Chương 2B. TRIGGER

1.1 Giao tác

1.1.1 Giao tác không tường minh

- Mặc định các lệnh bên trong lô (batch) chứa các câu lệnh không tường minh, điều này có nghĩa là **nếu có ít nhất 1 câu lệnh thực hiện không thành công trong lô thì tất cả các lệnh còn lại sẽ không được ghi nhận lại.**

Ví dụ: chúng ta cho **thực hiện cùng lúc 3 lệnh để cập nhật dữ liệu vào 3 bảng khác nhau** trong cùng một lô.

```
--Thêm mặt hàng mới
INSERT INTO MAT_HANG(MaMH, TenMH, DVT)
VALUES('D001', 'đèn ngủ', 'cái' )

--Sửa đổi tên nhà cung cấp 'NCC01'
UPDATE NHACC
SET TenNCC = 'Le Khai Hoan'
Where MaNCC = 'NCC01'

--Xoá đơn đặt hàng 'DH001'
DELETE HOA_DONDH
Where MaDH = 'DH001'
GO
```

Vi phạm toàn vẹn DL về khóa ngoại. Nên các lệnh trước đó không thực hiện

Tại sao việc xóa 'DH001' có thể vi phạm toàn vẹn tham chiếu?

- Giả sử có một bảng khác, ví dụ như CT_HOADON (Chi tiết hóa đơn), lưu trữ thông tin chi tiết về từng đơn đặt hàng. Trong bảng CT_HOADON, có một cột là MaDH (Mã đơn đặt hàng) đóng vai trò là **khóa ngoại**, tham chiếu đến cột MaDH (khóa chính) trong bảng HOA_DONDH.
- Nếu bạn xóa đơn đặt hàng 'DH001' khỏi bảng HOA_DONDH, nhưng vẫn còn các bản ghi trong bảng CT_HOADON có MaDH là 'DH001', thì sẽ xảy ra tình huống **dữ liệu bị "mồ côi" (orphaned data)**. Các bản ghi trong CT_HOADON sẽ tham chiếu đến một đơn đặt hàng không còn tồn tại, gây mất tính nhất quán và toàn vẹn của dữ liệu.

Ảnh hưởng đến các lệnh trước đó: Trong nhiều hệ quản trị cơ sở dữ liệu (DBMS), khi một lệnh trong một lô (batch) gặp lỗi, toàn bộ lô sẽ bị hủy bỏ (rollback). Điều này có nghĩa là nếu lệnh DELETE vi phạm ràng buộc toàn vẹn tham chiếu, các lệnh INSERT và UPDATE trước đó cũng sẽ không được thực hiện. Đây là cơ chế bảo vệ tính toàn vẹn dữ liệu, đảm bảo rằng cơ sở dữ liệu luôn ở trạng thái nhất quán.

1.1.2 Giao tác tường minh

- Giao tác tường minh trong những trường hợp **cập nhật dữ liệu trên nhiều bảng khác nhau** và phải đảm bảo **các hành động này nằm trong cùng một đơn vị xử lý (thỏa ACID).**
- **Bắt đầu một giao tác:** BEGIN TRAN[SACTION] [Tên_giao_tác]
- **Kết thúc một giao tác:**
  - **ROLLBACK TRAN[SACTION] [Tên_giao_tác]**

    → kết thúc giao tác nhưng **không** ghi nhận lại các hành động cập nhật dữ liệu **bên trong giao tác, tất cả các thay đổi được thực hiện "bên trong" giao tác sẽ bị *hủy bỏ*.**

    Hoặc
  - **COMMIT TRAN[SACTION] [Tên_giao_tác]**

    → kết thúc giao tác nhưng **đồng ý** ghi nhận lại **vĩnh viễn** các hành động cập nhật dữ liệu **bên trong giao tác vào cơ sở dữ liệu. (Các thay đổi được thực hiện trong phạm vi của giao tác).**

Ví dụ:

```
begin tran cap1
        insert into #test values(1,'aaa')
        begin tran cap2
                insert into #test values(2,'bbb')
                begin tran cap3
                        insert into #test values(3,'ccc')
                commit tran cap3
                go
        commit tran cap2
        go
rollback tran cap1
```

commit tran cap3: Xác nhận giao tác cap3. Điều này *không* có nghĩa là dữ liệu được lưu vào cơ sở dữ liệu ngay lập tức. Thay vào đó, nó báo hiệu rằng cap3 đã thành công.

rollback tran cap1: Lệnh này hủy bỏ giao tác cap1. Vì cap2 và cap3 được lồng trong cap1, tất cả các thay đổi được thực hiện trong cả ba giao tác (tức là việc chèn 3 bản ghi vào #test) đều bị hoàn tác.

Sau khi đoạn mã trên được thực thi, bảng #test sẽ *không* chứa bất kỳ bản ghi nào. Tất cả các thao tác chèn đã bị hủy bỏ bởi lệnh rollback tran cap1.

```
-- Tạo bảng tạm để minh họa (nếu chưa có)
CREATE TABLE #TestTransaction (ID INT, Value VARCHAR(10));
GO

-- Bắt đầu giao tác
BEGIN TRANSACTION MyTransaction;

-- Các thao tác "bên trong" giao tác
INSERT INTO #TestTransaction (ID, Value) VALUES (1, 'Value1');
UPDATE #TestTransaction SET Value = 'NewValue1' WHERE ID = 1;
DELETE FROM #TestTransaction WHERE ID = 1;

-- Kết thúc giao tác (một trong hai lựa chọn)
-- 1. COMMIT TRANSACTION MyTransaction; -- Ghi nhận thay đổi
-- 2. ROLLBACK TRANSACTION MyTransaction; -- Hủy bỏ thay đổi
GO

-- Kiểm tra dữ liệu sau giao tác
SELECT * FROM #TestTransaction;

-- Xóa bảng tạm
DROP TABLE #TestTransaction;
GO
```

- **Nếu bạn chọn COMMIT:** Sau khi chạy đoạn mã và kiểm tra bảng #TestTransaction, bạn sẽ thấy bảng trống (do lệnh DELETE được thực hiện và được ghi nhận). Các thay đổi đã trở thành "bên ngoài" giao tác, tức là được lưu vào cơ sở dữ liệu.
- **Nếu bạn chọn ROLLBACK:** Sau khi chạy đoạn mã và kiểm tra bảng #TestTransaction, bạn cũng sẽ thấy bảng trống. Tuy nhiên, điều này không phải do dữ liệu đã được chèn và xóa thành công. Thay vào đó, tất cả các thao tác (chèn, cập nhật, xóa) đã bị hủy bỏ bởi ROLLBACK. Các thay đổi không bao giờ được lưu vào cơ sở dữ liệu, chúng chỉ tồn tại "bên trong" phạm vi của giao tác và bị loại bỏ.

1.1.3 Phân vùng trong giao tác

- Ta chia nhỏ các hành động bên trong giao tác thành nhiều phần, tương ứng từng phần nhỏ ta có thể dễ dàng **chủ động** đồng ý ghi nhận hoặc không ghi nhận lại việc cập nhật dữ liệu.
- **Cú pháp:** **SAVE** TRAN[SACTION] **[Tên vùng]**
  Các lệnh

Ví dụ:

```
Begin tran
--Vùng 1
SAVE TRAN Vung_1_2
        insert into #test values(1,'aaa')
        insert into #test values(2,'bbb')
        --Vùng 2
        SAVE TRAN Vung_3
                insert into #test values(3,'ccc')
        RollBack Tran Vung_3
Commit Tran Vung_1_2
```

→ chỉ có mẩu tin thứ 1 và thứ 2 được ghi nhận lại.

1.2 Trigger

- **Khái niệm:** Trigger là một dạng đặc biệt của thủ tục nội tại. Tuy nhiên khác với thủ tục nội tại:
  - **Không có tham số.**
  - **Không thể gọi trực tiếp bằng lệnh EXECUTE** như thủ tục nội tại mà **thực hiện một cách tự động khi dữ liệu của bảng có liên quan đến trigger bị cập nhật** (như INSERT, UPDATE, hoặc DELETE).
- Trigger dùng cho các công việc sau:
  - **Kiểm tra ràng buộc toàn vẹn** dữ liệu phức tạp.
  - Thực hiện các **xử lý thiết kế thi hành tại Server** (trong mô hình Client/Server). Các *xử lý sẽ tự động thực hiện khi có thao tác **INSERT**, **UPDATE** hoặc **DELETE** xảy ra.*
  - Trigger dùng ***thay thế các constraint*** trong trường hợp ta muốn việc ***kiểm tra ràng buộc dữ liệu kèm theo các câu thông báo thích hợp*** theo ý muốn người dùng, hoặc thực hiện các hành động phức tạp hơn khi vi phạm ràng buộc.

1.2.1 Ràng buộc dữ liệu toàn vẹn với Trigger

- Để đảm bảo dữ liệu nhất quán và đúng đắn, ta cần **kiểm tra thực hiện 3 thao tác**: **Insert, Update và Delete.**
- **Có 2 cách kiểm tra**
  - **Kiểm tra mức giao diện:** là công việc lập trình trên các màn hình giao diện. Cách này phụ thuộc vào ứng dụng và có thể bỏ qua nếu người dùng tương tác trực tiếp với cơ sở dữ liệu.
  - **Kiểm tra mức CSDL:** thực hiện bởi các đối tượng constraint hoặc trigger. Cách này đảm bảo tính toàn vẹn dữ liệu bất kể người dùng tương tác với cơ sở dữ liệu bằng cách nào.

- **Các dạng ràng buộc toàn vẹn**
  - ***RBTV bằng phương pháp mô tả (Đối tượng Constraint)***
    - Xác định **khoá chính, khoá ngoại, miền giá trị**,… và mô tả chúng tại thời điểm tạo Table
    - Thực hiện **trước** khi cho phép thêm vào Table.

```
CREATE TABLE SanPham (
   MaSP INT PRIMARY KEY,
   TenSP VARCHAR(255) NOT NULL,
   Gia DECIMAL(10, 2) CHECK (Gia > 0),
   MaNCC INT FOREIGN KEY REFERENCES NhaCungCap(MaNCC)
);
```

  - ***RBTV theo phương pháp thủ tục (Đối tượng Trigger)***
    - Kiểm tra tính **toàn vẹn dữ liệu** trên nhiều cột hoặc nhiều dòng của các bảng khác nhau.
    - Xác định bởi **tập các câu lệnh T-SQL.** Các lệnh chứa bên trong đối tượng trigger.
    - Được **gọi thi hành khi có thao tác Thêm, xoá hoặc sửa** dữ liệu trên table tương ứng (ngoại trừ Trigger INSTEAD OF).
    - Thực hiện **sau** khi dữ liệu được ghi vào Table.
    - Cho phép thực hiện các hành động phức tạp hơn, bao gồm cả việc hoàn tác thao tác nếu vi phạm ràng buộc.

```
CREATE TRIGGER trg_KiemTraSoLuongTonKho
ON ChiTietDonHang
AFTER INSERT
AS
```

```
BEGIN
    -- Kiểm tra số lượng tồn kho
    IF EXISTS (SELECT 1 FROM inserted i JOIN SanPham sp ON i.MaSP = sp.MaSP
WHERE i.SoLuong > sp.SoLuongTon)
    BEGIN
        -- Hoàn tác thao tác INSERT và đưa ra thông báo lỗi
        RAISERROR('Số lượng đặt hàng vượt quá số lượng tồn kho.', 16, 1)
        ROLLBACK TRANSACTION
    END
END;
```

1.2.2 Bài toán

KHACH_HANG(**MaKH**, TenKhach)
PHIEU_XUAT(**MAPX**, Ngay_PX, **MaKH**)
CT_PHIEU_XUAT(**MAPX, MaHH**, SoLuong, DonGia)
HANG_HOA(**MAHH**,Ten_HH, DonGiaHienHanh)

- **KHACH_HANG:** một khách hàng có 1 mã duy nhất để phân biệt khách hàng này với khách hàng khác, có 1 tên khách hàng duy nhất
- **HANG_HOA:** một hàng hoá có 1 mã hàng duy nhất dùng để phân biệt với hàng hoá khác, có 1 tên hàng hoá và một đơn vị bán hiện tại duy nhất
- **PHIEU_XUAT:** một phiếu xuất có 1 mã duy nhất dùng để phân biệt với phiếu khác, có 1 ngày xuất xác định.
    - Một phiếu xuất liên quan đến 1 khách hàng duy nhất
    - Một phiếu xuất có **ít nhất 1** chi tiết xuất
- **CT_PHIEU_XUAT:** một chi tiết phiếu xuất có mã phiếu xuất và mã hàng hoá dùng để xác định khoá của quan hệ. Khoá này dùng để phân biệt với chi tiết xuất khác, có 1 số lượng xác định, 1 đơn giá tương ứng với đơn giá hiện hành của hàng hoá lấy từ thuộc tính DonGiaHienHanh của bảng HANG_HOA.

Ràng buộc **"đơn giá trong CT_PHIEU_XUAT phải tương ứng với đơn giá hiện hành của hàng hóa lấy từ thuộc tính DonGiaHienHanh của bảng HANG_HOA"** là ràng buộc *buộc phải* sử dụng Trigger để thực thi.

Tại sao Constraint không đủ?
Các Constraint như FOREIGN KEY, CHECK, UNIQUE, NOT NULL, và PRIMARY KEY rất hữu ích cho việc ràng buộc dữ liệu, nhưng chúng không thể xử lý logic phức tạp liên quan đến việc **tham chiếu và đồng bộ dữ liệu** giữa các bảng **dựa trên thời điểm thực hiện thao tác.**
Trong trường hợp này, bạn cần đảm bảo rằng mỗi khi một chi tiết phiếu xuất được thêm vào (INSERT) hoặc cập nhật (UPDATE), đơn giá của chi tiết đó phải khớp với DonGiaHienHanh của hàng hóa tương ứng *tại thời điểm đó.* **Constraint không thể tự động lấy giá trị hiện tại từ một bảng khác.**

Tại sao cần Trigger?
Trigger, đặc biệt là Trigger AFTER INSERT và AFTER UPDATE trên bảng CT_PHIEU_XUAT, **cho phép bạn thực hiện logic này một cách tự động.** Khi một bản ghi mới được chèn vào hoặc một bản ghi hiện có được cập nhật trong CT_PHIEU_XUAT, Trigger sẽ được kích hoạt và thực hiện các bước sau:

1. Lấy MaHH từ bản ghi vừa được chèn/cập nhật trong CT_PHIEU_XUAT.
2. Truy vấn bảng HANG_HOA để lấy DonGiaHienHanh tương ứng với MaHH vừa lấy.
3. So sánh DonGia trong CT_PHIEU_XUAT với DonGiaHienHanh vừa lấy từ HANG_HOA.

4. Nếu không khớp:
    - Đưa ra thông báo lỗi.
    - Hoàn tác (ROLLBACK) thao tác INSERT hoặc UPDATE trên CT_PHIEU_XUAT để đảm bảo tính toàn vẹn dữ liệu.

Ví dụ Trigger (T-SQL):

```
CREATE TRIGGER trg_KiemTraDonGia
ON CT_PHIEU_XUAT
AFTER INSERT, UPDATE
AS
BEGIN
   -- Kiểm tra xem có bản ghi nào bị sai đơn giá hay không
   IF EXISTS (
      SELECT 1
      FROM inserted i
      INNER JOIN HANG_HOA hh ON i.MaHH = hh.MAHH
      WHERE i.DonGia <> hh.DonGiaHienHanh
   )
   BEGIN
      -- Nếu có, đưa ra thông báo lỗi và hoàn tác giao dịch
      RAISERROR('Đơn giá trong chi tiết phiếu xuất không khớp với đơn giá hiện hành của hàng hóa.', 16, 1)
      ROLLBACK TRANSACTION
   END
END;
GO
```

Giải thích:

- CREATE TRIGGER trg_KiemTraDonGia: Tạo một Trigger với tên trg_KiemTraDonGia.
- ON CT_PHIEU_XUAT: Chỉ định bảng mà Trigger được liên kết (CT_PHIEU_XUAT).
- AFTER INSERT, UPDATE: Chỉ định Trigger sẽ được kích hoạt sau khi thực hiện thao tác INSERT hoặc UPDATE.
- inserted: Bảng ảo chứa các bản ghi vừa được chèn hoặc cập nhật.
- INNER JOIN HANG_HOA: Kết nối với bảng HANG_HOA để lấy DonGiaHienHanh.
- WHERE i.DonGia <> hh.DonGiaHienHanh: Kiểm tra xem đơn giá trong CT_PHIEU_XUAT có khác với đơn giá hiện hành trong HANG_HOA hay không.
- RAISERROR(...): Đưa ra thông báo lỗi nếu có sự không khớp.
- ROLLBACK TRANSACTION: Hoàn tác thao tác INSERT hoặc UPDATE để ngăn chặn việc dữ liệu không hợp lệ được lưu vào cơ sở dữ liệu.

1.2.3 Cơ chế hoạt động của Trigger

- **3 biến cố kích hoạt 1 trigger:** INSERT, UPDATE, DELETE

1.2.4 Tạo mới trigger

Cú pháp:

```
CREATE TRIGGER Tên_trigger
ON tên_table| tên_view
AFTER | INSTEAD OF biến_cố_kích_hoạt_trigger
AS
        -- Các câu lệnh T-SQL
```

Giải thích:

- **CREATE TRIGGER Tên_trigger:** Khai báo tạo một Trigger mới với tên được chỉ định (Tên_trigger). Tên Trigger phải tuân theo quy tắc đặt tên đối tượng trong SQL Server.
- **ON tên_table | tên_view:** Chỉ định đối tượng mà Trigger được liên kết. Trigger có thể được liên kết với một **bảng** (tên_table) hoặc một **view** (tên_view).
- **{AFTER | INSTEAD OF}:** Xác định thời điểm Trigger được kích hoạt so với sự kiện kích hoạt:
  - **AFTER:** Trigger được kích hoạt ***sau khi* sự kiện kích hoạt (INSERT, UPDATE, DELETE) đã được thực hiện thành công** trên bảng. Đây là giá trị mặc định nếu bạn không chỉ định gì cả. **Trigger AFTER không thể được sử dụng trên view.**
  - **INSTEAD OF: Trigger được kích hoạt *thay vì* sự kiện kích hoạt.** Điều này có nghĩa là câu lệnh INSERT, UPDATE hoặc DELETE ban đầu sẽ không được thực hiện trực tiếp. Thay vào đó, **Trigger sẽ thực hiện các hành động được định nghĩa bên trong nó.** Trigger INSTEAD OF **chỉ được sử dụng trên view** để cho phép cập nhật dữ liệu thông qua view (view thường không cho phép cập nhật trực tiếp).
- **biến_cố_kích_hoạt_trigger:** Chỉ định sự kiện nào sẽ kích hoạt Trigger. Có thể là một hoặc kết hợp của các sự kiện sau:
  - **INSERT**: Trigger được kích hoạt sau khi một bản ghi mới được chèn vào bảng.
  - **UPDATE**: Trigger được kích hoạt sau khi một bản ghi trong bảng được cập nhật. Bạn có thể *chỉ định các cột cụ thể mà khi cập nhật sẽ kích hoạt Trigger bằng mệnh đề* **FOR | AFTER UPDATE OF column1 [, column2...].**
  - **DELETE**: Trigger được kích hoạt sau khi một bản ghi bị xóa khỏi bảng.
- **AS BEGIN ... END:** Khối lệnh T-SQL chứa các hành động mà Trigger sẽ thực hiện khi được kích hoạt. Bên trong khối này, bạn có thể sử dụng các biến đặc biệt như inserted và deleted:
  - **inserted:** Một bảng ảo chứa các **bản ghi *mới* được chèn vào** (trong Trigger AFTER INSERT hoặc INSTEAD OF INSERT) hoặc các bản ghi ***sau khi cập nhật*** (trong Trigger AFTER UPDATE hoặc INSTEAD OF UPDATE).
  - **deleted:** Một bảng ảo chứa các **bản ghi *bị xóa*** (trong Trigger AFTER DELETE hoặc INSTEAD OF DELETE) hoặc các bản ghi ***trước khi cập nhật*** (trong Trigger AFTER UPDATE hoặc INSTEAD OF UPDATE).
  - Trigger lưu trữ dữ liệu của mẫu tin **vừa cập nhật** là **sự phối hợp của 2 table DELELTED và INSERTED**

- **AFTER có thể thay bằng FOR**. Thực ra, **FOR không thay thế hoàn toàn AFTER.** Trong ngữ cảnh của Trigger, FOR được sử dụng *kết hợp* với AFTER và thường được bỏ qua vì AFTER là mặc định. Ví dụ: FOR INSERT tương đương với AFTER INSERT. Cú pháp FOR được sử dụng nhiều hơn trong các ngữ cảnh khác của T-SQL, ví dụ như trong CURSOR.

Ví dụ:

```
CREATE TRIGGER Them_HH
ON HANG_HOA
AS
        Select * From Inserted
```

- **Chèn dữ liệu**

```
INSERT HANG_HOA(MaHH, TenHH)
VALUES('TV01', 'Tivi Sony')
CREATE TRIGGER SUA_HHON HANG_HOA
AFTER UPDATE
AS
    Select * From Inserted
    Select * From Deleted
```

- **Cập nhật dữ liệu**

```
UPDATEHANG_HOA
SET Ten_HH = 'Man Hinh Sony'
WHERE MaHH = 'TV01'
CREATE TRIGGER Xoa_HHON HANG_HOA
AFTER DELETE
AS
        Select * From Inserted
        Select * From Deleted
```

- **Xóa dữ liệu**

```
DELETE HANG_HOAWHERE MaHH = 'TV01'
```

1.2.5 Các thao tác Trigger phổ biến

1.2.5.1 Thêm mới mẩu tin

Kiểm tra ràng buộc dữ liệu:

- Khoá ngoại
- Miền giá trị
- Liên bộ trên một quan hệ
- Liên thuộc tính trong cùng một bảng
- Liên thuộc tính của nhiều bảng khác nhau

Ví dụ

HOADON_DH(**MaHD**, NgayDH, MaKH)
PHIEU_XUAT(**MaPX**, NgayXuat, **MaHD**)
CHITIET_DH(**MAHD, MaHH**, SoLuong, DonGia)

- **Xây dựng Trigger trong bảng PHIEU_XUAT** để kiểm tra các ràng buộc toàn vẹn dữ liệu khi **người dùng thêm mới thông tin của một phiếu xuất hàng cho một bảng hoá đơn đặt hàng trước đó.** Các ràng buộc toàn vẹn dữ liệu bao gồm
  - **Khoá ngoại:** cần kiểm tra **số đặt hàng phải tồn tại** trong bảng đơn đặt hàng.
  - **Miền giá trị**: cần kiểm tra **ngày giao hàng phải ở sau ngày đặt hàng.**

```
CREATE TRIGGER tg_PhieuXuat_Insert
ON PHIEU_XUAT
FOR INSERT
AS
BEGIN
        DECLARE @NgayHD datetime, @ErrMsg varchar(200)
        -- Kiểm tra số hoá đơn đã có trong bảng DONDH không?
        IF NOT EXISTS(Select * From Inserted I, HOADON_DH D
                        Where I.MaHD= D.MaHD)
        Begin
                Rollback TRAN
                Raierror('Số đơn đặt hàng không tồn tại', 16,1)
                Return
        End
```

```
        --Tính ra ngày đặt hàng
        Select @NgayDH=NgayDH
        From HoaDon_DH D, Inserted I
        Where D.MaHD = I.MaHD

        -- Kiểm tra ngày giao hàng phải sau ngày đặt hàng
        IF @NgayDH > (Select ngayxuat From Inserted)
        Begin
                Set @ErrMsg = 'ngày giao hàng phải ở sau ngày:' + Convert(char(10), ngayDH, 103)
                Raierror(@ErrMsg,16,1)
                Rollback TRAN
        End
END
```

Tuy nhiên đoạn mã trên chỉ cho kiểm tra trên 1 bản ghi được thêm vào bảng. Để thực hiện trên nhiều bản ghi:

```
CREATE TRIGGER tg_PhieuXuat_Insert
ON PHIEU_XUAT
AFTER INSERT -- Nên dùng AFTER thay vì FOR vì AFTER là mặc định và rõ ràng hơn
AS
BEGIN
   SET NOCOUNT ON; -- Tối ưu hiệu suất, ngăn chặn việc trả về số lượng hàng bị ảnh hưởng

   -- Kiểm tra số hoá đơn đã có trong bảng HOADON_DH hay chưa cho TẤT CẢ các bản ghi được chèn
   IF EXISTS (
      SELECT 1
      FROM inserted i
      WHERE NOT EXISTS (SELECT 1 FROM HOADON_DH d WHERE i.MaHD = d.MaHD)
   )
   BEGIN
      RAISERROR('Một hoặc nhiều số đơn đặt hàng không tồn tại.', 16, 1)
      ROLLBACK TRANSACTION
      RETURN
   END

   -- Kiểm tra ngày giao hàng phải sau ngày đặt hàng cho TẤT CẢ các bản ghi được chèn
   IF EXISTS (
      SELECT 1
      FROM inserted i
      INNER JOIN HOADON_DH d ON i.MaHD = d.MaHD
      WHERE d.NgayDH >= i.NgayXuat
   )
   BEGIN
      RAISERROR('Ngày giao hàng phải sau ngày đặt hàng.', 16, 1)
      ROLLBACK TRANSACTION
      RETURN
   END
END;
GO
```

1.5.2.2 Huỷ bỏ mẫu tin

Ví dụ: khi **xoá** một số hoá đơn đặt hàng **trong bảng HOADON_DH** cần phải kiểm tra các RBTV dữ liệu sau:

- Kiểm tra xem **đơn đặt hàng bị xoá đã được xuất hàng chưa?** Nếu **đã được xuất rồi thì thông báo không thể xoá đơn đặt hàng được.**
- Ngược lại thì xoá dữ liệu liên quan bên bảng chi tiết đơn đặt hàng (CHITIET_HD)

```
CREATE TRIGGER tg_HOADON_Delete
ON HOADON_DH
FOR DELETE
AS
        DECLARE @SoPX char(5), @ErrMsg char(200), @Delete_Err int

        -- Kiểm tra xem đơn hàng đã được xuất chưa
        IF EXISTS(Select MaPX From PHIEU_XUAT
                        Where MaHD IN(Select MaHD From Deleted))
        Begin
                Select @MaPX = MaPX From PHIEU_XUAT
                Where MaHD In (Select MaHD From Deleted)
                Set @ErrMsg = 'Đơn đặt hàng đã được nhập theo số xuất hàng '+ @SoPX + char(13)
                + '.Không thể huỷ được'
                RaiError(@ErrMsg,16,1)
                Rollback TRAN
        End
        Else
        Begin
                -- Xoá tự động chi tiết các đơn đặt hàng liên quan
                Delete CHITIET_DH
                Where MaHD In(Select MaHD From DELETED)
                Set @Delete_Err = @@ERROR
                IF @Delete_Err <> 0
                Begin
                        Set @ErrMsg = 'Lỗi vi phạm xóa trên bảng chi tiết đặt hàng'
                        RaiError(@ErrMsg, 16, 1)
                        Rollback Tran
                End
        End
```

1.5.2.3 Sửa đổi mẫu tin

- Kiểm tra ràng buộc dữ liệu
  - Khoá ngoại
  - Miền giá trị
  - Liên bộ trên một quan hệ
  - Liên thuộc tính trong cùng một bảng
  - Liên thuộc tính của nhiều bảng khác nhau

- **Hàm Update**
  - **Ý nghĩa:** kiểm tra **dữ liệu của cột** bên trong bảng **có bị thay đổi trong các trigger sửa đổi dữ liệu** hay không.
  - **Cú pháp: UPDATE (tên_cột) (biểu thức luận lý)**
    - **tên_cột:** tên cột mà chúng ta muốn kiểm tra xem dữ liệu tại đó có bị sửa đổi trong trigger không.
    - **Biểu thức luận lý:** trả về True khi giá trị dữ liệu của cột đã bị sửa đổi, ngược lại trả về False khi giá trị dữ liệu của cột không bị sửa đổi

Ví dụ: sửa đổi thông tin của một số đặt hàng bên trong bảng HOADON_DH cần phải kiểm tra các ràng buộc toàn vẹn dữ liệu sau:

- Không cho phép sửa đổi dữ liệu tại **cột MaDH hoặc MaKH** vì khi đó dữ liệu sẽ ảnh hưởng đến nhiều bảng.
- Sửa đổi giá trị cột **ngày đặt hàng** thì phải đảm bảo luôn luôn **trước ngày giao hàng đầu tiên của số đặt hàng đó (nếu đơn đặt hàng đã có giao hàng).**

```
CREATE TRIGGER tg_HOADON_DH_Update
ON DONDH
FOR UPDATE
AS
        Declare @MinNgayXH date, @ErrMsg varchar(200)
        -- Khi sửa đổi các cột MaDH hoặc MaKH
        IF Update(MaDH) OR Update(MaKH)
        Begin
                Rollback Tran
                Set @ErrMsg = 'Không thể thay đổi số đặt hàng hoặc mã khách hàng'
                RaisError(@ErrMsg, 16, 1)
                Return
        End

        -- Khi sửa đổi ngày đặt hàng
        IF Update(NgayDH)
        Begin
                -- Kiểm tra đơn đặt hàng đã được xuất chưa
                IF EXISTS (Select MaPX From PHIEU_XUAT PX, deleted d
                                where px.mapx=d.mapx)
                Begin
                        -- Tính ra ngày xuất hàng đầu tiên
                        -- Tính ngày xuất hàng sớm nhất (Min(NgayXuat)) cho các phiếu xuất liên quan đến đơn hàng đang được cập nhật.
                        Select @MinNgayXH = Min(NgayXuat)
                        From PHIEU_XUAT PX, DELETED D
                        Where PX.MaDH = D.MaDH

                        --kiểm tra giá trị ngày đặt hàng sau khi sửa đổi phải luôn trước ngày giao hàng đầu tiên
                        IF @MinNgayXH < (Select NgayDH From Inserted)
                        Begin
                                Rollback tran
```

```
                    Set @ErrMsg = ‘Ngày đặt hàng phải ở trước ngày:’ +
                    Convert(char(10),@MinNgayXH, 103)
                    RaisError(@ErrMsg, 16, 1)
                End
            End
        End
```

Nếu có nhiều bảng dữ liệu cần cập nhật đồng thời

```
CREATE TRIGGER tg_HOADON_DH_Update
ON DONDH
AFTER UPDATE -- Nên dùng AFTER rõ ràng hơn
AS
BEGIN
   SET NOCOUNT ON;

   -- Ngăn chặn thay đổi MaDH hoặc MaKH cho TẤT CẢ các bản ghi được cập nhật
   IF UPDATE(MaDH) OR UPDATE(MaKH)
   BEGIN
      RAISERROR('Không thể thay đổi số đặt hàng hoặc mã khách hàng.', 16, 1)
      ROLLBACK TRANSACTION
      RETURN
   END

   -- Kiểm tra ngày đặt hàng so với ngày xuất hàng cho TẤT CẢ các bản ghi được cập nhật
   IF EXISTS (
      SELECT 1
      FROM inserted i
      INNER JOIN deleted d ON i.MaDH = d.MaDH -- Kết hợp inserted và deleted dựa trên MaDH
      WHERE EXISTS (
         SELECT 1
         FROM PHIEU_XUAT px
         WHERE px.MaDH = i.MaDH AND px.NgayXuat <= i.NgayDH -- Kiểm tra NgayXuat <=
NgayDH
      )
   )
   BEGIN
      RAISERROR('Ngày đặt hàng không được sau hoặc bằng bất kỳ ngày xuất hàng nào liên quan.',
16, 1)
      ROLLBACK TRANSACTION
      RETURN
   END
END;
GO
```

1.5.2.4 Trigger cập nhật giá trị tự động

- **Ví dụ:** dữ liệu của bảng **TONKHO** sẽ được **tính tự động** từ dữ liệu của các bảng liên quan đến việc nhập hàng và việc xuất hàng, cụ thể sẽ là các bảng: **PNHAP**, **CTPNHAP**, **PXUAT**, và **CTPXUAT**
- Khi **thêm mới các thông tin** của chi tiết một phiếu nhập hàng vào bảng **CTNHAP**, chúng ta cần kiểm tra các RBTV dữ liệu
  - Kiểm tra ***số phiếu nhập phải có trong bảng PNHAP***.
  - Kiểm tra ***mã vật tư phải có trong danh sách chi tiết***, ***danh sách các mã vật tư phải có trong chi tiết đơn đặt hàng*** trước đó.
  - Kiểm tra ***tổng số lượng nhập hàng*** vẫn còn ***ít hơn số lượng đặt hàng*** của vật tư đó.
- Nếu tất cả các RBTV dữ liệu ở trên đều hợp lệ thì **tăng giá trị của cột tổng số lượng nhập** trong bảng TONKHO và **cột tổng trị giá** trong bảng PNHAP.

---

1. Bảng PNHAP (Phiếu Nhập)

```
CREATE TABLE PNHAP (
    SoPhieuNhap INT PRIMARY KEY,        -- Số phiếu nhập (khóa chính)
    NgayNhap DATE,                      -- Ngày nhập hàng
    NhaCungCap NVARCHAR(255),           -- Nhà cung cấp
    TongTriGia DECIMAL(18, 2) DEFAULT 0 -- Tổng trị giá phiếu nhập
);
```

2. Bảng CTPNHAP (Chi Tiết Phiếu Nhập)

```
CREATE TABLE CTPNHAP (
    ID INT IDENTITY(1,1) PRIMARY KEY,   -- Khóa chính tự tăng
    SoPhieuNhap INT,                    -- Số phiếu nhập (liên kết tới PNHAP)
    MaVatTu NVARCHAR(50),               -- Mã vật tư
    SoLuong INT,                        -- Số lượng nhập
    DonGia DECIMAL(18, 2),              -- Đơn giá nhập
    FOREIGN KEY (SoPhieuNhap) REFERENCES PNHAP(SoPhieuNhap)
);
```

3. Bảng PXUAT (Phiếu Xuất)

```
CREATE TABLE PXUAT (
    SoPhieuXuat INT PRIMARY KEY,        -- Số phiếu xuất (khóa chính)
    NgayXuat DATE,                      -- Ngày xuất hàng
    NguoiNhan NVARCHAR(255),            -- Người nhận hàng
    TongTriGia DECIMAL(18, 2) DEFAULT 0 -- Tổng trị giá phiếu xuất
);
```

4. Bảng CTPXUAT (Chi Tiết Phiếu Xuất)

```
CREATE TABLE CTPXUAT (
    ID INT IDENTITY(1,1) PRIMARY KEY,   -- Khóa chính tự tăng
    SoPhieuXuat INT,                    -- Số phiếu xuất (liên kết tới PXUAT)
    MaVatTu NVARCHAR(50),               -- Mã vật tư
    SoLuong INT,                        -- Số lượng xuất
    DonGia DECIMAL(18, 2),              -- Đơn giá xuất
    FOREIGN KEY (SoPhieuXuat) REFERENCES PXUAT(SoPhieuXuat)
);
```

5. Bảng TONKHO (Tồn Kho)

```
CREATE TABLE TONKHO (
    MaVatTu NVARCHAR(50) PRIMARY KEY,   -- Mã vật tư (khóa chính)
    TenVatTu NVARCHAR(255),             -- Tên vật tư
    TongSoLuongNhap INT DEFAULT 0,      -- Tổng số lượng nhập
    TongSoLuongXuat INT DEFAULT 0,      -- Tổng số lượng xuất
    SoLuongTon INT DEFAULT 0            -- Số lượng tồn (tự tính)
);
```

6. Bảng DatHang (Đơn Đặt Hàng)

```
CREATE TABLE DatHang (
    SoDonHang INT PRIMARY KEY,          -- Số đơn hàng (khóa chính)
    MaVatTu NVARCHAR(50),               -- Mã vật tư
    SoLuongDat INT,                     -- Số lượng đặt hàng
    NgayDat DATE,                       -- Ngày đặt hàng
    FOREIGN KEY (MaVatTu) REFERENCES TONKHO(MaVatTu)
);
```

Chức năng của từng bảng

1. **PNHAP**: Lưu thông tin các phiếu nhập hàng.
2. **CTPNHAP**: Lưu chi tiết từng vật tư nhập trong từng phiếu nhập.
3. **PXUAT**: Lưu thông tin các phiếu xuất hàng.
4. **CTPXUAT**: Lưu chi tiết từng vật tư xuất trong từng phiếu xuất.
5. **TONKHO**: Quản lý tồn kho của từng vật tư.
6. **DatHang**: Quản lý các đơn đặt hàng của từng vật tư.

```
CREATE TRIGGER trg_Insert_CTPNHAP
ON CTPNHAP
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    -- Kiểm tra ràng buộc: Số phiếu nhập phải tồn tại trong bảng PNHAP
    IF EXISTS (
        SELECT 1
        FROM INSERTED i
        LEFT JOIN PNHAP p ON i.SoPhieuNhap = p.SoPhieuNhap
        WHERE p.SoPhieuNhap IS NULL
    )
    BEGIN
        RAISERROR ('Số phiếu nhập không tồn tại trong bảng PNHAP.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    -- Kiểm tra ràng buộc: Mã vật tư phải có trong danh sách chi tiết đặt hàng
    IF EXISTS (
        SELECT 1
        FROM INSERTED i
        LEFT JOIN DatHang dh ON i.MaVatTu = dh.MaVatTu
```

```
        WHERE dh.MaVatTu IS NULL
    )
    BEGIN
        RAISERROR ('Mã vật tư không có trong danh sách đặt hàng.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    -- Kiểm tra ràng buộc: Tổng số lượng nhập không vượt quá số lượng đặt hàng
    IF EXISTS (
        SELECT 1
        FROM INSERTED i
        JOIN TONKHO tk ON i.MaVatTu = tk.MaVatTu
        JOIN DatHang dh ON i.MaVatTu = dh.MaVatTu
        WHERE (tk.TongSoLuongNhap + i.SoLuong) > dh.SoLuongDat
    )
    BEGIN
        RAISERROR ('Tổng số lượng nhập vượt quá số lượng đặt hàng.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    -- Cập nhật tổng số lượng nhập trong bảng TONKHO
    UPDATE tk
    SET tk.TongSoLuongNhap = tk.TongSoLuongNhap + i.SoLuong
    FROM TONKHO tk
    JOIN INSERTED i ON tk.MaVatTu = i.MaVatTu;

    -- Cập nhật tổng trị giá trong bảng PNHAP
    UPDATE p
    SET p.TongTriGia = p.TongTriGia + (i.SoLuong * i.DonGia)
    FROM PNHAP p
    JOIN INSERTED i ON p.SoPhieuNhap = i.SoPhieuNhap;
END;
GO
```

1.3 Hàm do người dùng đinh nghĩa (user defined functions)

- **Khái quát:** UDFs là một **chương trình con đảm trách một xử lý nào đó** với đặc tính là sẽ **nhận các tham số đầu vào và trả về một giá trị** kết quả xử lý *tính toán được dựa trên các tham số đầu vào đã nhận.*
- UDFs phân thành 2 nhóm:
  - **Hàm xác định (determinitic):** *luôn **trả về cùng giá trị*** nếu *giá trị các tham số được truyền vào là như nhau.*
  - **Hàm không xác định (non-determinitic):** Hàm không xác định có thể *cho ra kết quả **khác biệt tại mỗi thời điểm** chúng được gọi*
- UDFs là sự kết hợp của 2 đối tượng **View** và **Store Procedure**
- Khắc phục một số hạn chế của View và Store Procedure
  - *Store Procedure không thể là một phần của câu lệnh SELECT* nhưng **UDFs thì có**
  - *View không hỗ trợ đệ quy* trong khi **UDFs thì có thể làm được điều này.**

1.3.1 Tạo UDF

```
CREATE FUNCTION [Tên_Function] (tham số)
RETURNS kiểu_dữ_liệu_trả_về
AS
BEGIN
        --Các lệnh
        Return
END
```

Returns: thiết lập kiểu dữ liệu trả về của UDFs. Có 2 cách thiết lập chính

- Trả về **giá trị kiểu vô hướng:** một chuỗi, một giá trị logic hoặc một kiểu số.
- Trả về **một bảng**: có thể trả về hai loại bảng
  - **Inline table:** khắc phục được nhược điểm không có tham số của VIEW. Có nghĩa rằng UDFs loại inline table **giống như một VIEW có tham số.**
  - **Multistatement table:** UDFs loại này **giống với Store Procedure**. Loại này luôn trả về **1 biến table.** Thực hiện các câu SELECT phức tạp, cho phép thực hiện các câu lệnh logic khác như UPDATE, INSERT INTO…

Ví dụ 1: Tạo View như sau

```
CREATE VIEW DS_SinhVien
AS
Select * From SINHVIEN Where MaLop = 'C5CT09'
CREATE FUNCTION DS_SinhVien(@MaLop char(6))
RETURNS Table AS        -- RETURNS chứa duy nhất từ khoá table
        Return (Select * From SinhVien Where MaLop=@MaLop)
        -- Chứa một câu SELECT đơn giản nằm trong cặp dấu ngoặc đơn.
```

Ví dụ 2:

```
CREATE FUNCTION F_DSHangHoa(@LoaiHang varchar(50), @PhanTram numeric )
RETURNS @DSHangHoa Table  -- Biến kiểu table
(       MaMH char(10),
        TenMH varchar(50),
        DonGiaKhuyenMai numeric)
AS
Begin
        INSERT INTO @DSHangHoa(MaMH, TenMH, DonGiaKhuyenMai)
        Select MaMH, TenMH, DonGiaHienHanh
        From HANGHOA
        Where loaihang = @LoaiHang
        Update @DSHangHoa
        Set DonGiaKhuyenMai= DonGiaKhuyenMai- (DonGiaKhuyenMai * @Phantram)/100
        Return
End
```

Ví dụ 3:

```
CREATE FUNCTION DonGiaHienHanh(@MaMH varchar(50))
RETURNS numeric
AS
BEGIN
        Return(Select DonGiaHienHanh
        From HANGHOA
        WHERE MaMH = @MaMH)
END
```

Ví dụ 4:

```
CREATE FUNCTION Test_function(@b int, @c int)
RETURNS int
as
BEGIN
        declare @kq int
        if @b>2
                set @kq=@b+@c
        else
                set @kq=@b+@c+1
        return @kq
END
```

1.4 View trong SQL Server

Trong SQL Server, **View là đoạn lệnh truy vấn đã được viết sẵn và lưu bên trong cơ sở dữ liệu.** Một View thì bao gồm 1 câu lệnh SELECT và khi bạn chạy View thì bạn sẽ thấy kết quả giống như khi bạn mở 1 Table. Các bạn có thể tưởng tượng nó **giống như một Table ảo**. Bởi vì nó có thể **tổng hợp dữ liệu từ nhiều Table để tạo thành 1 Table ảo.**

1.4.1 Lợi ích của Views

View rất hữu dụng khi bạn muốn cho nhiều người người truy cập ở các permission khác nhau. Cụ thể là:

- **Hạn chế truy cập tới các Table cụ thể.** Chỉ cho phép được xem qua View.
- **Hạn chế truy cập vào vào Column của Table.** Khi truy cập thông qua View bạn không thể biết được tên Column mà View đó truy cập vào.
- **Liên kết các Column từ rất nhiều Table** vào thành Table mới được thể hiện qua View.
- **Trình bày các thông tin tổng hợp** (VD: sử dụng funtion như COUNT, SUM, ...)

1.4.2 Cú pháp View

Để tạo 1 View bạn có thể sử câu lệnh "CREATE VIEW". Cụ thể như sau:

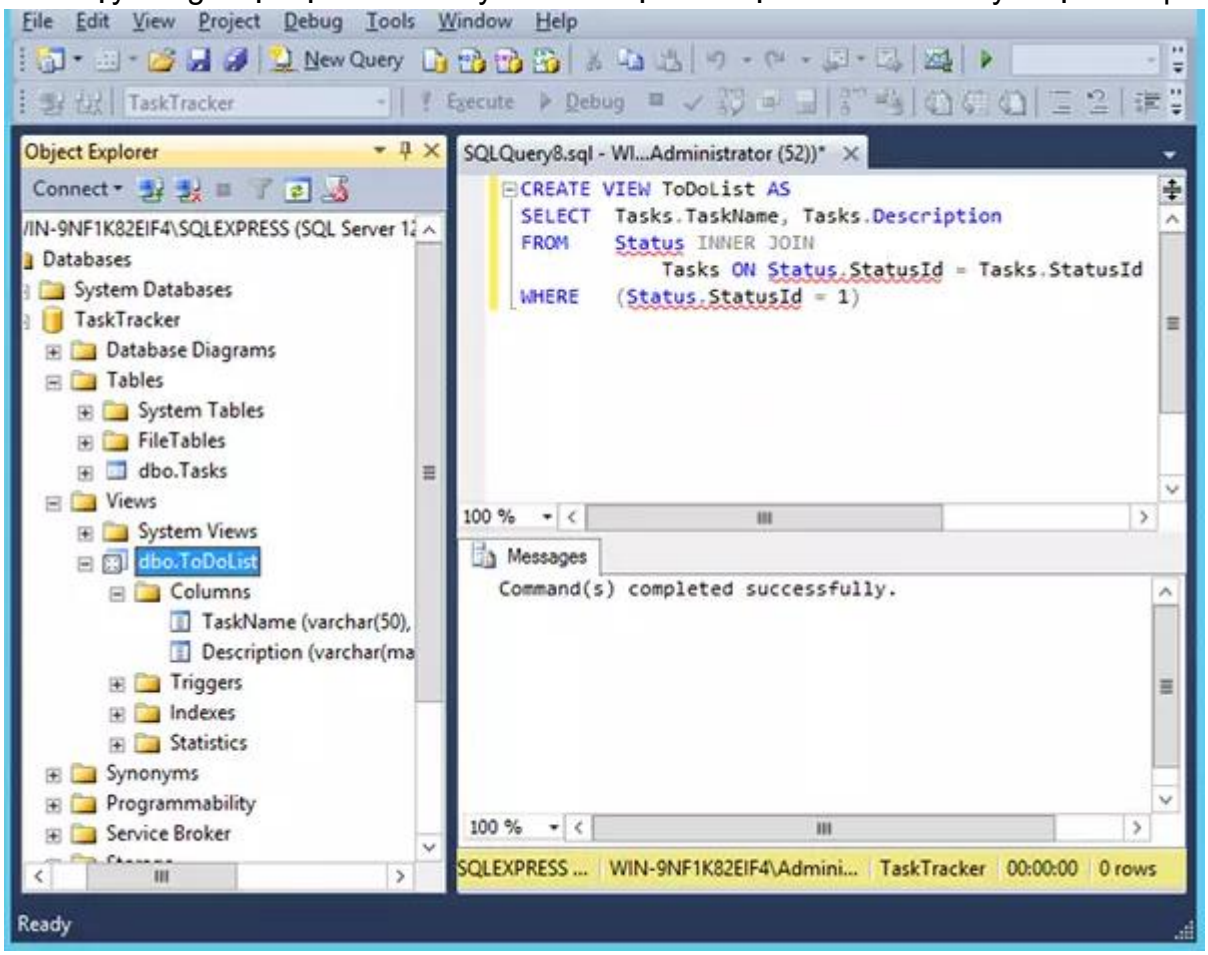
```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

1.4.3 Tạo View

Chúng ta hãy thử tạo 1 View có tên là "ToDoList" và lưu nó vào cơ sở dữ liệu nhé. Cơ bản thì tất cả những gì chúng ta làm là sử dụng lệnh "CREATE VIEW ToDoList AS" trước câu lệnh truy vấn, như sau:

```
CREATE VIEW ToDoList AS
SELECT      Tasks.TaskName, Tasks.Description
FROM        Status
INNER JOIN Tasks ON Status.StatusId = Tasks.StatusId
WHERE       (Status.StatusId = 1)
```

Khi bạn đã chạy xong đoạn lệnh trên. Hãy làm mới lại thư mục Views. Để thấy được kết quả:

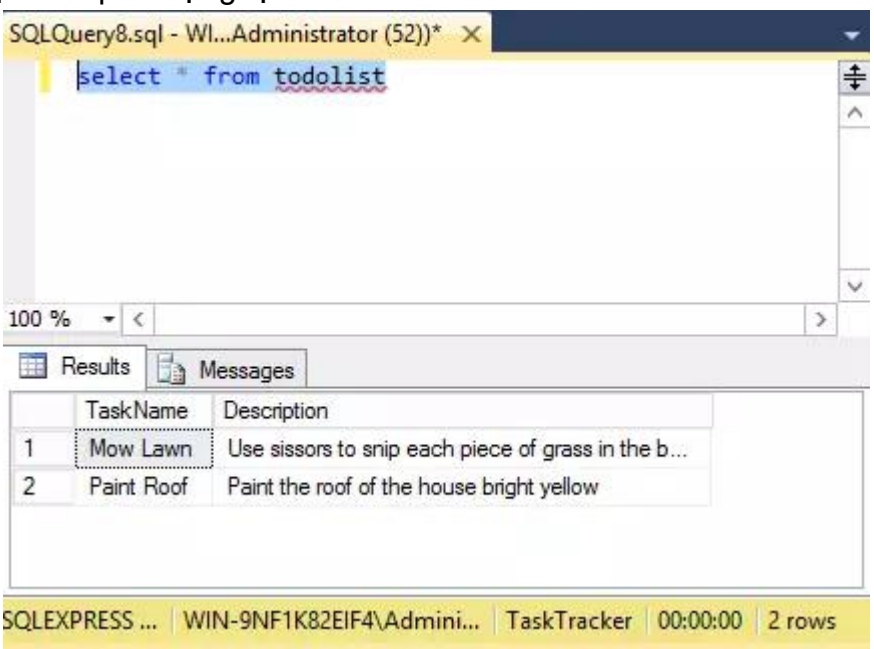


1.4.4 Mở View

Vì bây giờ bạn đã tạo xong View, bạn có thể **xem kết quả giống như khi mở 1 Table.** Và thay vì phải làm các câu lệnh INNER JOIN loằng ngoằng như phần trên thì bạn chỉ cần phải gọi câu lệnh:

```
select * from todolist
```

để có thể nhận được kết quả tượng tự như khi làm INNER JOIN.



1.4.5 Dữ liệu được cập nhật

Vì View lấy dữ liệu từ các Table của cơ sở dữ liệu nên khi dữ liệu bên trong các Table thay đổi thì khi thực hiện mở lại các View dữ liệu sẽ cũng thay đổi theo. **Cho nên sau khi cập nhật dữ liệu mới cho các Table thì chỉ cần mở lại View và các bạn sẽ có được những bản ghi mới nhất.**

1.4.6 Chỉnh sửa View

Bạn có thể *chỉnh sửa View* đã tồn tại bằng cách sử dụng **"ALTER"** thay vì "CREATE".
Chẳng hạn như với câu lệnh truy vấn đã sử dụng "CREATE" ở trên. Nếu muốn **thay đổi điều kiện từ "StatusId" thành "StatusName" thì sẽ như sau:**

```
ALTER VIEW ToDoList AS
SELECT      Tasks.TaskName, Tasks.Description
FROM        Status
INNER JOIN Tasks ON Status.StatusId = Tasks.StatusId
WHERE       (Status.StatusName = 'To Do')
```

1.4.7 SQL Updating a View

A view can be updated with the **CREATE OR REPLACE VIEW** statement.

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

The following SQL **adds the "City" column** to the "Brazil Customers" view:

Example

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = 'Brazil';
```

1.4.8 SQL Dropping a View

A view is **deleted** with the **DROP VIEW** statement.

SQL DROP VIEW Syntax

```
DROP VIEW view_name;
```

The following SQL drops the "Brazil Customers" view:

Example

```
DROP VIEW [Brazil Customers];
```