TRƯỜNG CAO ĐẲNG NGHỀ THÀNH PHỐ HỒ CHÍ MINH
KHOA: CÔNG NGHỆ THÔNG TIN

GIÁO TRÌNH

QUẢN TRỊ HỆ THỐNG CSDL NÂNG CAO (SQL SERVER)

(SỬ DỤNG CHO TRÌNH ĐỘ CAO ĐẲNG NGHỀ)

(LƯU HÀNH NỘI BỘ)

Tp. Hồ Chí Minh – 2013

MỤC LỤC

GIỚI THIỆU VỀ MÔ ĐUN QUẢN TRỊ HỆ THỐNG CƠ SỞ DỮ LIỆU NÂNG CAO (SQL SERVER)

I. VỊ TRÍ TÍNH CHẤT CỦA MÔ ĐUN:
Vị trí: Mô đun được bố trí cho sinh viên học sau các mô đun/ môn học Cơ sở dữ liệu, Quản trị hệ thống cơ sở dữ liệu cơ bản.
Tính chất: Là mô đun đào tạo chuyên môn nghề bắt buộc.
II. MỤC TIÊU CỦA MÔ ĐUN:
Kết nối hệ thống mạng để sử dụng hệ thống cơ sở dữ liệu.
Thực hiện thành thạo các thao tác quản trị tài khoản người dùng và tài khoản nhóm đối với hệ thống MS SQL Server.
Thiết lập cấu hình và giải quyết các vấn đề thường xảy ra trên mạng khi sử dụng truy cập cơ sở dữ liệu.
Bảo vệ tài nguyên dữ liệu trên các hệ thống MS SQL Server.
Lập được các chương trình bằng SQL hỗ trợ xử lý cơ sở dữ liệu và xử lý tự động.
Bố trí làm việc khoa học đảm bảo an toàn cho người và phương tiện học tập.
III. NỘI DUNG CỦA MÔ ĐUN
Bảo mật cơ sở dữ liệu
Sao lưu và phục hồi dữ liệu
Các hàm thông dụng
Thủ tục lưu trữ
Hàm người dùng
Trigger – thực thi tự động
Giao tác

BÀI 1: BẢO MẬT CƠ SỞ DỮ LIỆU

1.1 Các khái niệm về bảo mật

Bảo mật là một trong những yếu tố đóng vai trò quan trọng đối với sự sống còn của cơ sở dữ liệu. Hầu hết các hệ quản trị cơ sở dữ liệu thương mại hiện nay đều cung cấp khả năng bảo mật cơ sở dữ liệu với những chức năng như:

- Cấp phát quyền truy cập cơ sở dữ liệu cho người dùng và các nhóm người dùng, phát hiện và ngăn chặn những thao tác trái phép của người sử dụng trên cơ sở dữ liệu.
- Cấp phát quyền sử dụng các câu lệnh, các đối tượng cơ sở dữ liệu đối với người dùng.
- Thu hồi (huỷ bỏ) quyền của người dùng.

Bảo mật dữ liệu trong SQL được thực hiện dựa trên ba khái niệm chính sau đây:

- **Người dùng cơ sở dữ liệu (Database user):** Là đối tượng sử dụng cơ sở dữ liệu, thực thi các thao tác trên cơ sở dữ liệu như tạo bảng, truy xuất dữ liệu,... Mỗi một người dùng trong cơ sở dữ liệu được xác định thông qua tên người dùng (User ID). Một tập nhiều người dùng có thể được tổ chức trong một nhóm và được gọi là nhóm người dùng (User Group). Chính sách bảo mật cơ sở dữ liệu có thể được áp dụng cho mỗi người dùng hoặc cho các nhóm người dùng.

- **Các đối tượng cơ sở dữ liệu (Database objects):** Tập hợp các đối tượng, các cấu trúc lưu trữ được sử dụng trong cơ sở dữ liệu như bảng, khung nhìn, thủ tục, hàm được gọi là các đối tượng cơ sở dữ liệu. Đây là những đối tượng cần được bảo vệ trong chính sách bảo mật của cơ sở dữ liệu.

- **Đặc quyền (Privileges):** Là tập những thao tác được cấp phát cho người dùng trên các đối tượng cơ sở dữ liệu. Chẳng hạn một người dùng có thể truy

xuất dữ liệu trên một bảng bằng câu lệnh SELECT nhưng có thể không thể thực hiện các câu lệnh INSERT, UPDATE hay DELETE trên bảng đó.

SQL cung cấp hai câu lệnh cho phép chúng ta thiết lập các chính sách bảo mật trong cơ sở dữ liệu:

- Lệnh GRANT: Sử dụng để cấp phát quyền cho người sử dụng trên các đối tượng cơ sở dữ liệu hoặc quyền sử dụng các câu lệnh SQL trong cơ sở dữ liệu.
- Lệnh REVOKE: Được sử dụng để thu hồi quyền đối với người sử dụng.

1.2. Quyền người dùng

Là tập những thao tác được cấp phát cho người dùng trên các đối tượng cơ sở dữ liệu. Chẳng hạn một người dùng có thể truy xuất dữ liệu trên một bảng bằng câu lệnh SELECT nhưng có thể không thể thực hiện các câu lệnh INSERT, UPDATE hay DELETE để thêm, sửa, xóa dữ liệu trên bảng đó.

SQL cung cấp hai câu lệnh cho phép chúng ta thiết lập các chính sách bảo mật trong cơ sở dữ liệu:

- Lệnh GRANT: Sử dụng để cấp phát quyền cho người sử dụng trên các đối tượng cơ sở dữ liệu hoặc quyền sử dụng các câu lệnh SQL trong cơ sở dữ liệu.
- Lệnh REVOKE: Được sử dụng để thu hồi quyền đối với người sử dụng.

1.3 Cấp phát quyền

Câu lệnh GRANT được sử dụng để cấp phát quyền cho người dùng hay nhóm người dùng trên các đối tượng cơ sở dữ liệu. Câu lệnh này thường được sử dụng trong các trường hợp sau:

- Người sở hữu đối tượng cơ sở dữ liệu muốn cho phép người dùng khác quyền sử dụng những đối tượng mà anh ta đang sở hữu.
- Người sở hữu cơ sở dữ liệu cấp phát quyền thực thi các câu lệnh (như CREATE TABLE, CREATE VIEW,...) cho những người dùng khác.

1.3.1 Cấp phát quyền cho người dùng trên các đối tượng cơ sở dữ liệu

Chỉ có người sở hữu cơ sở dữ liệu hoặc người sở hữu đối tượng cơ sở dữ liệu mới có thể cấp phát quyền cho người dùng trên các đối tượng cơ sở dữ liệu. Câu lệnh GRANT trong trường hợp này có cú pháp như sau:

GRANT ALL [PRIVILEGES]| *các_quyền_cấp_phát*
[(*danh_sách_cột*)] ON *tên_bảng* | *tên_khung_nhìn*
|ON *tên_bảng* | *tên_khung_nhìn* [(*danh_sách_cột*)]
|ON *tên_thủ_tục*
|ON *tên_hàm*
TO *danh_sách_người_dùng* | *nhóm_người_dùng*

[WITH GRANT OPTION]

Trong đó:

ALL [PRIVILEGES]
Cấp phát tất cả các quyền cho người dùng trên đối tượng cơ sở dữ liệu được chỉ định. Các quyền có thể cấp phát cho người dùng bao gồm:
Đối với bảng, khung nhìn, và hàm trả về dữ liệu kiểu bảng: SELECT, INSERT, DELETE, UPDATE và REFERENCES (quyền REFERENCES được sử dụng nhằm cho phép tạo khóa ngoài tham chiếu đến bảng cấp phát)
Đối với cột trong bảng, khung nhìn: SELECT và UPDATE.
Đối với thủ tục lưu trữ và hàm vô hướng: EXECUTE.

các_quyền_cấp_phát	Danh sách các quyền cần cấp phát cho người dùng trên đối tượng cơ sở dữ liệu được chỉ định. Các quyền được phân cách nhau bởi dấu phẩy
tên_bảng|tên_khung_nhìn	Tên của bảng hoặc khung nhìn cần cấp phát quyền
danh_sách_cột	Danh sách các cột của bảng hoặc khung nhìn cần cấp phát quyền.
tên_thủ_tục	Tên của thủ tục được cấp phát cho người dùng.
tên_hàm	Tên hàm (do người dùng định nghĩa) được cấp phát quyền.
danh_sách_người_dùng	Danh sách tên người dùng nhận quyền được cấp phát. Tên của các người dùng được phân cách nhau bởi dấu phẩy.
WITH GRANT OPTION	Cho phép người dùng chuyển tiếp quyền cho người dùng khác.

Các ví dụ dưới đây sẽ minh hoạ cho ta cách sử dụng câu lệnh GRANT để cấp phát quyền cho người dùng trên các đối tượng cơ sở dữ liệu.

Ví dụ 1.1:
Cấp phát cho người dùng có tên *thuchanh* quyền thực thi các câu lệnh SELECT, INSERT và UPDATE trên bảng LOP
GRANT SELECT,INSERT,UPDATE ON lop
TO thuchanh
Cho phép người dùng *thuchanh* quyền xem họ tên và ngày sinh của các sinh viên (cột HODEM,TEN và NGAYSINH của bảng SINHVIEN)
GRANT SELECT

(hodem,ten,ngaysinh) ON sinhvien
TO thuchanh
hoặc:

GRANT SELECT
ON sinhvien(hodem,ten,ngaysinh) TO thuchanh

Với quyền được cấp phát như trên, người dùng *thuchanh* có thể thực hiện câu lệnh sau trên bảng SINHVIEN
SELECT hoden,ten,ngaysinh
FROM sinhvien
Nhưng câu lệnh dưới đây lại không thể thực hiện được
SELECT * FROM sinhvien
Trong trường hợp cần cấp phát tất cả các quyền có thể thực hiện được trên đối tượng cơ sở dữ liệu cho người dùng, thay vì liệt kê các câu lệnh, ta chỉ cần sử dụng từ khoá ALL PRIVILEGES (từ khóa PRIVILEGES có thể không cần chỉ định). Câu lệnh dưới đây cấp phát cho người dùng *thuchanh* các quyền SELECT, INSERT, UPDATE, DELETE VÀ REFERENCES trên bảng DIEMTHI
GRANT ALL ON DIEMTHI
TO thuchanh

Khi ta cấp phát quyền nào đó cho một người dùng trên một đối tượng cơ sở dữ liệu, người dùng đó có thể thực thi câu lệnh được cho phép trên đối tượng đã cấp phát. Tuy nhiên, người dùng đó không có quyền cấp phát những quyền mà mình được phép cho những người sử dụng khác.Trong một số trường hợp, khi ta cấp phát quyền cho một người dùng nào đó, ta có thể cho phép người đó chuyển tiếp quyền cho người dùng khác bằng cách chỉ định tuỳ chọn WITH GRANT OPTION trong câu lệnh GRANT.

Ví dụ 1.2:
Cho phép người dùng *thuchanh* quyền xem dữ liệu trên bảng SINHVIEN
đồng thời có thể chuyển tiếp quyền này cho người dùng khác
GRANT SELECT ON sinhvien TO thuchanh
WITH GRANT OPTION

1.3.2 Cấp phát quyền thực thi các câu lệnh

Ngoài chức năng cấp phát quyền cho người sử dụng trên các đối tượng cơ sở dữ liệu, câu lệnh GRANT còn có thể sử dụng để cấp phát cho người sử dụng một số quyền trên

hệ quản trị cơ sở dữ liệu hoặc cơ sở dữ liệu. Những quyền có thể cấp phát trong trường hợp này bao gồm:

- Tạo cơ sở dữ liệu: CREATE DATEBASE.
- Tạo bảng: CREATE RULE
- Tạo khung nhìn: CREATE VIEW

- Tạo thủ tục lưu trữ: CREATE PROCEDURE
- Tạo hàm: CREATE FUNCTION
- Sao lưu cơ sở dữ liệu: BACKUP DATABASE

Câu lệnh GRANT sử dụng trong trường hợp này có cú pháp như sau:
GRANT ALL | *danh_sách_câu_lênh*
TO *danh_sách_người_dùng*

Ví dụ 1.3: Để cấp phát quyền tạo bảng và khung nhìn cho người dùng có tên là
thuchanh, ta sử dụng câu lệnh như sau:
GRANT CREATE TABLE,CREATE VIEW TO thuchanh

Với câu lệnh GRANT, ta có thể cho phép người sử dụng tạo các đối tượng cơ sở dữ liệu trong cơ sở dữ liệu. Đối tượng cơ sở dữ liệu do người dùng nào tạo ra sẽ do người đó sở hữu và do đó người này có quyền cho người dùng khác sử dụng đối tượng và cũng có thể xóa bỏ (DROP) đối tượng do mình tạo ra.

Khác với trường hợp sử dụng câu lệnh GRANT để cấp phát quyền trên đối tượng cơ sở dữ liệu, câu lệnh GRANT trong trường hợp này không thể sử dụng tuỳ chọn WITH GRANT OPTION, tức là người dùng không thể chuyển tiếp được các quyền thực thi các câu lệnh đã được cấp phát.

1.4 Thu hồi quyền

Câu lệnh REVOKE được sử dZụng để thu hồi quyền đã được cấp phát cho người dùng. Tương ứng với câu lệnh GRANT, câu lệnh REVOKE được sử dụng trong hai trường hợp:
Thu hồi quyền đã cấp phát cho người dùng trên các đối tượng cơ sở dữ liệu.
Thu hồi quyền thực thi các câu lệnh trên cơ sở dữ liệu đã cấp phát cho người dùng.

1.4.1 Thu hồi quyền trên đối tượng cơ sở dữ liệu:

Cú pháp câu lệnh REVOKE sử dụng để thu hồi quyền đã cấp phát trên đối tượng cơ sở dữ liệu có cú pháp như sau:
REVOKE [GRANT OPTION FOR]
ALL [PRIVILEGES]| *các_quyền_cần_thu_hồi*
[(*danh_sách_cột*)] ON *tên_bảng* | *tên_khung_nhìn*
|ON *tên_bảng* | *tên_khung_nhìn* [(*danh_sách_cột*)]

|ON tên_*thủ_tục*
|ON *tên_hàm*
FROM *danh_sách_người_dùng*
[CASCADE]

Câu lệnh REVOKE có thể sử dụng để thu hồi một số quyền đã cấp phát cho người dùng hoặc là thu hồi tất cả các quyền (ALL PRIVILEGES).
Ví dụ 1.4: Thu hồi quyền thực thi lệnh INSERT trên bảng LOP đối với người dùng *thuchanh*.
REVOKE INSERT ON lop
FROM thuchanh
Giả sử người dùng *thuchanh* đã được cấp phát quyền xem dữ liệu trên các cột HODEM, TEN và NGAYSINH của bảng SINHVIEN, câu lệnh dưới đây sẽ thu hồi quyền đã cấp phát trên cột NGAYSINH (chỉ cho phép xem dữ liệu trên cột HODEM và TEN)
REVOKE SELECT
ON sinhvien(ngaysinh) FROM thuchanh

Khi ta sử dụng câu lệnh REVOKE để thu hồi quyền trên một đối tượng cơ sở dữ liêu từ một người dùng náo đó, chỉ những quyền mà ta đã cấp phát trước đó mới được thu hồi, những quyền mà người dùng này được cho phép bởi những người dùng khác vẫn còn có hiệu lực. Nói cách khác, nếu hai người dùng khác nhau cấp phát cùng các quyền trên cùng một đối tượng cơ sở dữ liệu cho một người dùng khác, sau đó người thu nhất thu hồi lại quyền đã cấp phát thì những quyền mà người dùng thứ hai cấp phát vẫn có hiệu lực.
Ví dụ 1.5: Giả sử trong cơ sở dữ liệu ta có 3 người dùng là *A*, *B* và *C*. A và B đều có quyền sử dụng và cấp phát quyền trên bảng R. A thực hiện lệnh sau để cấp phát quyền xem dữ liệu trên bảng R cho C:
GRANT SELECT ON R TO C
và B cấp phát quyền xem và bổ sung dữ liệu trên bảng R cho C bằng câu lệnh:
GRANT SELECT, INSERT ON R TO C
Như vậy, C có quyền xem và bổ sung dữ liệu trên bảng R. Bây giờ, nếu B thực hiện lệnh:

REVOKE SELECT, INSERT
ON R FROM C
Người dùng C sẽ không còn quyền bổ sung dữ liệu trên bảng R nhưng vẫn có thể xem được dữ liệu của bảng này (quyền này do A cấp cho C và vẫn còn hiệu lực).
Nếu ta đã cấp phát quyền cho người dùng nào đó bằng câu lệnh GRANT với tuỳ chọn WITH GRANT OPTION thì khi thu hồi quyền bằng câu lệnh REVOKE phải chỉ định tuỳ chọn CASCADE. Trong trường hợp này, các quyền được chuyển tiếp cho những người dùng khác cũng đồng thời được thu hồi.
Ví dụ 1.6: Ta cấp phát cho người dùng A trên bảng R với câu lệnh GRANT như sau:
GRANT SELECT ON R TO A
WITH GRANT OPTION
sau đó người dùng A lại cấp phát cho người dùng B quyền xem dữ liệu trên R với câu lệnh:
GRANT SELECT ON R TO B
Nếu muốn thu hồi quyền đã cấp phát cho người dùng A, ta sử dụng câu lệnh REVOKE như sau:
REVOKE SELECT ON NHANVIEN FROM A CASCADE
Câu lệnh trên sẽ đồng thời thu hồi quyền mà A đã cấp cho B và như vậy cả A và B đều không thể xem được dữ liệu trên bảng R.

Trong trường hợp cần thu hồi các quyền đã được chuyển tiếp và khả năng chuyển tiếp các quyền đối với những người đã được cấp phát quyền với tuỳ chọn WITH GRANT OPTION, trong câu lệnh REVOKE ta chỉ định mệnh đề GRANT OPTION FOR.
Ví dụ 1.7: Trong ví dụ trên, nếu ta thay câu lệnh:
REVOKE SELECT ON NHANVIEN FROM A CASCADE
bởi câu lệnh:
REVOKE GRANT OPTION FOR SELECT ON NHANVIEN
FROM A CASCADE

Thì B sẽ không còn quyền xem dữ liệu trên bảng R đồng thời A không thể chuyển tiếp quyền mà ta đã cấp phát cho những người dùng khác (tuy nhiên A vẫn còn quyền xem dữ liệu trên bảng R).

1.4.2 Thu hồi quyền thực thi các câu lênh:

Việc thu hồi quyền thực thi các câu lệnh trên cơ sở dữ liệu (CREATE DATABASE, CREATE TABLE, CREATE VIEW,...) được thực hiện đơn giản với câu lệnh REVOKE có cú pháp:

REVOKE ALL | *các_câu_lệnh_cần_thu_hồi*
FROM *danh_sách_người_dùng*

Ví dụ 1.8: Để không cho phép người dùng *thuchanh* thực hiện lệnh CREATE TABLE trên cơ sở dữ liệu, ta sử dụng câu lệnh:
REVOKE CREATE TABLE FROM thuchanh

BÀI 2: SAO LƯU VÀ PHỤC HỒI DỮ LIỆU

Bài này sẽ giới thiệu kỹ thuật sao lưu (backup) và khôi phục (restore) dữ liệu, là kỹ thuật thường được sử dụng bảo đảm an toàn dữ liệu phòng trường hợp CSDL có sự cố.

2.1 Sao lưu dữ liệu (Backup)

Trong quá trình thực hiện quản trị CSDL SQL Server thì một số nguyên nhân sau đây bắt buộc bạn phải xem xét đến kỹ thuật sao lưu và khôi phục dữ liệu:
Thiết bị lưu trữ (CSDL nằm trên các thiết bị lưu trữ này) bị hư hỏng.
Người dùng vô tình xóa dữ liệu.
Các hành động vô tình hay cố ý phá hoại CSDL.

2.2 Các loại Backup

Microsoft SQL Server 2005 cung cấp các kỹ thuật sao lưu CSDL chính: full backup, differential backup và Transaction log backup

2.2.1 Full backup và Differential backup

Full backup: Sao lưu một bản đầy đủ của CSDL trên các phương tiện lưu trữ. Quá trình full backup có thể tiến hành mà không cần offline CSDL, nhưng quá trình này lại chiếm một lượng lớn tài nguyên hệ thống và có thể ảnh hưởng nghiêm trọng tới thời gian đáp ứng các yêu cầu của hệ thống.
Differential backup: Được xây dựng nhằm làm giảm thời gian cần thiết để thực hiện quá trình full backup. Differential backup chỉ sao lưu những thay đổi trên dữ liệu kể từ lần full backup gần nhất. Trong những hệ thống CSDL lớn, quá trình differential backup sẽ sử dụng tài nguyên ít hơn rất nhiều so với quá trình full backup và có thể không ảnh hưởng đến hiệu suất của hệ thống.
Quá trình differential chỉ sao lưu những sự thay đổi của dữ liệu từ lần full backup gần nhất, do đó khi có sự cố với CSDL nếu không có bản sao lưu của quá trình full backup thì bản sao lưu của quá trình differential backup sẽ trở nên vô nghĩa.
Ví dụ:
Công ty XYZ thực hiện full backup vào cuối ngày thứ 6 hàng tuần và thực hiện differential backup vào tối các ngày từ thứ 2 tới thứ 5. Nếu CSDL có sự cố vào sáng thứ 4, quản trị viên CSDL sẽ phục hồi dữ liệu bằng bản sao lưu của quá trình full backup của ngày
thứ 6 tuần trước và sau đó phục hồi các thay đổi của dữ liệu bằng cách áp dụng bản sao lưu của
quá trình differential backup vào ngày thứ 3.

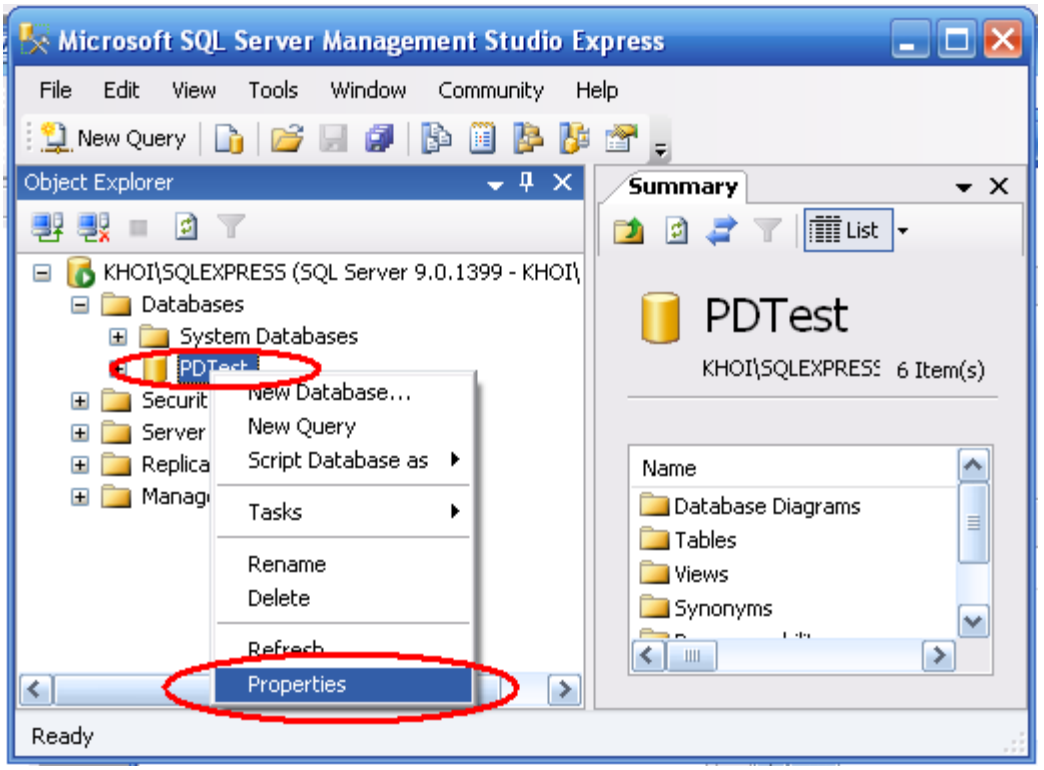
6.2.2 Transaction log backup

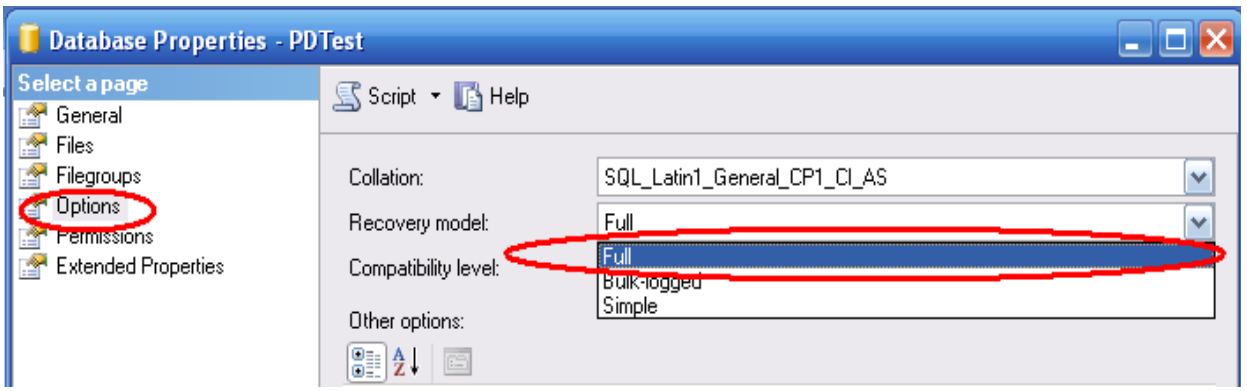
Quá trình full backup và differential backup chiếm nhiều tài nguyên hệ thống và ảnh hưởng đến hiệu suất làm việc hệ thống nên thường được thực hiện vào sau giờ làm việc. Tuy nhiên điều này có thể dẫn đến các mất mát dữ liệu trong một ngày làm việc nếu CSDL có sự cố trước khi quá trình sao lưu diễn ra. Transaction log backup là một giải pháp nhằm giảm thiểu tối đa lượng dữ liệu có thể mất khi có sự cố CSDL.
Trong quá trình hoạt động, SQL Server sử dụng transaction log để theo dõi tất cả các thay đổi trên CSDL. *Log* bảo đảm CSDL có thể phục hồi sau những sự cố đột xuất và cũng đảm bảo người dùng có thể quay ngược các kết quả trong các giao tác CSDL. Các giao tác chưa hoàn thành được lưu trong *log* trước khi được lưu vĩnh viễn trong CSDL.
Transaction log backup sao lưu transaction log của CSDL vào thiết bị lưu trữ. Mỗi khi transaction log được sao lưu, SQL Server bỏ đi các transaction đã thực hiện thành công (committed tracsaction) và ghi các transaction vào phương tiện sao lưu. Transaction log backup
sử dụng tài nguyên hệ thống ít hơn rất nhiều so với full backup và differential backup, do đó có

thể sử dụng transaction log backup bất kỳ thời gian nào mà không sợ ảnh hưởng đến hiệu suất
hệ thống.
Trở lại với ví dụ về công ty XYZ. Công ty này thực hiện full backup vào tối thứ 6 và differential backup vào tối từ thứ 2 tới thứ 5. Công ty thực hiện thêm quá trình transaction log backup mỗi giờ một lần. Giả sử sự cố CSDL xảy ra vào 9h:05 sáng thứ 4. Quá trình khôi phục
lại CSDL nhu sau: Dùng full backup và differential backup của tối thứ 6 và tối thứ 3 để phục
hồi lại trạng thái CSDL vào tối thứ 3. Tuy nhiên quá trình này vẫn còn để mất dữ liệu trong 2
giờ (7 – 9h) sáng thứ 4. Tiếp theo sử dụng 2 bản sao lưu transaction backup lúc 8h và 9h sáng
để khôi phục CSDL về trạng thái lúc 9h sáng thứ 4.

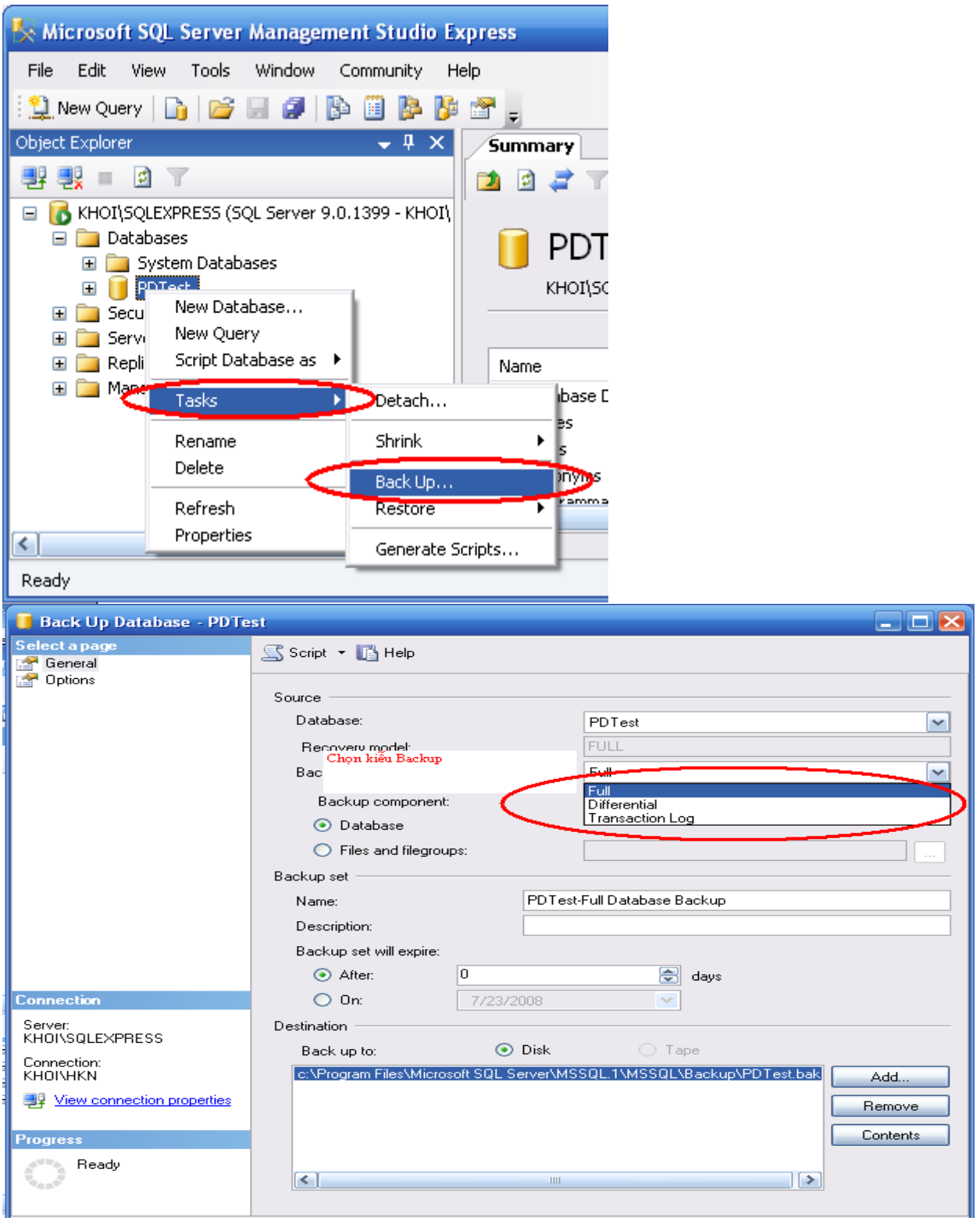
6.3 Các thao tác thực hiện quá trình Backup và Restore trong SQL Server 2005 Express Edition

6.3.1 Sao lưu (Backup)



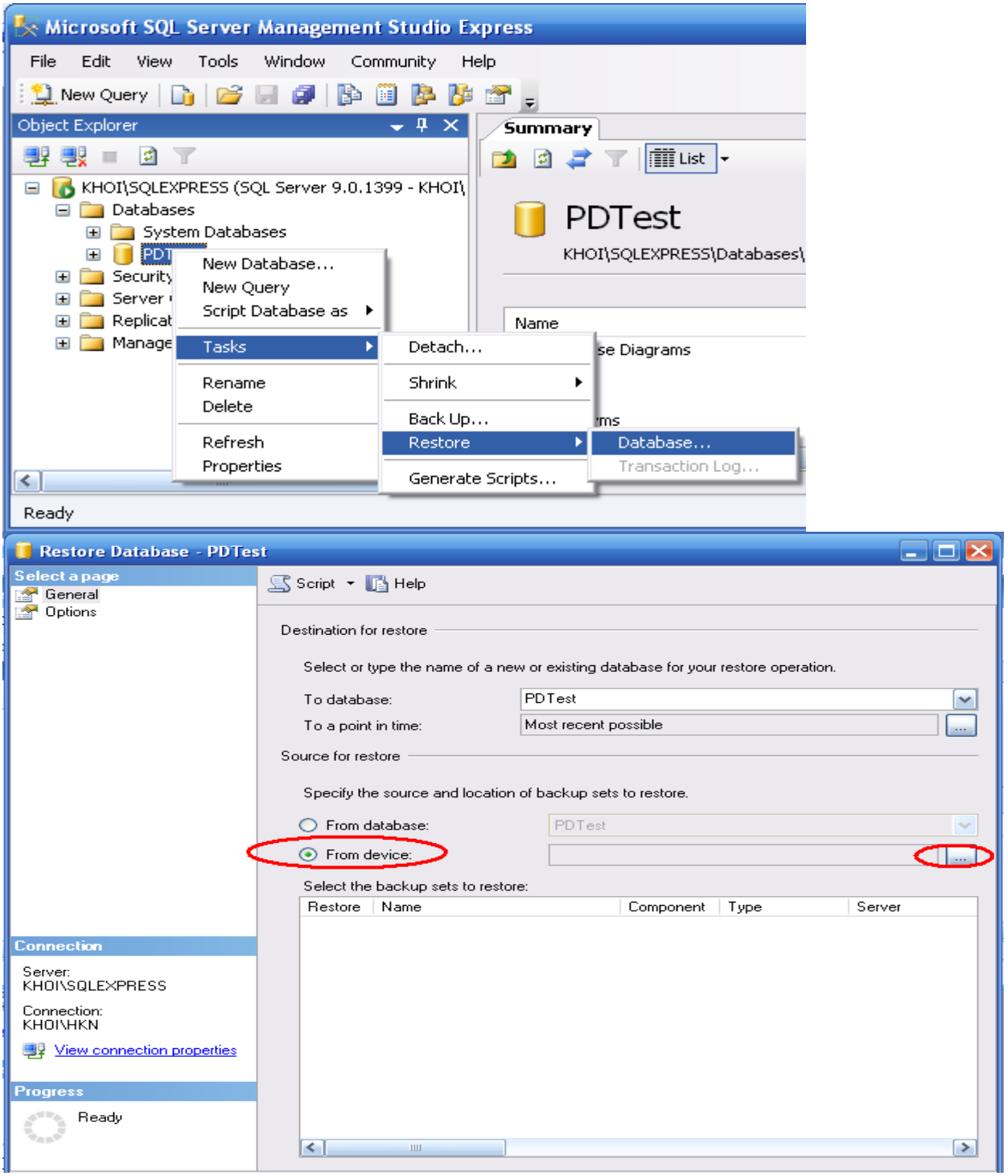

Database Properties - PDTest
Select a page
General
Files
Filegroups
Options
Permissions
Extended Properties
Script
Help
Collation:
SQL_Latin1_General_CP1_CI_AS
Recovery model:
Full
Compatibility level:
Full
Bulk-logged
Simple
Other options:

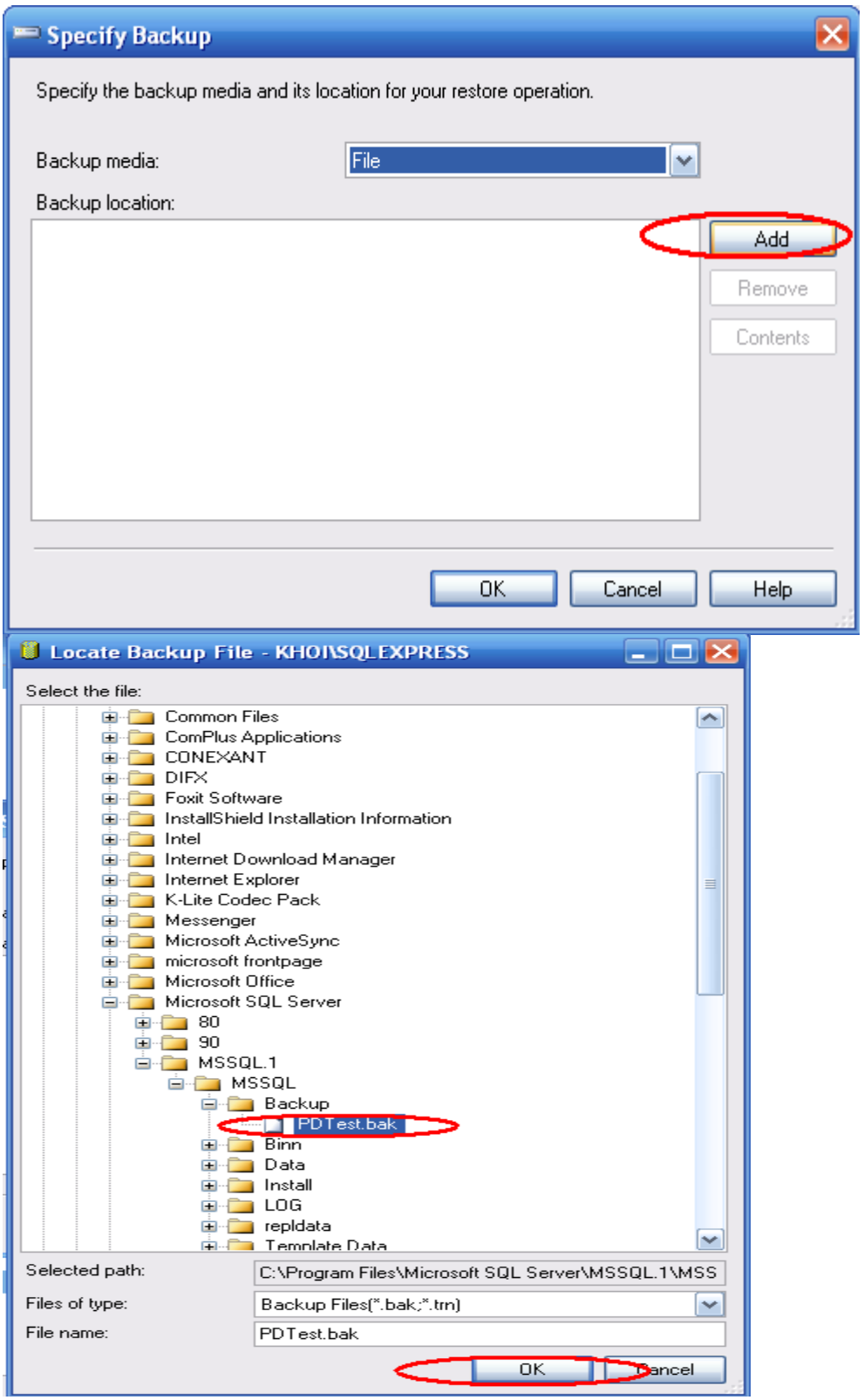
Click OK



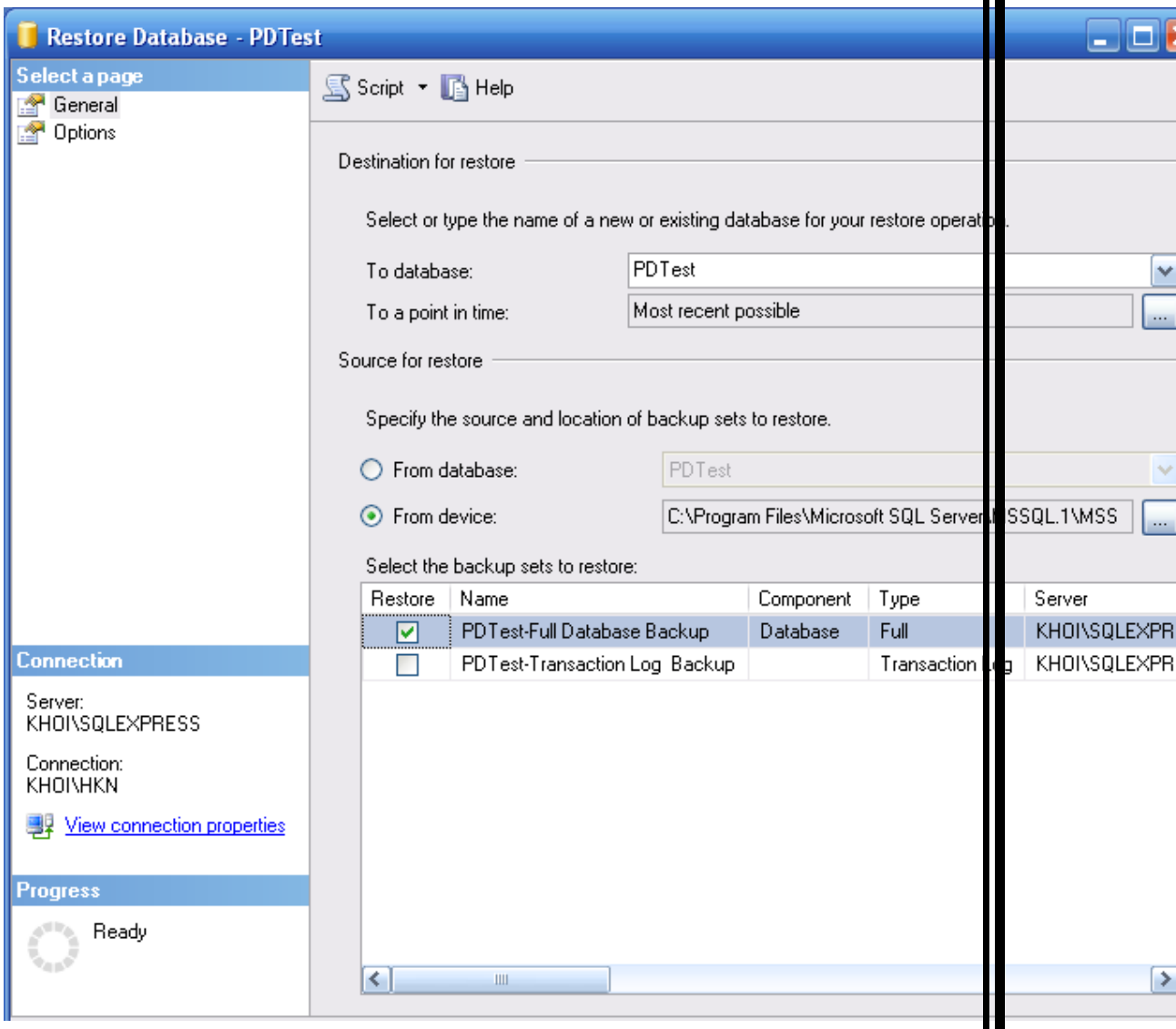
Click OK. Quá trình sao lưu hoàn tất

6.3.2 Phục hồi (Restore)





Click OK hai lần



Click OK. Quá trình phục hồi hoàn tất

BÀI 3: CÁC HÀM THÔNG DỤNG

Ngôn ngữ T-SQL có nhiều hàm có thể tham gia vào câu lệnh T-SQL. Những hàm này thực hiện các nhiệm vụ quan trọng khác nhau. Trong chương này sẽ trình bày một số các hàm thông dụng để làm việc với các kiểu dữ liệu số, chuỗi, ngày/thời gian và giá trị NULL trong SQL Server 2005.

3.1 Các hàm làm việc với kiểu dữ liệu chuỗi

Các hàm quan trọng bao gồm LEFT, RIGHT, LEN, REPLACE, STUFF, SUBSTRING, LOWER, UPPER, LTRIM, and RTRIM.

3.1.1 Hàm LEFT

Hàm LEFT trả về một chuỗi ký tự có chiều dài được chỉ định tính từ bên trái của chuỗi.
Ví dụ:
select left('Nha Trang', 5)

(No column name)
Nha T

3.1.2 Hàm RIGHT

Hàm RIGHT tương tự hàm LEFT nhưng tính từ bên phải của chuỗi
Ví dụ:
select right('Nha Trang', 5)

(No column name)
Trang

3.1.3 Hàm SUBSTRING

Hàm STRING trích xuất một chuỗi con từ một chuỗi cho trước.
Cấu trúc hàm SUBSTRING như sau:
SUBSTRING (chuỗi_ban_đầu, vị_trí_bắt_đầu, chiều_dài_chuỗi_con)
Ví dụ 1:
select substring ('Nha Trang', 2, 5)

(No column name)
ha Tr

Ví dụ 2:
Select substring('Nha Trang', -2, 5)

(No column name)
Nh

3.1.4 Hàm LEN

Hàm LEN trả về chiều dài một chuỗi
Ví dụ:
Select len('Nha Trang')

(No column name)
9

3.1.5 Hàm REPLACE

Hàm REPLACE thay thế một chuỗi bởi một chuỗi khác
Ví dụ 1: Câu lệnh dưới đây thay thế chữ "Nha" trong chuỗi Nha Trang bằng chữ "nha"
Select replace('Nha Trang', 'Nha', 'nha)

(No column name)
nha Trang

Ví dụ 2:
select replace(ADDRESS, 'Minh', 'Ninh')
from customers

(No column name)
33 Nguyen Trung Truc
99 Nguyen Thi Ninh Khai
45/2B Da Tuong
76 Tran Phu
56 Le Hong Phong
29A Phuong Sai
12 Nguyen Thien Thuat
14 Thong Nhat
14 Thong Nhat

3.1.6 Hàm STUFF

Hàm STUFF thay thế một số lượng xác định các ký tự trong một chuỗi bằng một chuỗi
khác bắt đầu từ một vị trí được chỉ định.
Ví dụ:
*select stuff('Nha Trang', 2, 3, '***')*

(No column name)
N***Trang

3.1.7 Hàm LOWER/UPPER

Hàm LOWER chuyển các ký tự hoa trong chuỗi thành các kí tự thường.
Hàm UPPER
chuyển các chuỗi ký tự thường trong chuỗi thành các ký tự hoa.
Ví dụ:
select lower('Nha Trang'), upper('Nha Trang')

(No column name)	(No column name)
nha trang	NHA TRANG

3.1.8 Hàm LTRIM/RTRIM

Hàm LTRIM cắt các khoảng trắng bên trái của chuỗi, hàm RTRIM cắt khoảng trắng bên phải chuỗi.
Ví dụ:
declare @llen int declare @rlen int declare @len int
select @llen = len(ltrim(' Nha Trang')),
@rlen = len(rtrim('Nha Trang ')),
@len = len('Nha Trang')
select @llen, @rlen, @len

(No column name)	(No column name)	(No column name)
9	9	9

3.2 Các hàm làm việc với kiểu dữ liệu số

Các hàm quan trọng làm việc với kiểu dữ liệu số là hàm ISNUMERIC và ham ROUND

3.2.1 Hàm ISNUMERIC

Hàm isNumeric kiểm tra một giá trị có phải thuộc kiểu dữ liệu số hay không.
Ví dụ: Câu lệnh dưới đây trả về tên khách hàng, và một cột có tên NUMERIC. Cột này sẽ
mang giá trị 0 nếu địa chỉ khách hàng không phải là số và ngược lại
select CUSTOMERNAME, isnumeric(ADDRESS) as ISNUMERIC
from customers

CUSTOMERNAME	ADDRESS
Cao Van Trung	0
Tran Van Phuc	0
Tran Viet Cuong	0
Nguyen Van Dai	0
Le Thi Hoa	0
Nguyen Thanh Thai	0
Cao Van Chung	0
NGuyen Van An	0
NGuyen Van An	0

3.2.2 Hàm ROUND

Hàm ROUND trả về một giá trị số, đã được làm tròn theo một độ đài chỉ định
Cấu trúc hàm ROUND như sau:
ROUND (số_làm_tròn , độ_dài_làm_tròn)
Khi sử dụng hàm ROUND cần lưu ý:
số_làm_tròn phải có kiểu dữ liệu số (numeric data type) như int, float, decimal… trừ kiểu
dữ liệu dạng nhị phân. Cho dù *số_làm_tròn* thuộc kiểu dữ liệu gì, kết quả hàm ROUND luôn
trả về kiểu số nguyên.
Nếu *độ_dài_làm_tròn* là số âm và lớn hơn số chữ số phía trước dấu thập phân thì hàm
ROUND trả về 0.
Ví dụ 1:
select ROUND(123.9994, 3), ROUND(123.9995, 3)

(No column name)	(No column name)
123.9990	124.0000

Ví dụ 2:
select ROUND(123.4545, 2),ROUND(123.45, -2)

(No column name)	(No column name)
123.4500	100.00

Ví dụ 3:
SELECT ROUND(150.75, 0), ROUND(150.75, 0, 1)

(No column name)	(No column name)
151.00	150.00

3.3 Các hàm làm việc với kiểu dữ liệu Ngày Giờ

3.3.1 Hàm GETDATE

Hàm GETDATE trả về ngày giờ lúc thực hiện câu truy vấn.
Ví dụ:
select getdate()

3.3.2 Hàm DAY/ MONTH/ YEAR
Hàm DAY trả về ngày của một một giá trị thuộc kiểu datetime. Hàm MONTH trả về tháng của một giá trị thuộc kiểu datetime
Hàm YEAR trả về năm của một giá trị thuộc kiểu datetime.
Ví dụ:
select day(orderdate) as DAYOFORDER, month(orderdate) as MONTHOFORDER, year(orderdate) as YEAROFORDER
from orders o inner join customers c on c.customerid = o.customerid where c.customerid = 3

DAYOFORDER	MONTHOFORDER	YEAROFORDER
1	1	2008
1	5	2008

3.3.3 Hàm DATEPART

Trong quá trình làm việc với các CSDL, đôi lúc ta muốn biết xem một ngày nào đó thuộc quý mấy trong năm, hay thuộc tuần thứ mấy trong tháng. Hàm DATEPART giúp giải quyết các yêu cầu trên một cách dễ dàng.
Cấu trúc hàm DATEPART như sau:

DATEPART (yêu_cầu_trích_xuất, giá_trị_trích_xuất)

giá_trị_trích_xuất là một giá trị thuộc kiểu datetime. *yêu_cầu_trích_xuất*: ngày, tháng, năm, quý,….

Khi có một yêu cầu trích xuất nào đó, chúng ta sẽ có các chữ viết tắt tương ứng với các yêu cầu đó. Bảng dưới đây mô tả các yêu chữ viết tắt và các yêu cầu trích xuất tương ứng.

Ý nghĩa	Chữ viết tắt
Năm	yy, yyyy
Quý	qq,q
Tháng	mm,m
Số ngày đã qua trong năm	dy,y
Ngày	dd,d
Tuần	wk,ww
Số ngày đã qua trong tuần	dw
Giờ	hh
Phút	mi,n
Giây	ss,s

Ví dụ:

select datepart(yyyy, orderdate)as YEAROFORDERDATE, datepart(qq, orderdate)as QUARTEROFORDERDATE,
datepart(m, orderdate) as MONTHOFORDERDATE, datepart(wk, orderdate) as WEEKOFORDERDATE, datepart(d, orderdate) as DATEOFORDERDATE, datepart(dy, Orderdate), datepart(dw, orderdate) from orders

YEAROFORDERDATE	QUARTEROFORDERDATE	MONTHOFORDERDATE	WEEKOFORDERDATE	DATEOFORDERDATE	(No column name)	(No column name)
2007	4	12	49	6	340	5
2008	1	1	1	1	1	1
2008	2	5	18	1	122	5

3.3.4 Hàm DATENAME

Tương tự hàm DATEPART nhưng hàm DATENAME trả về một chuỗi ký tự

Ví dụ:

select datename(yyyy, orderdate)as YEAROFORDERDATE, datename(qq, orderdate)as QUARTEROFORDERDATE, datename(m, orderdate) as MONTHOFORDERDATE, datename(wk, orderdate) as WEEKOFORDERDATE, datename(d, orderdate) as DATEOFORDERDATE, datename(dy, Orderdate), datename(dw, orderdate)
from orders

YEAROFORDERDATE	QUARTEROFORDERDATE	MONTHOFORDERDATE	WEEKOFORDERDATE	DATEOFORDERDATE	(No column name)	(No column name)
2007	4	December	49	6	340	Thursday
2008	1	January	1	1	1	Tuesday
2008	2	May	18	1	122	Thursday

3.4 Hàm chuyển đổi kiểu dữ liệu **CAST và CONVERTER**

Chuyển đổi một giá trị thuộc kiểu dữ liệu này sang một kiểu dữ liệu khác. Hàm CAST và CONVERTER cung cấp cùng một chức năng. Một điểm thuận lợi khi dùng CONVERTER là khi chuyển đổi, hàm này cũng cho phép người dùng sẽ định dạng lại giá tri kết quả theo ý muốn.

Cấu trúc hàm CAST và CONVERTER như sau:

CAST (biểu_thức/giá_ trị AS kiểu_dữ liệu [độ_dài_kiểu_dữ_liệu])
CONVERT (kiểu_dữ liệu [độ_dài_kiểu_dữ_liệu] , biểu_thức/giá_ trị [,kiểu_định_dạng])

Năm 2 chữ số	Năm 4 chữ số	*Đnh dạng khi xuát*
	0 hoặc 100	mon dd yyyy hh:mi AM (PM)
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 hoặc 109	mon dd yyyy hh:mi:ss:mmmAM (PM)
10	110	mm-dd-yy
11	111	yy/mm/dd
12	112	yymmdd
	13 hoặc 113	dd mon yyyy hh:mm:ss:mmm(24h)
14	114	hh:mi:ss:mmm(24h)

Ví dụ:

```
select  CUSTOMERNAME,
convert (varchar, BIRTHDAY, 103) as BIRTHDAY, ADDRESS
from Customers
where Customername = 'Le Thi Hoa'
and year(getdate()) - year(BIRTHDAY) > 20
```

Có thể sử dụng kết hợp hàm CONVERT và hàm CAST với nhau để cho kết qua như mong muốn.

Ví dụ:

```
select c.CUSTOMERID, c.CUSTOMERNAME,
convert(varchar(20),cast(SUM(i.UNITPRICE*od.QUANTITY)        as
money),1)
as SUMTOTAL
from   customers c inner join orders o on o.customerid = c.customerid inner
join orderdetail od on o.orderid = od.orderid
inner join items i on i.itemid = od.itemid
group by c.customerid, c.customername
```

BÀI 4: THỦ TỤC LƯU TRỮ (STORED PROCEDURE)

4.1. Khái niệm về thủ tục lưu trữ

Thủ tục lưu trữ là một đối tượng trong CSDL, bao gồm nhiều câu lệnh T-SQL được tập hợp lại với nhau thành một nhóm, và tất cả các lệnh này sẽ được thực thi khi thủ tục lưu trữ được thực thi.
Với thủ tục lưu trữ, một phần nào đó khả năng của ngôn ngữ lập trình được đưa vào trong
ngôn ngữ SQL. Thủ tục lưu trữ có thể có các thành phần sau:
Các cấu trúc điều khiển (IF, WHILE, FOR) có thể được sử dụng trong thủ tục.
Bên trong thủ tục lưu trữ có thể sử dụng các biến như trong ngôn ngữ lập trình nhằm lưu
giữ các giá trị tính toán được, các giá trị được truy xuất được từ cơ sở dữ liệu.

Một tập các câu lệnh SQL được kết hợp lại với nhau thành một khối lệnh bên trong một
thủ tục. Một thủ tục có thể nhận các tham số truyền vào cũng như có thể trả về các giá trị thông
qua các tham số (như trong các ngôn ngữ lập trình). Khi một thủ tục lưu trữ đã được định nghĩa, nó có thể được gọi thông qua tên thủ tục, nhận các tham số truyền vào, thực thi
các câu lệnh SQL bên trong thủ tục và có thể trả về các giá trị sau khi thực hiện xong.
Lợi ích của việc sử dụng thủ tục lưu trữ:
SQL Server chỉ biên dịch các thủ tục lưu trữ một lần và sử dụng lại kết quả biên dịch này trong các lần tiếp theo trừ khi người dùng có những thiết lập khác. Việc sử dụng lại kết quả biên dịch không làm ảnh hưởng đến hiệu suất hệ thống khi thủ tục lưu trữ được gọi liên tục nhiều lần.
Thủ tục lưu trữ được phân tích, tối ưu khi tạo ra nên việc thực thi chúng nhanh hơn nhiều
so với việc phải thực hiện một tập rời rạc các câu lệnh SQL tương đương theo cách thông thường.
Thủ tục lưu trữ cho phép chúng ta thực hiện cùng một yêu cầu bằng một câu lệnh đơn giản thay vì phải sử dụng nhiều dòng lệnh SQL. Điều này sẽ làm giảm thiểu sự lưu thông trên mạng.
Thay vì cấp phát quyền trực tiếp cho người sử dụng trên các câu lệnh SQL và trên các đối tượng cơ sở dữ liệu, ta có thể cấp phát quyền cho người sử dụng thông qua các thủ tục lưu trữ, nhờ đó tăng khả năng bảo mật đối với hệ thống.
Các thủ tục lưu trữ trả về kết quả theo 4 cách:
Sử dụng các tham số output
Sử dụng các lệnh trả về giá trị, các lệnh này luôn trả về giá trị số nguyên.
Tập các giá trị trả về của mỗi câu lệnh SELECT có trong thủ tục lưu trữ hoặc của quá
trình gọi một thủ tục lưu trữ khác trong một thủ tục lưu trữ.
Một biến con trỏ toàn cục có thể tham chiếu từ bên ngoài thủ tục.

4.2 Các cấu trúc lệnh : if, for, while, fetch trong một thủ tục lưu trữ.

4.2 .1 Cấu trúc lênh IF, IF .. Else

Cú pháp :
If Điều kiện
Begin
<tập lệnh>
end

Cơ chế:
Nếu điều kiện đúng : tập lệnh thực hiện
Nếu tập hợp lệnh chỉ có một lệnh thì không cần Begin .. End

If Điều kiện
Begin
<tập lệnh 1>
end
else
Begin
<tập lệnh 2>
End
Cơ chế:
Nếu điều kiện đúng : tập lệnh 1 thực hiện, ngược lại tập lệnh 2 thực hiện
Nếu tập hợp lệnh 1 ,lệnh 2 chỉ có 1 lệnh thì không cần Begin .. End

4.2 .2 Cấu trúc lặp :While
Cú pháp :
While Biểu thức Điều kiện
Begin
<Tập lệnh>
End

Cơ chế:
Tập lệnh sẽ được thực hiện đến khi biểu thức điều kiện trả về False
Có thể dùng lệnh **Break** để thoát khỏi vòng lặp

4.2 .1 Lệnh *FETCH*
Cú pháp :
FETCH Hướng di chuyển From Tên_biến_Cursor Into Danh sách biến
Trong đó:
Hướng di chuyển :
NEXT: Di chuyển về sau
PRIOR : Di chuyển về trước
FIRST : Di chuyển về đầu
LAST : Di chuyển về cuối
ABSOLUTE n : di chuyển đến mẩu tin thứ n tính từ mẩu tin đầu tiên ,nếu n<0 : tính từ mẩu tin cuối
RELATIVE n :di chuyển đến mẩu tin thứ n tính từ mẩu tin hiện hành

Trong quá trình di chuyển để kiểm tra việc di chuyển có thành công hay không ta kiểm tra biến hệ thống @@FETCH_STATUS nếu <>0 thất bại

4.3 Thao tác với thủ tục lưu trữ

4.3.1 Tạo thủ tục

Thủ tục lưu trữ được tạo thông qua câu lệnh CREATE PROCEDURE.

```
CREATE PROCEDURE tên_thủ_tục [(danh_sách_tham_số)]
[WITH RECOMPILE|ENCRYPTION|RECOMPILE,ENCRYPTION]
AS
BEGIN
Các_câu_lệnh_của_thủ_tục
END
```

Trong đó:
WITH RECOMPILE: yêu cầu SQL Server biên dịch lại thủ tục lưu trữ mỗi khi được gọi. WITH ENCRYPTION: yêu cầu SQL Server mã hóa thủ tục lưu trữ.
Các_câu_lệnh_của_thủ_tục: Các lệnh T-SQL. Các lệnh này có thể nằm trong cặp
BEGIN…END hoặc không.

Ví dụ: Giả sử cần thực hiện các công việc theo thứ tự như sau:
Nhập một đơn đặt hàng mới của khách hàng có mã khách hàng là 3
Nhập các chi tiết đơn đặt hàng cho đơn đặt hàng trên.
Để thực hiện các công việc trên chúng ta cần các câu lệnh như sau:
Trước tiên nhập đơn đặt hàng cho khách hàng có mã khách hàng là 3 insert into orders
values(3, '7/22/2008')
Tiếp theo thêm các chi tiết đơn đặt hàng cho hóa đơn này. Giả sử rằng đơn đặt hàng có mã là 4 và khách hàng đặt một mặt hàng có mã là 1.
insert into orderdetail values(4, 1, 10)

Cách viết như trên có hạn chế là: trong quá trình làm việc sẽ có rất nhiều đơn đặt hàng mới, do đó người dùng sẽ phải viết đi viết lại những câu lệnh tương tự nhau cho các khách hàng khác nhau. Một cách giải quyết vấn đề này là dùng thủ tục lưu trữ và dùng tham số để nhận các thông tin thay đổi.

```
create procedure sp_InsertOrderAndOrderDetail @customerid
int,@orderdate datetime,@orderid int,@itemid int, @quantity decimal
as
begin
insert into orders values(@customerid, @orderdate)


insert into orderdetail
values(@orderid, @itemid, @quantity)
end
```

Thực hiện thủ tực lưu trữ này như sau:
sp_InsertOrderAndOrderDetail ‘3’, ‘22/7/2008’, ‘4’, ‘1’, ‘10’)

3.3.2 Lời gọi thủ tục

Thủ tục lưu trữ được gọi theo cấu trúc
Tên_thủ_tục_lưu _trữ [danh_sách_tham_số]
Cần lưu ý là danh sách tham số truyền vào trong lời gọi phải theo đúng thứ tự khai báo
các tham số trong thủ tục lưu trữ.
Nếu thủ tục được gọi từ một thủ tục khác, thực hiện bên trong một trigger hay phối hợp
với câu lệnh SELECT, cấu trúc như sau;
Exec Tên_thủ_tục_lưu _trữ [danh_sách_tham_số]

4.3.3 Biến trong thủ tục lưu trữ

Trong thủ tục lưu trũ có thể có các biến nhằm lưu các kết quả tính toán hay truy xuất từ
CSDL. Các biến trong thủ tục được khai báo bằng từ khóa DECLARE theo cấu trúc như sau:
DECLARE @tên_biến kiểu_dữ_liệu
Ví dụ:

```
create procedure sp_SelectCustomerWithMaxAge as
begin
declare @maxAge int
select @maxAge = max(year(getdate())-year(BIRTHDAY))
from customers
select CUSTOMERNAME, BIRTHDAY
from customers
```

where year(getdate())-year(BIRTHDAY)=@maxAge
end

4.3.4 Giá trị trả về trong thủ tục lưu trữ

Trong các ví dụ trước, nếu đối số truyền cho thủ tục khi có lời gọi đến thủ tục là biến, những thay đổi giá trị của biền trong thủ tục sẽ không được giữ lại khi kết thúc quá trình thực hiện thủ tục.
Ví dụ: Có thủ tục lưu trữ như sau

```
create procedure sp_TestOutput @a int, @b int, @c int
as
begin
Set @c = @a + @b
end
```

Thực thi thủ tục:

```
Declare @tong int
set @tong = 0
sp_TestOutput 100, 200, @tong

select @tong
```

Kết quả là 0.

Sử dụng tham **số OUTPUT:**
Trong trường hợp cần phải giữ lại giá trị của đối số sau khi kết thúc thủ tục, ta phải khai
báo tham số của thủ tục theo cú pháp như sau:
@tên_tham_số kiểu_dữ_liệu OUTPUT

Ví dụ trên được viết lại như sau:

```
create procedure sp_TestOutput
@a int,
@b int,
@c int output as
begin
Set @c = @a + @b
End
```

Thực thi thủ tục:

```
Declare @tong int
set @tong = 0
sp_TestOutput 100, 100, @tong output

select @tong
```

Kết quả là 200.

Sử dụng lệnh RETURN

Tương như như việc sử dụng tham số OUTPUT, câu lệnh RETURN trả về giá trị cho đối tượng thực thi stored procedure.

Ví dụ:

```
create procedure sp_TestReturn as
begin
declare @out int
select @out = count(*)
from customers return @out
end
```

Thực thi thủ tục lưu trữ:

```
declare @a int
exec  @a = sp_TestReturn
```

4.3.5 Tham số với giá trị mặc định

Các tham số được khai báo trong thủ tục có thể nhận các giá trị mặc định. Giá trị mặc định sẽ được gán cho tham số trong trường hợp không truyền đối số cho tham số khi có lời gọi đến thủ tục.

Tham số với giá trị mặc định được khai báo theo cú pháp như sau:

@tên_tham_số kiểu_dữ_liệu = giá_trị_mặc_định

Ví dụ:

```
create procedure sp_TestDefault @customerid int = 3
as
begin
select * from customers
where customerid = @customerid
end
```

Thực thi thủ tục lưu trữ theo giá trị mặc định của tham số.

sp_TestDefault

CUSTOMERID	CUSTOMERNAME	BIRTHDAY	GENDER	ADDRESS
3	Tran Viet Cuong	1980-01-01 00:00:00.000	1	45/2B Da Tuong

Thực thi thủ tục và truyền giá trị cho tham số:
sp_TestDefault 4

CUSTOMERID	CUSTOMERNAME	BIRTHDAY	GENDER	ADDRESS
4	Nguyen Van Dai	1955-03-04 00:00:00.000	1	76 Tran Phu

4.4 Sửa đổi thủ tục

Khi một thủ tục đã được tạo ra, ta có thể tiến hành định nghĩa lại thủ tục đó bằng câu lệnh
Cú pháp như sau:
ALTER PROCEDURE tên_thủ_tục [(danh_sách_tham_số)]
[WITH RECOMPILE|ENCRYPTION|RECOMPILE,ENCRYPTION]
AS
Begin
Các_câu_lệnh_của_thủ_tục
End

Câu lệnh này sử dụng tương tự như câu lệnh CREATE PROCEDURE. Việc sửa đổi lại
một thủ tục đã có không làm thay đổi đến các quyền đã cấp phát trên thủ tục cũng như không tác động đến các thủ tục khác hay trigger phụ thuộc vào thủ tục này.

4.5 Xóa thủ tục

Để xoá một thủ tục đã có, ta sử dụng câu lệnh DROP PROCEDURE với cú pháp như sau:

DROP PROCEDURE tên_thủ_tục

Khi xoá một thủ tục, tất cả các quyền đã cấp cho người sử dụng trên thủ tục đó cũng đồng thời bị xoá bỏ. Do đó, nếu tạo lại thủ tục, ta phải tiến hành cấp phát lại các quyền trên thủ tục đó.

BÀI 5: HÀM NGƯỜI DÙNG (USER DEFINED FUNCTION-UDF)

5.2 Hàm do người dùng định nghĩa

Hàm do người dùng định nghĩa được chia làm 3 loại: (1) scalar (hàm vô hướng), (2) inline
table-valued (hàm nội tuyến, giá trị trả về dạng bảng), (3) multi-statement table-valued (hàm bao gồm nhiều câu lệnh SQL bên trong, trả về giá trị dạng bảng)
Scalar UDF: được sử dụng để trả về một duy nhất một giá trị dựa trên một các tham số truyền vào. Ví dụ: ta có thể tạo ra một UDF vô hướng nhận Customerid là tham số và trả về CustomerName.
Inline table-valued: trả về một tập mẫu tin dựa trên một câu lệnh SQL duy nhất định nghĩa các dòng và các cột trả về.
Multi-statement table-value: cũng trả về kết quả là một tập hợp nhưng có thể dựa trên nhiều câu lệnh SQL.

5.2.1 Hàm vô hướng (Scalar UDF)

Scarlar UDF được tạo ra bằng câu lệnh CREATE FUNCTION có cấu trúc như sau;

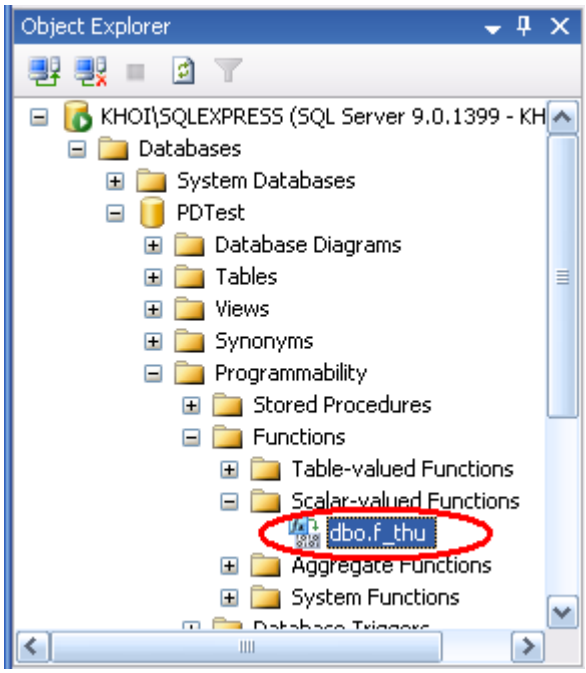
CREATE FUNCTION tên_hàm ([danh_sách_tham_số]) RETURNS (kiểu_trả_về_của_hàm) AS BEGIN các_câu_lệnh_của_hàm
END

Ví dụ:

Câu lệnh dưới đây định nghĩa hàm tính ngày trong tuần (thứ trong tuần) của một giá trị
kiểu ngày

```
create function  f_ thu(@ngay datetime)
returns nvarchar(10)
as begin
declare @st nvarchar(10)
select @st=case datepart(dw,@ngay)
when 1 then N'chủ nhật' when 2 then N'thứ  hai' when 3 then N 'thứ  ba'
when 4 then N 'thứ  tư' when 5 then N 'thứ  năm' when 6 then N 'thứ  sáu'
else N 'thứ  bảy'
end
return (@st)  /* trị  trả  về  của hàm */
end
```

Sau khi chạy thành công, hàm trở thành một đối tượng trong CSDL và có thể được truy
xuất như các hàm được xây dựng sẵn trong SQL Server 2005 Express Edition.



Ví dụ:

```
select CUSTOMERNAME, dbo.f_thu(BIRTHDAY)
from customers
```

CUSTOMERNAME	(No column name)
Cao Van Trung	thứ sáu
Tran Van Phuc	thứ hai
Tran Viet Cuong	thứ ba
Nguyen Van Dai	thứ sáu
Le Thi Hoa	thứ hai
Nguyen Thanh Thai	thứ sáu
Cao Van Chung	thứ bảy
NGuyen Van An	thứ sáu
NGuyen Van An	thứ sáu

5.2.2 Hàm nội tuyến - Inline UDF

Hàm nội tuyến được định nghĩa bằng lệnh CREATE FUNCTION.
CREATE FUNCTION tên_hàm ([danh_sách_tham_số])
RETURNS TABLE AS
RETURN (câu_lệnh_select)
Cú pháp của hàm nội tuyến phải tuân theo các qui tắc sau:
Kiểu trả về của hàm phải được chỉ định bởi mệnh đề RETURNS TABLE.
Trong phần thân của hàm chỉ có duy nhất một câu lệnh RETURN xác định giá trị trả về
của hàm thông qua duy nhất một câu lệnh SELECT. Ngoài ra, không sử dụng bất kỳ câu lệnh
nào khác trong phần thân của hàm.
Ví dụ: Ví dụ dưới đây lấy ra các khách hàng tùy thuộc vào giá trị mã khách hàng truyền
vào cho tham số.

```
create function f_SelectCustomer
(@customerid int)
returns table as
return (select * from customers
where customerid > @customerid)
```

Việc gọi các hàm nội tuyến cũng tương tự như việc gọi các hàm vô hướng.
Ví dụ:

```
select tmp.CUSTOMERNAME, o.ORDERDATE
from orders o inner join dbo.f_SelectCustomer(3) as tmp on o.customerid =
tmp.customerid
```

CUSTOMERNAME	ORDERDATE
Cao Van Trung	2007-12-06 00:00:00.000

5.2.3 Hàm bao gồm nhiều câu lệnh bên trong – Multi statement UDF

Hàm này cũng được định nghĩa bằng lệnh CREATE FUNCTION

CREATE FUNCTION tên_hàm ([danh_sách_tham_số])
RETURNS @biến_bảng TABLE định_nghĩa_bảng
AS
BEGIN
các_câu_lệnh_trong_thân_hàm
RETURN
END

Lưu ý: sau từ khóa RETURNS là một biến bảng được định nghĩa. Và sau từ khóa
RETURN ở cuối hàm không có tham số nào đi kèm.

Ví dụ:

```
create function f_SelectCustomer (@customerid int)
returns @myCustomers table
(
customerid int,
customername nvarchar(50), orderdate datetime
)
as begin
if @customerid = 0
insert into @myCustomers
select c.customerid, c.customername, o.orderdate
from customers c inner join orders o on o.customerid = c.customerid
else
return
end

insert into @myCustomers
select c.customerid, c.customername, o.orderdate
from customers c inner join orders o on c.customerid = o.customerid
where c.customerid = @customerid
```

Việc gọi hàm multi statement UDF cũng tương tự các loại hàm khác:

*select * from f_SelectCustomer(0)*

customerid	customername	orderdate
6	Cao Van Trung	2007-12-06 00:00:00.000
3	Tran Viet Cuong	2008-01-01 00:00:00.000
3	Tran Viet Cuong	2008-05-01 00:00:00.000

*select * from f_SelectCustomer(3)*

customerid	customername	orderdate
3	Tran Viet Cuong	2008-01-01 00:00:00.000
3	Tran Viet Cuong	2008-05-01 00:00:00.000

5.2.4 Thay đổi hàm

Dùng lệnh ALTER FUNCTION để thay đổi định nghĩa hàm. Cấu trúc của câu lệnh
ALTER FUNCTION tương tự như CREATE FUNCTION
Ví dụ:
alter function f_SelectCustomer
(@customerid int)
returns table as
*return (select * from customers*
where customerid > @customerid)

5.2.5 Xóa hàm

Dùng lệnh DROP FUNCTION để xóa hàm. Cấu trúc lệnh DROP FUNCTION như sau:
DROP FUNCTION tên_hàm
Ví dụ:
drop function f_thu
Tương tự như thủ tục lưu trữ, khi hàm bị xóa các quyền cấp cho người dùng trên hàm đócũng bị xóa. Do đó khi định nghĩa lại hàm này, ta phải cấp lại quyền cho các người dùng.

BÀI 6: TRIGGER

6.1 Trigger

Trigger là một dạng đặc biệt của thủ tục lưu trữ, được thực thi một cách tự động khi có sự thay đổi dữ liệu (do tác động của câu lệnh INSERT, UPDATE, DELETE) trên một bảng nào
đó.

6.1.1 Các đặc điểm của trigger

Trigger chỉ thực thi tự động thông qua các sự kiện mà không thực hiện bằng tay.
Trigger sử dụng được với khung nhìn.
Khi trigger thực thi theo các sự kiện Insert hoặc Delete thì dữ liệu khi thay đổi sẽ được
chuyển sang các bảng INSERTED và DELETED, là 2 bảng tạm thời chỉ chứa trong bộ nhớ,
các bảng này chỉ được sử dụng với các lệnh trong trigger. Các bảng này thường được sử dụng
để khôi phục lại phần dữ liệu đã thay đổi (roll back).
Trigger chia thành 2 loại INSTEAD OF và AFTER: INSTEAD OF là loại trigger mà hoạt động của sự kiện gọi trigger sẽ bị bỏ qua và thay vào đó là các lệnh trong trigger được thực hiện. AFTER trigger là loại ngầm định, khác với loại INSTEAD OF thì loại trigger này sẽ thực hiện các lệnh bênh trong sau khi đã thực hiện xong sự kiện kích hoạt trigger.

6.1.2 Các trường hợp sử dụng trigger

Sử dụng Trigger khi các biện pháp bảo đảm toàn vẹn dữ liệu khác không bảo đảm được. Các công cụ này sẽ thực hiện kiểm tra tính toán vẹn trước khi đưa dữ liệu vào CSDL, còn Trigger thực hiện kiểm tra tính toàn vẹn khi công việc đã thực hiện
Khi CSDL chưa được chuẩn hóa (Normalization) thì có thể xảy ra dữ liệu thừa, chứa ở
nhiều vị trí trong CSDL thì yêu cầu đặt ra là dữ liệu cần cập nhật thống nhất trong mọi nơi. Trong trường hợp này ta phải sử dụng Trigger.Khi xảy ra thay đổi dây chuyền dữ liệu giữa các bảng với nhau (khi dữ liệu bảng này thay đổi thì dữ liệu trong bảng khác cũng được thay đổi theo).

6.1.3 Khả năng sau của trigger

Một trigger có thể nhận biết, ngăn chặn và huỷ bỏ được những thao tác làm thay đổi trái
phép dữ liệu trong cơ sở dữ liệu.
Các thao tác trên dữ liệu (xoá, cập nhật và bổ sung) có thể được trigger phát hiện ra và tự động thực hiện một loạt các thao tác khác trên cơ sở dữ liệu nhằm đảm bảo tính hợp lệ của dữ liệu.
Thông qua trigger, ta có thể tạo và kiểm tra được những mối quan hệ phức tạp hơn giữa
các bảng trong cơ sở dữ liệu mà bản thân các ràng buộc không thể thực hiện được.

6.2 Định nghĩa trigger

Câu lệnh CREATE TRIGGER được sử dụng để đinh nghĩa trigger và có cấu trúc như sau:

CREATE TRIGGER tên_trigger
ON tên_bảng
FOR {[INSERT][,][UPDATE][,][DELETE]} AS
[IF UPDATE(tên_cột)
[AND UPDATE(tên_cột)|OR UPDATE(tên_cột)]
...]
các_câu_lệnh_của_trigger

Lưu ý: Như đã nói ở trên, chuẩn SQL định nghĩa hai bảng logic INSERTED và DELETED để sử dụng trong các trigger. Cấu trúc của hai bảng này tương tự như cấu trúc của bảng mà trigger tác động. Dữ liệu trong hai bảng này tuỳ thuộc vào câu lệnh tác động lên bảng làm kích hoạt trigger; cụ thể trong các trường hợp sau:

Khi câu lệnh DELETE được thực thi trên bảng, các dòng dữ liệu bị xoá sẽ được sao chép vào trong bảng DELETED. Bảng INSERTED trong trường hợp này không có dữ liệu.

Dữ liệu trong bảng INSERTED sẽ là dòng dữ liệu được bổ sung vào bảng gây nênsự kích hoạt đối với triggerbằng câu lệnh INSERT. Bảng DELETED trong trường hợp này không có dữ liệu.

Khi câu lệnh UPDATE được thực thi trên bảng, các dòng dữ liệu cũ chịu sự tác động của câu lệnh sẽ được sao chép vào bảng DELETED, còn trong bảng INSERTED sẽ là các dòng sau khi đã được cập nhật.

Hoạt động	Bảng INSERTED	Bảng DELETED
INSERT	dữ liệu được insert	không có dữ liệu
DELETE	không có dữ liệu	dữ liệu bị xóa
UPDATE	dữ liệu được cập nhật	dữ liệu trước khi cập nhật

6.3. Trigger với Insert, Update, Delete

Các thao tác trên dữ liệu (xoá, cập nhật và bổ sung) có thể được trigger phát hiện ra và tự động thực hiện một loạt các thao tác khác trên cơ sở dữ liệu nhằm đảm bảo tính hợp lệ của dữ liệu.

Ví dụ 1: Ví dụ dưới đây minh họa việc trigger được kích hoạt khi thêm dữ liệu vào bảng
CUSTOMERS

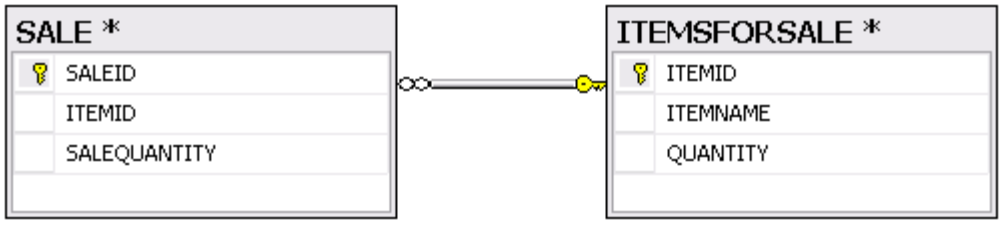
```
if exists (select name from sysobjects
where name = 't_CheckCustomerName' and type = 'TR')
drop trigger t_CheckCustomerName go
create trigger t_CheckCustomerName on customers for insert
as
declare @lengthOfName int
select @lengthOfName = len(inserted.customername)
from inserted
if @lengthOfName <=1
print N'Tên không hợp lệ' rollback tran
go
```

Thêm một khách hàng mới có tên là A

```
insert into customers
values('A', '5/5/1978', 'True', '35 Hung Vuong')
```

```
Tên không hợp lệ
Msg 3609, Level 16, State 1, Line 2
The transaction ended in the trigger. The batch has been aborted.
```

Ví dụ 2: Ví dụ dưới đây minh họa trigger được kích hoạt khi có sự thay đổi mang tính đây chuyền giữa các bảng.
Giả sử có CSDL như sau:



Với dữ liệu trong từng bảng là:

ITEMID	ITEMNAME	QUANTITY
1	LAPTOP	100.00
2	PPC	2000.00
3	IPOD	10.00

SALEID	ITEM...	SALEQUANTITY
1	1	10.00
2	2	10.00

Giả sử có một khách hàng mua 10 đơn vị mặt hàng LAPTOP. Khi đó số lượng LAPTOP
trong bảng ITEMFORSALE sẽ giảm xuống còn 90. Trigger dưới đây sẽ thực hiện công việc
đó.

```
if exists (select name from sysobjects
```

where name = 't_DecreaseQuantityOfItemForSale')
drop trigger t_DecreaseQuantityOfItemForSale go
create trigger t_DecreaseQuantityOfItemForSale on SALE
for insert as
update ITEMSFORSALE
set itemsforsale.quantity = itemsforsale.quantity - inserted.salequantity
from itemsforsale inner join inserted
on itemsforsale.itemid = inserted.itemid go
Thực hiện thêm dòng vào bảng SALE
insert into sale values(1, 10)

ITEMID	ITEMNAME	QUANTITY
1	LAPTOP	90.00
2	PPC	100.00
3	IPOD	20.00

SALEID	ITEM...	SALEQUANTITY
1	1	10.00
2	2	10.00
9	1	10.00

Ví dụ 3: Ví dụ này minh họa cũng minh họa trigger được kích hoạt khi có sự thay đổi
mang tính dây chuyền giữa các bảng nhưng trong trường hợp này dữ liệu thay đổi liên quan
đến nhiều dòng.
Giả sử người quản lý muốn thay đổi số lượng bán mặt hàng LAPTOP trong bãng SALE
lên thêm 5 đơn vị. Như vậy từ kết quả ví dụ 2, ta thấy cần phải giảm số lượng LAPTOP trong bảng ITEMSFORSALE xuống 10 đơn vị. Tuy nhiên, trong thực tế khi số lượng các dòng trong bảng SALE rất lớn, khi đó phải sử dụng trigger:
if exists (select name from sysobjects
where name = 't_DecreaseSumQuantityOfItemForSale')
drop trigger t_DecreaseSumQuantityOfItemForSale go
create trigger t_DecreaseSumQuantityOfItemForSale on ***SALE***
for update *as*
if update(salequantity)
update ITEMSFORSALE
set itemsforsale.quantity = itemsforsale.quantity -
(select sum(inserted.salequantity - deleted.salequantity)
from deleted inner join inserted
on deleted.saleid = inserted.saleid
where inserted.itemid = itemsforsale.itemid)
where itemsforsale.itemid in (select inserted.itemid
from inserted)

Thực hiện cập nhật cho bảng SALE:
update sale
set salequantity = salequantity + 10
where itemid = 1

SALEID	ITEM...	SALEQUANTITY
1	1	20.00
2	2	10.00
9	1	20.00

ITEMID	ITEMNAME	QUANTITY
1	LAPTOP	70.00
2	PPC	100.00
3	IPOD	20.00

Ví dụ 4: Ví dụ này minh họa INSTEAD OF trigger. Trigger dưới đây sẽ không cho thực
hiện thao tác xóa trên bảng CUSTOMERS.
create trigger t_RollbackDelete on customers
after delete
as
rollback tran

6.4 Kích hoạt trigger dựa trên sự thay đổi dữ liệu trên cột

Thay vì chỉ định một trigger được kích hoạt trên một bảng, ta có thể chỉ định trigger được
kích hoạt và thực hiện những thao tác cụ thể khi việc thay đổi dữ liệu chỉ liên quan đến một số
cột nhất định nào đó của cột. Trong trường hợp này, ta sử dụng mệnh đề IF UPDATE trong
trigger. IF UPDATE không sử dụng được đối với câu lệnh DELETE.

Trở lại ví dụ 3 trong phần định nghĩa trigger:
if exists (select name from sysobjects
where name = 't_DecreaseSumQuantityOfItemForSale')
drop trigger t_DecreaseSumQuantityOfItemForSale go
create trigger t_DecreaseSumQuantityOfItemForSale on SALE
for update as
if update(salequantity)
update ITEMSFORSALE
set itemsforsale.quantity = itemsforsale.quantity -
(select sum(inserted.salequantity - deleted.salequantity)
from deleted inner join inserted
on deleted.saleid = inserted.saleid
where inserted.itemid = itemsforsale.itemid)

where itemsforsale.itemid in (select inserted.itemid from inserted)

Trong ví dụ này trigger sẽ được kích hoạt khi có sự thay đổi dữ liệu trong cột *salequantity* của bảng Sale. Nếu có sự thay đổi dữ liệu trên các cột khác thì trigger sẽ không được kích hoạt.
Câu lệnh dưới đây không làm cho trigger kích hoạt.
update sale
set itemid = 3
where itemid = 2
Mệnh đề IF UPDATE có thể xuất hiện nhiều lần trong phần thân của trigger. Khi đó, mệnh đề IF UPDATE nào đúng thì phần câu lệnh của mệnh đề đó sẽ được thực thi khi trigger được kích hoạt.

6.5 Sử dụng trigger và giao tác (TRANSACTION)

Khi một trigger được kích hoạt, SQL Server luôn tạo ra một giao tác theo dõi những thay
đổi do câu lệnh kích hoạt trigger hoặc do bản thân trigger gây ra. Sự theo dõi này cho phép
CSDL quay trở lại trạng thái trước đó.
Ví dụ: Ví dụ dưới đây xây dựng trigger không cho phép nhập vào một bản ghi trong bảng SALE khi số lượng hàng bán lớn hơn số lượng hàng thực tế còn lại trong bảng ITEMSFORSALE
if exists (select name from sysobjects
where name = 't_CheckQuantity' and type = 'TR')
drop trigger t_CheckQuantity go

create trigger t_CheckQuantity on sale
for insert as
declare @insertedQuantity decimal(18,2) declare @currentQuantity decimal(18,2) declare @itemid int

select @itemid = itemid from inserted
select @insertedQuantity = salequantity from inserted select @currentQuantity = quantity
from itemsforsale
where itemid = @itemid

if(@currentquantity < @insertedquantity)
print N'số lượng nhập vào lớn hơn số lượng hiện có'
rollback tran
Tiến hành thêm vào bảng SALE số liệu như sau:
insert into sale values(2, 1000)

6.6 DDL TRIGGER

Được giới thiệu trong SQL Server 2005, khác với DML trigger được kích hoạt khi có sự thay đổi dữ liệu trên bảng, DDL trigger được thiết kế để đáp ứng lại các sự kiện diễn ra trên server hay trên CSDL. Một DDL trigger có thể được kích hoạt khi người dùng thực hiện các lệnh CREATE TABLE hay DROP TABLE. Ở cấp độ server, DDL trigger có thể đ ược kích hoạt khi có một tài khoản mới được tạo ra
DDL trigger được lưu trữ trong CSDL mà DDL trigger được gắn vào. Với các Server
DDL Trigger theo dõi các thay đổi ở cấp độ Server, được lưu trữ trong CSDL master.
DDL trigger được tạo ra cũng bằng câu lệnh CREATE TRIGGER với cấu trúc như sau:
CREATE TRIGGER tên_trigger
ON { ALL SERVER | DATABASE } FOR { loại_sự_kiện } [,...n]
AS { các_câu_lệnh_SQL}
Trong đó:
ALL SERVER | DATABASE: quy định trigger sẽ kích hoạt dựa trên các sự kiện diễn ra
trên Server hay các sự kiện diễn ra trên CSDL.
loại_sự_kiện: là một sự kiện đơn ở cấp độ Server hay cấp độ CSDL làm kích hoạt DDL
trigger như: CREATE_TABLE, ALTER_TABLE, DROP_TABLE…
Ví dụ 1: Câu lệnh dưới đây xây dựng một trigger được kích hoạt khi xảy ra các sự kiện ở
cấp độ CSDL. Trigger này sẽ ngăn chặn các lệnh DROP TABLE và ALTER TABLE.
create trigger t_safety on database
for CREATE_TABLE, DROP_TABLE

as
print N'Phải xóa trigger t_safety trước khi ALTER hay DROP bảng'
rollback tran

Tiến hành xóa bảng ORDERDETAIL
drop table orderdetail

Ví dụ 2: Câu lệnh dưới đây xây dựng một trigger được kích hoạt khi xảy ra các sự kiện ở cấp độ Server. Trigger này sẽ ngăn chặn việc tạo ra một account login mới

*IF EXISTS (SELECT * FROM sys.server_triggers WHERE name = 't_DoNotAllowCreateNewLogin') DROP TRIGGER t_DoNotAllowCreateNewLogin*
ON ALL SERVER GO
CREATE TRIGGER t_DoNotAllowCreateNewLogin
ON ALL SERVER
FOR CREATE_LOGIN AS
PRINT N'Phải DROP trigger t_DoNotAllowCreateNewLogin trước khi tạo account'
rollback
GO
Tiến hành tạo một account login mới:
create login test with password = '123456'

6.7 Vô hiệu hóa/Khôi phục lại trigger (Disable/Enable TRIGGER)

Trigger cần bị vô hiệu hóa trong một số trường hợp:
Trigger gây ra lỗi trong quá trình xử lý CSDL
Quá trình nhập hay khôi phục những dữ liệu không thỏa trigger.
Vô hiệu hóa trigger bằng lệnh DISABLE TRIGGER có cấu trúc như sau:
DISABLE TRIGGER tên_trigger
ON { tên_đối_tượng | DATABASE | SERVER }
Ví dụ 1: Ví dụ này sẽ vô hiệu hóa trigger *t_DoNotAllowCreateNewLogin*
disable trigger t_DoNotAllowCreateNewLogin

on all server
Tiến hành tạo một account login mới:
create login newLogin with password = '12345'

Ví dụ 2: Ví dụ này sẽ khôi phục lại trigger *t_ DoNotAllowCreateNewLogin enable trigger t_DoNotAllowCreateNewLogin*
on all server
Tiến hành tạo một account login mới:
create login newLogin1 with password = '12345'

BÀI 7: GIAO TÁC (TRANSACTION)

7.1 Khái niệm giao tác

Là tập hợp các lệnh sẽ được thực hiện nếu tất cả đều thành công, nếu có một lệnh thất bại ,thì sẽ không có lệnh nào được thực hiện
Ví dụ : giao tác chuyển tiền của ngân hàng : chuyển số lượng N từ tài khoản A sang tài khoản B, các công việc được thực hiện :
TaiKhoanA=TaiKhoanA-N
TaiKhoanB=TaiKhoanB+N
Hai công việc này sẽ được thực hiện nếu không có lệnh nào gây lổi

7.2 Tạo giao tác
Cú pháp xây dựng 1 giao tác trong SQL Server:
Lệnh Bắt đầu 1 Giao tác : Begin Tran
Lệnh kết thúc thành công 1 giao tác : Commit Tran
Lệnh kết thúc thất bại 1 giao tác : Rollback Tran

Để kiểm tra các lệnh thực hiện có thành công hay không :truy cập đến giá trị của biến @@Error, nếu @@Error<>0 : thất bại

Cú pháp :

```
Begin Tran
< tập các lệnh của giao tác>
If @@error<>0
begin
Rollback tran                    -- Giao tac that bai
end
Else
```

Commit tran

7.3 Sử dụng trigger và giao tác (TRANSACTION)

Khi một trigger được kích hoạt, SQL Server luôn tạo ra một giao tác theo dõi những thay đổi do câu lệnh kích hoạt trigger hoặc do bản thân trigger gây ra. Sự theo dõi này cho phép CSDL quay trở lại trạng thái trước đó.

Ví dụ: Ví dụ dưới đây xây dựng trigger không cho phép nhập vào một bản ghi trong bảng SALE khi số lượng hàng bán lớn hơn số lượng hàng thực tế còn lại trong bảng ITEMSFORSALE

```
if exists (select name from sysobjects
where name = 't_CheckQuantity' and type = 'TR')
drop trigger t_CheckQuantity go

create trigger t_CheckQuantity on sale
for insert as
declare @insertedQuantity decimal(18,2) declare @currentQuantity
decimal(18,2) declare @itemid int


select @itemid = itemid from inserted
select @insertedQuantity = salequantity  from inserted select
@currentQuantity = quantity
from itemsforsale
where itemid = @itemid


if(@currentquantity < @insertedquantity)
print N'số lượng nhập vào lớn hơn số lượng hiện có'
rollback tran
```

Tiến hành thêm vào bảng SALE số liệu như sau:

```
insert into sale values(2, 1000)
```

TÀI LIỆU THAM KHẢO

- Giáo trình SQL Server của Nhà xuất bản Lao Động - Xã Hội
- Giáo trình SQL Server 2000 của Đại Học Huế