TRƯỜNG CAO ĐẲNG LÝ TỰ TRỌNG

KHOA CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG

HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Tphcm, tháng 12 năm 2018

TRƯỜNG CAO ĐẲNG LÝ TỰ TRỌNG

KHOA CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG

HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Dành cho sinh viên

ngành Công nghệ thông tin

----TÀI LIỆU LƯU HÀNH NỘI BỘ---

Chương 1: TỔNG QUAN VỀ MICROSOFT SQL SERVER-MY SQL VÀ NGÔN NGỮ T-SQL

1. Mục tiêu: Sau khi học xong người học có khả năng:

- Cài đặt được phần mềm MS SQL Server
- Hiểu được các khái niệm trong Hệ quản trị MS SQL Server và My SQL
- Sử dụng được ngôn ngữ T_SQL trong SQL Sever

2. Nội dung chương:

2.1. TỔNG QUAN VỀ MS SQL SERVER

Hệ quản trị cơ sở dữ liệu (DBMS-Data Base Management System) là một phần mềm hay hệ thống được thiết kế ra nhằm mục đích quản trị (thêm, xóa, sửa, truy xuất thông tin) một Cơ sở dữ liệu.

Có rất nhiều loại Hệ quản trị cơ sở dữ liệu trên thị trường, đa số chúng đều sử dụng ngôn ngữ truy vấn có cấu trúc (SQL-Structured Query Language). Một số Hệ quản trị cơ sở dữ liệu thường dùng là: MySQL, SQL Server, Oracle, DB2,…Phần lớn các Hệ quản trị cơ sở dữ liệu đều hỗ trợ trên nhiều hệ điều hành khác nhau, riêng chỉ có SQL Server chỉ hỗ trợ trên hệ điều hành Windows.

Ưu điểm của Hệ quản trị cơ sở dữ liệu: quản lý việc dư thừa dữ liệu, đảm bảo tính nhất quán cho dữ liệu, chia sẻ dữ liệu, tính toàn vẹn dữ liệu. Tuy nhiên, một Hệ quản trị cơ sở dữ liệu tốt thì sẽ khá phức tạp, chiếm nhiều dung lượng bộ nhớ.

Microsoft SQL Server là hệ quản trị cơ sở dữ liệu quan hệ do Microsoft phát triển được hoạt động theo mô hình Client – Server và sử dụng ngôn ngữ truy vấn T-SQL.

2.1.1.Mô hình khách/chủ (Client/Server)

Mô hình Client/Server (mô hình khách/chủ) là một hệ thống mạng máy tính được kết nối với nhau, bao gồm các thành phần sau:

Hình 1.1. Mô hình Client/Server

- Server (máy chủ): Là máy tính có bộ xử lý với tốc độ cao, tài nguyên bộ nhớ lớn để hoạt động được tốt bởi cùng lúc sẽ có nhiều người dùng mạng truy xuất đến server thông qua các máy client. Trong một hệ thống mạng máy tính có thể có nhiều server với các chức năng độc lập riêng biệt khác nhau.

- Client (máy khách): là các máy tính được phép truy xuất đến các tài nguyên được chia sẻ trên mạng.

- Dây cáp mạng: là hệ thống dây kim loại hoặc quang học kết nối các máy tính, máy in lại với nhau.

- **Dữ liệu chung:** là các tập tin, thư mục mà người sử dụng trong hệ thống mạng có thể truy xuất đến server từ máy client.

Trong mô hình client/server, việc tổ chức các xử lý phải đảm bảo các yêu cầu (request) từ client phải được server hồi đáp (response) một cách nhanh chóng và không làm tắc nghẽn hệ thống mạng máy tính.

Ưu điểm của việc phát triển ứng dụng trên mô hình Client/Server:

- Giảm chi phí
- Tốc độ nhanh
- Tính tương thích cao

2.1.1.1 Khái niệm về cấu trúc vật lý

- Máy chủ (Server)
- Máy trạm (Client): là các máy tính được phép truy xuất các tài nguyên đã được chia sẻ trên mạng.
- 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

2.1.1.2. Khái niệm về các xử lý

- Các xử lý trong một ứng dụng có thể chia làm hai loại xử lý trên máy trạm và xử lý trên máy chủ
- Xử lý trên máy trạm
  - Đọc, cập nhật dữ liệu
  - Tính toán, hiển thị dữ liệu trên màn hình giao diện
  - Có thể sử dụng nhiều loại ngôn ngữ lập trình khác nhau: Java, C#...
- Xử lý trên máy chủ Database Server
  - Xử lý các yêu cầu đọc/ghi dữ liệu
  - Quản lý đồng bộ dữ liệu giữa các yêu cầu đọc ghi từ nhiều máy trạm gửi tới
  - Các dịch vụ quản trị dữ liệu tự động theo định kỳ như backup/restore dữ liệu

2.2.1 Microsoft SQL Server là gì?

SQL Server chính là một hệ quản trị dữ liệu quan hệ sử dụng câu lệnh SQL để trao đổi dữ liệu giữa máy cài SQL Server và máy Client. Một Relational Database Management System – RDBMS gồm có: databases, datase engine và các chương trình ứng dụng dùng để quản lý các bộ phận trong RDBMS và những dữ liệu khác.

2.2.2. Lịch sử ra đời

Năm 1989, phiên bản đầu tiên của SQL Server 1.0 ra đời được dùng cho các hệ điều hành 16 bit và được phát triển cho tới ngày nay.

Cho tới khi SQL Server ra phiên bản 6.5 thì được thị trường chấp nhận rộng rãi. Một đột phá cải tiến cho SQL Server 7.0 khi được Microsoft viết lại một engine hoàn toàn mới. Đến khi SQL Server từ phiên bản 7.0 cải tiến lên 8.0 chủ yếu phát triển về tính năng thiết kế web.

Cho đến ngày nay thì phiên bản mới nhất đó là SQL Server 2016 hỗ trợ bộ vi xử lý 64 bit ra đời vào ngày 1 tháng 6 năm 2016.

2.2.3. Cài đặt MS SQL Server

Để cài đặt SQL Server thì bạn cần phải có phiên bản Developer Edition và ít nhất là 500MB ổ cứng cùng với 64 MB Ram và có thể thể cài đặt trên hầu hết các hệ điều hành Windows.

Các bước cài đặt không có gì khó khăn cũng tương tự như các ứng dụng khác. Tuy nhiên trong quá trình cài đặt bạn cần lữu ý những điều sau:

+ Sau khi lựa chọn Install Database Server và chọn cài đặt SQL Server thì bạn có thể cài đặt thêm Analysis Service nếu bạn thích.

+ Ở màn hình Installation Definition thì bạn nên chọn Server and Client Tools.

+ Sau đó bạn nên chọn chọn tất cả các bộ phận của SQL Server và chọn kiểu Custom. Ngoài ra, bạn còn nên chọn các giá trị mặc định – default.

+ Sau khi cài đặt thành công, bạn sẽ thấy một icon nằm ở góc phải bên dưới của màn hình, đây chính là Service Manager. Bạn nên lưu ý rằng SQL Server có thể dùng chế độ bảo mật riêng của nó cũng có thể dùng chế độ security của hệ điều hành Windows.

2.2.4. Các tiện ích trong MS SQL Server

Các thành cơ bản trong SQL Server gồm có: Reporting Services, Database Engine, Integration Services, Notification Services, Full Text Search Service,… Tất cả kết hợp với nhau tạo thành một giải pháp hoàn chỉnh giúp cho việc phân tích và lưu trữ dữ liệu trở nên dễ dàng hơn.

+ **Database Engine**: Đây là một engine có khả năng chứa dữ liệu ở các quy mô dưới dạng support và table. Ngoài ra, nó còn có khả năng tự điều chỉnh ví dụ: trả lại tài nguyên cho ệ điều hành khi một user log off và sử dụng thêm các tài nguyên của máy khi cần.

+ **Integration Services**: là tập hợp các đối tượng lập trình và các công cụ đồ họa cho việc sao chép, di chuyển và chuyển đổi dữ liệu. Khi bạn làm việc trong một công ty lớn thì dữ liệu được lưu trữ ở nhiều nơi khác nhau như được chứa trong: Oracle, SQL Server, DB2, Microsoft Access,… và bạn chắc chắn sẽ có nhu cầu di chuyển dữ liệu giữa các server này. Ngoài ra, bạn còn muốn định dạng dữ liệu trước khi lưu vào database. Chắc chắn Integration Services sẽ giúp bạn giải quyết được công việc này dễ dàng.

+ **Analysis Services**: Đây là một dịch vụ phân tích dữ liệu rất hay của Microsoft. Dữ liệu khi được lưu trữ vào trong database mà bạn không thể lấy được những thông tin bổ ích thì coi như không có ý nghĩa gì. Chính vì thế, công cụ này ra đời giúp bạn trong việc phân tích dữ liệu một cách hiệu quả và dễ dàng bằng cách dùng kỹ thuật khai thác dữ liệu – datamining và khái niệm hình khối nhiều chiều – multi dimendion cubes.

+ **Notification Services**: Dịch vụ thông báo này là nền tảng cho sự phát triển và triển khai các ứng dụng soạn và gửi thông báo. Ngoài ra, dịch vụ này còn có chức năng gửi thông báo theo dịch thời đến hàng ngàn người dăng ký sử dụng trên nhiều loại thiết bị khác nhau.

+ **Reporting Services**: là một công cụ tạo, quản lý và triển khai báo cáo bao gồm: server và client. Ngoài ra, nó còn là nền tảng cho việc phát triển và xây dựng các ứng dụng báo cáo.

+ **Full Text Search Service**: là một thành phần đặc biệt trong việc truy vấn và đánh chỉ mục dữ liệu văn bản không cấu trúc được lưu trữ trong các cơ sở dữ liệu SQL Server.

+ **Service Broker**: là một môi trường lập trình cho việc tạo ra các ứng dụng trong việc nhảy qua các Instance.

2.2.5. Đăng ký quản trị Microsoft SQL Server

Thông thường một số thao tác trên CSDL hình thành một đơn vị logic công việc. Điều này có nghĩa là hoặc tất cả các thao tác được thực hiện hoặc không thao tác nào được thực hiện. Hơn nữa sự thực hiện các thao tác này phải đảm bảo tính nhất quán của CSDL. Một giao dịch là một tập hợp các thao tác mà xử lý như một đơn vị không chia cắt được. Các hệ quản trị CSDL điển hình cho phép người sử dụng một hay nhiều nhóm thao tác tra cứu hay thay đổi CSDL thành một giao dịch

2.2. TỔNG QUAN VỀ My SQL

2.2.1. Giới thiệu về My SQL

MySQL là hệ quản trị cơ sở dữ liệu tự do nguồn mở phổ biến nhất thế giới và được các nhà phát triển rất ưa chuộng trong quá trình phát triển ứng dụng. Vì MySQL là cơ sở dữ liệu tốc độ cao, ổn định và dễ sử dụng, có tính khả chuyển, hoạt động trên nhiều hệ điều hành cung cấp một hệ thống lớn các hàm tiện ích rất mạnh. Với tốc độ và tính bảo mật cao, MySQL rất thích hợp cho các ứng dụng có truy cập CSDL trên internet. MySQL miễn phí hoàn toàn cho nên bạn có thể tải về MySQL từ trang chủ. Nó có nhiều phiên bản cho các hệ điều hành khác nhau: phiên bản Win32 cho các hệ điều hành dòng Windows, Linux, Mac OS X, Unix, FreeBSD, NetBSD, Novell NetWare, SGI Irix, Solaris, SunOS,…

MySQL là một trong những ví dụ rất cơ bản về Hệ Quản trị Cơ sở dữ liệu quan hệ sử dụng Ngôn ngữ truy vấn có cấu trúc (SQL).

MySQL được sử dụng cho việc bổ trợ PHP, Perl, và nhiều ngôn ngữ khác, nó làm nơi lưu trữ những thông tin trên các trang web viết bằng PHP hay Perl,…

2.2.1.2. Lịch sử ra đời

Công ty Thuy Điển MySQL AB phát triển MySQL vào năm 1994. Công ty công nghệ Mỹ Sun Microsystem sau đó giữ quyền sở hữu MySQL sau khi mua lại MySQL vào năm 2008. Năm 2010, gã khổng lồ Oracle mua Sun Microsystems và MySQL thuộc quyền sở hữu của Oracle từ đó.

2.2.1.3. Cài đặt My SQL

Để cài đặt MySQL trên nền Windows bạn theo các bước sau:

- Trước tiên bạn cần download chương trình về từ website của MySQL
- Chạy tập tin Setup.exe
- Sau khi cài đặt thành công, bạn kiểm tra trong Windows Services xuất hiện dịch vụ mySQL hay không?. Để sử dụng được MySQL thì trạng thái của dịch vụ này phải ở chế độ Started.

2.2.1.4. Các tiện ích trong My SQL

MySQL cho phép dữ liệu được lưu trữ và truy cập trên nhiều công cụ lưu trữ, bao gồm InnoDB, CSV và NDB. MySQL cũng có khả năng sao chép dữ liệu và phân vùng bảng để có hiệu suất và độ bền tốt hơn. Người dùng MySQL không bắt buộc phải học các lệnh mới; họ có thể truy cập dữ liệu của mình bằng các lệnh SQL tiêu chuẩn.

MySQL được viết bằng C và C++ và có thể truy cập và có sẵn trên hơn 20 nền tảng, bao gồm Mac, Windows, Linux và Unix. RDBMS hỗ trợ cơ sở dữ liệu lớn với hàng triệu bản ghi và hỗ trợ nhiều loại dữ liệu bao gồm các số nguyên có chữ ký hoặc không dấu có độ dài 1, 2, 3, 4 và 8 byte(s); FLOAT; DOUBLE; CHAR; VARCHAR; BINARY; VARBINARY; TEXT; BLOB; DATE; TIME; DATETIME; TIMESTAMP; YEAR; SET; ENUM; và các kiểu OpenGIS. Các loại chuỗi có độ dài cố định và biến đổi cũng được hỗ trợ.

Để bảo mật, MySQL sử dụng một đặc quyền truy cập và hệ thống mật khẩu được mã hóa cho phép xác minh dựa trên máy chủ. Các máy khách MySQL có thể kết nối với Máy chủ MySQL bằng một số giao thức, bao gồm cả giao thức TCP/IP trên bất kỳ nền tảng nào. MySQL cũng hỗ trợ một số chương trình máy khách và tiện ích, chương trình dòng lệnh và công cụ quản trị như MySQL Workbench.

Các nhánh của MySQL bao gồm:

- **Drijection**: một hệ thống quản lý cơ sở dữ liệu nguồn mở nhẹ được phát triển dựa trên MySQL 6.0.
- **MariaDB**: một sự thay thế phổ biến do cộng đồng phát triển cho MySQL sử dụng các lệnh và API của MySQL.
- **Percona Server với XtraDB**: một phiên bản nâng cao của MySQL được biết đến với khả năng mở rộng theo chiều ngang.

2.2.2. Một số điểm khác biệt giữa hệ quản trị CƠ SỞ DỮ LIỆU SQL Server và My SQL

Tiêu chí	**SQL**	**MYSQL**
Định nghĩa	SQL là một ngôn ngữ truy vấn có cấu trúc (Structured Query Language). Nó rất hữu ích để quản lý cơ sở dữ liệu quan hệ.	MySQL là một RDBMS để lưu trữ, truy xuất, sửa đổi và quản trị cơ sở dữ liệu bằng cách sử dụng MySQL.
Kiểu	SQL là một ngôn ngữ truy vấn.	MySQL là phần mềm cơ sở dữ liệu. Nó đã sử dụng ngôn ngữ "SQL" để truy vấn cơ sở dữ liệu.
Hỗ trợ kết nối	SQL không cung cấp trình kết nối.	MySQL cung cấp một công cụ tích hợp được gọi là 'MySQL workbench' để thiết kế và phát triển cơ sở dữ liệu.
Mục đích	Để truy vấn và vận hành hệ thống cơ sở dữ liệu.	Cho phép xử lý dữ liệu, lưu trữ, sửa đổi, xóa theo định dạng bảng.
Sử dụng	Mã và lệnh SQL được sử dụng trong các hệ thống DBMS và RDMS khác nhau bao gồm MYSQL.	MYSQL được sử dụng làm cơ sở dữ liệu RDBMS.

Cập nhật	Ngôn ngữ là cố định, và lệnh vẫn giữ nguyên.	Nhận cập nhật thường xuyên.

2.3. CÁC KHÁI NIỆM NGÔN NGỮ T-SQL

2.3.1. Giới thiệu T-SQL

Transact-SQL là ngôn ngữ SQL mở rộng dựa trên SQL chuẩn của ISO (International Organization for Standardization) và ANSI (American National Standards Institute) được sử dụng trong SQL Server khác với PL-SQL (Procedural Language/Structured Query Language) dùng trong Oracle.

2.3.1.1. Biến trong T-SQL

Biến trong T-SQL cũng có chức năng tương tự như trong các ngôn ngữ lập trình khác nghĩa là cần khai báo trước loại dữ liệu trước khi sử dụng. Biến được bắt đầu bằng dấu @ (Đối với các biến toàn cục - global variable - thì có hai dấu @@)

Ví dụ: Ví dụ dưới đây khai báo một biến có tên @numberOfCustomers thông qua từ khóa declare. Biến này lưu số khách hàng đếm được thông qua hàm count. Sau đó in ra giá trị của biến.

declare @numberOfCustomers int select @numberOfCustomers = count(*) from Customers print @numberOfCustomers

2.3.1.2. Kiểu dữ liệu trong T-SQL

Kiểu dữ liệu	Ghi chú
Char(n)	Kiểu chuỗi với độ dài cố định
Nchar(n)	Kiểu chuỗi với độ dài cố định hỗ trợ UNICODE
Varchar(n)	Kiểu chuỗi với độ dài chính xác
Nvarchar(n)	Kiểu chuỗi với độ dài chính xác hỗ trợ UNICODE
Int	Số nguyên có giá trị từ -2^{31} đến $2^{31} - 1$
Tinyint	Số nguyên có giá tr ị từ 0 đến 255.
Smallint	Số nguyên có giá trị từ -2^{15} đến $2^{15} - 1$

Bigint	Số nguyên có giá trị từ -263 đến 263-1
Numeric	Kiểu số với độ chính xác cố định.
Decimal	Tương tự kiểu Numeric
Float	Số thực có giá trị từ 1.79E+308 đến 1.79E+308
Real	Số thực có giá trị từ 3.40E + 38 đến 3.40E + 38
Money	Kiểu tiền tệ
Bit	Kiểu bit (có giá trị 0 hoặc 1)
Datetime	Kiểu ngày giờ (chính xác đến ph ần trăm của giây)
Smalldatetime	Kiểu ngày gi ờ (chính xác đến phút)
Binary	Dữ liệu nhị phân với độ dài c ố định (tối đa 8000 bytes)
Varbinary	Dữ liệu nhị phân với độ dài chính xác (t ối đa 8000 bytes)
Image	Dữ liệu nhị phân với độ dài chính xác (tối đa 2,147,483,647 bytes)
Text	Dữ liệu kiểu chuỗi với độ dài l ớn (tối đa 2,147,483,647 ký tự)
Ntext	Dữ liệu kiểu chuỗi với độ dài lớn và h ỗ trợ UNICODE (tối đa 1,073,741,823 ký tự)

2.3.1.3. Chú thích trong T-SQL

T-SQL dùng kí hiệu -- để chú thích cho một dòng đơn và kí hiệu /*…*/ để chú thích cho một nhóm dòng

Ví dụ:

```
/* Minh họa chú thích

Chú thích cho một dòng đơn và một nhóm các dòng*/

DECLARE @MyNumber int   -- khai báo biến

SET @MyNumber = 4 - 2 + 27

-- kết quả là 29

SELECT @MyNumber
```

2.3.1.4. Cấu trúc điều khiển

IF…ELSE

Cú pháp:

```
  IF (Biểu_thức)
{ Câu lệnh hoặc nhóm lệnh được thực thi }
ELSE
    { Câu lệnh hoặc nhóm lệnh được thực thi }
```

Lưu ý: Trong SQL nếu ta muốn thực thi 1 nhóm lệnh thì nhóm lệnh đó phải nằm trong từ khóa BEGIN…END

Ví dụ:

```
 DECLARE @CharGender Char(1),
  @Gender  Varchar(20);
SET @CharGender = 'F';
IF (@CharGender<>'F')
  SET @Gender='Male'
ELSE
  SET @Gender='Female'

SELECT @Gender AS [Giới Tính]
```

CASE…WHEN

Khi chúng ta sử dụng nhiều If..else thì có thể dung Case..When để thay thế.

Cú pháp:

```
CASE Biểu thức
WHEN Giá trị 1 THEN kết quả
WHEN Giá trị 2 THEN kết quả

WHEN Giá trị n THEN kết quả
END
```

Ví dụ:

```
DECLARE @CharGender Char(1),
 @Gender  Varchar(20);
```

```
SET @CharGender = 'F';
SET @Gender =
 CASE @CharGender
  WHEN 'm' THEN 'Male'
  WHEN 'M' THEN 'Male'
  WHEN 'f' THEN 'Female'
  WHEN 'F' THEN 'Female'
 END;
SELECT @Gender AS [Giới Tính]
```

Khối lệnh : BEGIN…END

Cú pháp:

```
   BEGIN
    { Câu lệnh hoặc nhóm lệnh được thực thi}
    END
```

Vòng lặp : WHILE

Cú pháp:

```
  WHILE Biểu thức       {
   Câu lệnh hoặc nhóm lệnh được thực thi
  }
```

Ví dụ:

```
DECLARE @Number As int SET @Number = 1
WHILE @Number < 5
   BEGIN
 SELECT @Number AS Number
 SET @Number = @Number + 1
   END
```

2.3.2. Ngôn ngữ định nghĩa dữ liệu (DDL)

- Ngôn ngữ định nghĩa dữ liệu (DDL – Data Definition Language) gồm các lệnh cho phép tạo ra, thay đổi hoặc xóa các bảng
- Chúng ta cũng có thể định nghĩa các khoá (key), chỉ mục (index), chỉ định các liên kết giữa các bảng và thiết lập các quan hệ ràng buộc giữa các bảng trong CSDL

2.3.2.1. Câu lệnh CREATE TABLE

- Lệnh **CREATE TABLE** trong SQL được sử dụng để tạo một bảng mới.

- **Cú pháp**

```
CREATE TABLE ten_bang(
  kieu_du_lieu cot1,
  kieu_du_lieu cot2,
  kieu_du_lieu cot3,
  .....
  kieu_du_lieu cotN,
  PRIMARY KEY( mot hoac nhieu cot )
);
```

2.3.2.2. Câu lệnh ALTER TABLE

- Lệnh **ALTER TABLE** trong SQL được sử dụng để thêm, xóa hoặc sửa đổi các cột trong một bảng hiện có. Bạn cũng nên sử dụng lệnh ALTER TABLE để thêm và bỏ các ràng buộc khác nhau trên một bảng hiện có.

- Cú pháp:

Để thêm một cột cho bảng ta sử dụng cú pháp sau:

```
ALTER TABLE  ten_bang
ADD  ten_cot  Kieudulieu ;
```

2.3.2.3. Câu lệnh DROP TABLE

Lệnh **DROP TABLE** trong SQL được sử dụng để xóa một bảng và tất cả dữ liệu, chỉ mục (index), trigger, ràng buộc và quyền được trao cho bảng đó.

GHI CHÚ: Bạn phải thật cẩn thận trong khi sử dụng lệnh này, bởi vì một khi một bảng bị xóa thì tất cả thông tin có sẵn trong bảng đó cũng sẽ bị xóa vĩnh viễn.

Cú pháp

Cú pháp cơ bản của lệnh DROP TABLE trong SQL như sau:

DROP TABLE ten_bang;

2.3.3. Ngôn ngữ thao tác dữ liệu (DML) và điều khiển dữ liệu (DCL)

2.3.3.1. Câu lệnh SELECT, INSERT, UPDATE, DELETE

TRUY VẤN DỮ LIỆU

Khối câu lệnh phổ dụng: SELECT - FROM – WHERE. Ta có thể sử dụng theo cú pháp chung như sau:

SELECT [*| DISTINCT] <Danh sách các cột [AS <Bí danh>]>

FROM <Danh sách Tên bảng/Tên View>

[WHERE <Biểu thức điều kiện>]

[GROUP BY <Danh sách cột>]

[HAVING <Điều kiện>]

[ORDER BY <Tên cột/ Số thứ tự cột/Biểu thức> [ASC/DESC]]

Lệnh Insert

- Ý nghĩa: Dùng để chèn một hàng hoặc một số hàng cho bảng.

- Cú pháp:

+ INSERT INTO <Tên bảng> (Danh sách các cột)

VALUES (Danh sách các giá trị)

hoặc

+ INSERT INTO <Tên bảng> (Danh sách các cột) (Các câu hỏi con);

Lệnh Update

- Ý nghĩa: Dùng để sửa đổi dữ liệu.

- Cú pháp:

UPDATE <Tên bảng>

SET <Tên_cột_1=Biểu_thức_1, Tên_cột_2=Biểu_thức_2,… >

[WHERE <điều kiện>]

Lệnh Delete

- Ý nghĩa: Xoá một số hàng trong bảng.

- Cú pháp: DELETE FROM <Tên bảng> WHERE <Điều kiện>

2.3.3.2. Câu lệnh GRANT, REVOKE, DENY

Trao quyền GRANT

- Ý nghĩa: Dùng để trao quyền cho một acount nào đó.

- Cú pháp: GRANT <Quyền> ON <Tên bảng/ Tên View> TO <user> [WITH GRANT OPITION] Các quyền có thể trao là: All, Select, update, delete, insert, index, alter, read, write,… User có thể là: Public, tên một user cụ thể,… - Chú ý: Nếu được trao quyền với chỉ định WITH GRANT OPITION thì anh ta có thể trao lại quyền ấy cho người khác.

Ví dụ : Trao quyền Select cho acount Lannt GRANT Select ON SINHVIEN TO Lannt WITH GRANT OPITION

Thu hồi quyền REVOTE

- Ý nghĩa: Dùng để thu hồi quyền của một acount nào đó. - Cú pháp: REVOTE <Quyền> ON <Tên bảng/ Tên View> FROM <user> Ví dụ 2.8. Thu hồi quyền Select của acount Lannt REVOTE Select ON SINHVIEN FROM Lannt;

2.3.4. Thực thi Câu lệnh T-SQL

2.3.4.1. Câu lệnh đơn

Một câu lệnh SQL được phân ra thành các thành phần cú pháp như trên bởi một parser, sau đó SQL Optimizer (một bộ phận quan trọng của SQL Server) sẽ phân tích và tìm cách thực thi (Execute Plan) tối ưu nhất ví dụ như cách nào nhanh và tốn ít tài nguyên của máy nhất… và sau đó SQL Server Engine sẽ thực thi và trả về kết quả.

2.3.4.2. Batches (Lô)

Khi thực thi một nhóm lệnh SQL Server sẽ phân tích và tìm biện pháp tối ưu cho các câu lệnh như một câu lệnh đơn và chứa execution plan đã được biên dịch

(compiled) trong bộ nhớ sau đó nếu nhóm lệnh trên được gọi lại lần nữa thì SQL Server không cần biên dịch mà có thể thực thi ngay điều này giúp cho một batch chạy nhanh hơn.

Chương 2: NGÔN NGỮ ĐỊNH NGHĨA DỮ LIỆU (DDL)

1. Mục tiêu: Sau khi học xong người học có khả năng:

- Sử dụng ngôn ngữ T-SQL tạo cơ sở dữ liệu
- Thay đổi cấu trúc bảng
- Tạo các ràng buộc ở mức khia báo cho cơ sở dữ liệu

2. Nội dung chương:

2.1. TẠO VÀ THAY ĐỔI CẤU TRÚC BẢNG

2.1.1. Cơ sở dữ liệu của SQL Server – My SQL

2.1.1.1. Khái niệm về cơ sở dữ liệu

Cơ sở dữ liệu (CSDL) là một hệ thống các thông tin có cấu trúc được lưu trữ trên các thiết bị lưu trữ thông tin thứ cấp (như băng từ, đĩa từ ...) để có thể thỏa mãn yêu cầu khai thác thông tin đồng thời của nhiều người sử dụng hay nhiều chương trình ứng dụng với nhiều mục đích khác nhau.

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

- Một database bao gồm tối thiểu hai file
    - .mdf: lưu trữ các đối tượng trong database như table, view, …
        - Có thể bổ sung thêm các tập tin lưu trữ khác
        - Tổ chức tốt các tập tin lưu trữ giúp tăng tốc độ xử lý
    - .ldf: lưu trữ quá trình cập nhật/thay đổi dữ liệu
        - Hỗ trợ phục hồi dữ liệu
        - Hỗ trợ backup/restore dữ liệu
- Các thông số về kích thước
    - Initial size
    - File growth
    - Maximum file size

2.1.1.3. Tạo mới cơ sở dữ liệu

```
CREATE DATABASE Tên_ CSDL
```

2.1.1.4. Xóa cơ sở dữ liệu đã có

```
DROP DATABASE Tên_ CSDL
```

2.1.2.Bảng dữ liệu (Table)

2.1.2.1. Khái niệm về bảng

- Bảng dùng để lưu trữ các thông tin của một đối tượng trong thực tế
    - Gồm có các dòng và các cột
    - Bảng trong CSDL thường có khoá chính (primary key)
    - Các bảng thường liên hệ với nhau bằng các mối quan hệ
    - Bảng được tạo trong các Schema (mặc định là schema dbo)
- Bảng có thể có các ràng buộc (constraint), trigger

2.1.2.2. Các thuộc tính của bảng

- Tên bảng
- Tên cột
- Kiểu dữ liệu
    - Độ dài dữ liệu
    - Số ký số lưu trữ
    - Số số lẻ lưu trữ
- Thuộc tính trên cột
    - Allow null
    - Identity
    - Default value

2.1.2.3. Tạo cấu trúc bảng dữ liệu

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

2.1.2.4. Xóa bảng đã tồn tại

```
DROP TABLE ten_bang;
```

2.1.3. Thay đổi cấu trúc bảng

2.1.3.1. Thêm cột

```
ALTER TABLE ten_bang ADD ten_cot kieu_du_lieu;
```

2.1.3.2. Hủy bỏ cột hiện có

```
ALTER TABLE ten_bang DROP COLUMN ten_cot;
```

2.1.3.3. Sửa đổi kiểu dữ liệu của cột

ALTER TABLE ten_bang MODIFY COLUMN ten_cot kieu_du_lieu;

2.1.3.4. Đổi tên cột, tên bảng dữ liệu.

- Đổi tên cột trong bảng

Cú pháp

sp_rename ‘ten_bang.ten_cot_cu’, ‘ten_cot_moi’, “COLUMN’;

- Đổi tên bảng

Cú pháp

sp_rename ‘ten_bang_cu’, ‘ten_bang_moi’;

2.1.3.5. So sánh sự khác biệt trên hê quản trị CƠ SỞ DỮ LIỆU SQL – My SQL.

2.2. TÍNH TOÀN VẸN DỮ LIỆU TRONG CƠ SỞ DỮ LIỆU

2.2.1. Giới thiệu về ràng buộc toàn vẹn

Ràng buộc toàn vẹn (*Integrity Constraint / Rule* viết tắt là: RBTV) và kiểm tra sự vi phạm ràng buộc toàn vẹn là hai trong những vấn đề rất quan trọng trong quá trình phân tích, thiết kế và khai thác CSDL. Trong quá trình phân tích - thiết kế cơ sở dữ liệu, nếu không quan tâm đúng mức đến những vấn đề trên, thì có thể dẫn đến những hậu quả rất nghiêm trọng về tính an toàn và toàn vẹn dữ liệu, đặc biệt trong những CSDL tương đối lớn.

2.2.2 Các loại quy tắc kiểm tra tính toàn vẹn dữ liệu

2.2.2.1. Kiểm tra duy nhất dữ liệu

Ràng buộc dữ liệu duy nhất (Unique constraint) không cho phép dữ liệu bị trùng lặp. Ràng buộc này được sử dụng để thiết lập cho những cột mà dữ liệu nếu được nhập thì không được trùng lặp. Tuy nhiên đối với những cột này, chúng ta có thể không chỉ định dữ liệu (null). Đây chính là điểm khác biệt giữa primary key và unique.

Cú pháp:

alter table tên_bảng add constraint tên_constraint unique(tên_cột)

2.2.2.2. Kiểm tra tồn tại dữ liệu

Loại ràng buộc toàn vẹn này cho phép kiểm tra tính tồn tại của dữ liệu (khóa ngoại), 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 dữ liệu khác. Bạn có thể sử dụng thành phần **FOREIGN KEY** trong câu lệnh **CREATE TABLE** để thực hiện việc kiểm tra tính tồn tại của dữ liệu.

2.2.2.3. Kiểm tra miền giá trị

Ràng buộc check (Check constraint) qui định dữ liệu được thêm mới hoặc thay đổi phải thoả điều kiện được chỉ định trong lúc thiết lập.

Cú pháp:

alter table tên_bảng add constraint tên_constraint check(điều_kiện)

Ràng buộc dữ liệu mặc định (Default constraint) sẽ chỉ định dữ liệu mặc định cho những cột khi thêm mới bị bỏ qua

Cú pháp:

alter table tên_bảng add constraint tên_constraint default giá_trị

2.2.2.4. Thêm ràng buộc mới vào trong bảng

ALTER TABLE <Ten Bang>

ADD CONSTRAINT <Ten Constraint> <Loai Constraint> <Tham So>

2.2.2.5. Hủy bỏ ràng buộc đã có trong bảng

ALTER TABLE <Ten Bang>

DROP CONSTRAINT <Ten Constraint>

2.2.2.6. Tắt/ Bật quy tắc kiểm tra toàn vẹn dữ liệu

- Tắt quy tắc kiểm tra toàn vẹn dữ liệu

  ALTER TABLE <Ten Bang>

  NOCHECK CONSTRAINT ALL <Ten Constraint>
- Bật quy tắc kiểm tra toàn vẹn dữ liệu

  ALTER TABLE <Ten Bang>

  CHECK CONSTRAINT ALL <Ten Bang>

2.2.3. Mô hình quan hệ dữ liệu

2.2.3.1. Khái niệm về mô hình quan hệ dữ liệu

Mô hình CSDL quan hệ lần đầu tiên được E.F.Codd và tiếp sau đó được công ty IBM giới thiệu vào năm 1970. Ngày nay, hầu hết các tổ chức đã áp dụng CSDL quan hệ để quản lý dữ liệu trong đơn vị mình.

Mô hình cơ sở dữ liệu quan hệ

Cấu trúc dữ liệu: dữ liệu được tổ chức dưới dạng quan hệ hay còn gọi là bảng.

Thao tác dữ liệu: sử dụng những phép toán mạnh (bằng ngôn ngữ SQL).

2.2.3.2. Tạo mới mô hình quan hệ dữ liệu

2.2.3.3. Các chức năng trong mô hình quan hệ dữ liệu

- Thiết lập mối quan hệ khoá ngoại (FOREIGN KEY)
- Chỉnh sửa cấu trúc bảng
- Chỉnh sửa thuộc tính bảng
- Tạo bảng mới

2.2.4. Index

2.2.4.1. Index là gì?

Chỉ mục (INDEX) trong SQL là bảng tra cứu đặc biệt mà công cụ tìm kiếm cơ sở dữ liệu có thể sử dụng để tăng nhanh thời gian và hiệu suất truy xuất dữ liệu.

Hiểu đơn giản, một chỉ mục là một con trỏ chỉ tới từng giá trị xuất hiện trong bảng/cột được đánh chỉ mục. Chỉ mục trong Database có ý nghĩa tương tự như các mục trong xuất hiện trong Mục lục của một cuốn sách.

INDEX giúp tăng tốc các truy vấn SELECT chứa các mệnh đề WHERE hoặc ORDER, nhưng nó làm chậm việc dữ liệu nhập vào với các lệnh UPDATE và INSERT. Các chỉ mục có thể được tạo hoặc xóa mà không ảnh hưởng tới dữ liệu.

2.2.4.2. Các kiểu Index

- Single-Column Index
- Unique Index
- Composite Index
- Implicit Index

2.2.4. 3. Cách tạo Index

Để tạo một chỉ mục ta sử dụng **lệnh CREATE INDEX**, có thể đặt tên cho chỉ mục, xác định bảng, các cột muốn lập chỉ mục và xác định chỉ mục là theo thứ tự tăng dần hoặc giảm dần.

Lệnh CREATE INDEX

Cú pháp cơ bản của lệnh **CREATE INDEX** trong SQL như sau:

```
CREATE INDEX ten_index ON ten_bang;
```

Chỉ mục SINGLE-COLUMN

Single-Column Index được tạo cho duy nhất 1 cột trong bảng. Cú pháp cơ bản như sau:

```
CREATE INDEX ten_index
ON ten_bang (ten_cot);
```

Chỉ mục UNIQUE

Unique Index là chỉ mục duy nhất, được sử dụng để tăng hiệu suất và đảm bảo tính toàn vẹn dữ liệu. Một chỉ mục duy nhất không cho phép chèn bất kỳ giá trị trùng lặp nào được chèn vào bảng. Cú pháp cơ bản như sau.

```
CREATE UNIQUE INDEX ten_index
ON ten_bang (ten_cot);
```

Chỉ mục COMPOSITE

Composite Index là chỉ mục kết hợp dành cho hai hoặc nhiều cột trong một bảng. Cú pháp cơ bản của nó như sau:

```
CREATE INDEX ten_index
ON ten_bang (cot1, cot2);
```

Lưu ý:

- Việc tạo **Single-Column Index** hay **Composite Index** tùy thuộc vào tần suất bạn sử dụng mệnh đề WHERE của truy vấn dưới dạng điều kiện bộ lọc.
- Nếu chỉ có một cột được sử dụng, thì lựa chọn tốt nhất là **Single-column Index**. Nếu có hai hoặc nhiều cột được sử dụng thường xuyên trong mệnh đề WHERE như là các bộ lọc thì dạng chỉ mục **Composite Index** là lựa chọn tối ưu hơn.

IMPLICIT INDEX

Implicit Index (Index ngầm định) là chỉ mục mà được tạo tự động bởi Database Server khi một bảng được tạo. Các Index ngầm định được tạo tự động cho các ràng buộc Primary key và các ràng buộc Unique.

2.2.4.4. Sửa và xóa Index

Lệnh DROP INDEX

Khi không cần sử dụng INDEX bạn có thể DROP theo cú pháp sau:
DROP INDEX ten_index;

Chương 3: NGÔN NGỮ THAO TÁC DỮ LIỆU (DML)

3.1. TRUY VẤN ĐƠN GIẢN

3.1.1. Hiệu chỉnh dữ liệu:

3.1.1.1. 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.

3.1.1.2. 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. Cẩn 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_xóadl] Trong đó: 9 Tên bảng: tên bảng có các dòng dữ liệu muốn hủy bỏ. 9 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. 9 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 10/2012. Chúng ta sử dụng lệnh DELETE như sau:

```
DELETE DONDH WHERE CONVERT(CHAR(7),
 NGAYDH, 21)= ‘2012-10’
```

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 10/2012 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)=’2012-10’
```

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 10/2012. Sau đó tiếp tục cho thực hiện lại truy vấn hủy bỏ các đơn đặt hàng trong tháng 10/2012 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 10/2012 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.

3.1.1.3. Lệnh UPDATE SET

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 cẩn 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. 9 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
```

Ví dụ: *Để giảm giá 10% cho tất cả các phiếu bán hàng trong ngày cuối cùng của tháng 10/2012. 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="2012-01-31"
```

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. Trước khi thực hiện các lệnh này các bạn nên tạo ra bảng dữ liệu 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.

3.1.2. Câu lệnh truy vấn đơn giản

3.1.2.1. 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**.

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.

*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

3.1.2.2. 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 lắp 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 bây 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.

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.*

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
```

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

*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 H, NHACC NCC
WHERE DH.MANHACC=NCC.MANHACC
ORDER BY NCC.MANHACC
```

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**.

3.1.2.3. 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
```

3.1.3. Hàm thường dùng:

3.1.3.1. 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.

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(PHANTRAM AS
VARCHAR(3))+"%" FROM VATTU
```

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
```

3.1.3.2. 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
yy	Năm	1900-9999
qq	Quí	1-4
mm	Tháng	1-12
dy	Ngày trong năm	1-366
dd	Ngày trong tháng	1-31
wk	Tuần	1-53
dw	Ngày trong tuần	1-7(Chủ nhật-thứ bảy)
hh	Giờ trong ngày	0-23
mi	Phút trong giờ	0-59
ss	Giây trong phút	0-59
ms	Phần trăm mili giây	0-999

.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) Ỉ 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
```

.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 lệnh 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
```

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
```

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)
```

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 2012. 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)=2012
 ORDER BY THANG
```

3.1.3.3. 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.

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)
```

.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()
```

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)
```

3.1.3.4. Hàm xử lý chuỗi

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

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_ktự) → Chuỗi_con

RIGHT (Chuỗi nguồn, Số_ktự) → Chuỗi_con

SUBSTRING (Chuỗi nguồn,Vị_trí,Số_ktự) → 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.

3. 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 hoc"

. SELECT LEN('Trung tam tin hoc')

3.2. TRUY VẤN NÂNG CAO

3.2.1. Các mệnh đề trong truy vấn nâng cao

3.2.1.1. 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 t 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_nhómdl
            ORDER BY Tên_cột [DESC] [, ...] ]
```

Trong đó:

- Hàm thống kê: là tên của các hàm thống kê và các hàm 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
```

3.2.1.2. 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_nhómdl

        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.

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
```

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.

3.2.1.3. 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)
```

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
```

3.2.2. Các mệnh đề trong truy vấn nâng cao:

3.2.2. 1. Mệnh đề liên kết dữ liệu từ nhiều 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 mối 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.

*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
```

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.

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
```

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!

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 qui 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.

3.2.2. 2. 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.

*Để tính ra đồng thời tổng số lượng nhập, tổng số lượng xuất của các vật tư trong tháng 10/2012. 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)="2012-10"
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)="2012-10"
GROUP BY MAVTU
```

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 10/2012.*

```
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)= ‘2012-10’
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)=’2012-10’
GROUP BY MAVTU ORDER BY MAVTU, LOAI
```

3.2.2. 3. Truy vấn con (SubQuery)

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 10/2012. Chúng ta có thể thực hiện câu truy vấn như sau:*

```
SELECT  MANHACC
FROM DONDH
WHERE CONVERT(CHAR(7), NGAYDH,21)=’2012-10’
```

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")
```

Ví dụ: *Đâu đảm bảo rằng trong tháng 10/2012 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 10/2012, 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)="2012-10")
```

Các bạn cũng có thể sử dụng từ khóa **EXISTS** hoặc mệnh đề **JOIN** để thực hiện việc so sánh các dòng dữ liệu rong 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.

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
```

Hoặc

```
SELECT TENNHACC, DIENTHOAI
```

```
FROM NHACC NCC INNER JOIN DONDH DH ON
DH.MANHACC=NCC.MANHACC

WHERE CONVERT(CHAR(7), NGAYDH, 21)="2012-10"
```

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 qui tắc mà các bạn nên nhớ là:

"**IN s**ẽ 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)
```

*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 tòan!*

```
SELECT TENNHACC, DIENTHOAI
FROM NHACC
WHERE MANHACC <> ANY (SELECT DISTINCT MANHACC
                      FROM DONDH)
```

Việc chúng tơi trình bày thêm cácc 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 c nhn của chng tơi thì cc bạn chỉ cần nhớ cc tốn

tử **IN** hoặc **NOT IN** v sử dụng cho đng trong cc trường hợp so snh cần thiết đối với dạng truy vấn con trả về danh sách các giá trị.

3.3. BẢNG ẢO (VIRTUAL TABLE - VIEW)

3.3.1. 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).

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.

3.3.2. Tạo mới bảng ảo

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 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_sldat)
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_sldat
FROM CTDONDH CTDH INNER JOIN VATTU VT ON
CTDH.MAVTU=VT.MAVTU GROUP BY CTDH.Mavtu, Tenvtu
GO
```

3.3.3. Xem và cập nhật 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 lệnh như sau:

SELECT *|Danh_sách_cột FROM tên_bảng_ảo|tên_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"
```

3.3.4 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. Sau đó chọn nút Drop All để đồng ý hủy bỏ.

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.*

```
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'.

Chương4: THỦ TỤC NỘI TẠI (STORED PROCEDURE) VÀ TRIGGER

4.1. BIẾN CON TRỎ (CURSOR)

4.1.1. Khai báo và gán giá trị cho biến:

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.

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.

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.**

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 @Tongsldat INT, @Hotenncc CHAR(50) DECLARE @Ngayxh DATETIME

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 25/03/2012 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='2012-03-25'
```

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 @Tongsldat INT
SELECT   @Tongsldat = SUM(SLDAT) FROM CTDONDH
```

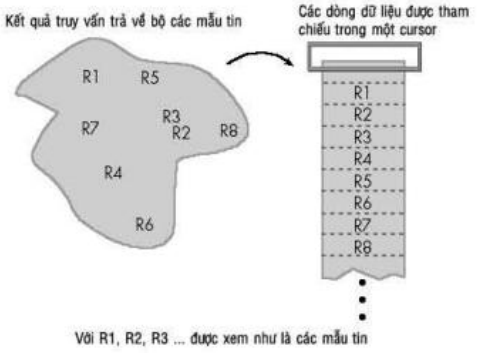
Ví dụ:*Để tính ra đồng thời giá trị số lượng đặt hàng thấp nhất và cao nhất. Chúng ta chỉ cần sử dụng duy nhất một lệnh SELECT như sau:*

```
DECLARE @MinSldat INT, @MaxSldat INT
```

SELECT @MinSldat=MIN(SLDAT), @MaXSldat=MAX(SLDAT) FROM CTDONDH

4.1.2. Khái niệm về biến con trỏ

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.

4.1.3. Các bước sử dụng biến con trỏ:

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 thứ tự 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

4.1.3.1. Định nghĩa biến con trỏ

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) 9 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 cusor 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.

4.1.3.2 Đọc và xử lý dữ liệu trong con trỏ

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.

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
```

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(SLNHAP*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 dùng để chỉ định việc cập nhật dữ liệu trên dòng dữ liệu hiện hành của cursor.

4.1.3.3. Đóng con trỏ

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. Các bạn thấy rằng tiện ích duy nhất khi làm việc với cursor là chúng cho phép các bạn có thể duyệt từng tự trên các dòng dữ, 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.

Các bạn thấy rằng tiện ích duy nhất khi làm việc với cursor là chúng cho phép các bạn có thể duyệt từng tự trên các dòng dữ liệu ,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.

4.2. THỦ TỤC NỘI TẠI (STORED PROCEDURE)

4.2.1. Khái niệm về thủ tục nội tại

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 nó, 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.

4.2.2 Các hành động cơ bản với thủ tục nội tại

4.2.2.1. Tạo mới thủ tục nội tại

Tạo mới thủ tục nội tại bằng lệnh **CREATE PROCEDURE**.

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 10/2012. Chúng ta thực hiện lệnh CREATE PROCEDURE như sau:*

```
CREATE PROC spud_MaxSLVattu_201210
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.

4.2.2.2. Gọi thực hiện thủ tục nội tại

Chúng ta chỉ có thể **gọi thực hiện thủ tục nội tại** bằng lệnh **EXECUTE**. Cú pháp bên dưới mô tả việc gọi thực hiện một thủ tục nội tại đơn giản – không có ham số gởi vào hoặc đón nhận giá trị trả về gì 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_201210 đã được tạo ra trong thí dụ trước đó. Chúng ta thực hiện lệnh EXECUTE như sau:*

EXECUTE spud_MaxSLVattu_201210

4.2.2.3. 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 khôngcòn 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

4.2.2.4. 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.**

dùng lệnh **ALTER PROCEDURE** để thay đổi nội dung.

Cú pháp:

ALTER PROC[EDURE] Tên_thủ_tục

AS

[DECLARE Biến_cục_bộ]

Các_lệnh

Ví dụ:*Cầ n bổ sung thêm phần kiểm tra dữ liệu đã có bán hàng trong tháng 10/2012 trong thủ tục nội tại spud_MaxSLVattu_201210 đã được tạo ra ở thí dụ trước đó. Chúng ta thực hiện lệnh ALTER PROCEDURE như sau:*

```
ALTER PROC spud_MaxSLVattu_201210 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)="2012-10")
   BEGIN
   PRINT "Tháng 10/2012 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)="2012-10"
   AND SLXUAT = (SELECT MAX(SLXUAT)
   FROM CTPXUAT INNER JOIN PXUAT ON
   PXUAT.SOPX=CTPXUAT.SOPX WHERE
   CONVERT(CHAR(7),NGAYXUAT,21)="2012-10")
   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 10/2012. 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**.

4.2.2.5. 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.

4.2.2.6. 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.

4.2.2.7. 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

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.

4.2.3. Một số vấn đề khác trong thủ tục nội tại

4.2.3.1. Mã hóa nội dung thủ tục nội tại

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),
@sChuoira 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 @sChuoira = 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.

4.2.3.2. Biên dịch thủ tục nội tại

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 tạo 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***

4.2.4.3. Sử dụng lệnh RETURN trong thủ tục

Thông thường chng ta sẽ sử dụng lệnh **RETURN** dùng để **thóat ra khỏi thủ tục** trong các trường hợp dữ liệu không hợp lệ. Ngòai ra lệnh **RETURN** cũng cho phép chúng ta trả về **một số nguyên** tại nơi đã gọi thực hiện thủ tục. Khi đĩ thủ tục sẽ trả về giá trị là một số nguyên, khái niệm này giúp cho **thủ tục trở thành một hm**. (chỉ trả về giá trị)
Mặc định lệnh **RETURN** không có giá trị chỉ định thì thủ tục sẽ **trả về giá trị là không** . 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 gi trị **khác không** dúng để chỉ định **các lỗi xảy ra** khi thực hiện thủ tục.

Cú pháp:

RETURN [Số_nguyên]

Khi bên trong một thủ tục nội tại **chĩ sử dụng lệnh RETURN** thì thông thường chúng ta sẽ phải **sử dụng một biến cục bộ** để đn nhận **gi trị trả về** của thủ tục đĩ. Khi đó cú pháp của lệnh gọi thực hiện thủ tục phải được bổ sung như sau:

EXEC @Biến=Tn_thủ_tục [Các_tham_số]

Trong đó:

- **Biến**: biến cục bộ được định nghĩa để để n nhận gi 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 hng của một vật tư đối với một nhà cung cấp chỉ định, chng ta cần phải kiểm tra xem gi trị của mã vật tư và mã nhà cung cấp mà người dùng truyền vào thủ tục có đng hay không? Qui định rằng thủ tục trả sẽ về 1à khi mã vậttư 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 "M vật tư khơng đng"
ELSE
IF @nKetqua=2
PRINT "M nh cung cấp khơng đng"
ELSE
PRINT "Tổng số lượng đặt là: " +
CAST(@nTongSLdat AS VARCHAR(10))
GO
```

Kết quả trả về

Tổng số lượng đặt là: 20

Nhận xét thấy rằng trong thí dụ này chúng ta định nghĩa ra một biến cục bộ tn **@nKetqua** cĩ kiểu dữ liệu là số nguyẹn (INT) dùng để đĩn 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.

4.2.4.4. Tham số kiểu con trỏ bên trong thủ tục

Trong các trường hợp cần khai báo **tham số kiểu cursor** trong các thủ tục nội tại chúng ta can 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**: bêntrong 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 can 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 nàođó. Chúng ta tạo một thủ tục có tham số* ***kiểu dữ liệu cursor*** *chứa danh sách các vật tư đã bánra 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 201210.

```
--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'
```

4.2.3.6. Thủ tục cập nhật dữ liệu

Trong các thí dụ mà chúng tôi trình bày ở những phần trên hầu như chức năng chính của thu 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 bean trong một môi trường khác

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 cac RBTV hop le 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.

4.2.3.7. 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 gởi 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 gởi 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 lànhiều hoặc ít dòng dữ liệu có trong bảng DONDH và các bảng liên quan

4.3. Giao tác

4.3. 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 gởi 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 qui 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.

4.3.2. 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 hoàntoàn 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)

4.3.3. 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 đo 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 nhấttrong 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 lắp 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 lắp. 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 @sChuoiTam = LTRIM(STR(@nSoctuke))
SET @sSoctuke = @sKytu +
REPLICATE("0", 3-LEN(@sChuoiTam)) +
@sChuoiTam
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.

4.4. TRIGGER

4.4.1. Khái quát về trigger

4.4.2. 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, xoá hay sửa.

Chính nhờ vào tính năng đặc biệt là **tự động thực hiện** mà phần lớn nội dung các lệnh bean 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.

4.4.3. Mở rộng ràng buộc toàn vẹn dữ liệu với trigger

Phần lớn các **ràng buộc toàn vẹn dữ liệu đơn giản** như là: **khóa nội, khóa ngoại, miền 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.

4.4.4. Các dạng ràng buộc toàn vẹn nên dùng 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.

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.

4.4.5. 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 tạitrong bộ nhớ** của máy tính (RAM) được xem như là hai **bảng luận lý** mà chúng ta **có thể sửdụng** trong các xử lý của trigger. Chúng ta không thể tham chiếu trực tiếp tới những bảng nàytrong 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**, khicâu 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.

4.4.6. Làm việc với trigger

4.4.6.1. Tạo mới trigger

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 các khác, nếu mỗi hành động cập nhật dữ liệu (thêm, xoá 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 nghiã 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<> "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’, ‘2012-04-01’, ‘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’, ‘2012-04-01’, ‘2012-03-
15’,’"C01’)
```

Kết quả lỗi trả về

Ngày dự kiến nhập hàng phải sau ngày đặt hàng

4.4.6.2. Sửa 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

4.4.6.3. Xóa trigger

Khi một trigger **không còn cần** sử dụng nữa thì chúng ta có thể xóa bỏ nó ra khỏi cơ sở dữliệu. 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 sửdụng nữa.

Ngoài ra chúng ta có thể xóa bỏ trigger khi có nhu cầu cần tạo lại nội dung mới, có nghĩa là khiđó 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

4.4.7. Lập trình trigger

4.4.7.1. Các thao tác lập trình trigger phổ biến

4.4.7.2. 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, 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
```

4.4.7.5. 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.

.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", "2012-01-15")

Kết quả lỗi trả về

Server: Msg 50000, Level 16, State 1, Line 0

Ngày giao hàng phải sau ngày : 10/02/2012

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", "2012-02-15")

Kết quả trả về hành động thêm dữ liệu mới thành công!

(1 row(s) affected)

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 đắng 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. 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 đắng 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 30/01/2012, ngày nhập hàng dự kiến là 02/02/2012 và đã được nhập hàng lần đầu vào ngày 31/01/2012. Giả sử chúng ta ra lệnh để sửa đổi giá trị ngày đặt hàng lại thành ngày 01/02/2012.

```
UPDATE DONDH
 SET NGAYDH= ‘2012-02-01’
WHERE SODH= ‘D002’
```

Kết quả lỗi trả về

Server: Msg 50000, Level 16, State 1, Line 0

Ngày đặt hàng phải trước ngày : 31/01/2012

Trường hợp 3: Đơn đặt hàng D005 có ngày đặt hàng là 01/03/2012, ngày nhập hàng dự kiến là 05/03/2012 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 07/03/2012.

```
UPDATE DONDH
SET NGAYDH= ‘2012-03-07"
WHERE SODH= ‘D005’
```

Hoặc

```
UPDATE DONDH
SET NGAYDKNH= ‘2012-02-01"
 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.

4.4.7.6. 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 thi 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 chi 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 chi 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 để 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.

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.