BÀI 5: TRIGGER

Ngày học | @November 13, 2023

5.3.1 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?: ON
> • Trigger được kích hoạt khi câu lệnh nào được thực thi trên bảng: INSERT, UPDATE, DELETE?
> • Trigger sẽ làm gì khi được kích hoạt?

💡
```
CREATE TRIGGER trg_xxxx
ON tên_bảng /Khung nhìn / ALL SERVER
FOR / ALTER {[INSERT][,][UPDATE][,][DELETE]}
AS
BEGIN
        SET NOCOUNT ON;
        [IF UPDATE(tên_cột)
        [AND UPDATE(tên_cột)|OR UPDATE(tên_cột)]
        ...]
        các_câu_lệnh_của_trigger
END
GO
```

ALTER: dùng khi đã tạo rồi và muốn sửa

> 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**.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.

```
CREATE TABLE mathang
(
mahang NVARCHAR(5) PRIMARY KEY,
tenhang NVARCHAR(50) NOT NULL,
soluong INT,
)
CREATE TABLE nhatkybanhang
(
stt INT IDENTITY PRIMARY KEY,
ngay DATETIME,
nguoimua NVARCHAR(30),
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
```

Trigger với INSTEAD OF

Khi tạo trigger trên 1 bảng với mệnh đề INSTEAD OF thì có nghĩa là các lệnh bên trong thân của trigger sẽ chạy thay thế cho lệnh kích hoạt trigger.

VD:

```
CREATE TABLE NhanVien
(
    MaNV nvarchar(50),
    Ten nvarchar(50)
)
GO
```

```
CREATE TABLE TempNV
(
    Ten nvarchar(50)
)
GO
```

Cần: Nếu INSERT dữ liệu vào bảng NhanVien thì không đưa dữ liệu vào bảng NhanVien mà đưa vào bảng TempNV

```
CREATE TRIGGER trg_NhanVien_InsteadOf_Insert
ON NhanVien
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO TempNV(Ten)
    SELECT Ten from inserted;
END
GO
```

```
INSERT INTO NhanVien(MaNV, Ten)
VALUES  ('NV01', 'Minh'),
        ('NV02', 'Long')

SELECT * FROM NhanVien
SELECT * FRO, TempNV
```

VD: Code sau chỉ đúng khi INSERT TỪNG DÒNG 1

```
CREATE TABLE UserAccount
(
    AccountId int identity(1,1) primary key,
    UserName nvarchar(50) unique
)
GO
```

```
CREATE TABLE Comment
(
    CommentId int identity(1,1) primary key,
    Content nvarchar(255),
```

```
    AccountId int
)
GO
```

```
CREATE VIEW viewComment
AS
    SELECT c.CommentId, c.Content, c.AccountId, u.UserName
    FROM Comment as c
          JOIN UserAccount as u ON c.AccountId = u.AccountId
GO
```

```
CREATE TRIGGER trg_viewComment_InsteadOf_Insert
ON viewComment
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE   @AccountId int,
              @Username nvarchar(50),
              @Content nvarchar(255);
    SELECT  @Username = Username,
            @Content = Content
    from inserted

    SELECT  @AccountId = AccountId
    FROM UserAccount
    WHERE UserName = @UserName

    IF (@AccountId IS NULL)
        BEGIN
            INSERT INTO UserAccount(UserName) VALUES (@UserName);
            SET @AccountId = @@IDENTITY
        END
    INSERT INTO Comment(Content, AccountId)
    VALUES (@Content, @AccountId);
END
GO
```

```
INSERT INTO viewComment(Content, UserName)
VALUES  ('ABC', 'a1'),
        ('ABCD', 'a2'),
        ('ABCDE', 'a3')
```

```
SELECT * FROM UserAccount
SELECT * FROM Comment
```

```
CREATE TRIGGER trg_viewComment_InsteadOf_Insert
ON viewComment
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO UserAccount(UserName)
        SELECT UserName FROM inserted
                        WHERE UserName NOT IN (SELECT Username FROM UserAccount)
    INSERT INTO Comment(Content, AccountId)
        SELECT Content, u.AccountId
        FROM inserted AS i JOIN
              UserAccount as u ON i.UserName = u.UserName
```

Trong trường hợp Insert nhiều dòng

5.3.2 Sử dụng mệnh đề IF UPDATE trong trigger

```
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 5.14: 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

5.3.3 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

Ví dụ: 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: 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
    */
```

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

5.3.4.1 Sử dụng truy vấn con

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 -
(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)
```

5.3.4.2 Sử dụng biến con trỏ

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** = 0 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
[INTO danh_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: 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 5.19: Trigger dưới đây là một cách giải quyết khác của trường hợp được đề cập ở ví dụ 5.17

```
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
```