CHƯƠNG I THỦ TỤC LƯU TRỮ

1.1 Thủ tục lưu trữ (stored procedure)

1.1.1 Các khái niệm

Như đã đề cập ở các chương trong SQL Server 1, SQL được thiết kế và cài đặt như là một ngôn ngữ để thực hiện các thao tác trên cơ sở dữ liệu như tạo lập các cấu trúc trong cơ sở dữ liệu, bổ sung, cập nhật, xoá và truy vấn dữ liệu trong cơ sở dữ liệu. Các câu lệnh SQL được người sử dụng viết và yêu cầu hệ quản trị cơ sở dữ liệu thực hiện theo chế độ tương tác.

Các câu lệnh SQL có thể được nhúng vào trong các ngôn ngữ lập trình, thông qua đó chuỗi các thao tác trên cơ sở dữ liệu được xác định và thực thi nhờ vào các câu lệnh, các cấu trúc điều khiển của bản thân ngôn ngữ lập trình được sử dụng.

Với thủ tục lưu trữ, một phần nào đó khả năng của ngôn ngữ lập trình được đưa vào trong ngôn ngữ SQL. Một thủ tục là một đối tượng trong cơ sở dữ liệu bao gồm một tập nhiều câu lệnh SQL được nhóm lại với nhau thành một nhóm với những khả năng 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. Sử dụng các thủ tục lưu trữ trong cơ sở dữ liệu sẽ giúp tăng hiệu năng của cơ sở dữ liệu, mang lại các lợi ích sau:

Đơn giản hoá các thao tác trên cơ sở dữ liệu nhờ vào khả năng module hoá các thao tác này.

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.

1.1.2 Tạo thủ tục lưu trữ

Thủ tục lưu trữ được tạo bởi câu lệnh CREATE PROCEDURE với cú pháp như sau:

CREATE PROCEDURE tên_thủ_tục [(danh_sách_tham_số)]

[WITH RECOMPILE|ENCRYPTION|RECOMPILE,ENCRYPTION]

AS

Các_câu_lệnh_của_thủ_tục

Trong đó:

tên_thủ_tục	Tên của thủ tục cần tạo. Tên phải tuân theo qui tắc định danh và không được vượt quá 128 ký tự.
danh_sách_tham_số	Các tham số của thủ tục được khai báo ngay sau tên thủ tục và nếu thủ tục có nhiều tham số thì các khai báo phân cách nhau bởi dấu phẩy. Khai báo của mỗi một tham số tối thiểu phải bao gồm hai phần: • Tên tham số được bắt đầu bởi dấu @. • Kiểu dữ liệu của tham số **Ví dụ**: @mamonhoc nvarchar(10)
RECOMPILE	Thông thường, thủ tục sẽ được phân tích, tối ưu và dịch sẵn ở lần gọi đầu tiên. Nếu tuỳ chọn WITH RECOMPILE được chỉ định, thủ tục sẽ được dịch lại mỗi khi được gọi
ENCRYPTION	Thủ tục sẽ được mã hoá nếu tuỳ chọn WITH ENCRYPTION được chỉ định. Nếu thủ tục đã được mã hoá, ta không thể xem được nội dung của thủ tục.

các_câu_lệnh_của_thủ_tục	Tập hợp các câu lệnh sử dụng trong nội dung thủ tục. Các câu lệnh này có thể đặt trong cặp từ khoá BEGIN...END hoặc có thể không.

Ví dụ 1.1: Giả sử ta cần thực hiện một chuỗi các thao tác như sau trên cơ sở dữ liệu

1. Bổ sung thêm môn học cơ sở dữ liệu có mã TI-005 và số đơn vị học trình là 5 vào bảng MONHOC
2. Lên danh sách nhập điểm thi môn cơ sở dữ liệu cho các sinh viên học lớp có mã C24102 (tức là bổ sung thêm vào bảng DIEMTHI các bản ghi với cột MAMONHOC nhận giá trị TI-005, cột MASV nhận giá trị lần lượt là mã các sinh viên học lớp có mã C24105 và các cột điểm là NULL).

Nếu thực hiện yêu cầu trên thông qua các câu lệnh SQL như thông thường, ta phải thực thi hai câu lệnh như sau:

```
INSERT INTO MONHOC
VALUES('TI-005','Cơ sở dữ liệu',5)

INSERT INTO DIEMTHI(MAMONHOC,MASV)
  SELECT 'TI-005',MASV
  FROM SINHVIEN
  WHERE MALOP='C24102'
```

Thay vì phải sử dụng hai câu lệnh như trên, ta có thể định nghĩa môt thủ tục lưu trữ với các tham số vào là @mamonhoc, @tenmonhoc, @sodvht và @malop như sau:

```
CREATE PROC sp_LenDanhSachDiem(
@mamonhoc NVARCHAR(10),
@tenmonhoc NVARCHAR(50),
@sodvht SMALLINT,
@malop NVARCHAR(10))
AS
BEGIN
INSERT INTO monhoc
```

VALUES(@mamonhoc,@tenmonhoc,@sodvht)

INSERT INTO diemthi(mamonhoc,masv)
SELECT @mamonhoc,masv
FROM sinhvien
WHERE malop=@malop
END

Khi thủ tục trên đã được tạo ra, ta có thể thực hiện được hai yêu cầu đặt ra ở trên một cách đơn giản thông qua lời gọi thủ tục:

sp_LenDanhSachDiem 'TI-005','Cơ sở dữ liệu',5,'C24102'

1.1.3 Lời gọi thủ tục lưu trữ

Như đã thấy ở ví dụ ở trên, khi một thủ tục lưu trữ đã được tạo ra, ta có thể yêu cầu hệ quản trị cơ sở dữ liệu thực thi thủ tục bằng lời gọi thủ tục có dạng:

tên_thủ_tục [***danh_sách_các_đối_số***]

Số lượng các đối số cũng như thứ tự của chúng phải phù hợp với số lượng và thứ tự của các tham số khi định nghĩa thủ tục.

Trong trường hợp lời gọi thủ tục được thực hiện bên trong một thủ tục khác, bên trong một trigger hay kết hợp với các câu lệnh SQL khác, ta sử dụng cú pháp như sau:

EXECUTE tên_thủ_tục [danh_sách_các_đối_số]

Thứ tự của các đối số được truyền cho thủ tục có thể không cần phải tuân theo thứ tự của các tham số như khi định nghĩa thủ tục nếu tất cả các đối số được viết dưới dạng:

@tên_tham_số = giá_trị

Ví dụ 1.2: Lời gọi thủ tục ở ví dụ trên có thể viết như sau:

sp_LenDanhSachDiem @malop='C24102',
@tenmonhoc='Cơ sở dữ liệu',
@mamonhoc='TI-005',
@sodvht=5

1.1.4 Sử dụng biến trong thủ tục

Ngoài những tham số được truyền cho thủ tục, bên trong thủ tục còn có thể sử dụng các biến nhằm lưu giữ các giá trị tính toán được hoặc truy xuất được từ cơ sở dữ liệu. Các biến trong thủ tục được khai báo bằng từ khoá DECLARE theo cú pháp như sau:

DECLARE @tên_biến kiểu_dữ_liệu

Tên biến phải bắt đầu bởi ký tự @ và tuân theo qui tắc về định danh. Ví dụ dưới đây minh hoạ việc sử dụng biến trong thủ tục

Ví dụ 1.3: Trong định nghĩa của thủ tục dưới đây sử dung các biến chứa các giá trị truy xuất được từ cơ sở dữ liệu.

```
CREATE PROCEDURE sp_Vidu(
@malop1 NVARCHAR(10),
@malop2 NVARCHAR(10))
AS
DECLARE @tenlop1 NVARCHAR(30)
DECLARE @namnhaphoc1 INT
DECLARE @tenlop2 NVARCHAR(30)
DECLARE @namnhaphoc2 INT
SELECT @tenlop1=tenlop,
@namnhaphoc1=namnhaphoc
FROM lop WHERE malop=@malop1

SELECT @tenlop2=tenlop,
  @namnhaphoc2=namnhaphoc
FROM lop WHERE malop=@malop2

PRINT @tenlop1+' nhap hoc nam '+str(@namnhaphoc1)
print @tenlop2+' nhap hoc nam '+str(@namnhaphoc2)

IF @namnhaphoc1=@namnhaphoc2
PRINT 'Hai lớp nhập học cùng năm'
```

```
ELSE
PRINT 'Hai lớp nhập học khác năm'
```

1.1.5 Giá trị trả về của tham số 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 biến trong thủ tục sẽ không được giữ lại khi kết thúc quá trình thực hiện thủ tục.

Ví dụ 1.4: Xét câu lệnh sau đây

```
CREATE PROCEDURE sp_Conghaiso(@a  INT,@b INT, @c INT)
AS
SELECT @c=@a+@b
```

Nếu sau khi đã tạo thủ tục với câu lệnh trên, ta thực thi một tập các câu lệnh như sau:

```
DECLARE @tong INT
SELECT @tong=0
EXECUTE sp_Conghaiso 100,200,@tong
SELECT @tong
```

Câu lệnh "SELECT @tong" cuối cùng trong loạt các câu lệnh trên sẽ cho kết quả là: 0

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
```

hoặc:

```
@tên_tham_số  kiểu_dữ_liệu  OUT
```

và trong lời gọi thủ tục, sau đối số được truyền cho thủ tục, ta cũng phải chỉ định thêm từ khoá OUTPUT (hoặc OUT)

Ví dụ 1.5: Ta định nghĩa lại thủ tục ở ví dụ 1.4 như sau:

```
CREATE PROCEDURE sp_Conghaiso(
@a     INT,
@b     INT,
@c     INT OUTPUT)
```

AS

SELECT @c=@a+@b

và thực hiện lời gọi thủ tục trong một tập các câu lệnh như sau:

DECLARE @tong INT

SELECT @tong=0

EXECUTE sp_Conghaiso 100,200,@tong OUTPUT

SELECT @tong

thì câu lệnh "SELECT @tong" sẽ cho kết quả là: 300

1.1.6 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 5.6: Trong câu lệnh dưới đây:

CREATE PROC sp_TestDefault(

@tenlop NVARCHAR(30)=NULL,

@noisinh NVARCHAR(100)='Huế')

AS

BEGIN

IF @tenlop IS NULL

SELECT hodem,ten

FROM sinhvien INNER JOIN lop

ON sinhvien.malop=lop.malop

WHERE noisinh=@noisinh

ELSE

SELECT hodem,ten

FROM sinhvien INNER JOIN lop

ON sinhvien.malop=lop.malop

WHERE noisinh=@noisinh AND

tenlop=@tenlop

END

thủ tục sp_TestDefault được định nghĩa với tham số @tenlop có giá trị mặc định là NULL và tham số @noisinh có giá trị mặc định là Huế. Với thủ tục được định nghĩa như trên, ta có thể thực hiện các lời gọi với các mục đích khác nhau như sau:

Cho biết họ tên của các sinh viên sinh tại Huế:

sp_testdefault

Cho biết họ tên của các sinh viên lớp Tin K24 sinh tại Huế:

sp_testdefault @tenlop='Tin K24'

Cho biết họ tên của các sinh viên sinh tại Nghệ An:

sp_testDefault @noisinh=N'Nghệ An'

Cho biết họ tên của các sinh viên lớp Tin K26 sinh tại Đà Nẵng:

sp_testdefault @tenlop='Tin K26',@noisinh='Đà Nẵng'

1.1.7 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 ALTER PROCEDURE có cú pháp như sau:

ALTER PROCEDURE tên_thủ_tục [(danh_sách_tham_số)]

[WITH RECOMPILE|ENCRYPTION|RECOMPILE,ENCRYPTION]

AS

Các_câu_lệnh_Của_thủ_tục

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.

1.1.8 Xoá 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.

CHƯƠNG II TRIGGER

Trong chương 1, ta đã biết các ràng buộc được sử dụng để đảm bảo tính toàn vẹn dữ liệu trong cơ sở dữ liệu. Một đối tượng khác cũng thường được sử dụng trong các cơ sở dữ liệu cũng với mục đích này là các trigger. Cũng tương tự như thủ tục lưu trữ, một trigger là một đối tượng chứa một tập các câu lệnh SQL và tập các câu lệnh này sẽ được thực thi khi trigger được gọi. Điểm khác biệt giữa thủ tục lưu trữ và trigger là: các thủ tục lưu trữ được thực thi khi người sử dụng có lời gọi đến chúng còn các trigger lại được "gọi" tự động khi xảy ra những giao tác làm thay đổi dữ liệu trong các bảng.

Mỗi một trigger được tạo ra và gắn liền với một bảng nào đó trong cơ sở dữ liệu. Khi dữ liệu trong bảng bị thay đổi (tức là khi bảng chịu tác động của các câu lệnh INSERT, UPDATE hay DELETE) thì trigger sẽ được tự đông kích hoạt.

Sử dụng trigger một cách hợp lý trong cơ sở dữ liệu sẽ có tác động rất lớn trong việc tăng hiệu năng của cơ sở dữ liệu. Các trigger thực sự hữu dụng với những khả năng sau:

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.

2.1 Định nghĩa trigger

Một trigger là một đối tượng gắn liền với một bảng và được tự động kích hoạt khi xảy ra những giao tác làm thay đổi dữ liệu trong bảng. Định nghĩa một trigger bao gồm các yếu tố sau:

Trigger sẽ được áp dụng đối với bảng nào?

Trigger được kích hoạt khi câu lệnh nào được thực thi trên bảng: INSERT,

UPDATE, DELETE?

Trigger sẽ làm gì khi được kích hoạt?

Câu lệnh CREATE TRIGGER được sử dụng để đinh nghĩa trigger và có cú pháp 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
```

Ví dụ 2.1: Ta định nghĩa các bảng như sau:

Bảng MATHANG lưu trữ dữ liệu về các mặt hàng:

```
CREATE TABLE mathang
(
Mahang NVARCHAR(5) PRIMARY KEY, /*mã hàng*/
Tenhang NVARCHAR(50) NOT NULL, /*tên hàng*/
Soluong INT, /*Số lượng hàng hiện có*/
)
```

Bảng NHATKYBANHANG lưu trữ thông tin về các lần bán hàng

```
CREATE TABLE nhatkybanhang
(
stt INT IDENTITY PRIMARY KEY,
ngay  DATETIME,  /*ngày bán hàng*/
nguoimua      NVARCHAR(30),   /*tên người mua hàng*/
mahang NVARCHAR(5)
FOREIGN KEY REFERENCES mathang(mahang),
Soluong INT,
Giaban MONEY
)
```

Câu lệnh dưới đây định nghĩa trigger trg_nhatkybanhang_insert. Trigger này có chức năng tự động giảm số lượng hàng hiện có khi một mặt hàng nào đó được bán (tức là khi câu lệnh INSERT được thực thi trên bảng NHATKYBANHANG).

```
CREATE TRIGGER trg_nhatkybanhang_insert
ON nhatkybanhang
FOR INSERT
AS
  UPDATE mathang
  SET mathang.soluong=mathang.soluong-inserted.soluong
  FROM mathang INNER JOIN inserted
 ON mathang.mahang=inserted.mahang
```

Với trigger vừa tạo ở trên, nếu dữ liệu trong bảng MATHANG là:

thì sau khi ta thực hiện câu lênh:

```
INSERT INTO nhatkybanhang
(ngay,nguoimua,mahang,soluong,giaban)
VALUES('5/5/2004','Tran Ngoc Thanh','H1',10,5200)
```

dữ liệu trong bảng MATHANG sẽ như sau:

MAHANG	TENHANG	SOLUONG
H1	Xà phòng	20
H2	Kem đánh răng	45

MAHANG	TENHANG	SOLUONG
H1	Xà phòng	30
H2	Kem đánh răng	45

Trong câu lệnh CREATE TRIGGER ở ví dụ trên, sau mệnh đề ON là tên của bảng mà trigger cần tạo sẽ tác động đến. Mệnh đề tiếp theo chỉ định câu lệnh sẽ kích hoạt trigger (FOR INSERT). Ngoài INSERT, ta còn có thể chỉ định UPDATE hoặc DELETE cho mệnh đề này, hoặc có thể kết hợp chúng lại với nhau. Phần thân của trigger nằm sau từ khoá AS bao gồm các câu lệnh mà trigger sẽ thực thi khi được kích hoạt.

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 nên sự kích hoạt đối với trigger bằng 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.

2.2 Sử dụng mệnh đề IF UPDATE trong trigger

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.

Ví dụ 2.2: Xét lại ví dụ với hai bảng MATHANG và NHATKYBANHANG, trigger dưới đây được kích hoạt khi ta tiến hành cập nhật cột SOLUONG cho một bản ghi của bảng NHATKYBANHANG (lưu ý là chỉ cập nhật đúng một bản ghi)

```
CREATE TRIGGER trg_nhatkybanhang_update_soluong
ON nhatkybanhang
FOR UPDATE
AS
IF UPDATE(soluong)
  UPDATE mathang
  SET mathang.soluong = mathang.soluong –
  (inserted.soluong-deleted.soluong)
  FROM (deleted INNER JOIN inserted ON
    deleted.stt = inserted.stt) INNER JOIN mathang
   ON mathang.mahang = deleted.mahang
```

Với trigger ở ví dụ trên, câu lệnh:

UPDATE nhatkybanhang

SET soluong=soluong+20

WHERE stt=1

sẽ kích hoạt trigger ứng với mệnh đề IF UPDATE (soluong) và câu lệnh UPDATE trong trigger sẽ được thực thi. Tuy nhiên câu lệnh:

UPDATE nhatkybanhang

SET nguoimua='Mai Hữu Toàn'

WHERE stt=3

lại không kích hoạt trigger này.

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.

Ví dụ 2.3: Giả sử ta định nghĩa bảng R như sau:

CREATE TABLE R

(

A INT,

B INT,

C INT

)

và trigger trg_R_update cho bảng R:

CREATE TRIGGER trg_R_test

ON R

FOR UPDATE

AS

IF UPDATE(A)

Print 'A updated'

IF UPDATE(C)

Print 'C updated'

Câu lệnh:

UPDATE R SET A=100 WHERE A=1

sẽ kích hoạt trigger và cho kết quả là:

A updated

và câu lệnh:

UPDATE R SET C=100 WHERE C=2

cũng kích hoạt trigger và cho kết quả là:

C updated

còn câu lệnh:

UPDATE R SET B=100 WHERE B=3

hiển nhiên sẽ không kích hoạt trigger

2.3 ROLLBACK TRANSACTION và trigger

Một trigger có khả năng nhận biết được sự thay đổi về mặt dữ liệu trên bảng dữ liệu, từ đó có thể phát hiện và huỷ bỏ những thao tác không đảm bảo tính toàn vẹn dữ liệu. Trong một trigger, để huỷ bỏ tác dụng của câu lệnh làm kích hoạt trigger, ta sử dụng câu lệnh:

ROLLBACK TRANSACTION

(Câu lệnh ROLLBACK TRANSACTION sẽ chi tiết ở chương 6)

Ví dụ 2.4: Nếu trên bảng MATHANG, ta tạo một trigger như sau:

```
CREATE TRIGGER trg_mathang_delete
ON mathang
FOR DELETE
AS
ROLLBACK TRANSACTION
```

Thì câu lệnh DELETE sẽ không thể có tác dụng đối với bảng MATHANG. Hay nói cách khác, ta không thể xoá được dữ liệu trong bảng.

Ví dụ 2.5: Trigger dưới đây được kích hoạt khi câu lệnh INSERT được sử dụng để bổ sung một bản ghi mới cho bảng NHATKYBANHANG. Trong trigger này kiểm tra điều kiện hợp lệ của dữ liệu là số lượng hàng bán ra phải nhỏ hơn hoặc bằng số lượng hàng hiện có. Nếu điều kiện này không thoả mãn thì huỷ bỏ thao tác bổ sung dữ liệu.

```
CREATE TRIGGER trg_nhatkybanhang_insert
ON NHATKYBANHANG
```

```
FOR INSERT
AS
  DECLARE @sl_co int    /* Số lượng hàng hiện có */
  DECLARE @sl_ban int        /*  Số lượng hàng được bán */
  DECLARE @mahang nvarchar(5)   /*  Mã hàng được bán */
  SELECT @mahang=mahang,@sl_ban=soluong
  FROM inserted
  SELECT @sl_co = soluong
  FROM mathang where mahang=@mahang
  /*Nếu số lượng hàng hiện có nhỏ hơn số lượng bán  thì huỷ bỏ thao tác bổ sung dữ liệu*/
IF @sl_co<@sl_ban
ROLLBACK TRANSACTION
    /*Nếu dữ liệu hợp lệ  thì giảm số lượng hàng hiện có */
  ELSE
 UPDATE mathang
 SET soluong=soluong-@sl_ban
 WHERE mahang=@mahang
```

2.4 Sử dụng trigger trong trường hợp câu lệnh INSERT, UPDATE và DELETE có tác động đến nhiều dòng dữ liệu

Trong các ví dụ trước, các trigger chỉ thực sự hoạt động đúng mục đích khi các câu lệnh kích hoạt trigger chỉ có tác dụng đối với đúng một dòng dữ liêu. Ta có thể nhận thấy là câu lệnh UPDATE và DELETE thường có tác dụng trên nhiều dòng, câu lệnh INSERT mặc dù ít rơi vào trường hợp này nhưng không phải là không gặp; đó là khi ta sử dụng câu lệnh có dạng INSERT INTO ... SELECT ... Vậy làm thế nào để trigger hoạt động đúng trong trường hợp những câu lệnh có tác động lên nhiều dòng dữ liệu?

Có hai giải pháp có thể sử dụng đối với vấn đề này:

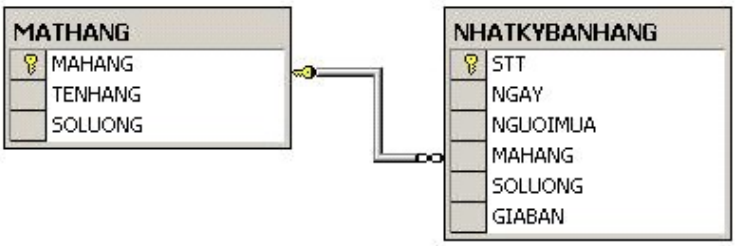
Sử dụng truy vấn con.

Sử dụng biến con trỏ.

2.4.1 Sử dụng truy vấn con

Ta hình dung vấn đề này và cách khắc phục qua ví dụ dưới đây:

Ví dụ 2.6: Ta xét lại trường hợp của hai bảng MATHANG và NHATKYBANHANG

như sơ đồ 

MAHANG	TENHANG	SOLUONG
H1	Xà phòng	30
H2	Kem đánh răng	45

STT	NGAY	NGUOIMUA	MAHANG	SOLUONG	GIABAN
1	1-1-2004	Ha	H1	10	10000.0000
2	2-2-2004	Phong	H2	20	5000.0000
3	3-3-2004	Thuy	H2	30	6000.0000

Trigger dưới đây cập nhật lại số lượng hàng của bảng MATHANG khi câu lệnh UPDATE được sử dụng để cập nhật cột SOLUONG của bảng NHATKYBANHANG.

```
CREATE TRIGGER trg_nhatkybanhang_update_soluong
ON nhatkybanhang
FOR UPDATE
AS
IF UPDATE(soluong)
  UPDATE mathang
  SET mathang.soluong = mathang.soluong –
  (inserted.soluong-deleted.soluong)
  FROM (deleted INNER JOIN inserted ON
    deleted.stt = inserted.stt) INNER JOIN mathang
   ON mathang.mahang = deleted.mahang
```

Với trigger được định nghĩa như trên, nếu thực hiện câu lệnh:

```
UPDATE nhatkybanhang
SET soluong = soluong + 10
WHERE stt = 1
```

thì dữ liệu trong hai bảng MATHANG và NHATKYBANHANG sẽ là:

MAHANG	TENHANG	SOLUONG
H1	Xà phòng	20
H2	Kem đánh răng	45

STT	NGAY	NGUOIMUA	MAHANG	SOLUONG	GIABAN
1	1-1-2004	Ha	H1	20	10000.0000
2	2-2-2004	Phong	H2	20	5000.0000
3	3-3-2004	Thuy	H2	30	6000.0000
4	4-4-2004	Dung	H1	40	9000.0000

Tức là số lượng của mặt hàng có mã H1 đã được giảm đi 10. Nhưng nếu thực hiện tiếp câu lệnh:

```
UPDATE nhatkybanhang
SET soluong=soluong + 5
WHERE mahang='H2'
```

dữ liệu trong hai bảng sau khi câu lệnh thực hiện xong sẽ như sau:

MAHANG	TENHANG	SOLUONG
H1	Xà phòng	20
H2	Kem đánh răng	40

STT	NGAY	NGUOIMUA	MAHANG	SOLUONG	GIABAN
1	1-1-2004	Ha	H1	20	10000.0000
2	2-2-2004	Phong	H2	25	5000.0000
3	3-3-2004	Thuy	H2	35	6000.0000

Ta có thể nhận thấy số lượng của mặt hàng có mã H2 còn lại 40 (giảm đi 5) trong khi đúng ra phải là 35 (tức là phải giảm 10). Như vậy, trigger ở trên không hoạt động đúng
trong trường hợp này.
Để khắc phục lỗi gặp phải như trên, ta định nghĩa lại trigger như sau:

```
CREATE TRIGGER trg_nhatkybanhang_update_soluong
ON nhatkybanhang
FOR UPDATE
AS
IF UPDATE(soluong)
  UPDATE mathang
  SET mathang.soluong = mathang.soluong -
   (SELECT SUM(inserted.soluong-deleted.soluong)
    FROM inserted INNER JOIN deleted
                ON inserted.stt=deleted.stt
WHERE inserted.mahang = mathang.mahang)
```

```
  WHERE mathang.mahang IN (SELECT mahang
FROM inserted)
hoặc:
CREATE TRIGGER trg_nhatkybanhang_update_soluong
ON nhatkybanhang
FOR UPDATE
AS
IF UPDATE(soluong)
/*Nếu số lượng dòng được cập nhật bằng 1 */
IF @@ROWCOUNT = 1
 BEGIN
   UPDATE mathang
   SET mathang.soluong = mathang.soluong –
   (inserted.soluong-deleted.soluong)
   FROM (deleted INNER JOIN inserted ON
    deleted.stt = inserted.stt) INNER JOIN mathang
   ON mathang.mahang = deleted.mahang
END
 ELSE
  BEGIN
    UPDATE mathang
    SET mathang.soluong = mathang.soluong -
(SELECT SUM(inserted.soluong-deleted.soluong)
    FROM inserted INNER JOIN deleted
                ON inserted.stt=deleted.stt
    WHERE inserted.mahang = mathang.mahang)
    WHERE mathang.mahang IN (SELECT mahang
FROM inserted)
END
```

2.4.2 Sử dụng biến con trỏ

Một cách khác để khắc phục lỗi xảy ra như trong ví dụ 5.17 là sử dụng con trỏ để duyệt qua các dòng dữ liệu và kiểm tra trên từng dòng. Tuy nhiên, sử dụng biến con trỏ trong trigger là giải pháp nên chọn trong trường hợp thực sự cần thiết.

Một biến con trỏ được sử dụng để duyệt qua các dòng dữ liệu trong kết quả của một truy vấn và được khai báo theo cú pháp như sau:

DECLARE tên_con_trỏ CURSOR

FOR câu_lệnh_SELECT

Trong đó câu lệnh SELECT phải có kết quả dưới dạng bảng. Tức là trong câu lệnh không sử dụng mệnh đề COMPUTE và INTO.

Để mở một biến con trỏ ta sử dụng câu lệnh:

OPEN tên_con_trỏ

Để sử dụng biến con trỏ duyệt qua các dòng dữ liệu của truy vấn, ta sử dụng câu lệnh FETCH. Giá trị của biến trạng thái @@FETCH_STATUS bằng không nếu chưa duyệt hết các dòng trong kết quả truy vấn.

Câu lệnh FETCH có cú pháp như sau:

FETCH [[NEXT|PRIOR|FIST|LAST] FROM] tên_con_trỏ

[INTOdanh_sách_biến]

Trong đó các biến trong danh sách biến được sử dụng để chứa các giá trị của các trường ứng với dòng dữ liệu mà con trỏ trỏ đến. Số lượng các biến phải bằng với số lượng các cột của kết quả truy vấn trong câu lệnh DECLARE CURSOR.

Ví dụ 2.7: Tập các câu lệnh trong ví dụ dưới đây minh hoạ cách sử dụng biến con trỏ để duyệt qua các dòng trong kết quả của câu lệnh SELECT

```
DECLARE contro CURSOR
FOR SELECT mahang,tenhang,soluong FROM mathang
OPEN contro
DECLARE @mahang  NVARCHAR(10)
DECLARE @tenhang NVARCHAR(10)
DECLARE @soluong INT
/*Bắt  đầu duyệt qua các dòng trong kết quả truy vấn*/
FETCH NEXT FROM contro
```

```
    INTO @mahang,@tenhang,@soluong
WHILE @@FETCH_STATUS=0
BEGIN
PRINT 'Ma hang:'+@mahang
PRINT 'Ten hang:'+@tenhang
PRINT 'So luong:'+STR(@soluong)
FETCH NEXT FROM contro
    INTO @mahang,@tenhang,@soluong
END
/*Đóng con trỏ và giải phóng vùng nhớ*/
CLOSE contro
DEALLOCATE contro
```

Ví dụ 2.8: Trigger dưới đây là một cách giải quyết khác của trường hợp được đề cập ở ví dụ 2.6

```
CREATE TRIGGER trg_nhatkybanhang_update_soluong
ON nhatkybanhang
FOR UPDATE
AS
IF UPDATE(soluong)
BEGIN
DECLARE @mahang NVARCHAR(10)
DECLARE @soluong INT
DECLARE contro CURSOR FOR
  SELECT inserted.mahang,  inserted.soluong - deleted.soluong AS soluong
  FROM inserted INNER JOIN deleted
  ON inserted.stt=deleted.stt
OPEN contro
FETCH NEXT FROM contro INTO @mahang,@soluong
WHILE @@FETCH_STATUS=0
  BEGIN
UPDATE mathang SET soluong=soluong-@soluong
```

```
WHERE mahang=@mahang
FETCH NEXT FROM contro INTO @mahang,@soluong
  END
CLOSE contro
DEALLOCATE contro
END
END
```

CHƯƠNG III HÀM

3.1. Hàm do người dùng định nghĩa

Hàm là đối tượng cơ sở dữ liệu tương tự như thủ tục. Điểm khác biệt giữa hàm và thủ tục là hàm trả về một giá trị thông qua tên hàm còn thủ tục thì không. Điều này cho phép ta sử dụng hàm như là một thành phần của một biêu thức (chẳng hạn trong danh sách chọn của câu lệnh SELECT).

Ngoài những hàm do hệ quản trị cơ sở dữ liệu cung cấp sẵn, người sử dụng có thể định nghĩa thêm các hàm nhằm phục vụ cho mục đích riêng của mình.

3.1.1 Định nghĩa và sử dụng hàm

Hàm được định nghĩa thông qua câu lệnh CREATE FUNCTION với cú pháp 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 3.1: Câu lệnh dưới đây định nghĩa hàm tính ngày trong tuần (thứ trong tuần) của một giá trị kiểu ngày

```
CREATE FUNCTION thu(@ngay DATETIME)
RETURNS NVARCHAR(10)
AS
BEGIN
DECLARE @st NVARCHAR(10)
SELECT @st=CASE DATEPART(DW,@ngay)
WHEN 1 THEN 'Chu nhật'
WHEN 2 THEN 'Thứ hai'
WHEN 3 THEN 'Thứ ba'
WHEN 4 THEN 'Thứ tư'
```

```
WHEN 5 THEN 'Thứ năm'
WHEN 6 THEN 'Thứ sáu'
ELSE 'Thứ bảy'
END
RETURN (@st)  /*  Trị trả về của hàm  */
END
```

Một hàm khi đã được định nghĩa có thể được sử dụng như các hàm do hệ quản trị cơ sở dữ liệu cung cấp (thông thường trước tên hàm ta phải chỉ định thêm tên của người sở hữu hàm)

Ví dụ 3.2: Câu lệnh SELECT dưới đây sử dụng hàm đã được định nghĩa ở ví dụ trước:

```
SELECT masv,hodem,ten,dbo.thu(ngaysinh),ngaysinh
FROM sinhvien
WHERE malop='C24102'
```

có kết quả là:

MASV	HODEM	TEN		NGAYSINH
0241020001	Nguyễn Tuấn	Anh	Chủ nhật	1979-07-15 00:00:00
0241020002	Trần Thị Kim	Anh	Thứ năm	1982-11-04 00:00:00
0241020003	Võ Đức	Ân	Thứ hai	1982-05-24 00:00:00
0241020004	Nguyễn Công	Bình	Thứ tư	1979-06-06 00:00:00
0241020005	Nguyễn Thanh	Bình	Thứ bảy	1982-04-24 00:00:00
0241020006	Lê Thị Thanh	Châu	Thứ ba	1982-05-25 00:00:00
0241020007	Bùi Đình	Chiến	Thứ ba	1981-04-07 00:00:00
0241020008	Nguyễn Công	Chính	Chủ nhật	1981-11-01 00:00:00

3.1.2 Hàm với giá trị trả về là "dữ liệu kiểu bảng"

Ta đã biết được chức năng cũng như sự tiện lợi của việc sử dụng các khung nhìn trong cơ sở dữ liệu. Tuy nhiên, nếu cần phải sử dụng các tham số trong khung nhìn (chẳng hạn các tham số trong mệnh đề WHERE của câu lệnh SELECT) thì ta lại không thể thực hiện được. Điều này phần nào đó làm giảm tính linh hoạt trong việc sử dụng khung nhìn.

Ví dụ 3.3: Xét khung nhìn được định nghĩa như sau:

```
CREATE VIEW  sinhvien_k25
```

AS

SELECT masv,hodem,ten,ngaysinh

FROM sinhvien INNER JOIN lop

ON sinhvien.malop=lop.malop

WHERE khoa=25

với khung nhìn trên, thông qua câu lệnh:

SELECT * FROM sinhvien_K25

ta có thể biết được danh sách các sinh viên khoá 25 một cách dễ dàng nhưng rõ ràng không thể thông qua khung nhìn này để biết được danh sách sinh viên các khoá khác do không thể sử dụng điều kiện có dạng KHOA = @thamso trong mệnh đề WHERE của câu lệnh SELECT được.

Nhược điểm trên của khung nhìn có thể khắc phục bằng cách sử dụng hàm với giá trị trả về dưới dạng bảng và được gọi là hàm nội tuyến (inline function). Việc sử dụng hàm loại này cung cấp khả năng như khung nhìn nhưng cho phép chúng ta sử dụng được các tham số và nhờ đó tính linh hoạt sẽ cao hơn.

Một hàm nội tuyến được định nghĩa bởi câu lệnh CREATE TABLE với cú pháp như sau:

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 3.4: Ta định nghĩa hàm func_XemSV như sau:

CREATE FUNCTION func_XemSV(@khoa SMALLINT)

RETURNS TABLE

AS

RETURN(SELECT masv,hodem,ten,ngaysinh

```
    FROM sinhvien INNER JOIN lop
        ON sinhvien.malop=lop.malop
    WHERE khoa=@khoa)
```

hàm trên nhận tham số đầu vào là khóa của sinh viên cần xem và giá trị trả về của hàm là tập các dòng dữ liệu cho biết thông tin về các sinh viên của khoá đó. Các hàm trả về giá trị dưới dạng bảng được sử dụng như là các bảng hay khung nhìn trong các câu lệnh SQL.

Với hàm được định nghĩa như trên, để biết danh sách các sinh viên khoá 25, ta sử dụng câu lệnh như sau:

```
SELECT * FROM dbo.func_XemSV(25)
```

còn câu lệnh dưới đây cho ta biết được danh sách sinh viên khoá 26

```
SELECT * FROM dbo.func_XemSV(26)
```

Đối với hàm nội tuyến, phần thân của hàm chỉ cho phép sự xuất hiện duy nhất của câu lệnh RETURN. Trong trường hợp cần phải sử dụng đến nhiều câu lệnh trong phần thân của hàm, ta sử dụng cú pháp như sau để định nghĩa hàm:

```
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
```

Khi định nghĩa hàm dạng này cần lưu ý một số điểm sau:

- Cấu trúc của bảng trả về bởi hàm được xác định dựa vào định nghĩa của bảng trong mệnh đề RETURNS. Biến @biến_bảng trong mệnh đề RETURNS có phạm vi sử dụng trong hàm và được sử dụng như là một tên bảng.
- Câu lệnh RETURN trong thân hàm không chỉ định giá trị trả về. Giá trị trả về của hàm chính là các dòng dữ liệu trong bảng có tên là @biếnbảng được định nghĩa trong mệnh đề RETURNS

Cũng tương tự như hàm nội tuyến, dạng hàm này cũng được sử dụng trong các câu

lệnh SQL với vai trò như bảng hay khung nhìn. Ví dụ dưới đây minh hoạ cách sử dụng dạng hàm này trong SQL.

Ví dụ 3.5: Ta định nghĩa hàm func_TongSV như sau:

```
CREATE FUNCTION Func_Tongsv(@khoa SMALLINT)
RETURNS @bangthongke TABLE
(
makhoa        NVARCHAR(5),
tenkhoa       NVARCHAR(50),
tongsosv      INT
)
AS
BEGIN
IF @khoa=0
INSERT INTO @bangthongke
SELECT khoa.makhoa,tenkhoa,COUNT(masv)
FROM (khoa INNER JOIN lop
    ON khoa.makhoa=lop.makhoa)
                INNER JOIN sinhvien
on lop.malop=sinhvien.malop
GROUP BY khoa.makhoa,tenkhoa
ELSE
INSERT INTO @bangthongke
SELECT khoa.makhoa,tenkhoa,COUNT(masv)
FROM (khoa INNER JOIN lop
    ON khoa.makhoa=lop.makhoa)
                INNER JOIN sinhvien
ON lop.malop=sinhvien.malop
WHERE khoa=@khoa
GROUP BY khoa.makhoa,tenkhoa
RETURN /*Trả kết quả về cho hàm*/
END
```

Với hàm được định nghĩa như trên, câu lệnh:

SELECT * FROM dbo.func_TongSV(25)

Sẽ cho kết quả thống kê tổng số sinh viên khoá 25 của mỗi khoa

MAKHOA	TENKHOA	TONGSOSV
DHTO1	Khoa Toán cơ - Tin học	5
DHTO2	Khoa Công nghệ thông tin	6
DHTO3	Khoa Vật lý	6
DHTO5	Khoa Sinh học	8

Còn câu lệnh:

SELECT * FROM dbo.func_TongSV(0)

Cho ta biết tổng số sinh viên hiện có (tất cả các khoá) của mỗi khoa:

MAKHOA	TENKHOA	TONGSOSV
DHTO1	Khoa Toán cơ - Tin học	15
DHTO2	Khoa Công nghệ thông tin	19
DHTO3	Khoa Vật lý	13
DHTO5	Khoa Sinh học	13

Bài tập

Dựa trên cơ sở dữ liệu ở bài tập chương 1, thực hiện các yêu cầu sau:

1. Tạo thủ tục lưu trữ để thông qua thủ tục này có thể bổ sung thêm một bản ghi mới cho bảng MATHANG (thủ tục phải thực hiện kiểm tra tính hợp lệ của dữ liệu cần bổ sung: không trùng khoá chính và đảm bảo toàn vẹn tham chiếu)
2. Tạo thủ tục lưu trữ có chức năng thống kê tổng số lượng hàng bán được của một mặt hàng có mã bất kỳ (mã mặt hàng cần thống kê là tham số của thủ tục).
3. Viết hàm trả về một bảng trong đó cho biết tổng số lượng hàng bán được của mỗi mặt hàng. Sử dụng hàm này để thống kê xem tổng số lượng hàng (hiện có và đã bán) của mỗi mặt hàng là bao nhiêu.
4. Viết trigger cho bảng CHITIETDATHANG theo yêu cầu sau:

Khi một bản ghi mới được bổ sung vào bảng này thì giảm số lượng hàng hiện có nếu số lượng hàng hiện có lớn hơn hoặc bằng số lượng hàng được bán ra. Ngược lại thì huỷ bỏ thao tác bổ sung.

Khi cập nhật lại số lượng hàng được bán, kiểm tra số lượng hàng được cập nhật lại có phù hợp hay không (số lượng hàng bán ra không được vượt quá số lượng hàng hiện có và không được nhỏ hơn 1). Nếu dữ liệu hợp lệ thì giảm (hoặc tăng) số lượng hàng hiện có trong công ty, ngược lại thì huỷ bỏ thao tác cập nhật.

5. Viết trigger cho bảng CHITIETDATHANG để sao cho chỉ chấp nhận giá hàng bán ra phải nhỏ hơn hoặc bằng giá gốc (giá của mặt hàng trong bảng MATHANG)
6. Để quản lý các bản tin trong một Website, người ta sử dụng hai bảng sau: Bảng LOAIBANTIN (loại bản tin)

```
CREATE TABLE loaibantin
(
maphanloai INT NOT NULL  PRIMARY KEY,
tenphanloai NVARCHAR(100)  NOT NULL ,
```

bantinmoinhat DEFAULT(0)

)

Bảng BANTIN (bản tin)

CREATE TABLE bantin

(

maso INT NOT NULL PRIMARY KEY,

ngayduatin DATETIME NULL ,

tieude NVARCHAR(200) NULL ,

noidung NTEXT NULL ,

maphanloai INT NULL FOREIGN KEY

REFERENCES loaibantin(maphanloai)

)

Trong bảng LOAIBANTIN, giá trị cột BANTINMOINHAT cho biết mã số của bản tin thuộc loại tương ứng mới nhất (được bổ sung sau cùng).

Hãy viết các trigger cho bảng BANTIN sao cho:

Khi một bản tin mới được bổ sung, cập nhật lại cột BANTINMOINHAT của dòng tương ứng với loại bản tin vừa bổ sung.

Khi một bản tin bị xoá, cập nhật lại giá trị của cột BANTINMOINHAT trong bảng LOAIBANTIN của dòng ứng với loại bản tin vừa xóa là mã số của bản tin trước đó (dựa vào ngày đưa tin). Nếu không còn bản tin nào cùng loại thì giá trị của cột này bằng 0.

Khi cập nhật lại mã số của một bản tin và nếu đó là bản tin mới nhất thì cập nhật lại giá trị cột BANTINMOINHAT là mã số mới.

CHƯƠNG IV BẢO MẬT TRONG SQL

4.1 Các khái niệm

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.

4.2 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 trongcác 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.

4.2.1 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.

	• Đối với cột trong bảng, khung nhìn: SELECT và UPDATE. • Đối với thủ tục lưu trữ và hàm vô hướng: EXECUTE. Trong các quyền được đề cập đến ở trên, quyền REFERENCES được sử dụng nhằm cho phép tạo khóa ngoài tham chiếu đến bảng cấp phát.
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 4.1: 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 nàođó, 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 4.2: 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

4.2.2 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 4.3: 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.

4.3 Thu hồi quyền

Câu lệnh REVOKE được sử dụng 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.

4.3.1 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 4.4: 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 4.5: 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 4.6: 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 4.7: 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).

4.3.2 Thu hồi quyền thực thi các câu lệnh:

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 4.8: 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

CHƯƠNG V GIAO TÁC SQL

5.1 Giao tác và các tính chất của giao tác

Một giao tác (transaction) là một chuỗi một hoặc nhiều câu lệnh SQL được kết hợp lại với nhau thành một khối công việc. Các câu lệnh SQL xuất hiện trong giao tác thường có mối quan hệ tương đối mật thiết với nhau và thực hiện các thao tác độc lập.

Việc kết hợp các câu lệnh lại với nhau trong một giao tác nhằm đảm bảo tính toàn vẹn dữ liệu và khả năng phục hồi dữ liệu. Trong một giao tác, các câu lệnh có thể độc lập với nhau nhưng tất cả các câu lệnh trong một giao tác đòi hỏi hoặc phải thực thi trọn vẹn hoặc không một câu lệnh nào được thực thi.

Các cơ sở dữ liệu sử dụng nhật ký giao tác (transaction log) để ghi lại các thay đổi mà giao tác tạo ra trên cơ sở dữ liệu và thông qua đó có thể phục hồi dữ liệu trong trường hợp gặp lỗi hay hệ thống có sự cố.

Một giao tác đòi hỏi phải có được bốn tính chất sau đây:

- Tính nguyên tử (Atomicity): Mọi thay đổi về mặt dữ liệu hoặc phải được thực hiện trọn vẹn khi giao tác thực hiện thành công hoặc không có bất kỳ sự thay đổi nào về dữ liệu xảy ra nếu giao tác không thực hiện được trọn vẹn. Nói cách khác, tác dụng của các câu lệnh trong một giao tác phải như là một câu lệnh đơn.

  Tính nhất quán (Consistency): Tính nhất quan đòi hỏi sau khi giao tác kết thúc, cho dù là thành công hay bị lỗi, tất cả dữ liệu phải ở trạng thái nhất quán (tức là sự toàn vẹn dữ liệu phải luôn được bảo toàn).

  Tính độc lập (Isolation): Tính độc lập của giao tác có nghĩa là tác dụng của mỗi một giao tác phải giống như khi chỉ mình nó được thực hiện trên chính hệ thống đó. Nói cách khác, một giao tác khi được thực thi đồng thời với những giao tác khác trên cùng hệ thống không chịu bất kỳ sự ảnh hưởng nào của các giao tác đó.

  Tính bền vững (Durability): Sau khi một giao tác đã thực hiện thành công, mọi tác dụng mà nó đã tạo ra phải tồn tại bền vững trong cơ sở dữ liệu, cho dù là hệ thống có bị lỗi đi chăng nữa.

5.2 Mô hình giao tác trong SQL

Giao tác SQL được định nghĩa dựa trên các câu lệnh xử lý giao tác sau đây:

BEGIN TRANSACTION: Bắt đầu một giao tác

SAVE TRANSACTION:Đánh dấu một vị trí trong giao tác (gọi là điểm đánh dấu).

ROLLBACK TRANSACTION: Quay lui trở lại đầu giao tác hoặc một điểm đánh dấu trước đó trong giao tác.

COMMIT TRANSACTION: Đánh dấu điểm kết thúc một giao tác. Khi câu lệnh này thực thi cũng có nghĩa là giao tác đã thực hiện thành công.

ROLLBACK [WORK]: Quay lui trở lại đầu giao tác.

COMMIT [WORK]: Đánh dấu kết thúc giao tác.

Một giao tác trong SQL được bắtđấu bởi câu lệnh BEGIN TRANSACTION. Câu lệnh này đánh dấu điểm bắt đầu của một giao tác và có cú pháp như sau:

BEGIN TRANSACTION [tên_giao_tác]

Một giao tác sẽ kết thúc trong các trường hợp sau:

Câu lệnh COMMIT TRANSACTION (hoặc COMMIT WORK) được thực thi. Câu lệnh này báo hiệu sự kết thúc thành công của một giao tác. Sau câu lệnh này, một giao tác mới sẽ được bắt đầu.

Khi câu lệnh ROLLBACK TRANSACTION (hoặc ROLLBACK WORK) được thực thi để huỷ bỏ một giao tác và đưa cơ sở dữ liệu về trạng thái như trước khi giao tác bắt đầu. Một giao tác mới sẽ bắt đầu sau khi câu lệnh ROLLBACK được thực thi.

Một giao tác cũng sẽ kết thúc nếu trong quá trình thực hiện gặp lỗi (chẳng hạn hệ thống gặp lỗi, kết nối mạng bị "đứt",...). Trong trường hợp này, hệ thống sẽ tự động phục hồi lại trạng thái cơ sở dữ liệu như trước khi giao tác bắt đầu (tương tự như khi câu lệnh ROLLBACK được thực thi để huỷ bỏ một giao tác). Tuy nhiên, trong trường hợp này sẽ không có giao tác mới được bắt đầu.

Ví dụ 5.1: Giao tác dưới đây kết thúc do lệnh ROLLBACK TRANSACTION và mọi thay đổi về mặt dữ liệu mà giao tác đã thực hiện (UPDATE) đều không có tác dụng.

BEGIN TRANSACTION giaotac1

UPDATE monhoc SET sodvht=4 WHERE sodvht=3

UPDATE diemthi SET diemlan2=0 WHERE diemlan2 IS NULL

ROLLBACK TRANSACTION giaotac1

còn giao tác dưới đây kết thúc bởi lệnh COMMIT và thực hiện thành công việc cập nhật dữ liệu trên các bảng MONHOC và DIEMTHI.

BEGIN TRANSACTION giaotac2

UPDATE monhoc SET sodvht=4 WHERE sodvht=3

UPDATE diemthi SET diemlan2=0 WHERE diemlan2 IS NULL

COMMIT TRANSACTION giaotac2

Câu lệnh:

SAVE TRANSACTION tên_điểm_dánh_dấu

được sử dụng để đánh dấu một vị trí trong giao tác. Khi câu lệnh này được thực thi, trạng thái của cơ sở dữ liệu tại thời điểm đó sẽ được ghi lại trong nhật ký giao tác.

Trong quá trình thực thi giao tác có thể quay trở lại một điểm đánh dấu bằng cách sử dụng câu lệnh:

ROLLBACK TRANSACTION tên_điểm_đánh_dấu

Trong trường hợp này, những thay đổi về mặt dữ liệu mà giao tác đã thực hiện từ điểm đánh dấu đến trước khi câu lệnh ROLLBACK được triệu gọi sẽ bị huỷ bỏ. Giao tác sẽ được tiếp tục với trạng thái cơ sở dữ liệu có được tại điểm đánh dấu . Hình 6.2 mô tả cho ta thấy hoạt động của một giao tác có sử dụng các điểm đánh dấu:

Sau khi câu lệnh ROLLBACK TRANSACTION được sử dụng để quay lui lại một điểm đánh dấu trong giao tác, giao tác vẫn được tiếp tục với các câu lệnh sau đó.

Nhưng nếu câu lệnh này được sử dụng để quay lui lại đầu giao tác (tức là huỷ bỏ giao tác), giao tác sẽ kết thúc và do đó câu lệnh COMMIT TRANSACTION trong trường hợp này sẽ gặp lỗi.

Ví dụ 5.2: Câu lệnh COMMIT TRANSACTION trong giao tác dưới đây kết thúc thành công một giao tác

BEGIN TRANSACTION giaotac3

UPDATE diemthi SET diemlan2=0 WHERE diemlan2 IS NULL

SAVE TRANSACTION a

UPDATE monhoc SET sodvht=4 WHERE sodvht=3

ROLLBACK TRANSACTION a

UPDATE monhoc SET sodvht=2 WHERE sodvht=3

COMMIT TRANSACTION giaotac3

và trong ví dụ dưới đây, câu lệnh COMMIT TRANSACTION gặp lỗi:

BEGIN TRANSACTION giaotac4

UPDATE diemthi SET diemlan2=0 WHERE diemlan2 IS NULL

SAVE TRANSACTION a

UPDATE monhoc SET sodvht=4 WHERE sodvht=3

ROLLBACK TRANSACTION giaotac4

UPDATE monhoc SET sodvht=2 WHERE sodvht=3

COMMIT TRANSACTION giaotac4

5.3 Giao tác lồng nhau

Các giao tác trong SQL có thể được lồng vào nhau theo từng cấp. Điều này thường gặp đối với các giao tác trong các thủ tục lưu trữ được gọi hoặc từ một tiến trình trong một giao tác khác.

Ví dụ dưới đây minh hoạ cho ta trường hợp các giao tác lồng nhau.

Ví dụ 5.3: Ta định nghĩa bảng T như sau:

CREATE TABLE T

(

A INT PRIMARY KEY,

B INT

)

và thủ tục sp_TransEx:

CREATE PROC sp_TranEx(@a INT,@b INT)

AS

BEGIN

BEGIN TRANSACTION T1

```
IF NOT EXISTS (SELECT * FROM T WHERE A=@A )
INSERT INTO T VALUES(@A,@B)
IF NOT EXISTS (SELECT * FROM T WHERE A=@A+1)
INSERT INTO T VALUES(@A+1,@B+1)
COMMIT TRANSACTION T1
END
```

Lời gọi đến thủ tuch sp_TransEx được thực hiện trong một giao tác khác như sau:

```
BEGIN TRANSACTION T3
EXECUTE sp_tranex 10,20
ROLLBACK TRANSACTION T3
```

Trong giao tác trên, câu lệnh ROLLBACK TRANSACTION T3 huỷ bỏ giao tác và do đó tác dụng của lời gọi thủ tục trong giao tác không còn tác dụng, tức là không có dòng dữ liệu nào mới được bổ sung vào bảng T (cho dù giao tác T1 trong thủ tục sp_tranex đã thực hiện thành công với lệnh COMMIT TRANSACTION T1).

Ta xét tiếp một trường hợp của một giao tác khác trong đó có lời gọi đến thủ tục sp_tranex như sau:

```
BEGIN TRANSACTION
EXECUTE sp_tranex 20,40
SAVE TRANSACTION a
EXECUTE sp_tranex 30,60
ROLLBACK TRANSACTION a
EXECUTE sp_tranex 40,80
COMMIT TRANSACTION
```

sau khi giao tác trên thực hiện xong, dữ liệu trong bảng T sẽ là:

```
A    B
20   40
21   41
40   80
41   81
```

Như vậy, tác dụng của lời gọi thủ tục sp_tranex 30,60 trong giao tác đã bị huỷ bỏ bởi câu lệnh ROLLBACK TRANSACTION trong giao tác.

Như đã thấy trong ví dụ trên, khi các giao tác SQL được lồng vào nhau, giao tác ngoài cùng nhất là giao tác có vai trò quyết định. Nếu giao tác ngoài cùng nhất được uỷ thác (commit) thì các giao tác được lồng bên trong cũng đồng thời uỷ thác; Và nếu giao tác ngoài cùng nhất thực hiện lệnh ROLLBACK thì những giao tác lồng bên trong cũng chịu tác động của câu lệnh này (cho dù những giao tác lồng bên trong đã thực hiện lệnh COMMIT TRANSACTION).

MỤC LỤC