TÀI LIỆU HƯỚNG DẪN THỰC HÀNH MÔN CƠ SỞ DỮ LIỆU – IT004

BUỔI THỰC HÀNH 05

Hướng dẫn thực hành

Lê Võ Đình Kha – 18520872@gm.uit.edu.vn
Lê Thị Trà My – 19521861@gm.uit.edu.vn

GIỚI THIỆU NỘI DUNG BUỔI THỰC HÀNH SỐ 4

NỘI DUNG

Phần 1: Ôn tập buổi Thực hành số 4.

- Phép chia.
- Hàm tính toán và gom nhóm.

Phần 2: Ràng buộc toàn vẹn

- Loại phức tạp: Sử dụng TRIGGER

CÂU LỆNH TRUY VẤN - PHÉP CHIA

Ví dụ: Tìm SoHD,NGHD đã mua tất cả sản phẩm.

=> Tìm hóa đơn mà không có sản phẩm nào là không mua

- Tập bị chia R : **CTHD(SoHD,MaSP)**
- Tập chia S: **SANPHAM(MaSP)**
- Tập kết quả: **KQ(SoHD)**
- Kết KQ với HOADON để lấy ra **NGHD**

CÂU LỆNH TRUY VẤN - PHÉP CHIA

Ví dụ: Tìm SoHD,NGHD đã mua tất cả sản phẩm.

```
SELECT SoHD,NGHD
FROM HOADON
WHERE NOT EXISTS ( SELECT *
                   FROM SANPHAM
                   WHERE NOT EXISTS ( SELECT *
                                      FROM CTHD
                                      WHERE CTHD.MASP= SANPHAM.MASP
                                      AND CTHD.SoHD= HOADON.SoHD))
```

CÂU LỆNH TRUY VẤN - PHÉP CHIA

Ví dụ: Tìm SoHD đã mua tất cả sản phẩm.

```
SELECT SoHD
FROM CTHD C1
WHERE NOT EXISTS ( SELECT *
                   FROM SANPHAM
                   WHERE NOT EXISTS ( SELECT *
                                      FROM CTHD C2
                                      WHERE C2.MASP= SANPHAM.MASP
                                      AND C2.SoHD= C1.SoHD))
```

CÂU LỆNH TRUY VẤN – HÀM TÍNH TOÁN

Các hàm kết hợp được đặt ở mệnh đề SELECT

– **COUNT**

+ **COUNT (*)** đếm số dòng

+ **COUNT (<tên thuộc tính>)** đếm số giá trị **khác NULL** của thuộc tính

+ **COUNT (DISTINCT <tên thuộc tính>)** đếm số giá trị **khác nhau** và **khác NULL** của thuộc tính.

– **MIN** (<tên thuộc tính>)

– **MAX** (<tên thuộc tính>)

– **SUM** (<tên thuộc tính>)

– **AVG** (<tên thuộc tính>)

CÂU LỆNH TRUY VẤN – HÀM GOM NHÓM

SELECT <danh sách các cột>

FROM <danh sách các bảng>

WHERE <điều kiện>

GROUP BY <danh sách các cột gom nhóm>

Sau khi gom nhóm, mỗi nhóm các bộ sẽ có cùng giá trị tại các thuộc tính gom nhóm

VD: Cho biết số lượng nhân viên của từng phòng ban

```
SELECT PHG, COUNT(*) AS SL_NV
FROM NHANVIEN
GROUP BY PHG
```

CÂU LỆNH TRUY VẤN – HÀM GOM NHÓM

SELECT <danh sách các cột>

FROM <danh sách các bảng>

WHERE <điều kiện>

GROUP BY <danh sách các cột gom nhóm>

HAVING <điều kiện trên nhóm>

Ví dụ: Cho biết những nhân viên tham gia từ 2 đề án trở lên

```
SELECT MA_NVIEN
FROM PHANCONG
GROUP BY MA_NVIEN
HAVING COUNT(*) >= 2
```

CÂU LỆNH TRUY VẤN – HÀM GOM NHÓM

SELECT TOP 4 <danh sách các cột>

FROM <danh sách các bảng>

WHERE <điều kiện>

GROUP BY <danh sách các cột gom nhóm>

HAVING <điều kiện trên nhóm>

- **Select Top 4:** Trả về 4 dòng dữ liệu đầu tiên tìm thấy.
- **Kết hợp Order by:** lấy top theo thuộc tính cần xếp hạng.
- **Select Top 4 With Ties:** nếu có nhiều giá trị bằng với vị trí thứ 4 thì lấy hết. **Bắt buộc phải có Order by**
- Có thể sử dụng **SELECT DISTINCT TOP** để lọc các giá trị trùng

CÂU LỆNH TRUY VẤN

SELECT <danh sách các cột>

FROM <danh sách các bảng>

WHERE <điều kiện>

GROUP BY <danh sách các cột gom nhóm

HAVING <điều kiện trên nhóm>

ORDER BY < các thuộc tính sắp thứ tự>

RÀNG BUỘC TOÀN VẸN - TRIGGER

- Một quan hệ

+ Miền giá trị

+ Liên bộ

+ Liên thuộc tính

Ràng buộc toàn vẹn Constraint:

NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT

- Nhiều quan hệ

+ **Tham chiếu**

+ Liên bộ, liên quan hệ

+ Liên thuộc tính, liên quan hệ

+ Thuộc tính tổng hợp

+ Chu trình

Ràng buộc toàn vẹn sử dụng Trigger

RÀNG BUỘC TOÀN VẸN - TRIGGER

1. KHÁI NIỆM VỀ TRIGGER

- **Trigger** là một thủ tục đặc biệt, có thể xem như một công cụ kiểm tra lỗi. Công cụ này được sử dụng để khai báo ràng buộc dữ liệu cho một đối tượng table hoặc view và tự động thực thi khi một trong 3 phát biểu **Insert, Update, Delete** làm thay đổi dữ liệu trên đối tượng đó.
- **Trigger** không thể được gọi trực tiếp như các hàm, thủ tục bình thường, nó cũng không có tham số và giá trị trả về.
- **Trigger** chỉ được chạy khi toàn bộ câu lệnh **thêm, xóa, cập nhật** được thực hiện trọn vẹn. Lợi ích chính của trigger so với các công cụ kiểm tra khác đó là chúng có thể chứa các xử lý phức tạp trên các table có liên quan đến table đang thay đổi.

RÀNG BUỘC TOÀN VẸN - TRIGGER

1. KHÁI NIỆM VỀ TRIGGER

- **Trigger** có thể chứa phát biểu **ROLLBACK TRAN** ngay cả khi không có phát biểu **BEGIN TRAN**. Trong trường hợp phát biểu **ROLLBACK TRANSACTION** bên trong một trigger được thực hiện:

+ Nếu **trigger** này được kích hoạt bởi 1 phát biểu cập nhật từ bên trong một **transaction khác**, thì toàn bộ **transaction đó** bị hủy bỏ.

+ Nếu **trigger** được kích hoạt bởi 1 phát biểu cập nhật từ bên trong một gói lệnh, thì sẽ hủy bỏ toàn bộ gói lệnh đó.

- Dựa vào ứng dụng, có 3 loại trigger như sau: **Insert trigger, Update trigger, Delete trigger.**

RÀNG BUỘC TOÀN VẸN - TRIGGER

2. CÚ PHÁP TẠO TRIGGER

```
CREATE TRIGGER <trigger_name>
ON <table name>
( WITH <Encryption> )
FOR | AFTER | INSTEAD OF
(INSERT), (UPDATE), (DELETE)
AS
BEGIN
(Tập hợp các câu lệnh)
END
```

RÀNG BUỘC TOÀN VẸN - TRIGGER

Trong đó cần lưu ý:

- **trigger_name:** Tên trigger phải phân biệt.

- **ON <table name>:** tên table mà trigger sẽ thực hiện. Không sử dụng cú pháp trigger này cho View.

- **WITH ENCRYPTION:** Mã hóa trigger, không cho xem và sửa đổi câu lệnh tạo trigger..

RÀNG BUỘC TOÀN VẸN - TRIGGER

- **FOR | AFTER | INSTEAD OF DELETE, INSERT, UPDATE**: Dùng chỉ định những phát biểu cập nhật nào nào trên table sẽ kích hoạt trigger. Từ khóa AFTER và FOR là tương đương nhau. Khi thực hiện trigger, SQL sẽ tạo các bảng tạm: **INSERTED và DELETED.**

+ Khi chèn một hoặc nhiều dòng dữ liệu mới vào bảng thông qua một câu lệnh **INSERT** thì những dòng dữ liệu mới đó cũng được lưu trong **table INSERTED**.

+ Khi xóa một hoặc nhiều dòng dữ liệu trong bảng thông qua một câu lệnh **DELETE** thì các đối tượng bị xoá đó sẽ được di chuyển sang bảng tạm **DELETED**.

+ Khi cập nhật một hoặc nhiều dòng dữ liệu trong bảng thông qua một câu lệnh **UPDATE** thì: bảng đó và bảng **INSERTED** đều chứa các dòng dữ liệu mới, còn bảng **DELETED** chứa các dòng có nội dung cũ.

RÀNG BUỘC TOÀN VẸN - TRIGGER

+ Bạn không thể thay đổi dữ liệu trên các bảng tạm **DELETED và INSERTED**. Nhưng bạn có thể dùng 2 bảng này để xử lý các dòng dữ liệu trên các bảng có liên quan. Ngoài ra, trong **trigger Insert và Update**, bạn có thể thay đổi nội dung của các dòng dữ liệu mới thay đổi bằng lệnh Update trên bảng hiện tại (bảng mà trigger đang thao tác).

- **AS:** Từ khóa bắt đầu các hành động bên trong trigger. Trigger có thể chứa hầu hết các lệnh của T-SQL ngoại trừ một số lệnh sau:

+ Các lệnh **CREATE, ALTER và DROP**.

+ **TRUNCATE TABLE**

+ **SELECT INTO** (vì câu lệnh này cũng tạo ra bảng).

RÀNG BUỘC TOÀN VẸN - TRIGGER

- **Khai báo biến:** VD: declare @sl int, @hoten varchar(20)
- **IF ... ELSE :** tương tự trong lập trình C.
- **Print 'In dòng chữ ra màn hình' :** in chuỗi ra màn hình.
- **@@ERROR:** biến toàn cục chứa mã lỗi.
- **@@ROWCOUNT:** biến toàn cục chứa số dòng dữ liệu.
- **Lệnh về transaction:** (SQL Server thường mặc định auto commit)
  - + Commit / Commit tran/ Commit transaction
  - + Rollback/ Rollback tran/ Rollback transaction
- **Các bảng tạm:** Inserted, Deleted

RÀNG BUỘC TOÀN VẸN - TRIGGER

```
CREATE TRIGGER <trigger_name>
ON <table name>
FOR (INSERT), (UPDATE), (DELETE)
AS
BEGIN
    <Khai báo biến>
    If <Kiểm tra điều kiện>
        BEGIN
            Rollback Transaction
        END
END
```

RÀNG BUỘC TOÀN VẸN - TRIGGER

3. KÍCH HOẠT/VÔ HIỆU HÓA MỘT TRIGGER

ALTER TABLE *table* ENABLE | DISABLE TRIGGER ALL | trigger_name(,...n)

4. HIỆU CHỈNH TRIGGER

Bạn có thể thay đổi các lệnh cần thực hiện cũng như hành động cập nhật mà Trigger sẽ được gọi thực hiện.

ALTER TRIGGER trigger_name ...

5. XÓA TRIGGER

DROP TRIGGER {trigger} (,...n)

Nếu xóa một table thì tất cả trigger của nó cũng bị xóa.

STORE PROCEDURE

- Tạo Store Procedure

```
CREATE PROCEDURE < Tên Procedure> <Các tham số>
AS
    < Khai báo biến>
    < Nội dung chương trình>
GO
```

- Gọi Store Procedure

```
EXEC < Tên Procedure> <Truyền tham số tương ứng>
```

STORE PROCEDURE

- Tạo Store Procedure

```
CREATE PROCEDURE TIM_SP_NUOCSX @NUOC nvarchar(40)
AS
SELECT * FROM SANPHAM WHERE NUOCSX= @NUOC
GO
```

- Gọi Store Procedure

```
EXEC TIM_SP_NUOCSX 'Trung Quoc'
```

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

1. KHÁI NIỆM

- Cursor là kiểu dữ liệu cho phép truy xuất đến trên từng mẫu tin trong tập kết quả trả về bởi câu lệnh Select. Ngoài ra, bạn có thể sử dụng các phát biểu Update hoặc Delete để cập nhật hay xóa mẫu tin hiện hành trên các bảng cơ sở của Select bằng mệnh đề **WHERE CURRENT OF <Tên Cursor>** .

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

2. CÁC THAO TÁC CHUNG TRÊN CURSOR

- Khai báo cursor: **DECLARE <cursor_name> CURSOR FOR <lệnh Select>**
- Mở cursor : **OPEN <cursor_name>**

  Sau lệnh mở cursor, con trỏ mẫu tin hiện hành nằm ở vùng BOF.
- Xử lý mẫu tin trên cursor:

  Di chuyển mẫu tin hiện hành: **FETCH NEXT FROM cursor_name**

  Sử dụng phát biểu Update hoặc Delete để cập nhật hay xóa mẫu tin hiện hành
- Đóng cursor: **CLOSE <tên cursor>**
- Hủy bỏ cursor: **DEALLOCATE <TÊN CURSOR>**

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

3. KHAI BÁO CURSOR

DECLARE <CursorName> CURSOR

(LOCAL | GLOBAL) -- Phạm vi hoạt động

(FORWARD_ONLY | SCROLL) -- Phương thức di chuyển

(STATIC | KEYSET | DYNAMIC) -- Loại Cursor

(READ_ONLY | SCROLL_LOCKS | OPTIMISTIC) -- Xử lý đồng thời

(TYPE_WARNING)

FOR <lệnh Select>

(FOR UPDATE (OF ColumnName (,...n)))

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

4. PHẠM VI HOẠT ĐỘNG CURSOR

- Mặc định, cursor có phạm vi Global trên kết nối mà nó đã được tạo. Nghĩa là, bạn có thể sử dụng cursor trên các gói lệnh thực hiện trên kết nối đó, trừ Khi bạn đóng và giải phóng Cursor. Nếu bạn mở Cursor chưa đóng thì sẽ bị lỗi và có khi bị treo cho đến khi đóng kết nối. Với lý do đó, khi không sử dụng Cursor Global, bạn nên đóng và giải phóng Cursor.
- Cursor Local có phạm vi hoạt động bên trong gói lệnh đã tạo nó. Và tự giải phóng khi kết thúc gói lệnh.

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

5. PHƯƠNG THỨC DI CHUYỂN TRÊN CURSOR

Có 2 phương thức di chuyển:

- **FORWARD_ONLY:** là phương thức mặc định, chỉ cho phép di chuyển sang mẫu tin kế tiếp.
- **SCROLL:** Cho phép di chuyển lên xuống trong tập mẫu tin.

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

6. CÁC LOẠI CURSOR

Có 3 loại Cursor:

1. STATIC: có thuộc tính READ ONLY, do đó không thể cập nhật các bảng gốc thông qua Cursor này. Khi tạo Cursor Static, dữ liệu từ các bảng gốc sẽ được Copy sang một bảng tạm trong CSDL tempdb. Do đó, Nếu các table nguồn của Cursor bị thay đổi dữ liệu thì các dữ liệu không xuất hiện trên Cursor.

2. DYNAMIC: Cho phép cập nhật dữ liệu trên các table nguồn (dùng mệnh đề **WHERE CURRENT OF <tênCursor>** trong các phát biểu UPDATE or DELETE), và tự động hiển thị tất cả những thay đổi từ table nguồn. Tuy nhiên, dữ liệu và thứ tự của các mẫu tin trong tập mẫu tin có thể bị thay đổi.

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

6. CÁC LOẠI CURSOR

3. **KEYSET:** Giống như cursor Dynamic. Nhưng nó chỉ được tạo khi bảng nguồn có khai báo khóa, nếu không thì SQL tự động chuyển sang loại STATIC. Khi tạo Cursor KEYSET, Tập các khóa của bảng nguồn được lưu trên một table của CSDL tempdb. Do đó, việc xóa mẫu tin hoặc thay đổi giá trị khóa trên các bảng nguồn không thông qua Cursor sẽ không phản hồi trên tập mẫu tin.

Cursor kiểu **STATIC, KEYSET**, và **DYNAMIC** mặc định dùng phương thức **SCROLL**.

TYPE_WARNING: Gởi thông báo chú ý về client nếu Cursor thực hiện chuyển đổi ngầm định từ kiểu yêu cầu sang một kiểu khác.

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

7. XỬ LÝ ĐỒNG THỜI

Trong môi trường nhiều người dùng cùng làm việc trên cùng tập dữ liệu, Làm thế nào để người dùng chắc chắn rằng những thay đổi của họ không bị thay đổi bởi người dùng khác? Phụ thuộc vào kiểu Cursor mà bạn đã sử dụng, bạn không thể nhận thấy được những thay đổi cho đến khi bạn đóng Cursor và mở lại nó.

Trừ khi sử dụng cursor trong một transaction, nếu không các table nguồn của cursor không tự động khóa dữ liệu. SQL Server cung cấp **4 chọn lựa** cho phép ngăn cản việc sửa đổi mẫu tin cho tới khi thực hiện xong hoặc bằng cách khóa các table nguồn của cursor để bảo vệ các thay đổi của bạn.

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

7. XỬ LÝ ĐỒNG THỜI

4 chọn lựa bao gồm:

- **READ_ONLY:** Dùng khi chỉ truy xuất dữ liệu mà không sửa đổi dữ liệu.
- **SCROLL_LOCKS:** Khoá các dòng đã được đọc vào Cursor đối với các User khác.
- **OPTIMISTIC WITH VALUES:** Chỉ khóa các giá trị mà bạn vừa thay đổi. Nếu người dùng khác thay đổi các giá trị đó sẽ nhận được thông báo lỗi.
- **OPTIMISTIC WITH ROW VERSIONING:** Khi muốn cả dòng được cập nhật, không chỉ một vài Fields trong nó.

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

8. KHAI BÁO CỘT TRONG CURSOR ĐƯỢC PHÉP CẬP NHẬT

UPDATE (OF column_name (,...n))

- Nếu chỉ định **OF column_name (,...n)** chỉ những cột liệt kê mới được sửa đổi.
- Nếu chỉ định **UPDATE** mà không chỉ định danh sách cột, thì tất cả các cột đều có khả năng cập nhật trừ phi chỉ định **READ_ONLY.**

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

9. TRUY XUẤT DỮ LIỆU TRÊN CURSOR

```
FETCH ( NEXT | PRIOR | FIRST | LAST
        | ABSOLUTE { n | @nvar } | RELATIVE { n | @nvar } )
    FROM ( GLOBAL ) cursor_name
    ( INTO @variable_name ( ,...n ) )
```

- **NEXT:** Chuyển sang mẫu tin kế tiếp.
- **PRIOR:** Chuyển về mẫu tin trước đó.
- **FIRST:** Chuyển về mẫu tin đầu tiên.
- **LAST:** Chuyển đến mẫu tin cuối cùng.

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

9. TRUY XUẤT DỮ LIỆU TRÊN CURSOR

```
FETCH ( NEXT | PRIOR | FIRST | LAST
        | ABSOLUTE { n | @nvar } | RELATIVE { n | @nvar } )
    FROM ( GLOBAL ) cursor_name
    ( INTO @variable_name ( ,...n ) )
```

- **ABSOLUTE {n | @nvar}**: Nếu n hoặc @nvar > 0, tìm đến dòng thứ n tính từ dòng đầu tiên đếm xuống trong tập mẫu tin. Nếu n hoặc @nvar < 0, tìm đến dòng thứ n tính từ dòng cuối cùng đếm lên. Nếu n hoặc @nvar = 0, chuyển đến vùng BOF và không có giá trị trả về. Hằng số n phải là số nguyên và biến @nvar phải thuộc kiểu smallint, tinyint, hoặc int. Không sử dụng phương thức ABSOLUTE cho kiểu DYNAMIC.

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

9. TRUY XUẤT DỮ LIỆU TRÊN CURSOR

```
FETCH ( NEXT | PRIOR | FIRST | LAST
        | ABSOLUTE { n | @nvar } | RELATIVE { n | @nvar } )
    FROM ( GLOBAL ) cursor_name
    ( INTO @variable_name ( ,...n ) )
```

- **RELATIVE {n | @nvar}**: Nếu n hoặc @nvar > 0, chuyển xuống n dòng tính từ dòng kề dưới dòng hiện hành. Nếu n or @nvar < 0, Chuyển lên n dòng trước dòng hiện hành. Nếu n or @nvar = 0, trả về dòng hiện hành. o cursor_name: Tên cursor đang mở. Nếu tồn tại cursor cục bộ và cursor toàn cục có cùng tên thì tên cursor được sử dụng sẽ là cursor cục bộ nếu không có từ khóa GLOBAL.

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

9. TRUY XUẤT DỮ LIỆU TRÊN CURSOR

```
FETCH ( NEXT | PRIOR | FIRST | LAST
        | ABSOLUTE { n | @nvar } | RELATIVE { n | @nvar } )
FROM ( GLOBAL ) cursor_name
( INTO @variable_name ( ,...n ) )
```

- **INTO @varname(,...n):** Danh sách biến cục bộ nhận dữ liệu tương ứng từ các cột trên mẫu tin hiện hành, theo thứ tự từ trái sang phải. Số biến phải bằng số cột đã liệt kê trong câu lệnh Select khi tạo Cursor. Kiểu dữ liệu của mỗi biến phải tương thích với kiểu dữ liệu của cột hoặc được hổ trợ chuyển kiểu ngầm định theo kiểu của cột.

ĐỌC THÊM - KIỂU DỮ LIỆU CURSOR

9. TRUY XUẤT DỮ LIỆU TRÊN CURSOR

```
FETCH ( NEXT | PRIOR | FIRST | LAST
        | ABSOLUTE { n | @nvar } | RELATIVE { n | @nvar } )
    FROM ( GLOBAL ) cursor_name
    ( INTO @variable_name ( ,...n ) )
```

Kiểm tra kết quả của lệnh **FETCH:** Sử dụng hàm @@FETCH_STATUS sau lệnh FETCH. Hàm trả về một trong 3 giá trị:

0	Nếu lệnh FETCH chuyển đến 1 mẫu tin trong danh sách.
-1	Nếu lệnh FETCH chuyển đến vùng BOF hoặc EOF
-2	Nếu chuyển đến 1 dòng đã bị xóa trên Server (Keyset).

VÍ DỤ 1 - TRIGGER

Yêu cầu: (**Bài tập Quản lý Bán hàng – Phần 1 – Câu 11**) Cài đặt trigger kiểm tra Ngày mua hàng (NGHD) của một khách hàng thành viên sẽ lớn hơn hoặc bằng ngày khách hàng đó đăng ký thành viên (NGDK).

- **Tạo trigger trên bảng KHÁCH HÀNG để kiểm tra khi cập nhật ngày đăng ký.**
- **Tạo trigger trên bảng HÓA ĐƠN để kiểm tra khi cập nhật và thêm hóa đơn mới.**

VÍ DỤ 1 - TRIGGER

Tạo trigger trên bảng KHÁCH HÀNG để kiểm tra khi cập nhật ngày đăng ký.

```
create TRIGGER TRG_CHECK_NGHD_NGDK        -- CẬP NHẬT NGÀY ĐĂNG KÝ CỦA KHÁCH HÀNG
ON KHACHHANG FOR UPDATE
AS
BEGIN
        DECLARE @NGHD SMALLDATETIME
        DECLARE @NGDK SMALLDATETIME
        SELECT @NGDK=NGDK FROM inserted
        IF (@NGDK > ANY (SELECT HOADON.NGHD FROM HOADON, inserted WHERE HOADON.MAKH=inserted.MAKH))
        BEGIN
                PRINT N'NGÀY ĐĂNG KÝ PHẢI NHỎ HƠN NGÀY MUA HÀNG'
                ROLLBACK TRANSACTION
        END
END
```

VÍ DỤ 1 - TRIGGER

Tạo trigger trên bảng HÓA ĐƠN để kiểm tra khi cập nhật và thêm hóa đơn mới.

```
CREATE TRIGGER TRG_CHECK_NGHD
ON HOADON FOR INSERT, UPDATE
AS
BEGIN
        DECLARE @NGDK SMALLDATETIME
        DECLARE @NGHD SMALLDATETIME
        SELECT @NGDK=NGDK, @NGHD= NGHD FROM inserted,KHACHHANG
        WHERE KHACHHANG.MAKH=inserted.MAKH
        IF (@NGDK>@NGHD)
        BEGIN
                PRINT N'Ngày lập hóa đơn phải lớn hơn ngày đăng ký của khách hàng'
                ROLLBACK TRANSACTION
        END
        ELSE
                PRINT N'CẬP NHẬT/THÊM THÀNH CÔNG'
END
```

VÍ DỤ 2- TRIGGER

Yêu cầu: Cài đặt trigger thực hiện việc cập nhật tự động doanh số tổng của các hóa đơn khi có một chi tiết hóa đơn mới được thêm vào.

```
CREATE TRIGGER TANGDOANHSO
ON HOADON
FOR INSERT
AS
BEGIN
    DECLARE @TRIGIA2 int, @MAKH2 char(4)
    SELECT @TRIGIA2 = TRIGIA, @MAKH2=MAKH FROM inserted
    UPDATE KHACHHANG SET DOANHSO= DOANHSO + @TRIGIA2 WHERE MAKH=@MAKH2
END
```

BÀI TẬP

- **Sinh viên hoàn thành:**
  - Phần **I** bài tập **QuanLyBanHang** từ câu **11** đến câu **14.**
  - Phần **III** bài tập **QuanLyBanHang** từ câu **36** đến câu **46.**
  - Phần **I** bài tập **QuanLyGiaoVu** câu **9, 10** và từ câu **15** đến câu **24**.

HỎI - ĐÁP