhật vào bảng.

Ngược lại với After trigger, Instead of trigger xảy ra trước các hành động kiểm tra dữ liệu khác của SQL Server. Do đó khi Instead of trigger thực hiện, dữ liệu chưa thực sự thêm vào bảng giống như tên của trigger mô tả. Sử dụng Instead of trigger, bạn có thể sửa đổi hành động cập nhật dữ liệu vào bảng này trở thành hành động cập nhật dữ liệu vào một hay nhiều bảng khác.

Những người làm việc quen với SQL Server và ở những phiên bản trước SQL Server 2000 đều quen với việc chỉ có một loại trigger là after trigger. Do đó, khi nói đến trigger thì bạn nên hiểu đó là after trigger, instead of trigger khi đề cập đến thường sẽ được dùng tên đầy đủ với từ “instead of” đi kèm.

### 1.5. Các bảng trung gian Inserted và Deleted

Phần lớn khi xây dựng các xử lý bên trong các trigger chúng ta thường xuyên tham chiếu đến



dữ liệu bên trong hai bảng **Inserted** và **Deleted**. Cấu trúc của hai bảng này **hoàn toàn giống** với cấu trúc của bảng dữ liệu liên quan đến trigger khi tạo ra. Thật ra hai bảng này **chỉ tồn tại trong bộ nhớ** của máy tính (RAM) được xem như là hai **bảng luận lý** mà chúng ta **có thể sử dụng** trong các xử lý của trigger. Chúng ta không thể tham chiếu trực tiếp tới những bảng này trong trong các thủ tục nội tại.

Bảng **Deleted** qui định dùng để lưu trữ **các dòng dữ liệu bị hủy bỏ** trong các lệnh **DELETE**, khi câu lệnh DELETE được thực hiện thì các dòng dữ liệu nào bị xóa sẽ được lưu trữ vào bên trong bảng Deleted.

Tương tự như thế bảng **Inserted** qui định dùng để lưu trữ các **dòng dữ liệu được thêm mới** trong các lệnh **INSERT**, khi câu lệnh INSERT được thực hiện thì các dòng dữ liệu vừa được thêm mới sẽ được lưu trữ vào bên trong bảng Inserted.

Ngoài ra lệnh **UPDATE** trong Microsoft SQL Server được xem như là **sự phối hợp** của hai lệnh **DELETE** và **INSERT** (xóa bỏ dữ liệu cũ và thêm vào dữ liệu mới sau khi sửa đổi) do thế mà đối với các trigger liên quan đến việc sửa đổi dữ liệu thì chúng ta có thể **tham chiếu đến cả hai bảng** trung gian Inserted và Deleted.

Trong đó bảng Deleted sẽ chứa đựng thông tin của các dòng dữ liệu đã bị hủy bỏ – các dòng dữ liệu cũ trước khi sửa đổi, bảng Inserted sẽ chứa đựng thông tin của các dòng dữ liệu mới vừa thêm vào – các dòng dữ liệu sau khi sửa đổi.

Dữ liệu của hai bảng **Inserted**, **Deleted** sẽ lưu trữ các dòng dữ liệu vừa được thêm mới hoặc vừa bị hủy bỏ trong bảng dữ liệu liên quan đến trigger. Tuy nhiên như đã nói về After trigger, các dữ liệu thật bên dưới bảng dữ liệu cũng bị ảnh hưởng thật sự. Vì thế trong after trigger **không bao giờ** có lặp lại các lệnh **INSERT INTO** hoặc **DELETE** trên dữ liệu của chính bảng mà trigger đang tham chiếu đến. Ngược lại khi dữ liệu vi phạm các ràng buộc toàn vẹn được kiểm tra bên trong trigger thì các cập nhật dữ liệu trước đó phải gỡ bỏ ra khỏi bảng bằng lệnh **ROLLBACK TRAN**.

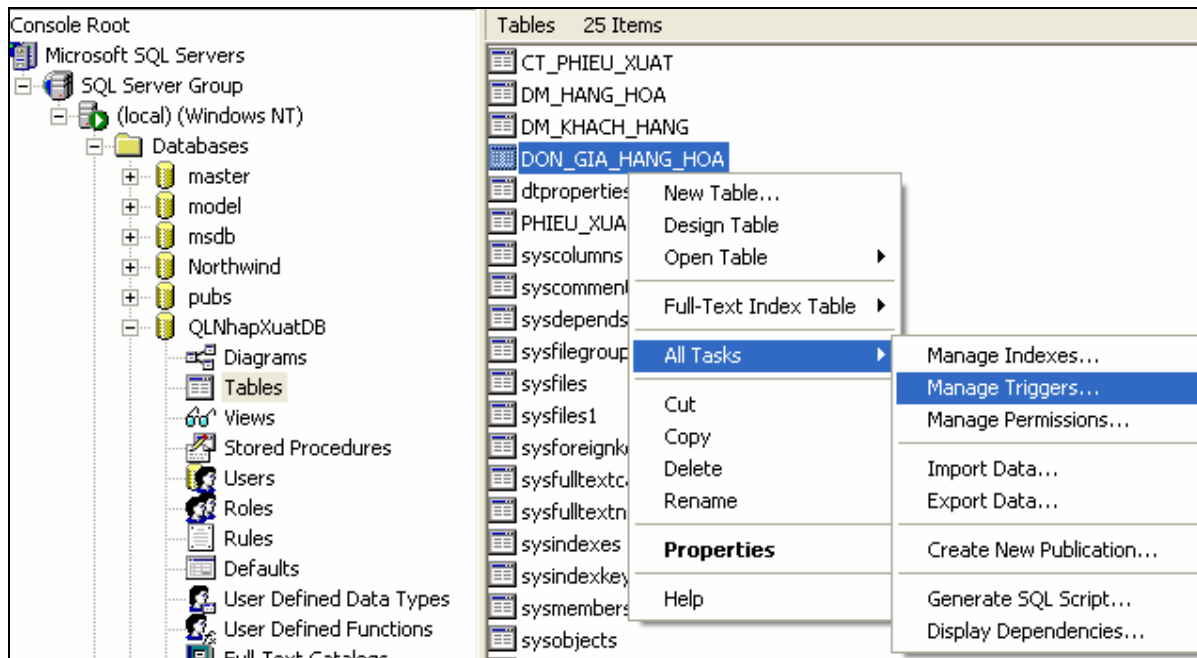
Đối với Instead of trigger thì ngược lại, dữ liệu sẽ không được thêm vào bảng. Một khi bạn đã định nghĩa một Instead of trigger cho một hành động cập nhật dữ liệu trên bảng thì chắc chắn dữ liệu sẽ không tự động cập nhật vào bảng, thay vào đó, dữ liệu chỉ được đưa vào bảng insterted hay deleted mà thôi. Lúc này, để hành động cập nhật dữ liệu thật sự có hiệu lực, bạn phải viết câu lệnh INSERT, DELETE hay UPDATE trực tiếp bên trong instead of trigger.

## II. Làm việc với trigger

### II.1. Tạo mới trigger

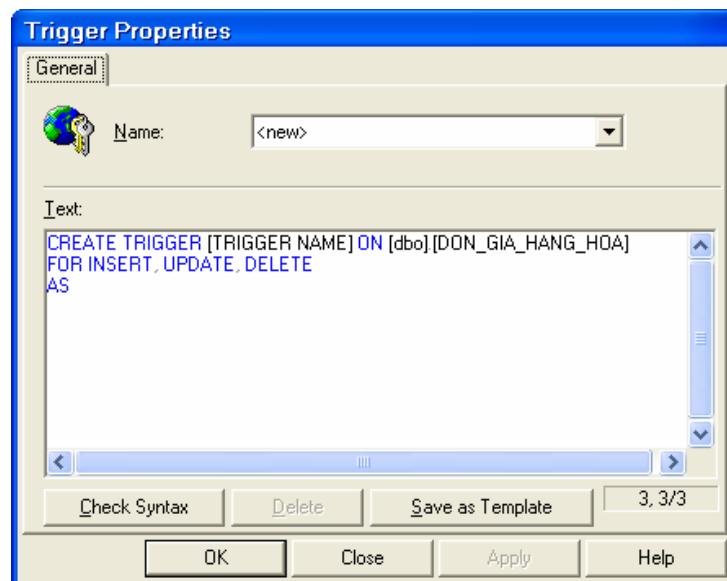
Giống như các đối tượng khác trong Microsoft SQL Server mà chúng tôi đã trình bày trước đây ở các chương trước, các bạn có hai cách để có thể tạo mới một trigger. Các bước bên dưới sẽ hướng dẫn các bạn cách thức **tạo mới** một trigger bằng tiện ích Enterprise Manager.

- ✓ Bước 1: Khởi động tiện ích Enterprise Manager. Chọn tên một bảng muốn tạo mới trigger có liên quan, kích hoạt thực đơn tắt và chọn tiếp chức năng **All Task** → **Manage Triggers**.



**Hình 7-1.** Chọn Manage Trigger để tạo trigger mới.

- ✓ Bước 2: Trong màn hình định nghĩa trigger mới lần lượt chỉ định **tên của trigger** và **các câu lệnh cần thiết** bên trong trigger để thực hiện kiểm tra các ràng buộc toàn vẹn dữ liệu. Nhấn vào nút **Check Syntax** để hệ thống kiểm tra cú pháp các lệnh bên trong trigger có hợp lệ hay không. Sau cùng nhấn nút **OK** để lưu lại nội dung của trigger mà chúng ta đã tạo ra.



**Hình 7-2.** Màn hình chỉ định các thuộc tính liên quan đến trigger.

Ngoài ra chúng ta có cũng có thể tạo mới trigger bằng lệnh **CREATE TRIGGER** trong tiện ích Query Analyzer. Cú pháp được mô tả như bên dưới.

Cú pháp:

```
CREATE TRIGGER Tên_Trigger ON Tên_bảng
```



```
{ [ INSTEAD OF ] | [ FOR | AFTER ] } { [ INSERT [, UPDATE [,DELETE ] ] ] }
AS
[DECLARE Biến_cục_bộ]
Các_lệnh
```

Trong đó:

- ✓ **Tên trigger:** tên trigger được tạo mới, tên trigger này phải là **duy nhất** trong một cơ sở dữ liệu.
- ✓ **Tên bảng:** tên bảng có trong cơ sở dữ liệu mà trigger tạo mới có liên quan đến.
- ✓ **INSTEAD OF:** chỉ định đây là trigger loại instead of trigger. Chú ý rằng với mỗi bảng, bạn chỉ có quyền tạo một instead of trigger cho một hành động cập nhật dữ liệu. Nói cách khác, nếu mỗi hành động cập nhật dữ liệu (thêm, xóa và sửa) trên bảng bạn đều viết instead of trigger thì bạn chỉ có tối đa 3 instead of trigger trên bảng.
- ✓ **FOR hoặc AFTER:** Nếu tạo trigger thông thường hay after trigger, bạn dùng từ khoá For hoặc After. Hai từ khoá này đều có ý nghĩa xác định trigger được tạo là loại after trigger, tuy nhiên, một số chức năng mở rộng của câu lệnh CREATE TRIGGER chỉ có thể viết với từ khoá For hoặc từ khoá After mà thôi.
- ✓ **INSERT, UPDATE, DELETE:** các hành động cập nhật dữ liệu có liên quan tác động vào bảng để kích hoạt trigger.
- ✓ **Biến cục bộ:** là những biến cục bộ được sử dụng trong trigger, những biến này chỉ có phạm vi cục bộ bên trong một trigger.
- ✓ **Các lệnh:** các lệnh bên trong trigger dùng để kiểm tra các ràng buộc toàn vẹn dữ liệu.



#### Ví dụ:

Tạo trigger cho việc thêm mới một đơn đặt hàng cần phải kiểm tra các ràng buộc: mã nhà cung cấp phải tồn tại trong bảng NHACC, ngày nhập hàng dự kiến phải sau ngày đặt hàng. Chúng ta thực hiện lệnh CREATE TRIGGER như sau:

```
CREATE TRIGGER tg_DONDH_Insert ON DONDH
FOR INSERT
AS
--Khai báo biến cục bộ
DECLARE @nMancc_Hople INT, @nNgaydknh_Hople INT
DECLARE @dNgaydh DATETIME, @dNgaydknh DATETIME
DECLARE @sChuoiloi CHAR(100)

--Giả sử các ràng buộc đều hợp lệ
SET @nMancc_Hople = 0
SET @nNgaydknh_Hople = 0

-- Nếu Manhacc không có trong bảng NHACC
IF EXISTS (SELECT MANHACC FROM INSERTED
           WHERE MANHACC NOT IN
           (SELECT MANHACC FROM NHACC))
```





```

SET @nMancc_Hople = 1

SELECT @dNgaydh = NGAYDH
FROM INSERTED
SELECT @dNgaydknh = NGAYDKNH
FROM INSERTED
-- Nếu Ngaydknh trước Ngaydh
IF @dNgaydknh < @dNgaydh
SET @nNgaydknh_Hople = 1

--Kiểm tra tính hợp lệ các ràng buộc
SET @sChuoiloi = CASE
    WHEN @nMancc_Hople = 0 AND @nNgaydknh_Hople = 0
        THEN "Dữ liệu hợp lệ"
    WHEN @nMancc_Hople = 1 AND @nNgaydknh_Hople = 0
        THEN "Mã nhà cung cấp không có, xem lại"
    WHEN @nMancc_Hople = 0 AND @nNgaydknh_Hople = 1
        THEN "Ngày dự kiến nhập hàng phải sau ngày đặt hàng"
    WHEN @nMancc_Hople = 1 AND @nNgaydknh_Hople = 1
        THEN "Mã nhà cung cấp và ngày dự kiến nhập hàng sai"
    END
IF @sChuoiloi <> "Dữ liệu hợp lệ"
BEGIN
    PRINT @sChuoiloi
    ROLLBACK TRAN
END
GO

```

Trước khi kiểm tra các hoạt động của trigger trong bảng DONDH mà chúng ta đã xây dựng ở thí dụ trên có đúng không, các bạn phải **tạm thời ngưng lại** các kiểm tra ràng buộc toàn vẹn dữ liệu bằng đối tượng constraint đã định nghĩa trước đó bằng lệnh:

```
ALTER TABLE DONDH NOCHECK CONSTRAINT ALL
```

Sau đó sử dụng lệnh INSERT INTO để thêm dữ liệu vào bảng DONDH nhằm kiểm tra các hoạt động của trigger có đúng theo mong muốn không. Thực hiện lệnh thêm mới một đơn đặt hàng với mã nhà cung cấp **không có** trong bảng NHACC:

```

INSERT INTO DONDH VALUES
("D009", "2002-04-01", "2002-04-15", "C99")

```

Kết quả lỗi trả về

```
Mã nhà cung cấp không có, xem lại
```



Hoặc khi thêm mới một đơn đặt hàng với ngày nhập hàng dự kiến **trước** ngày đặt hàng:

```
INSERT INTO DONDH VALUES
("D009", "2002-04-01", "2002-03-15", "C01")
```

Kết quả lỗi trả về

Ngày dự kiến nhập hàng phải sau ngày đặt hàng

## II.2. Xóa trigger

Khi một trigger **không còn cần** sử dụng nữa thì chúng ta có thể xóa bỏ nó ra khỏi cơ sở dữ liệu. Tuy nhiên thông thường khi đã xóa bỏ các trigger thì chúng ta **cần phải bật lại** các kiểm tra ràng buộc toàn vẹn dữ liệu trong đối tượng constraint mà chúng ta đã tạm thời tắt đi trước đây. Cú pháp lệnh **DROP TRIGGER** bên dưới cho phép chúng ta có thể xóa bỏ một trigger.

Cú pháp:

```
DROP TRIGGER Tên_trigger
```

Trong đó:

- ✓ **Tên trigger:** tên trigger đã được tạo trước đó mà chúng ta muốn xóa bỏ khi không còn sử dụng nữa.

Ngoài ra chúng ta có thể xóa bỏ trigger khi có nhu cầu cần tạo lại nội dung mới, có nghĩa là khi đó bên trong nội dung của trigger sẽ cần bổ sung thêm các lệnh, xử lý nào đó.



### Ví dụ:

Để xóa bỏ trigger tg\_DONDH\_Insert đã được tạo ra trong thí dụ trước đó. Chúng ta thực hiện lệnh **DROP TRIGGER** như sau:

```
DROP TRIGGER tg_DONDH_Insert
```

## II.3. Sửa nội dung trigger

Đôi khi nội dung của các trigger **cần phải thay đổi** lại để cho các kiểm tra ràng buộc toàn vẹn dữ liệu bên trong đó được thực hiện đúng đắn. Chúng ta có hai cách để thay đổi nội dung của trigger: hoặc là dùng lệnh xóa bỏ trigger và tạo lại trigger với nội dung mới (như phần trên chúng tôi đã gợi ý) hoặc là dùng lệnh **ALTER TRIGGER**. Cú pháp của lệnh **ALTER TRIGGER** bên dưới hoàn toàn giống cú pháp của lệnh **CREATE TRIGGER** mà chúng tôi đã trình bày trước đây.

Cú pháp:

```
ALTER TRIGGER Tên_Trigger ON Tên_bảng
FOR INSERT [, UPDATE [,DELETE ]]
AS
[DECLARE Biến_cục_bộ]
Các_lệnh
```

## II.4. Trigger lồng nhau

Khái niệm **trigger lồng nhau** đối với after trigger hoàn toàn giống khái niệm thủ tục lồng nhau trước đây mà chúng tôi đã trình bày trong chương 4. Bản thân bên trong trigger chúng ta được phép gọi thực hiện các lệnh INSERT, UPDATE, DELETE để cập nhật dữ liệu của các bảng khác, chính các lệnh này sẽ làm **kích hoạt** các trigger liên quan khác (nếu có) và cứ thế các trigger có thể gọi thực hiện lồng nhau!

Với loại instead of trigger mà Microsoft mới thêm vào SQL Server 2000, việc gọi các trigger lồng nhau không hoàn toàn giống như after trigger. Do trong bản thân instead of trigger, bạn cũng phải gọi lệnh cập nhật dữ liệu insert, update, delete nên nếu hành động này tác động vào chính bảng liên kết với trigger thì instead of trigger sẽ không thể xảy ra một lần nữa, nếu không sẽ trở thành một vòng lặp vô hạn. Một khi hành động insert, update, delete được gọi thực hiện trên một bảng khác thì trigger liên kết với những bảng đó sẽ được thi hành. Như vậy, có thể nói rằng instead of trigger trên một bảng chỉ xảy ra một lần mà thôi.

Cấp độ lồng tối đa của các trigger **không vượt quá 32 cấp**. Chúng ta cũng có thể sử dụng biến hệ thống @@NESTLEVEL để biết được cấp độ lồng hiện hành của trigger. Mặc định các trigger được phép lồng nhau, tuy nhiên chúng ta cũng có thể tạm thời tắt chế độ lồng của trigger bằng lệnh.

```
EXEC sp_configure 'nested triggers', 0
```

Hoặc có thể bật trở lại chế độ lồng nhau của trigger bằng lệnh.

```
EXEC sp_configure 'nested triggers', 1
```



### Ví dụ:

Thông thường khi dữ liệu cập nhật trong các bảng CTPXUAT hoặc CTPNHAP thì chúng ta phải **cập nhật tự động** các cột: tổng số lượng nhập, tổng số lượng xuất và số lượng cuối kỳ của các vật tư tương ứng trong bảng TONKHO. Các xử lý này có thể **chia ra làm hai nơi**:

Các xử lý cập nhật cột tổng số lượng nhập hoặc tổng số lượng xuất được viết trong trigger của các bảng CTPNHAP hoặc CTPXUAT.

Xử lý cập nhật lại giá trị của cột số lượng cuối kỳ được viết trong trigger của bảng TONKHO.

Khi đó bản thân các trigger của các bảng CTPNHAP và CTPXUAT sẽ thực hiện các lệnh cập nhật dữ liệu trên bảng TONKHO, lúc này trigger của bảng TONKHO sẽ được kích hoạt thực hiện chỉ để tính lại giá trị cột số lượng cuối kỳ khi giá trị các cột số lượng nhập hoặc số lượng xuất bị thay đổi. Chúng ta thực hiện lệnh CREATE TRIGGER như sau:

```
CREATE TRIGGER tg_TONKHO_InsertUpdate ON TONKHO
FOR INSERT, UPDATE
AS
UPDATE TONKHO
SET SLCK = SLDK + TSLNHAP - TSLXUAT WHERE NAMTHANG+MAVTU IN
(SELECT NAMTHANG+MAVTU FROM INSERTED)
GO
```

### III. Trigger kiểm tra ràng buộc dữ liệu

Trong các phần còn lại của bài này, chúng tôi sẽ giới thiệu cho các bạn **chi tiết các xử lý** bên trong trigger để có thể thực hiện việc kiểm tra các **ràng buộc toàn vẹn dữ liệu phức tạp**. Để dễ hiểu chúng tôi chia các hành động cập nhật dữ liệu ra làm ba sự kiện rời rạc: **thêm, sửa và xóa** nhằm giúp các bạn phân biệt rõ ràng các xử lý thường dùng bên trong các sự kiện đó.

Như đã trình bày, trigger được tự động thực hiện khi hành động cập nhật dữ liệu tương ứng xảy ra. **Các câu lệnh bên trong trigger và câu lệnh cập nhật dữ liệu hợp lại thành một transaction**. Điều này nghĩa là nếu trong trigger, bạn gọi lệnh **rollback tran** thì toàn bộ transaction sẽ bị huỷ bỏ và do đó, hành động cập nhật dữ liệu cũng sẽ bị huỷ bỏ theo. Đây là điểm quan trọng bạn cần chú ý khi sử dụng trigger để thực hiện việc kiểm tra các ràng buộc toàn vẹn.

#### III.1. Khi thêm mới mẫu tin

Trigger của sự kiện này sẽ **tự động kích hoạt** khi dữ liệu trong bảng được **thêm mới** vào bảng dữ liệu. Thông thường bên trong trigger sẽ có một số các kiểm tra ràng buộc toàn vẹn dữ liệu như là:

- ✓ Khóa ngoại.
- ✓ Miền giá trị.
- ✓ Liên thuộc tính trong cùng một bảng.
- ✓ Liên thuộc tính của nhiều bảng khác nhau.

Khi các giá trị dữ liệu thêm mới **vi phạm** các ràng buộc toàn vẹn dữ liệu thì trigger sẽ **thông báo** cho người dùng biết và **không lưu lại** các thông tin của dòng dữ liệu vừa được thêm mới vào bên trong bảng. Trong các trigger thêm mới dữ liệu thì các dữ liệu vừa mới thêm vào sẽ được lưu trữ tạm thời trong bảng **Inserted**, do đó chúng ta sẽ tham chiếu đến bảng Inserted để lấy ra các giá trị dữ liệu vừa mới thêm dùng trong những kiểm tra ràng buộc toàn vẹn dữ liệu.



##### Ví dụ:

*Xây dựng trigger trong bảng PNHAP để kiểm tra các ràng buộc toàn vẹn dữ liệu khi người dùng **thêm mới** thông tin của **một phiếu nhập hàng** cho một đơn đặt hàng trước đó. Các ràng buộc toàn vẹn dữ liệu bao gồm:*

*Khóa ngoại: cần kiểm tra số đặt hàng phải tồn tại trong bảng đơn đặt hàng.*

*Miền giá trị: cần kiểm tra ngày giao hàng phải sau ngày đặt hàng.*

*Chúng ta thực hiện lệnh **CREATE TRIGGER** như sau:*

```
CREATE TRIGGER tg_PNHAP_Insert ON PNHAP
FOR INSERT
AS
    DECLARE @Ngaydh DATETIME, @ErrMsg CHAR(200)

    -- Kiểm xem sodh đã có trong bảng DONDH không?
```



```

IF NOT EXISTS (SELECT *
                FROM INSERTED I, DONDH D
                WHERE I.SODH=D.SODH)
BEGIN
    ROLLBACK TRAN
    RAISERROR ("Số đơn đặt hàng không tồn tại", 16, 1)
    RETURN
END

-- Tính ra ngày đặt hàng
SELECT @Ngaydh=NGAYDH
FROM DONDH D, INSERTED I
WHERE D.SODH=I.SODH

-- Kiểm xem ngày giao hàng phải sau ngày đặt hàng
IF @Ngaydh > (SELECT NGAYNHAP
              FROM INSERTED)
BEGIN
    SET @ErrMsg = "Ngày giao hàng phải sau ngày : "
    + CONVERT(CHAR(10), @Ngaydh, 103)
    RAISERROR(@ErrMsg, 16, 1)
    ROLLBACK TRAN
END
GO

```

Để kiểm tra các hoạt động bên trong trigger có đúng hay không? Chúng ta cần phải **lần lượt** thực hiện các lệnh **INSERT INTO** để thêm dữ liệu vào bảng PNHAP cho các trường hợp mà chúng ta đã biện luận trong trigger.

**Trường hợp 1: Vi phạm** ràng buộc toàn vẹn số đơn đặt hàng **không tồn tại** trong bảng DONDH.

```

ALTER TABLE PNHAP
    NOCHECK CONSTRAINT ALL
INSERT INTO PNHAP (SOPN, SODH, NGAYNHAP)
VALUES ("N004", "D999", "2002-04-15")

```

Kết quả lỗi trả về

```

Server: Msg 50000, Level 16, State 1, Line 0
Số đơn đặt hàng không tồn tại

```

**Trường hợp 2: Vi phạm** ràng buộc toàn vẹn ngày giao hàng **trước** ngày đặt hàng.

```

INSERT INTO PNHAP (SOPN, SODH, NGAYNHAP)
VALUES ("N004", "D003", "2001-01-15")

```

Kết quả lỗi trả về

```

Server: Msg 50000, Level 16, State 1, Line 0
Ngày giao hàng phải sau ngày : 10/02/2002

```

Cuối cùng khi nhập thông tin của một phiếu nhập hàng **không vi phạm** các ràng buộc toàn vẹn

dữ liệu.

```
INSERT INTO PNHAP (SOPN, SODH, NGAYNHAP)
VALUES ("N004", "D003", "2002-02-15")
```

Kết quả trả về hành động thêm dữ liệu mới thành công!

```
(1 row(s) affected)
```

### III.2. Khi hủy bỏ mẫu tin

Trigger của sự kiện này sẽ **tự động** kích hoạt khi dữ liệu bên trong bảng bị **hủy bỏ**. Thông thường bên trong trigger sẽ có một số các kiểm tra ràng buộc toàn vẹn dữ liệu như là: kiểm tra ràng buộc toàn vẹn dữ liệu **khóa ngoại** dùng để **xóa tự động dữ liệu** bên bảng con có liên quan hoặc **thông báo lỗi** đã vi phạm ràng buộc toàn vẹn khi xóa dữ liệu bên bảng cha.

Khi giá trị dữ liệu trong bảng bị hủy nếu có vi phạm ràng buộc toàn vẹn dữ liệu khóa ngoại thì trigger sẽ thông báo cho người dùng biết và không hủy thông tin của các dòng dữ liệu. Trong các trigger hủy bỏ dữ liệu thì dữ liệu vừa bị hủy bỏ sẽ được lưu trữ tạm thời trong bảng Deleted, chúng ta sẽ tham chiếu đến bảng Deleted để lấy ra giá trị của các dữ liệu vừa bị hủy bỏ dùng cho các kiểm tra ràng buộc toàn vẹn dữ liệu khóa ngoại.



#### Ví dụ:

Khi xóa một số đặt hàng bên trong bảng DONDH cần phải kiểm tra các ràng buộc toàn vẹn dữ liệu sau:

Kiểm tra xem đơn đặt hàng bị xóa **đã được nhập hàng chưa**? Nếu đã được nhập hàng rồi thì thông báo **không thể xóa** đơn đặt hàng được.

Ngược lại thì **xóa tự động** dữ liệu liên quan bên bảng chi tiết đơn đặt hàng (CTDONDH).

Chúng ta thực hiện lệnh **CREATE TRIGGER** như sau:

```
CREATE TRIGGER tg_DONDH_Delete ON DONDH
FOR DELETE
AS
    DECLARE @Sopn CHAR(4), @ErrMsg CHAR(200),
            @Delete_Err INT

    -- Kiểm tra xem đơn đặt hàng đã được nhập chưa ?
    IF EXISTS (SELECT SOPN FROM PNHAP
                WHERE SODH IN (SELECT SODH
                                FROM DELETED))
    BEGIN
        SELECT @Sopn=MIN(SOPN)
        FROM PNHAP
        WHERE SODH IN (SELECT SODH
                        FROM DELETED)

        SET @ErrMsg = "Đơn đặt hàng đã được nhập theo " +
            " số nhập hàng " + @Sopn + CHAR(13) +
```



```

". Không thể hủy được"
RAISERROR(@ErrMsg, 16,1)
ROLLBACK TRAN
END
ELSE
BEGIN
-- Xóa tự động chi tiết các đơn đặt hàng liên quan
DELETE CTDONDH
WHERE SODH IN (SELECT SODH
                FROM DELETED)
SET @Delete_Err = @@ERROR
IF @Delete_Err <> 0
BEGIN
SET @ErrMsg = "Lỗi vi phạm xóa bên bảng " +
              "chi tiết đặt hàng"
RAISERROR(@ErrMsg, 16, 1)
ROLLBACK TRAN
END
END
GO

```

Cũng hoàn toàn giống như thí dụ trên, chúng ta cần phải lần lượt thực hiện các lệnh **DELETE** để hủy bỏ các dòng dữ liệu trong bảng **DONDH** nhằm kiểm tra tính đúng đắn của các hành động bên trong trigger đã chúng ta đã viết.

**Trường hợp 1: Vi phạm** ràng buộc toàn vẹn khóa ngoại vì số đơn đặt hàng **D002** đã được nhập hàng rồi.

```

ALTER TABLE CTDONDH
NOCHECK CONSTRAINT ALL

DELETE DONDH
WHERE SODH="D002"

```

Kết quả lỗi trả về

```

Server: Msg 50000, Level 16, State 1, Line 0
Đơn đặt hàng đã được nhập theo số nhập hàng N003
Không thể hủy được

```

**Trường hợp 2: Xóa thành công** toàn bộ thông tin đơn đặt hàng **D006** trong cả hai bảng **DONDH** và **CTDONDH** (vì đơn đặt hàng này chưa được nhập hàng về).

```

DELETE DONDH
WHERE SODH="D006"

```

Kết quả trả về 3 dòng bị xóa trong bảng **CTDONDH** và 1 dòng trong **DONDH**

```

(3 row(s) affected)

```

```

(1 row(s) affected)

```

### III.3. Khi sửa đổi mẫu tin

Trigger của sự kiện này sẽ **tự động** kích hoạt khi dữ liệu trong bảng bị **sửa đổi**. Thông thường bên trong trigger sẽ có một số các kiểm tra ràng buộc toàn vẹn dữ liệu như là: kiểm tra ràng buộc toàn vẹn dữ liệu **khóa ngoại**, **miền giá trị**, **liên thuộc tính** trong cùng một bảng dữ liệu, liên thuộc tính của nhiều bảng dữ liệu khác nhau. Tuy nhiên thông thường đối với việc thay đổi dữ liệu trên bảng chúng ta sẽ **hạn chế** việc sửa đổi dữ liệu, chỉ cho phép người sử dụng sửa đổi giá trị dữ liệu trên **một số cột nhất định** nào đó bên trong bảng.

Để kiểm tra giá trị dữ liệu của một cột bên trong bảng có bị thay đổi trong các trigger sửa đổi dữ liệu chúng ta sẽ sử dụng hàm **UPDATE**. Cú pháp đầy đủ của hàm UPDATE sẽ được mô tả như sau:

Cú pháp:

```
UPDATE (Tên_cột) → Biểu_thức_luận_lý
```

Trong đó:

- ✓ **Tên cột:** tên cột mà chúng ta muốn kiểm tra xem dữ liệu tại đó có bị sửa đổi trong trigger không.
- ✓ **Biểu thức luận lý:** trả về **True** khi giá trị dữ liệu của cột đã bị sửa đổi, ngược lại trả về **False** khi giá trị dữ liệu của cột không bị sửa đổi.

Hành động sửa đổi dữ liệu bên dưới của Microsoft SQL Server thật chất là sự kết hợp của **hai hành động** đi kèm là: **xóa** dữ liệu cũ hiện có và **thêm** lại dữ liệu mới đã được sửa đổi. Do đó bên trong trigger sửa đổi dữ liệu khi đó bảng **Inserted** sẽ chứa đựng dữ liệu mới sau khi sửa đổi và bảng **Deleted** sẽ chứa đựng dữ liệu cũ trước khi sửa đổi. Thông thường trong trigger sửa đổi chúng ta có thể tham chiếu đến **cùng lúc hai bảng** luận lý **Inserted và Deleted**.

Bản thân trigger sửa đổi dữ liệu sẽ **tự động** gọi đến trigger thêm mới dữ liệu của cùng một bảng. Do đó nếu trong trigger thêm mới nếu đã có các kiểm tra ràng buộc toàn vẹn dữ liệu **miền giá trị**, **khóa ngoại**, **liên thuộc tính** trong cùng một bảng dữ liệu thì chúng ta **không cần thực hiện** lại các kiểm tra ràng buộc toàn vẹn dữ liệu đó bên trong trigger sửa đổi dữ liệu.



#### Ví dụ:

Khi sửa đổi thông tin của một số đặt hàng bên trong bảng *DONDH* cần phải kiểm tra các ràng buộc toàn vẹn dữ liệu sau:

**Không cho phép** sửa đổi dữ liệu tại các cột **số đặt hàng** hoặc **mã nhà cung cấp** vì khi đó dữ liệu sẽ **bị ảnh hưởng** đến **nhiều** bảng liên quan khác.

Khi sửa đổi giá trị cột ngày đặt hàng thì phải đảm bảo luôn luôn trước ngày nhập hàng đầu tiên của số đặt hàng đó (nếu đơn đặt hàng đã có nhập hàng).

Chúng ta thực hiện lệnh **CREATE TRIGGER** như sau:

```
CREATE TRIGGER tg_DONDH_Update ON DONDH
FOR UPDATE
AS
DECLARE @MinNgaynh DATETIME, @ErrMsg CHAR(200)
```





```
--Khi sửa đổi các cột SODH hoặc MANHACC
IF UPDATE(SODH) OR UPDATE(MANHACC)
BEGIN
    ROLLBACK TRAN
    SET @ErrMsg = "Không thể thay đổi số đặt hàng hoặc mã nhà cung cấp"
    RAISERROR (@ErrMsg, 16, 1)
    RETURN
END
-- Khi sửa đổi cột NGAYDH
IF UPDATE(NGAYDH)
BEGIN
    -- Kiểm tra đơn đặt hàng đã được về nhập chưa?
    IF EXISTS (SELECT SOPN
                FROM PNHAP
                WHERE SODH IN (SELECT SODH
                              FROM DELETED))
    BEGIN
        -- Tính ra ngày nhập hàng đầu tiên
        SELECT @MinNgaynh=MIN(NGAYNHAP)
        FROM PNHAP PN, DELETED D
        WHERE PN.SODH=D.SODH
        -- Kiểm tra giá trị ngày đặt hàng sau khi sửa đổi
        -- phải luôn trước ngày nhập hàng đầu tiên
        IF @MinNgaynh < (SELECT NGAYDH
                        FROM INSERTED)
        BEGIN
            ROLLBACK TRAN
            SET @ErrMsg = "Ngày đặt hàng phải trước ngày : "
                        + CONVERT(CHAR(10), @MinNgaynh, 103)
            RAISERROR(@ErrMsg, 16, 1)
        END
    END
END
GO
```

*Chúng ta cần phải lần lượt thực hiện các lệnh UPDATE để sửa đổi dữ liệu trong bảng DONDH nhằm kiểm tra tính đúng đắn của trigger đã viết.*

**Trường hợp 1:** Sửa đổi dữ liệu cột số đặt hàng hoặc cột mã nhà cung cấp.

```
ALTER TABLE DONDH
    NOCHECK CONSTRAINT ALL
UPDATE DONDH
    SET SODH="D055"
    WHERE SODH="D005"
```



Hoặc

```
UPDATE DONDH
SET MANHACC="C01"
WHERE SODH="D002"
```

Kết quả lỗi trả về

Server: Msg 50000, Level 16, State 1, Line 0

Không thể thay đổi số đặt hàng hoặc mã nhà cung cấp

**Trường hợp 2:** Đơn đặt hàng D002 có ngày đặt hàng là 30/01/2002, ngày nhập hàng dự kiến là 02/02/2002 và **đã được nhập hàng** lần đầu vào ngày 31/01/2002. Giả sử chúng ta ra lệnh để sửa đổi giá trị ngày đặt hàng lại thành ngày 01/02/2002.

```
UPDATE DONDH
SET NGAYDH="2002-02-01"
WHERE SODH="D002"
```

Kết quả lỗi trả về

Server: Msg 50000, Level 16, State 1, Line 0

Ngày đặt hàng phải trước ngày : 31/01/2002

**Trường hợp 3:** Đơn đặt hàng D005 có ngày đặt hàng là 01/03/2002, ngày nhập hàng dự kiến là 05/03/2002 và chưa được nhập hàng về. Giả sử vô tình chúng ta ra lệnh để sửa đổi giá trị ngày đặt hàng lại thành ngày 07/03/2002.

```
UPDATE DONDH
SET NGAYDH="2002-03-07"
WHERE SODH="D005"
```

Hoặc

```
UPDATE DONDH
SET NGAYDKNH="2002-02-01"
WHERE SODH="D005"
```

Kết quả lỗi trả về

Server: Msg 50000, Level 16, State 1, Line 0

Ngày dự kiến nhập hàng phải sau ngày đặt hàng

Nhận xét thấy rằng chuỗi lỗi này **hoàn toàn không có** trong phần biện luận của trigger sửa đổi dữ liệu ở thí dụ trên, thật ra ràng buộc toàn vẹn dữ liệu kiểm tra **ngày đặt hàng phải trước ngày nhập hàng dự kiến** đã được chúng tôi biện luận **trong trigger thêm mới dữ liệu** của bảng DONDH có tên là tg\_DONDH\_Insert ở thí dụ đầu tiên trong phần tạo trigger bằng lệnh **CREATE TRIGGER** trước đây.

Qua đây các bạn thấy rằng trigger sửa đổi dữ liệu sẽ **tự động** gọi thực hiện trigger thêm mới dữ liệu trong cùng một bảng là vì như trước đây chúng tôi đã nói việc sửa đổi dữ liệu thật ra bên trong Microsoft SQL Server là sự kết hợp đồng thời của cả hai hành động xóa dữ liệu và thêm mới dữ liệu.

## IV. Trigger cập nhật giá trị tự động

Thông thường dữ liệu của các bảng này được hình thành dựa vào số liệu của một hoặc nhiều bảng khác bên trong cơ sở dữ liệu. Do đó, khi dữ liệu của một bảng được cập nhật thì có thể sẽ ảnh hưởng đến dữ liệu của một hay nhiều bảng khác.



### Ví dụ:

Dữ liệu của bảng TONKHO sẽ được tính tự động từ dữ liệu của các bảng liên quan đến việc nhập hàng và việc xuất hàng, cụ thể sẽ là các bảng: PNHAP, CTPNHAP, PXUAT và CTPXUAT.

Những mối quan hệ này như bạn đã biết, tạo ra những ràng buộc dữ liệu mà bạn cần phải đảm bảo tính đúng đắn (hay tính toàn vẹn) của chúng trong cơ sở dữ liệu. Bạn vẫn có thể làm việc này bằng cách yêu cầu người dùng sau khi nhập dữ liệu trên bảng này cần phải sửa đổi dữ liệu trên các bảng liên quan cho phù hợp. Tuy vậy, cách làm này thực sự không thể áp dụng trong thực tế đối với những công việc phức tạp hay dữ liệu liên quan đến quá nhiều bảng.



### Ví dụ:

Khi lập một chỉ tiết xuất vật tư (thêm dữ liệu vào bảng CTXUAT) người dùng phải kiểm tra xem trong bảng tồn kho, số lượng tồn kho của vật tư tương ứng trong năm tháng ứng với ngày lập phiếu xuất có đủ hay không, nếu đủ mới có thể xuất. Việc này bạn có thể thực hiện bằng trigger kiểm tra dữ liệu như đã trình bày ở trên.

Tuy nhiên, sau khi thêm chỉ tiết xuất đó, số lượng tồn kho của vật tư đã bị giảm xuống nên người dùng cần phải cập nhật lại thông tin tồn kho này trong bảng TONKHO. Việc này nếu người dùng thực hiện sẽ phải đi qua các bước sau:

- ✓ Xác định năm tháng của phiếu nhập vật tư
- ✓ Xác định mã vật tư vừa xuất và số lượng vừa xuất
- ✓ Sử dụng mã vật tư, số lượng và năm tháng vừa xác định để cập nhật bảng TONKHO

Việc cập nhật từ phía người dùng như vậy có thể dẫn đến việc dữ liệu xác định bởi người dùng bị sai. Nếu người dùng chậm cập nhật dữ liệu vào bảng tồn kho, một phiếu xuất khác có thể được nhập vào bởi một người dùng khác và do thông tin tồn kho lúc này tạm thời không chính xác, việc xuất mới này có thể được ghi nhận vào cơ sở dữ liệu nhưng trên thực tế lại không thực hiện được.

Như vậy, bạn có thể thấy rằng mặc dù trên một ràng buộc dữ liệu rất đơn giản, chỉ liên quan đến hai bảng nhưng nếu không được thực hiện một cách **chính xác** và **kịp thời** thì tính đúng đắn của dữ liệu trong cơ sở dữ liệu sẽ bị vi phạm.

Với khả năng tham gia vào transaction được khởi tạo bởi câu lệnh cập nhật dữ liệu, trigger khi sử dụng để thực hiện các công việc tính toán, cập nhật giá trị tự động sẽ đảm bảo tính **kịp thời** như trong ví dụ trên yêu cầu. Nếu hành động cập nhật dữ liệu xảy ra thì chắc chắn những hành động cập nhật dữ liệu trong trigger cũng sẽ xảy ra ngay tức thì. Vấn đề còn lại là tính chính xác trong việc cập nhật dữ liệu của trigger.

Trong phần cuối của chương này chúng tôi muốn giới thiệu thêm các **xử lý cập nhật giá trị tự**

**động** bên trong các trigger, các xử lý này cũng sẽ được chia ra làm thành các sự kiện rời rạc nhằm giúp các bạn dễ hiểu. Tuy nhiên chúng ta **không nên** viết các xử lý cập nhật giá trị tự động này thành **một trigger mới hoàn toàn** mà chỉ cần **bổ sung** các xử lý này vào bên dưới trong các trigger kiểm tra ràng buộc toàn vẹn đã có. Bởi vì các **xử lý cập nhật giá trị tự động** chỉ được thực hiện khi nào các **ràng buộc toàn vẹn dữ liệu** đã được **kiểm tra hợp lệ** trước đó.

#### IV.1. Khi thêm mới mẫu tin

Khi giá trị của các dữ liệu thêm mới **đã được kiểm tra hợp lệ** so với các ràng buộc toàn vẹn dữ liệu thì các dữ liệu này sẽ được **cập nhật tăng giá trị** tại một cột nào đó của các bảng liên quan.



##### Ví dụ:

*Khi thêm mới các thông tin của chi tiết một phiếu nhập hàng vào bảng CTPNHAP, chúng ta cần kiểm tra các ràng buộc toàn vẹn dữ liệu:*

*Kiểm tra số phiếu nhập phải có trong bảng PNHAP.*

*Kiểm tra mã vật tư phải có trong danh sách chi tiết danh sách các mã vật tư có trong chi tiết đơn đặt hàng trước đó.*

*Kiểm tra tổng số lượng nhập hàng vẫn còn ít hơn số lượng đặt hàng của vật tư đó.*

*Nếu tất cả các ràng buộc toàn vẹn dữ liệu ở trên đều hợp lệ thì **tăng giá trị** của **cột tổng số lượng nhập** trong bảng **TONKHO** và **cột tổng trị giá** trong bảng **PNHAP**. Chúng ta thực hiện lệnh **CREATE TRIGGER** như sau:*

```
CREATE TRIGGER tg_CTPNHAP_Insert ON CTPNHAP
FOR INSERT
AS
    DECLARE @TongSinh INT, @Namthang CHAR(7),
            @Sodh CHAR(4), @Mavtu CHAR(4),
            @ErrMsg CHAR(200)
    -- Kiểm tra sự tồn tại của SOPN trong bảng PNHAP
    IF NOT EXISTS(SELECT I.SOPN
                  FROM INSERTED I, PNHAP PN
                  WHERE PN.SOPN=I.SOPN)
    BEGIN
        ROLLBACK TRAN
        SET @ErrMsg = "Số phiếu nhập hàng không có, xem      lại"
        RAISERROR(@ErrMsg, 16, 1)
        RETURN
    END
    -- Kiểm tra MAVTU phải có trong bảng CTDONDH
    -- đúng theo số đặt hàng tương ứng
    IF NOT EXISTS(SELECT I.MAVTU
                  FROM INSERTED I, PNHAP PN, CTDONDH CTDH
                  WHERE PN.SODH=CTDH.SODH
```



```

        AND I.MAVTU=CTDH.MAVTU)
BEGIN
    ROLLBACK TRAN
    SET @ErrMsg = "Mã vật tư không có trong danh sách          đặt hàng, xem lại"
    RAISERROR(@ErrMsg, 16, 1)
    RETURN
END
-- Tính tổng số lượng đã nhập hàng
SELECT @Mavtu=MAVTU
    FROM INSERTED
SELECT @Sodh=SODH
    FROM INSERTED I, PNHAP PN
    WHERE PN.SOPN=I.SOPN

-- Gọi thủ tục tính tổng số lượng đã nhập hàng
EXEC TinhSLDaNhap @Sodh, @Mavtu, @Tongslnh OUTPUT
-- Kiểm tra tổng số lượng đã nhập có lớn hơn số lượng đặt?
IF @Tongslnh> (SELECT SLDAT FROM
                CTDONDH CTDH, PNHAP PN, INSERTED I
                WHERE I.SOPN=PN.SOPN
                AND CTDH.SODH=PN.SODH
                AND CTDH.MAVTU=I.MAVTU)
BEGIN
    ROLLBACK TRAN
    SET @ErrMsg = "Tổng số lượng nhập hàng đã vượt hơn số lượng đặt hàng"
    RAISERROR(@ErrMsg, 16, 1)
    RETURN
END
-- Khi tất cả các ràng buộc toàn vẹn đều hợp lệ thì :
-- 1. Tăng tự động cột TONGTGNHAP trong bảng PNHAP
UPDATE PNHAP
    SET TONGTGNHAP = TONGTGNHAP
        + (SLNHAP*DGNHAP)
    FROM INSERTED I
    WHERE I.SOPN=PNHAP.SOPN

-- 2. Tăng tự động cột TSLNHAP trong bảng TONKHO
-- Tính Năm tháng của NGAYNHAP
SELECT @Namthang = CONVERT(CHAR(7), NGAYNHAP, 21)
    FROM PNHAP PN, INSERTED I WHERE PN.SOPN=I.SOPN

-- Kiểm tra khi đã có mã vật tư trong bảng TONKHO

```



```

IF EXISTS (SELECT NAMTHANG
           FROM TONKHO TK, INSERTED I
           WHERE TK.MAVTU=I.MAVTU
                AND TK.NAMTHANG=@Namthang)

UPDATE TONKHO
SET TSLNHAP = TSLNHAP + I.SLNHAP
FROM TONKHO TK INNER JOIN INSERTED I
ON I.MAVTU=TK.MAVTU
AND TK.NAMTHANG=@Namthang
ELSE
-- Khi mã vật tư chưa có trong bảng TONKHO
INSERT INTO TONKHO (NAMTHANG, MAVTU, SLDK,          TSLNHAP, TSLXUAT,
SLCK)

SELECT  @Namthang, MAVTU, 0 ,
        SLNHAP , 0, 0
        FROM INSERTED
GO

```

## IV.2. Khi hủy bỏ mẫu tin

Ngược lại với việc thêm mới, khi giá trị **các dòng dữ liệu** trong bảng **bị hủy bỏ** thì chúng ta **cập nhật giảm** giá trị tại một cột nào đó của các bảng liên quan. Ở các trigger trong các thí dụ trước chỉ áp dụng đúng khi việc cập nhật dữ liệu trên **một và chỉ một dòng dữ liệu** mà thôi, nếu việc cập nhật tác động cùng lúc trên nhiều dòng dữ liệu thì các trigger trên sẽ hoạt động **không còn đúng nữa**.

Do đó trong thí dụ bên dưới chúng tôi muốn minh họa cho các bạn thấy được cách viết trigger áp dụng trong các trường hợp người dùng **cập nhật dữ liệu** cùng lúc **một hoặc nhiều dòng dữ liệu**. Về nguyên tắc chung thường phải sử dụng hàm **SUM** để tính ra số tổng của cùng một nhóm dữ liệu và cập nhật giá trị đó vào trong các bảng khác.



### Ví dụ:

Khi hủy bỏ thông tin chi tiết các phiếu nhập hàng trong bảng CTPNHAP, chúng ta sẽ giảm giá trị cột tổng số lượng nhập trong bảng TONKHO và cột tổng trị giá nhập trong bảng PNHAP. Chúng ta thực hiện lệnh CREATE TRIGGER như sau:

```

CREATE TRIGGER tg_CTPNHAP_Delete ON CTPNHAP
FOR DELETE
AS

-- Khi dữ liệu bị xóa chỉ có một dòng
IF @@ROWCOUNT=1
BEGIN

```



```
-- Giảm giá trị cột TONGTGNHAP trong bảng PNHAP
UPDATE PNHAP
SET TONGTGNHAP = TONGTGNHAP - (SLNHAP*DGNHAP)
FROM DELETED D
WHERE D.SOPN=PNHAP.SOPN

-- Giảm giá trị cột TSLNHAP trong bảng TONKHO
UPDATE TONKHO
SET TSLNHAP = TSLNHAP - SLNHAP
FROM DELETED
WHERE NAMTHANG+TONKHO.MAVTU IN
(SELECT CONVERT(CHAR(7), NGAYNHAP, 21)+MAVTU
FROM PNHAP PN INNER JOIN DELETED D
ON PN.SOPN=D.SOPN)

END
ELSE
-- Khi dữ liệu bị xóa nhiều dòng
BEGIN
-- Giảm giá trị cột TONGTGNHAP trong bảng PNHAP
UPDATE PNHAP
SET TONGTGNHAP = TONGTGNHAP -
(SELECT SUM(SLNHAP*DGNHAP)
FROM DELETED D
WHERE D.SOPN=PNHAP.SOPN)
WHERE SOPN IN (SELECT SOPN
FROM DELETED)

-- Giảm giá trị cột TSLNHAP trong bảng TONKHO
UPDATE TONKHO
SET TSLNHAP = TSLNHAP -
(SELECT SUM(SLNHAP)
FROM DELETED D INNER JOIN PNHAP PN
ON D.SOPN=PN.SOPN
WHERE TONKHO.MAVTU=D.MAVTU AND
TONKHO.NAMTHANG=CONVERT(CHAR(7),          NGAYNHAP,
21))
WHERE NAMTHANG+TONKHO.MAVTU IN
(SELECT CONVERT(CHAR(7), NGAYNHAP, 21)+MAVTU
FROM PNHAP PN INNER JOIN DELETED D
ON PN.SOPN=D.SOPN)

END
GO
```

### IV.3. Khi sửa đổi mẫu tin

Giống như các hành động thêm và hủy bỏ trước đây, sau khi mẫu tin bị sửa đổi chúng ta **phải tính toán để cập nhật lại** giá trị các cột của các bảng có liên quan khớp với giá trị mới vừa được sửa đổi. Việc cập nhật này có thể là **tăng hoặc giảm** so với **giá trị cũ trước đó** hoàn toàn tùy thuộc vào **giá trị chênh lệch** giữa giá trị mới được sửa đổi và giá trị cũ trước đó.)

Do thế nguyên tắc chung trong các trigger sửa đổi dữ liệu là **cập nhật lại giá trị chênh lệch** sau khi và trước khi sửa đổi dữ liệu, tùy thuộc vào số chênh lệch này là âm hoặc dương mà giá trị được cập nhật sẽ giảm hoặc tăng so với giá trị hiện hành.



#### Ví dụ:

Khi sửa đổi thông tin của chi tiết một phiếu nhập hàng vào bảng CTPNHAP, chúng ta cần kiểm tra các ràng buộc toàn vẹn dữ liệu:

**Không cho phép** sửa đổi các cột: số phiếu nhập hoặc mã vật tư.

Số lượng nhập hàng sau khi sửa đổi phải đảm bảo: tổng số lượng nhập không vượt quá số lượng đặt hàng ban đầu.

Nếu tất cả các ràng buộc toàn vẹn dữ liệu ở trên đều hợp lệ thì cập nhật lại giá trị của **cột tổng số lượng nhập** trong bảng TONKHO và **cột tổng trị giá** trong bảng PNHAP. Chúng ta thực hiện lệnh CREATE TRIGGER như sau:

```
CREATE TRIGGER tg_CTPNHAP_Update ON CTPNHAP
FOR UPDATE
AS
    DECLARE @Tongslnh INT,          @Namthang CHAR(7),
            @Sodh          CHAR(4), @Mavtu CHAR(4),
            @Tongtg        MONEY,   @Soluongcl INT,
            @Trigiacl MONEY,   @ErrMsg CHAR(100)

    -- Kiểm tra nếu sửa đổi cột SOPN hoặc MAVTU
    IF UPDATE(SOPN) OR UPDATE(MAVTU)
    BEGIN
        SET @ErrMsg = "Không được sửa dữ liệu các cột số      phiếu nhập hoặc mã vật tư"
        RAISERROR(@ErrMsg, 16, 1)
        ROLLBACK TRAN
        RETURN
    END

    -- Tính tổng số lượng đã nhập hàng
    SELECT @Mavtu=MAVTU
    FROM INSERTED
    SELECT @Sodh=SODH
    FROM INSERTED I, PNHAP PN WHERE PN.SOPN=I.SOPN

    -- Gọi thủ tục tính tổng số lượng đã nhập hàng
    EXEC TinhSLDaNhap @Sodh, @Mavtu, @Tongslnh OUTPUT
```





```
-- Kiểm tra số lượng sau khi sửa có vi phạm
-- tổng số lượng nhập > số lượng đặt
IF @Tongslnh> (SELECT SLDAT
                FROM CTDONDH CTDH, PNHAP PN, INSERTED I
                WHERE I.SOPN=PN.SOPN AND
                     CTDH.SODH=PN.SODH AND
                     CTDH.MAVTU=I.MAVTU)

BEGIN
    SET @ErrMsg = "Số lượng nhập hàng sửa lại vượt quá số lượng đặt"
    RAISERROR(@ErrMsg, 16, 1)
    ROLLBACK TRAN
    RETURN
END
-- Tính số lượng chênh lệch sau và trước khi sửa đổi
SELECT @Soluongcl = (I.SLNHAP - D.SLNHAP)
FROM INSERTED I, DELETED D
WHERE I.SOPN=D.SOPN
-- Tính trị giá chênh lệch sau và trước khi sửa đổi
SELECT @Trigiactl = (I.SLNHAP*I.DGNHAP) -
                   (D.SLNHAP*D.DGNHAP)
FROM INSERTED I, DELETED D
WHERE I.SOPN=D.SOPN
-- Tính Năm tháng của NGAYNHAP
SELECT @Namthang = CONVERT(CHAR(7), NGAYNHAP, 21)
FROM PNHAP PN, INSERTED I WHERE PN.SOPN=I.SOPN
-- Tất cả dữ liệu đều hợp lệ thì
-- 1. Cập nhật TONKHO
UPDATE TONKHO
    SET TSLNHAP = TSLNHAP + @Soluongcl
    FROM TONKHO TK INNER JOIN INSERTED I
    ON I.MAVTU=TK.MAVTU AND TK.NAMTHANG=@Namthang
-- 2. Cập nhật PNHAP
UPDATE PNHAP
    SET TONGTGNHAP= TONGTGNHAP + @Trigiactl
    FROM INSERTED I
    WHERE PNHAP.SOPN=I.SOPN
GO
```

#### IV.4. Instead of trigger và cập nhật dữ liệu trên bảng ảo

Với instead of trigger, việc cập nhật dữ liệu trên bảng ảo (view) sẽ được thực hiện dễ dàng. Trước đây:

- ✓ Việc cập nhật dữ liệu chỉ có tác dụng trên một bảng nếu bảng ảo được xây dựng từ nhiều bảng gốc
- ✓ Một bảng ảo xây dựng dựa trên nhiều bảng sẽ không thể dùng để xoá dữ liệu
- ✓ Danh sách các cột của bảng ảo đó phải có đầy đủ các cột của bảng gốc muốn cập nhật (nếu các cột này tham gia vào khoá chính, khoá ngoại hay có thuộc tính NOT NULL và không có giá trị mặc định)
- ✓ Nếu trong bảng ảo có một cột là calculated column thì bảng ảo đó không cập nhật được
- ✓ Một bảng ảo được tạo từ câu lệnh SELECT với từ khoá DISTINCT thì không thể cập nhật dữ liệu được
- ✓ Một bảng ảo được tạo từ câu lệnh SELECT với từ khoá TOP và có sử dụng ORDER BY thì không thể cập nhật dữ liệu được
- ✓ Một bảng ảo được tạo từ câu lệnh SELECT có phân nhóm (GROUP BY) thì không thể cập nhật dữ liệu được
- ✓ Nếu bảng ảo có thể cập nhật dữ liệu được (không rơi vào các trường hợp loại trừ ở trên) nhưng do các bảng gốc có các mối ràng buộc khoá ngoại thì việc cập nhật có thể sẽ không thực hiện thành công



##### Ví dụ:

Giả sử bạn muốn tạo một bảng ảo có tên là vw\_CTDONDH (chi tiết đơn đặt hàng) để liệt kê chi tiết đặt hàng của các đơn đặt hàng với dữ liệu lấy từ ba bảng DonDH(SoDH, NgayDH, MaNhaCC), CTDONDH(SoDH, MaVTu, SoLuong, DonGia) và VatTu(MaVTu, TenVTu, DVTinh, PhanTram). Trong vw\_CTDONDH bạn muốn liệt kê các thông tin Số đặt hàng, Ngày đặt hàng, Tên vật tư, Số lượng, Đơn giá.

Để có thể sử dụng vw\_CTDONDH để cập nhật dữ liệu vào bảng DonDH, bạn cần có thêm cột MaNhaCC, đồng thời cột SoDH phải lấy từ bảng DonDH. Tương tự, nếu muốn dùng vw\_CTDONDH để cập nhật dữ liệu vào bảng CTDONDH bạn sẽ cần thêm cột MaVTu và cột SoDH phải lấy từ bảng CTDONDH.

Như vậy, trên một bảng ảo, bạn không thể cùng lúc cập nhật dữ liệu vào hai bảng vì cột SoDH chỉ lấy từ bảng CTDONDH hoặc DonDH mà thôi. Nếu bạn sử dụng alias để lấy cả hai cột SoDH từ DonDH và CTDONDH thì lúc này bảng ảo vw\_CTDONDH sẽ trở nên rắc rối và chứa hai cột dữ liệu giống nhau.

Ngoài ra, vw\_CTDONDH cũng sẽ không thể dùng để xoá một chi tiết đặt hàng vì lấy dữ liệu từ nhiều bảng gốc.

Với phiên bản SQL Server 2000, Microsoft cung cấp instead of trigger cho phép bạn loại bỏ những giới hạn trên, ngoại trừ trường hợp bảng ảo có mệnh đề ORDER BY.

Vì bản chất của instead of trigger là không để cho dữ liệu được cập nhật xuống bảng (hay bảng ảo mà instead of trigger đó được định nghĩa) nên các điều kiện ở trên sẽ không bị SQL Server kiểm tra. Lúc này, dữ liệu được đưa vào bảng inserted hay deleted và bạn có thể sử dụng



những bảng này để thực hiện việc cập nhật dữ liệu xuống các bảng gốc.



**Ví dụ:**

Bạn xây dựng vw\_CTDONDH với câu lệnh *SELECT* như sau:

```
Select D.SoDH, NgayDH, MaNhaCC, V.MaVTu, TenVTu, SoLuong, DonGia From CTDONDH CT,
DONDH D, VATTU V Where CT.SoDH = D.SoDH And CT.MaVTu = V.MaVTu
```

*Instead of trigger* sau sẽ thực hiện cập nhật dữ liệu một cách linh hoạt vào bảng CTDONDH và cả bảng DONDH, VATTU khi cần.

```
CREATE TRIGGER tg_vw_CTDONDH_BI
INSTEAD OF INSEART ON vw_CTDONDH
AS
-- Nếu chưa có đơn đặt hàng, thêm đơn đặt hàng vào DONDH
Insert Into DONDH Select SoDH, NgayDH, MaNhaCC From Inserted Where SoDH Not In
(Select SoDH From DonDH)

-- Nếu chưa có vật tư, thêm vật tư vào bảng VATTU
Insert Into VATTU(MaVTu, TenVTu) Select MaVTu, TenVTu From Inserted Where MaVTu Not In
(Select MaVTu From VATTU)

-- Thêm các chi tiết đặt hàng vào CTDONDH
Insert Into CTDONDH Select SoDH, MaVTu, SoLuong, DonGia From Inserted
```

*Trigger* sau giúp xóa dữ liệu trên bảng CTDONDH qua bảng ảo

```
CREATE TRIGGER tg_vw_CTDONDH_BD
INSTEAD OF DELETE ON vw_CTDONDH
AS
Delete CTDONDH From CTDONDH CT Inner Join Inserted I On CT.SoDH = I.SoDH And
CT.MaVTu = I.MaVTu
```

*Trigger* sau cập nhật giá trị SoLuong và DonGia của chi tiết đặt hàng

```
CREATE TRIGGER tg_vw_CTDONDH_BU
INSTEAD OF UPDATE ON vw_CTDONDH
AS
Update CTDONDH
From CTDONDH CT inner join Inserted I on I.SoDH=CT.SoDH and I.MaVTu=CT.MaVTu Set
SoLuong = i.SoLuong, DonGia = i.DonGia
```

Với các *instead of trigger* đã được định nghĩa trên vw\_CTDONDH như trên, bạn có thể coi vw\_CTDONDH như một bảng và có thể thực hiện các câu lệnh *INSERT*, *UPDATE*, *DELETE* trên bảng ảo này bình thường.

Tóm lại trong chương này chúng tôi đã trình bày về **đối tượng trigger** của Microsoft SQL Server. Việc phân chia, tổ chức các kiểm tra ràng buộc toàn vẹn dữ liệu phức tạp hoặc các cập nhật dữ liệu tự động trong đối tượng trigger sẽ làm cho các xử lý được tập trung tại máy chủ và độc lập với ngôn ngữ lập trình tại máy trạm. Điều này làm cho tốc độ của các ứng dụng theo mô hình khách chủ được nhanh hơn.

**Đề thi:**

**SQL Server**

**Thời gian: 150 phút**

\*\*\* Các kết quả lưu vào đĩa làm việc để chấm điểm \*\*\*

**Phần 1: Tạo cấu trúc các bảng. (2đ)** (Các trường gạch dưới và in đậm là khoá chính của bảng)

a- Tạo bảng **CHU\_DE**(Chủ đề phim) với các cột dữ liệu và yêu cầu như sau:

Column Name	Description	Datatype	Length	Allow Null
Macd	Mã chủ đề phim	Char	2	No
<b>Tencd</b>	Tên chủ đề phim	VarChar	50	No

b- Tạo bảng PHIM (Danh mục phim) với các cột dữ liệu và yêu cầu như sau:

Column Name	Description	Datatype	Length	Allow Null
Maphim	Mã phim	Char	4	No
<b>Tenphim</b>	Tên phim	VarChar	50	No
<b>NgayPH</b>	Ngày phát hành	Date		No
<b>Sldia</b>	Số lượng đĩa của phim	Int		No
<b>Dongia</b>	Đơn giá phim	Int		No

Kiểm tra ràng buộc: Sldia>0 và Dongia>0

c- Tạo bảng PHIM\_CHU\_DE(Phim và chủ đề) với các cột dữ liệu và yêu cầu như sau:

Column Name	Description	Datatype	Length	Allow Null
Maphim	Mã phim	Char	4	No
Macd	Mã chủ đề phim	Char	2	No

Cài đặt ràng buộc: Maphim là khóa ngoại tham chiếu đến bảng PHIM

Cài đặt ràng buộc: Macd là khóa ngoại tham chiếu đến bảng CHU\_DE

d- Tạo bảng GIAO\_PHIM(Các phim được mua) với các cột dữ liệu và yêu cầu như sau:

Column Name	Description	Datatype	Length	Allow Null
Sttgiao	Số thứ tự các lần giao phim	Identity	10	No
<b>Ngaygiao</b>	Ngày giao phim	Date		No
<b>Maphim</b>	Mã phim	Char	4	No
<b>Sobo</b>	Số bộ phim mang đi giao	Int		No
<b>Sotien</b>	Số tiền bán được phim	Int		Yes

# TRUNG TÂM TIN HỌC ĐH KHOA HỌC TỰ NHIÊN TP.HỒ CHÍ MINH

Cài đặt ràng buộc: Maphim là khóa ngoại tham chiếu đến bảng PHIM

Kiểm tra ràng buộc: Sobo>0 và Sotien>=0

Mô tả: Số tiền bán phim = Số bộ phim x Đơn giá phim

## **Phần 2: Xây dựng các view liên kết dữ liệu (3đ)**

a. Liệt kê thông tin các phim đem giao ngày 15/4/2004. **(0.5đ)**

Mẫu:

Ma phim      Ten phim   So bo giao

b. Liệt kê các chủ đề nào chưa có phim. **(0.5đ)**

Mẫu:

Ma chu de      Ten chu de

c. Liệt kê các phim bán được với số tiền nhiều nhất trong năm 2004. **(1đ)**

Mẫu:

Ma phim      Ten phim      Tong so luot giao      Tong so tien ban duoc

d. Thống kê theo tháng năm doanh số bán phim. **(1đ)**

Mẫu:

Thang nam      Tong so tien ban

## **Phần 3: Xây dựng thủ tục nội tại để tính toán yêu cầu nào đó. (2.5đ)**

a- Xây dựng thủ tục nội tại trả về số tiền bán phim trong tháng, thủ tục được mô tả như sau: **(1.5đ)**

- Tham số vào: Mã phim, Tháng năm (dạng YYYYMM), Số tiền bán phim (biến OUTPUT)
- Tham số ra: không có.
- Yêu cầu:
  - ☐ Kiểm tra Tháng năm có đúng dạng YYYYMM
  - ☐ Kiểm tra Mã phim phải tồn tại trong bảng PHIM
  - ☐ Gán giá trị đúng cho biến OUTPUT Số tiền bán phim là tiền bán phim trong tháng (cẩn cú trên ngày giao)

Nếu vi phạm các yêu cầu kiểm tra thì thông báo lỗi

b- Xây dựng thủ tục nội tại xuất ra cursor, thủ tục được mô tả như sau: **(1đ)**

- Tham số vào: không có
- Tham số ra: Danh sách các phim có kèm theo chủ đề, gồm các cột sau: Mã phim, Tên phim, Ngày phát hành, Các chủ đề.
- Mẫu:

<u>Ma phim</u>	<u>Ten phim</u>	<u>Ngày PH</u>	<u>Các chủ đề</u>
0001	The Legend of Evil Lake	1/4/2004	Hành động, Tình cảm
	Truyện thuyết hồ Quỷ Cốc		

...

## **Phần 4: Xây dựng trigger. (2.5đ)**

Tạo trigger cho hành động xóa dữ liệu trên bảng PHIM. Khi người dùng xóa một dòng dữ liệu trong bảng PHIM thì trigger này sẽ:

- Kiểm tra tính tồn tại của maphim trong bảng GIAO\_PHIM
- Kiểm tra tính tồn tại của maphim trong bảng PHIM\_CHU\_DE
- Khi một trong các kiểm tra ở trên đúng thì không cho xóa dòng dữ liệu trong bảng PHIM và hiển thị thông báo lỗi tương ứng cho người dùng biết
- Ngược lại thì cho xóa dòng dữ liệu trong bảng PHIM

# TRUNG TÂM TIN HỌC ĐH KHOA HỌC TỰ NHIÊN TP.HỒ CHÍ MINH

## Dữ liệu thử (không bắt buộc nhập)

CHU_DE	
Macd	Tencd
HD	Hành động
HS	Hình sự
TC	Tình cảm
KD	Kinh dị

PHIM				
Maphim	Tenphim	NgayPH	Sldia	Dongia
0001	The Legend of Evil Lake Truyền thuyết hồ Quỷ Cốc	1/4/2004	2	18000
0002	The Pledge - Lời cam kết	1/4/2004	2	18000
0003	Gothika – Linh hồn bí ẩn	1/4/2004	3	26000

PHIM_CHU_DE	
Maphim	Macd
0001	HD
0001	TC
0002	HS
0003	HS
0003	KD

GIAO_PHIM				
sttgiao	Ngaygiao	Maphim	Sobo	Sotien
1	15/4/2004	0001	3	
2	15/4/2004	0002	1	
3	16/4/2004	0003	2	
4	20/4/2004	0001	5	

## **ĐỀ KIỂM TRA GIÁO VIÊN**

### **Kiểm tra chuyên môn Giáo viên - SQL Server**

Thời gian : 150 phút

*\*\*\* Các kết quả lưu trên giấy để chấm điểm \*\*\**

#### **A. Phần trắc nghiệm**

1. Giá trị Null trong cột của biểu thức ON được xử lý thế nào khi tiến hành JOIN hai bảng
  - a. Các dòng có chứa giá trị NULL của bảng này chỉ kết hợp các dòng có giá trị NULL ở bảng còn lại
  - b. Các dòng có chứa giá trị NULL sẽ không được trả về trừ khi sử dụng OUTER JOIN
  - c. Các dòng có chứa giá trị NULL của bảng này kết hợp với tất cả các dòng bảng còn lại
  - d. Các dòng có chứa giá trị NULL của bảng này không kết hợp được với bất cứ dòng nào ở bảng còn lại
2. Để sử dụng cursor làm tham số trong thủ tục nội tại spud\_MyProc thì khai báo nào dưới đây là ĐÚNG:
  - a. CREATE PROC spud\_MyProc @cur\_sor CURSOR VARYING OUTPUT IS ...
  - b. CREATE PROC spud\_MyProc @cur\_sor CURSOR VARYING OUTPUT AS ...
  - c. CREATE PROC spud\_MyProc @cur\_sor CURSOR VARYING IS ...
  - d. CREATE PROC spud\_MyProc @cur\_sor CURSOR IS ...
3. Xét bảng PHIEU\_NHAP(sopn, ngaynhap).có các mẫu tin sau:

Phieu_Nhap	
Sopn	Ngaynhap
N001	2004-04-01
N002	2004-04-03
N003	2004-04-10
N004	2004-05-01
N005	2004-05-01

Để lọc ra các phiếu nhập phát hành trong tháng 4/2004 thì câu lệnh nào dưới đây sẽ lọc có dữ liệu:

- a. SELECT \* FROM Phieu\_Nhap WHERE CONVERT(CHAR(6),ngaynhap,112)='2004-04'
  - b. SELECT \* FROM Phieu\_Nhap WHERE CONVERT(CHAR(6),ngaynhap,112)='200404'
  - c. SELECT \* FROM Phieu\_Nhap WHERE CONVERT(CHAR(7),ngaynhap,112)='2004-04'
  - d. SELECT \* FROM Phieu\_Nhap WHERE CONVERT(CHAR(6),ngaynhap,12)='200404'
4. Câu lệnh xoá cột trong bảng nào dưới đây là ĐÚNG cú pháp:
    - a. ALTER TABLE <tên bảng> DROP <tên cột>
    - b. ALTER TABLE <tên bảng> DROP COLUMN ALL
    - c. ALTER TABLE <tên bảng> DROP COLUMN <tên cột>
    - d. Các câu a,b và c đều sai
  5. Câu lệnh tạo view nào dưới đây là ĐÚNG:
    - a. CREATE VIEW v\_SinhVien AS SELECT sv.\*, kh.tenkh FROM Sinh\_Vien sv INNER JOIN Khoa kh ON sv.makh=kh.makh ORDER BY makh
    - b. CREATE VIEW v\_SinhVien AS SELECT sv.\*, kh.tenkh FROM Sinh\_Vien sv INNER JOIN Khoa kh ON sv.makh=kh.makh ORDER BY sv.makh
    - c. CREATE VIEW v\_SinhVien AS SELECT sv.\*, kh.tenkh FROM Sinh\_Vien sv INNER JOIN Khoa kh ON sv.makh=kh.makh ORDER BY kh.makh
    - d. Các câu a,b và c đều sai

## TRUNG TÂM TIN HỌC ĐH KHOA HỌC TỰ NHIÊN TP.HỒ CHÍ MINH

6. Giả sử bảng KHOA là bảng rỗng và có cấu trúc: KHOA(makh,tenkh), cột tenkh không cho phép NULL. Chọn các câu lệnh truy vấn nào dưới đây gây ra lỗi “Cannot insert the value NULL into column 'tenkh'” (chọn nhiều câu ĐÚNG)

- a. INSERT INTO Khoa (makh,tenkh) VALUES ('LY',NULL)
- b. INSERT INTO Khoa (makh,tenkh) VALUES ('LY','')
- c. INSERT INTO Khoa VALUES ('LY',NULL)
- d. INSERT INTO Khoa (makh) VALUES ('LY')

7. Giả sử bảng KHOA là bảng rỗng và có cấu trúc: KHOA(makh,tenkh). Xét giao tác sau:

```
BEGIN TRAN Cap1
INSERT INTO Khoa (makh,tenkh) VALUES ('AV','Khoa Anh')
BEGIN TRAN Cap2
INSERT INTO Khoa (makh,tenkh) VALUES ('LY','Khoa Vat ly')
COMMIT TRAN Cap2
ROLLBACK TRAN Cap1
```

Cho biết kết quả nào dưới đây là ĐÚNG khi thực hiện lệnh SELECT \* FROM Khoa:

- a. AV, LY
- b. AV
- c. LY
- d. Bảng KHOA vẫn là bảng rỗng

8. lệnh RETURN có ý nghĩa gì trong Stored Procedure. Chọn nhiều câu ĐÚNG (chọn nhiều câu):

- a. Trả về một số nguyên
- b. Trả về một số nguyên khác 0
- c. Kết thúc một thủ tục
- d. Luôn bắt buộc phải chỉ ra giá trị trả về. Ví dụ: RETURN 0

9. Hàm nào sau đây không có trong SQL Server

- a. @@Fetch\_Status
- b. @@FetchStatus
- c. @@Error
- d. @@Version

10. Để tạo bảng lưu VatTu\_BK từ bảng VatTu. Chọn câu lệnh ĐÚNG:

- a. SELECT \* FROM VatTu INTO VatTu\_BK
- b. CREATE TABLE VatTu\_BK FROM VatTu
- c. CREATE TABLE VatTu\_BK AS SELECT \* FROM VatTu
- d. SELECT \* INTO VatTu\_BK FROM VatTu

11. Thực hiện các câu lệnh sau:

```
DECLARE @thong_bao CHAR(10), @ma_loi INT
SET @thong_bao = 'Lỗi: '
SET @ma_loi = 1
IF @ma_loi=1
SET @thong_bao = @thong_bao+'không xác định'
PRINT @thong_bao
```

Kết quả in ra biến @thong\_bao nào dưới đây là ĐÚNG:



- a. 'Lỗi: '
- b. 'Lỗi: không xác định'
- c. Hệ thống báo lỗi tràn chuỗi trong biến @thong\_bao
- d. Các câu a,b và c đều sai
12. Để kết thúc một giao tác (transaction) mà không ghi nhận thay đổi thì các trường hợp nào chỉ ra dưới đây là ĐÚNG:
- a. ROLLBACK
- b. ROLLBACK TRAN
- c. Các câu a và b đều đúng
- d. Các câu a và b đều sai
13. Để liệt kê các sinh viên thì các phát biểu nào dưới đây là ĐÚNG:
- a. 

```
SELECT masv,ho,ten,  
phai=CASE ISNULL(nam,2)  
WHEN 1 THEN 'Nam'  
WHEN 0 THEN 'Nu'  
ELSE 2 THEN 'Khong xac dinh' END  
FROM Sinh_Vien
```
- b. 

```
SELECT masv,ho,ten,  
phai=CASE ISNULL(nam,2)  
WHEN 1 THEN 'Nam'  
WHEN 0 THEN 'Nu'  
WHEN 2 THEN 'Khong xac dinh' END  
FROM Sinh_Vien
```
- c. 

```
SELECT masv,ho,ten,  
phai=CASE ISNULL(nam,2)  
WHEN 1 THEN 'Nam'  
WHEN 0 THEN 'Nu'  
ELSE 'Khong xac dinh' END  
FROM Sinh_Vien
```
- d. Các câu a, b và c đều sai
14. Giả sử trigger thêm trên bảng KHOA được tạo bởi các lệnh sau:
- ```
CREATE TRIGGER tg_Khoa_Insert ON Khoa FOR INSERT  
AS  
DECLARE @tenkh VARCHAR(50)  
SELECT @tenkh=tenkh FROM INSERTED  
IF @tenkh = NULL OR @tenkh = ''  
BEGIN  
ROLLBACK TRAN  
RAISERROR('Tên khoa không được rỗng',16,1)  
RETURN  
END
```

Cho biết câu lệnh INSERT nào dưới đây là ĐÚNG (thực hiện thành công):

## TRUNG TÂM TIN HỌC ĐH KHOA HỌC TỰ NHIÊN TP.HỒ CHÍ MINH

---

- a. INSERT INTO Khoa (makh,tenkh) VALUES('00',NULL)
  - b. INSERT INTO Khoa (makh,tenkh) VALUES('00','')
  - c. Các câu a, b đều đúng
  - d. Các câu a, b đều sai
- 15.** Để di chuyển mẫu tin trong Cursor. Chọn các câu lệnh ĐÚNG (chọn nhiều câu ĐÚNG):
- a. FETCH FIRST
  - b. FETCH LAST
  - c. FETCH PRIOR
  - d. FETCH PREVIOUS
- 16.** Hàm Fetch\_Status dùng để xác định trạng thái sau khi đọc mẫu tin trong cursor. Chọn các phát biểu ĐÚNG (chọn nhiều câu ĐÚNG):
- a. Trả về 0 nếu đọc thành công
  - b. Trả về 0 nếu đọc không thành công
  - c. Trả về -1 nếu đọc không thành công (giá trị duy nhất)
  - d. Trả về -1 hoặc -2 nếu đọc không thành công (nhiều giá trị)
- 17.** Để đổi từ mã ASCII sang ký tự. Chọn phát biểu ĐÚNG:
- a. Hàm ASCII
  - b. Hàm CHR
  - c. Hàm CHAR
  - d. Các câu a, b và c đều đúng
- 18.** Để xác định các cấp giao tác (transaction). Chọn phát biểu ĐÚNG:
- a. Hàm @@TRANSCOUNT
  - b. Hàm @@TRANCOUNT
  - c. Hàm @@ROWCOUNT
  - d. Hàm @@TRAN\_COUNT
- 19.** Cho biết thứ tự của các mệnh đề trong câu truy vấn SQL
- a. Select, From, Where, Group By, Order By, Having
  - b. Select, From, Where, Order By, Group By, Having
  - c. Select, From, Where, Having, Group By, Order By
  - d. Select, From, Where, Group By, Having, Order By
- 20.** Trong các đoạn lệnh sau đây, đoạn lệnh nào ĐÚNG
- a. declare @a int  
@a=1
  - b. declare @a int  
select @a=1
  - c. declare @a int  
set a=1
  - d. declare @a int  
Set @a=select 1
- 21.** Thứ tự nào sau đây được dùng để làm việc với Cursor
- a. Declare, Open, Fetch, Deallocate, Close
  - b. Declare, Fetch, Open, Deallocate, Close
  - c. Open, Declare, Fetch, Close, Deallocate
  - d. Declare, Open, Fetch, Close, Deallocate

## TRUNG TÂM TIN HỌC ĐH KHOA HỌC TỰ NHIÊN TP.HỒ CHÍ MINH

---

22. Trong bảng [Don Dat Hang] có hai cột Ngay\_Dat và Ngay\_Giao, để ràng buộc dữ liệu nhập vào có Ngay\_Dat phải trước Ngay\_Giao, đoạn lệnh nào theo sau ALTER TABLE [Don Dat Hang] dưới đây là ĐÚNG:

- a. ADD CHECK Ngay\_Dat < Ngay\_Giao
- b. ADD CHECK (Ngay\_Dat < Ngay\_Giao)
- c. ADD CONSTRAINT CHECK (Ngay\_Dat < Ngay\_Giao)
- d. ADD CONSTRAINT (Ngay\_Dat < Ngay\_Giao)

23. Từ khoá nào sau đây cho biết câu truy vấn con không trả về dòng nào hoặc trả về nhiều dòng

- a. IN
- b. ANY
- c. ALL
- d. EXISTS

24. Trong các phát biểu sau về VIEW, câu nào KHÔNG đúng?

- a. Một view chiếm một khoảng không gian lưu trữ dữ liệu tương tự như một table bình thường
- b. Một view coi như một table đối với người dùng
- c. Một view là một câu query được định nghĩa trước và lưu trong cơ sở dữ liệu
- d. Một view có thể được sử dụng để giới hạn người dùng chỉ được truy cập một số cột hay dòng dữ liệu

25. Giả sử trigger trên bảng NHAN\_VIEN được tạo bởi các lệnh sau:

```
CREATE TRIGGER trig1 ON NHAN_VIEN FOR INSERT
AS
IF (SELECT COUNT(*) FROM ???
WHERE mucluong > 100000) > 0
BEGIN
    PRINT "Loi: khong the co muc luong > $100,000"
    ROLLBACK TRAN
END
```

Cho biết bảng nào dưới đây là ĐÚNG khi được điền vào sau mệnh đề FROM:

- a. bảng DELETE
- b. bảng DELETED
- c. bảng INSERT
- d. bảng INSERTED

26. Bảng sysobjects chứa thông tin về các table, stored proc, trigger, và các đối tượng khác trong SQL Server. Các database nào có chứa bảng sysobject

- a. Pubs
- b. Master
- c. Msdb
- d. Tất cả các database

27. Những hàm nào sau đây dùng để chuyển giá trị trong cột Diem kiểu SmallInt sang kiểu chuỗi (chọn nhiều câu ĐÚNG)

- a. STR(Diem)
- b. STR(Char(3),Diem)
- c. CONVERT(Char(3),Diem)
- d. CONVERT(SmallInt, Diem)

## TRUNG TÂM TIN HỌC ĐH KHOA HỌC TỰ NHIÊN TP.HỒ CHÍ MINH

28. Trên bảng Luong, cột TienLuong được đặt các thuộc tính Permission để không cho phép người dùng trong Role Public được quyền Đọc hay Ghi. Việc gì sẽ xảy ra khi một người dùng thuộc Role Public thực hiện câu truy vấn: `Select * From Luong`

- a. Kết quả hiển thị những cột khác trừ TienLuong
- b. Kết quả hiển thị tất cả các cột kể cả TienLuong
- c. Nhận được thông báo 'permission denied'
- d. Các câu a và b đều sai

29. Để tính tổng các số nguyên từ 1 đến 10 bằng cách sử dụng vòng lặp WHILE. Xét đoạn lệnh dưới đây:

```
DECLARE @i INT, @tong INT
SET @i=1
SET @tong=0
WHILE @i<=10
    SET @tong=@tong+@i
    SET @i=@i+1
PRINT @tong
```

Trong các trường hợp thì trường hợp nào dưới đây là ĐÚNG:

- a. Kết quả in ra của biến @tong là: 55
- b. Kết quả in ra của biến @tong là: 0
- c. Kết quả in ra của biến @tong là: 1
- d. Vòng lặp không bao giờ dừng

30. Câu lệnh tạo bảng nào dưới đây là ĐÚNG:

- a. 

```
CREATE TABLE VatTu
(mavt CHAR(4) NOT NULL,
tenvt VARCHAR(30),
phantram TINYINT
CONSTRAINT PRK_VatTu_mavt PRIMARY KEY(mavt),
CONSTRAINT DEF_VatTu_phantram DEFAULT 20 FOR phantram)
```
- b. 

```
CREATE TABLE VatTu
(mavt CHAR(4) NOT NULL,
tenvt VARCHAR(30),
phantram TINYINT
CONSTRAINT DEF_VatTu_phantram DEFAULT 20 FOR phantram
CONSTRAINT PRK_VatTu_mavt PRIMARY KEY(mavt))
```
- c. 

```
CREATE TABLE VatTu
(mavt CHAR(4) NOT NULL,
tenvt VARCHAR(30),
phantram TINYINT
CONSTRAINT DEF_VatTu_phantram DEFAULT 20
CONSTRAINT PRK_VatTu_mavt PRIMARY KEY(mavt))
```
- d. 

```
CREATE TABLE VatTu
(mavt CHAR(4) NOT NULL,
tenvt VARCHAR(30),
phantram TINYINT,
CONSTRAINT DEF_VatTu_phantram DEFAULT 20
CONSTRAINT PRK_VatTu_mavt PRIMARY KEY(mavt))
```

# TRUNG TÂM TIN HỌC ĐH KHOA HỌC TỰ NHIÊN TP.HỒ CHÍ MINH

## B. Phần thực hành

### Phần 1: Tạo cấu trúc các bảng. (Các trường gạch dưới và in đậm là khoá chính của bảng)

a- Tạo bảng **NHAN\_VIEN**(Nhân viên) với các cột dữ liệu và yêu cầu như sau:

| Column Name        | Description   | Datatype | Length | Allow Null |
|--------------------|---------------|----------|--------|------------|
| <b><u>Manv</u></b> | Mã nhân viên  | Char     | 3      | No         |
| <u>Tennv</u>       | Tên nhân viên | VarChar  | 50     | No         |

b- Tạo bảng **DE\_NGHI** (Đề nghị tạm ứng) với các cột dữ liệu và yêu cầu như sau:

| Column Name        | Description               | Datatype | Length | Allow Null |
|--------------------|---------------------------|----------|--------|------------|
| <b><u>Sotu</u></b> | Số đề nghị tạm ứng        | Char     | 7      | No         |
| <u>Ngay_tu</u>     | Ngày đề nghị              | Date     |        | No         |
| <u>Manv</u>        | Mã nhân viên được tạm ứng | Char     | 3      | No         |
| <u>Sotien_tu</u>   | Số tiền tạm ứng           | Int      |        | No         |

Cài đặt ràng buộc: Manv là khóa ngoại tham chiếu đến bảng NHAN\_VIEN

Kiểm tra ràng buộc: Sotien\_tu>0

c- Tạo bảng **PHIEU\_CHI**(Phiếu chi tiền mặt) với các cột dữ liệu và yêu cầu như sau:

| Column Name        | Description                 | Datatype | Length | Allow Null |
|--------------------|-----------------------------|----------|--------|------------|
| <b><u>Sopc</u></b> | Số phiếu chi                | Char     | 5      | No         |
| <u>Ngay_chi</u>    | Ngày chi                    | Date     |        | No         |
| <u>Sotu</u>        | Chi theo Số đề nghị tạm ứng | Char     | 7      | No         |
| <u>Manv</u>        | Mã nhân viên được chi       | Char     | 3      | No         |
| <u>Sotien_chi</u>  | Số tiền chi                 | Int      |        | No         |

Cài đặt ràng buộc: Sotu là khóa ngoại tham chiếu đến bảng DE\_NGHI

Cài đặt ràng buộc: Manv là khóa ngoại tham chiếu đến bảng NHAN\_VIEN

Kiểm tra ràng buộc: Sotien\_chi>0

d- Tạo bảng **GIAI\_TRINH**(Giải trình tạm ứng) với các cột dữ liệu và yêu cầu như sau:

| Column Name        | Description                          | Datatype | Length | Allow Null |
|--------------------|--------------------------------------|----------|--------|------------|
| <b><u>Sott</u></b> | Số thứ tự các lần giải trình tạm ứng | Identity | 10     | No         |
| <u>Ngay_gt</u>     | Ngày giải trình                      | Date     |        | No         |
| <u>Sotu</u>        | Giải trình cho Số đề nghị tạm ứng    | Char     | 4      | No         |
| <u>Sotien_gt</u>   | Số tiền giải trình tạm ứng           | Int      |        | No         |

Cài đặt ràng buộc: Sotu là khóa ngoại tham chiếu đến bảng DE\_NGHI

Kiểm tra ràng buộc: Sotien\_gt>=0

### Phần 2: Xây dựng các view liên kết dữ liệu

a. Liệt kê danh sách các nhân viên có tạm ứng trong năm 2004. Mô tả: một nhân viên có thể có nhiều phiếu đề nghị tạm ứng

Mẫu:

Ma nv      Ten      Ngay tam ung      So tien tam ung

b. Liệt kê các phiếu đề nghị tạm ứng nào chưa có giải trình.

Mẫu:

So tam ung      Ngay tam ung      Ma nv      Ten      So tien tam ung

## TRUNG TÂM TIN HỌC ĐH KHOA HỌC TỰ NHIÊN TP.HỒ CHÍ MINH

- c. Liệt kê thông tin tạm ứng và giải trình tạm ứng. Mô tả: một phiếu đề nghị tạm ứng có thể có nhiều lần giải trình. Tiền còn lại = Số tiền tạm ứng – Tổng tiền giải trình

Mẫu:

Số tạm ứng   Ngày tạm ứng   Ma nv   Tên   Số tiền tạm ứng   Tổng tiền giải trình   Tiền còn lại

- d. Thống kê tình hình tạm ứng và giải trình theo nhân viên.

Mẫu:

Ma nv   Tên   Tổng số phiếu tạm ứng   Tổng số tiền tạm ứng   Tổng số tiền giải trình

### **Phần 3: Xây dựng thủ tục nội tại để tính toán yêu cầu nào đó.**

- a- Xây dựng thủ tục nội tại kiểm tra trạng thái của một phiếu tạm ứng, thủ tục được mô tả như sau:

- *Tham số vào:* Số tạm ứng
- *Tham số ra:* không có.
- *Yêu cầu:* kiểm tra lần lượt theo thứ tự sau:
  - ☐ Kiểm tra Số tạm ứng có trong bảng DE\_NGHI không ? Nếu có thì kiểm tra tiếp:
  - ☐ Kiểm tra Số tạm ứng này đã có phiếu chi nào chưa ? Nếu có thì kiểm tra tiếp:
  - ☐ Kiểm tra Số tạm ứng này đã có ít nhất một lần giải trình nào chưa ? Nếu có thì kiểm tra tiếp:
  - ☐ Kiểm tra Số tạm ứng này đã giải trình đủ chưa ? nếu đã giải trình đủ thì thông báo "đã giải trình đủ" ngược lại thì "chưa giải trình đủ"

Nếu vi phạm các yêu cầu kiểm tra thì thông báo lỗi

- b- Xây dựng thủ tục nội tại xuất ra cursor, thủ tục được mô tả như sau:

- *Tham số vào:* Mã nhân viên
- *Tham số ra:* Danh sách các phiếu chi, gồm các cột sau: Mã nhân viên, Tên nhân viên, Số phiếu chi, Ngày chi, Số tiền chi.

### **Phần 4: Xây dựng trigger.**

Tạo trigger cho hành động sửa dữ liệu trên bảng PHIEU\_CHI (chỉ được phép sửa đúng 2 cột là: số tạm ứng và ngày chi). Khi người dùng sửa một dòng dữ liệu trong bảng PHIEU\_CHI thì trigger này sẽ:

- Kiểm tra nếu phiếu chi chỉ cho tạm ứng đã có giải trình thì không cho sửa và hiển thị thông báo lỗi
- Kiểm tra nếu sotu không có trong bảng DE\_NGHI thì không cho sửa và hiển thị thông báo lỗi
- Nếu sotu có trong bảng DE\_NGHI thì kiểm tra ngay\_chi >= ngay\_tu
- Nếu tất cả các kiểm tra trên đều thoả thì tự động gán lại manv và sotien\_chi ứng với sotu mới sửa

## TRUNG TÂM TIN HỌC ĐH KHOA HỌC TỰ NHIÊN TP.HỒ CHÍ MINH

### Dữ liệu tham khảo

| NHAN_VIEN |                  |
|-----------|------------------|
| Manv      | Tennv            |
| A01       | Hoàng Ngọc Anh   |
| B01       | Trần Văn Bình    |
| C01       | Lê Thị Châu      |
| C02       | Nguyễn Văn Chính |
| D01       | Lê Văn Dũng      |

| DE_NGHI |           |      |            |
|---------|-----------|------|------------|
| Sotu    | Ngay_tu   | Manv | Sotien_tu  |
| 2004001 | 15/4/2004 | A01  | 10 000 000 |
| 2004002 | 20/4/2004 | B01  | 2 000 000  |
| 2004003 | 28/4/2004 | C01  | 3 000 000  |
| 2004004 | 10/5/2004 | C02  | 5 000 000  |
| 2004005 | 12/7/2004 | A01  | 2 000 000  |

| PHIEU_CHI |           |         |      |            |
|-----------|-----------|---------|------|------------|
| Sopc      | Ngay_chi  | Sotu    | Manv | Sotien_chi |
| PC001     | 15/4/2004 | 2004001 | A01  | 10 000 000 |
| PC002     | 21/4/2004 | 2004002 | B01  | 2 000 000  |
| PC003     | 28/4/2004 | 2004003 | C01  | 3 000 000  |
| PC004     | 11/5/2004 | 2004004 | C02  | 5 000 000  |
| PC005     | 15/7/2004 | 2004005 | A01  | 2 000 000  |

| GIAI_TRINH |           |         |           |
|------------|-----------|---------|-----------|
| Sott       | Ngay_gt   | Sotu    | Sotien_gt |
| 1          | 15/5/2004 | 2004001 | 8 000 000 |
| 2          | 17/5/2004 | 2004001 | 2 000 000 |
| 3          | 18/5/2004 | 2004002 | 2 000 000 |
| 4          | 25/5/2004 | 2004003 | 2 500 000 |