



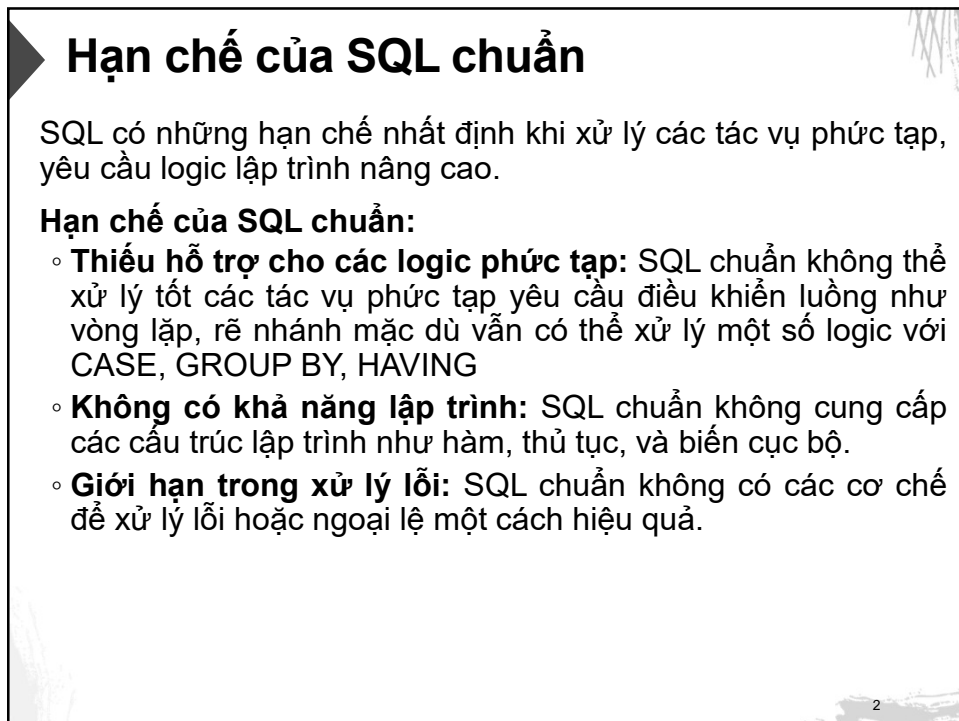
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA HỆ THỐNG THÔNG TIN

MÔN CƠ SỞ DỮ LIỆU - IT004

**CHƯƠNG 5 (tt):
TRIGGER**

ThS. TẠ VIỆT PHƯƠNG
phuongtv@uit.edu.vn

1



Hạn chế của SQL chuẩn

SQL có những hạn chế nhất định khi xử lý các tác vụ phức tạp, yêu cầu logic lập trình nâng cao.

Hạn chế của SQL chuẩn:

- **Thiếu hỗ trợ cho các logic phức tạp:** SQL chuẩn không thể xử lý tốt các tác vụ phức tạp yêu cầu điều khiển luồng như vòng lặp, rẽ nhánh mặc dù vẫn có thể xử lý một số logic với CASE, GROUP BY, HAVING
- **Không có khả năng lập trình:** SQL chuẩn không cung cấp các cấu trúc lập trình như hàm, thủ tục, và biến cục bộ.
- **Giới hạn trong xử lý lỗi:** SQL chuẩn không có các cơ chế để xử lý lỗi hoặc ngoại lệ một cách hiệu quả.

2

SQL mở rộng

Lý do cần mở rộng:

- **Tối ưu hóa hiệu suất:** Ngôn ngữ mở rộng cho phép xử lý các tác vụ phức tạp ngay trong cơ sở dữ liệu, giảm tải cho ứng dụng.
- **Hỗ trợ lập trình thủ tục** (Procedural Programming).
- **Tích hợp tốt hơn:** Hỗ trợ tích hợp với các ngôn ngữ lập trình khác như Java, C#, giúp dễ dàng phát triển các ứng dụng doanh nghiệp.
- **Bảo mật và quản lý:** Tăng cường khả năng kiểm soát truy cập và bảo mật dữ liệu.

3

3

SQL mở rộng

Để khắc phục những hạn chế của SQL chuẩn, các nhà cung cấp hệ quản trị cơ sở dữ liệu đã phát triển các **ngôn ngữ mở rộng của SQL**. Các ngôn ngữ này kết hợp SQL với các tính năng lập trình thủ tục, cho phép thực hiện các tác vụ phức tạp hơn như:

- **Kiểm soát luồng:** Sử dụng các cấu trúc điều khiển như IF-ELSE, WHILE, LOOP để thực hiện logic phức tạp.
- **Xử lý lỗi:** Sử dụng các khối TRY-CATCH hoặc EXCEPTION để xử lý lỗi phát sinh trong quá trình thực thi.
- **Thủ tục và hàm:** Tạo thủ tục (stored procedure) và hàm (function) để đóng gói và tái sử dụng code.
- **Trigger:** Tạo trigger để tự động thực thi các hành động khi có sự kiện xảy ra trên cơ sở dữ liệu.
- **Cursor:** Sử dụng cursor để xử lý dữ liệu từng dòng.

4

4

SQL mở rộng

Là **ngôn ngữ lập trình thủ tục** (procedural programming languages) được mở rộng từ SQL. Hỗ trợ các khái niệm của lập trình thủ tục như biến, vòng lặp, điều kiện, xử lý lỗi và khối mã logic. Là ngôn ngữ lập trình cấp cao dành riêng cho cơ sở dữ liệu. Không phải là ngôn ngữ lập trình đa mục đích như Python hay Java, mà chủ yếu được dùng để thao tác với cơ sở dữ liệu.

Một số ngôn ngữ mở rộng phổ biến:

T-SQL (Transact-SQL):

- Ngôn ngữ mở rộng của SQL được sử dụng trong Microsoft SQL Server.
- Cung cấp các tính năng như: cấu trúc điều khiển, xử lý lỗi, thủ tục, hàm, trigger, cursor.
- Hỗ trợ kiểu dữ liệu do người dùng định nghĩa (UDT).
- Tích hợp chặt chẽ với nền tảng .NET của Microsoft.

5

5

SQL mở rộng

◦ **PL/SQL (Procedural Language/Structured Query Language):**

- Ngôn ngữ mở rộng của SQL được sử dụng trong Oracle Database.
- Cung cấp các tính năng tương tự T-SQL, cùng với cấu trúc LOOP.
- Hỗ trợ kiểu dữ liệu thu thập (collection).
- Có thể được đóng gói thành các package để tăng khả năng tái sử dụng code.
- Tích hợp với các ngôn ngữ như Java, C++, Python.

Trong chương trình sẽ tập trung vào T-SQL (và có đề cập một chút về PL/SQL)

PL/pgSQL (Procedural Language for PostgreSQL): ngôn ngữ mở rộng của SQL được sử dụng trong PostgreSQL.

MySQL Stored Procedures: Ngôn ngữ mở rộng của SQL được sử dụng trong MySQL.

...

6

6

SQL mở rộng

Tiêu chí	SQL Server	Oracle
Nhà phát triển	Microsoft	Oracle Corporation
Kiểu dữ liệu	INT, VARCHAR, NVARCHAR, DATETIME, XML	NUMBER, VARCHAR2, CLOB, DATE, BLOB
Hệ thống lưu trữ	File .mdf/.ldf	Tablespace, Datafile
Cách quản lý giao tác	Mặc định COMMIT ngay	Giao tác cần COMMIT rõ ràng
Ngôn ngữ thủ tục	T-SQL	PL/SQL
Hỗ trợ JSON	Tích hợp sẵn	Dùng JSON_TABLE()
Cách xử lý batch	GO	/ (slash)
Cách xử lý lỗi	TRY...CATCH	EXCEPTION HANDLING

Trong bài này chủ yếu đề cập đến T-SQL phù hợp nội dung môn CSDL

7

7

Khối lệnh

Khối lệnh T-SQL (T-SQL Block) là một nhóm các câu lệnh T-SQL được đặt trong cặp từ khóa BEGIN và END. Khối lệnh cho phép nhóm các câu lệnh có liên quan về mặt logic để thực hiện một tác vụ cụ thể.

Đặc điểm:

- **Phân tách logic:** Giúp tổ chức và phân chia code thành các phần logic riêng biệt, giúp code dễ đọc và bảo trì hơn.
- **Phạm vi biến:** Biến được khai báo bên trong khối lệnh chỉ có hiệu lực trong phạm vi khối lệnh đó.
- **Xử lý luồng:** Cho phép sử dụng các cấu trúc điều khiển (IF-ELSE, WHILE, CASE) để điều khiển luồng thực thi các câu lệnh.
- **Xử lý lỗi:** Cho phép sử dụng khối TRY...CATCH để xử lý lỗi phát sinh trong khối lệnh.

8

8

Khởi lệnh

Ví dụ T-SQL:

```
DECLARE @student_name NVARCHAR(50);
BEGIN
    SET @student_name = 'Tiêu Bắc Thần';
    PRINT 'Tên sinh viên: ' + @student_name;
END;
```

9

9

Khởi lệnh

Ví dụ T-SQL: xử lý lỗi

```
BEGIN TRY
    DECLARE @num INT;
    SET @num = 10 / 0; -- Lỗi chia cho 0
    PRINT 'Kết quả: ' + CAST(@num AS NVARCHAR);
END TRY
BEGIN CATCH
    PRINT 'Lỗi xảy ra: ' + ERROR_MESSAGE();
END CATCH;
```

10

10

Khởi lệnh

Ví dụ:

```
DECLARE @SoLuongTon INT;  
SELECT @SoLuongTon = SoLuongTon FROM SanPham WHERE MaSP = 1;  
IF @SoLuongTon > 0  
BEGIN  
    UPDATE SanPham SET SoLuongTon = SoLuongTon - 1 WHERE MaSP = 1;  
    PRINT 'Đã bán 1 sản phẩm.';  
END  
ELSE  
BEGIN  
    PRINT 'Sản phẩm đã hết hàng.';  
END;
```

11

11

Biến

Biến (variable) là một đối tượng có thể lưu giữ một giá trị dữ liệu. Là một vùng nhớ dùng để lưu trữ dữ liệu tạm thời trong suốt quá trình thực thi chương trình.



12

12

Biến

Dữ liệu có thể được chuyển đến câu lệnh của SQL mở rộng bằng việc sử dụng biến.

Phạm vi của biến là cục bộ, chỉ có thể được sử dụng trong phạm vi của khối mà biến được khai báo.

Để gán giá trị cho biến ta có thể sử dụng các cách sau:

- Sử dụng toán tử gán
- Bằng cách selecting (hoặc fetching) dữ liệu.
- Bằng cách truyền cho nó như là một đối số OUT hoặc IN OUT trong chương trình con (subprogram), rồi sau đó gán giá trị trong subprogram

13

13

Biến

Trong T-SQL, biến được khai báo bằng DECLARE và gán giá trị bằng SET hoặc SELECT

Trong T-SQL biến cục bộ được tạo và được sử dụng cho việc lưu trữ tạm thời trong khi câu lệnh SQL được thực hiện. Tên của biến cục bộ phải bắt đầu với dấu '@'

Declare *tên_biến* Kiểu_dữ_liệu

tên_biến: xác định tên của biến, tên của biến phải bắt đầu với 1 dấu '@'.

Kiểu_dữ_liệu: là kiểu dữ liệu được định nghĩa bởi người sử dụng hoặc hệ thống.

Ví dụ:

```
Declare @MaSinhVien char(10)
```

```
Declare @HoTen nvarchar(30)
```

```
Declare @Sum float, @Count int
```

```
SELECT * FROM SinhVien WHERE HoTen = @HoTen;
```

14

14

Biến

Ví dụ T-SQL

```
DECLARE @student_name NVARCHAR(50);
DECLARE @student_age INT;
DECLARE @course_code NVARCHAR(10) = 'IS210';

SET @student_name = 'Nguyễn Thị Hòa';
SET @student_age = 20;

PRINT 'Tên sinh viên: ' + @student_name;
PRINT 'Tuổi: ' + CAST(@student_age AS NVARCHAR);
PRINT 'Mã khóa học: ' + @course_code;
```

15

15

Biến

Đưa kết quả truy vấn vào biến:

Ví dụ T-SQL:

```
SinhVien(MaSV: int; HoTen: nvarchar(30), Tuoi: int)
Select @Var1 = HoTen, @Var2 = Tuoi from SinhVien
where MaSV = 1
```

Lưu ý: nếu câu truy vấn trả về nhiều dòng, các biến chỉ nhận giá trị tương ứng của dòng đầu tiên

16

16

Hằng

Trong T-SQL, không có CONSTANT, chỉ có thể dùng DECLARE với SET và tránh thay đổi giá trị

Ví dụ T-SQL

```
DECLARE @pi FLOAT;  
SET @pi = 3.14159;  
PRINT 'Pi: ' + CAST(@pi AS NVARCHAR);
```

Biến @pi có thể bị thay đổi, không phải hằng số thực sự.

17

17

Kiểu dữ liệu

Kiểu dữ liệu là một thuộc tính định nghĩa loại dữ liệu mà một đối tượng (biến hoặc hằng) có thể chứa.

PL/SQL và T-SQL có các kiểu dữ liệu tương đồng nhưng cũng có khác biệt.

Sinh viên xem lại phần kiểu dữ liệu trong bài SQL

Ví dụ:

```
DECLARE @SoLuong INT;  
DECLARE @Gia DECIMAL(10,2);  
DECLARE @MoTa NVARCHAR(255);
```

18

18

Biến

Biến cục bộ:

- Trong T-SQL biến cục bộ được tạo và được sử dụng cho việc lưu trữ tạm thời trong khi câu lệnh SQL được thực hiện. Tên của biến cục bộ phải bắt đầu với dấu '@'.

Declare *tên_biến Kiểu_dữ_liệu*

tên_biến: xác định tên của biến, tên của biến phải bắt đầu với 1 dấu '@'.

Kiểu_dữ_liệu: là kiểu dữ liệu được định nghĩa bởi người sử dụng hoặc hệ thống.

Ví dụ:

```
Declare @MaSinhVien char(10)
```

```
Declare @HoTen nvarchar(30)
```

```
Declare @Sum float, @Count int
```

19

19

Biến

Câu lệnh SET hoặc SELECT được sử dụng để gán giá trị đến cho biến cục bộ

SET *tên_biến = giá_trị*
SET *tên_biến = tên_biến*
SET *tên_biến = biểu_thức*
SET *tên_biến = kết_quả_truy_vấn*

Ví dụ:

```
Declare @STT int
```

```
SET @STT = 1
```

```
SELECT HoTen FROM SinhVien
```

```
WHERE MaSinhVien=@STT
```

Ví dụ:

```
Set @SoSV = (select count (*) from SinhVien)
```

```
Set @MaLop = 'TH'+Year(@NgayTuyenSinh)
```

20

20

Biến

Đưa kết quả truy vấn vào biến:

Ví dụ: SinhVien(MaSV: int; HoTen: nvarchar(30), Tuoi int)

```
Select @Var1 = HoTen, @Var2 = Tuoi from SinhVien  
where MaSV = 1
```

Lưu ý: nếu câu truy vấn trả về nhiều dòng, các biến chỉ nhận giá trị tương ứng của dòng đầu tiên

21

21

Biến

Biến toàn cục: Biến toàn cục là biến được định nghĩa và xử lý bởi **hệ thống**. Biến toàn cục trong SQL Server được bắt đầu với 2 dấu '@'. Giá trị của các biến này có thể được truy lục bằng câu truy vấn SELECT đơn giản.

Ví dụ:

```
SELECT @@VERSION AS sqlServerVersionDetails
```

SQL tự cập nhật giá trị cho các biến này, user không thể gán giá trị trực tiếp

Bản chất là 1 hàm (function)

Một số biến thông dụng:

- @@error
- @@rowcount
- @@trancount
- @@fetch_status
- @@SERVERNAME: Trả về tên của server hiện tại.

22

22

► Biến

Gán giá trị: Sử dụng toán tử =

Ví dụ:

```
SET @HoTen = 'Nguyễn Văn A';
```

```
SET @Tuoi = 20;
```

Sử dụng biến trong câu lệnh SQL:

```
SELECT * FROM SinhVien WHERE HoTen = @HoTen;
```

23

23

► Stored procedure

Stored Procedure là đoạn chương trình kịch bản (programming scripts) với các câu lệnh SQL nhúng (embedded SQL) được lưu dưới dạng đã được biên dịch và thi hành trực tiếp bởi MS SQL server.

Stored Procedure: Đóng gói logic phức tạp

Còn gọi là: **Thủ tục “nội”, thủ tục nội tại, thủ tục thường trú**

Một stored procedure là một mã SQL đã được chuẩn bị sẵn mà có thể lưu, vì thế mã này có thể được sử dụng lại nhiều lần.

24

24

Stored procedure

Cú pháp:

```
CREATE PROCEDURE tên procedure  
{Parameter_name DataType [=default] [output] }  
AS  
Câu lệnh SQL  
GO;
```

Câu lệnh thực thi một Stored Procedure:

```
EXEC tên procedure;
```

Lưu ý:

- Tên tham số đặt theo quy tắc như tên biến cục bộ
- Chỉ có thể trả về giá trị kiểu int

25

25

Stored procedure

Ví dụ:

```
CREATE PROCEDURE sp_DanhSachNV  
AS  
BEGIN  
    SELECT HoTen, NTNS  
    FROM NhanVien  
    ORDER BY HoTen;  
END;
```

26

26

Stored procedure

Ví dụ:

```
CREATE PROCEDURE sp_DanhSachNV1 @max_luong INT
AS
    BEGIN
        SELECT HoTen, NTNS, Luong
        FROM NhanVien WHERE Luong < @max_luong
        ORDER BY HoTen;
    END;
```

27

27

Stored procedure

Ví dụ:

```
CREATE PROCEDURE usp_TinhTong
@So1 INT,          -- Tham số input
@So2 INT,          -- Tham số input
@Tong INT OUTPUT   -- Tham số output
AS
    BEGIN
        SET @Tong = @So1 + @So2;
    END;
```

Tham số input: Dùng để truyền dữ liệu vào stored procedure. Mặc định, mọi tham số được khai báo đều là tham số input.

Tham số output: Dùng để stored procedure trả về dữ liệu cho chương trình gọi nó. Cần thêm từ khóa OUTPUT sau kiểu dữ liệu.

28

28

▶ RBTV

Review Chương 4, phần 2.2

- ❖ **Ràng buộc toàn vẹn trong SQL được chia làm 2 loại chính:**
 - **Loại đơn giản:** Sử dụng **Constraint** để mô tả
 - **Loại phức tạp:** Sử dụng **Trigger** để thực hiện
- ❖ Có thể khai báo RBTV ở mức cột hoặc mức bảng

29

29

▶ RBTV

Trong SQL, có ba **cơ chế (mechanism)** duy trì toàn vẹn dữ liệu trong DBMS:

- **Constraint** – các ràng buộc logic tĩnh (Domain, Key, Entity, Referential)
- **Assertion** – Ràng buộc toàn cục, cho phép kiểm tra điều kiện phức tạp trên nhiều bảng (một dạng constraint đặc biệt, hiện ít được DBMS hỗ trợ, **không hỗ trợ trên SQL Server**)
- **Trigger** – thủ tục tự động thực thi khi có sự kiện (INSERT, UPDATE, DELETE), nhằm duy trì hoặc khôi phục toàn vẹn dữ liệu, chỉ dùng khi logic phức tạp, có aggregate, cross-table nặng

30

30

Constraint

Đã được giới thiệu trong chương 4 - SQL, phần DDL

Bổ sung: Dùng từ khóa **WITH CHECK (Mặc định)** hoặc **WITH NOCHECK** để tùy chỉnh cho Constraint áp dụng với dữ liệu cũ hay không

```
ALTER TABLE EMPLOYEE  
WITH NOCHECK
```

```
ADD CONSTRAINT CK_Salary_Positive CHECK (Salary >= 0);
```

Thêm ràng buộc mà không kiểm tra dữ liệu cũ, chỉ ngăn dữ liệu xấu mới phát sinh.

31

31

Constraint

```
ALTER TABLE EMPLOYEE  
NOCHECK CONSTRAINT CK_Salary_Positive;
```

```
ALTER TABLE EMPLOYEE  
WITH CHECK CHECK CONSTRAINT CK_Salary_Positive;
```

Tạm thời bật hoặc tắt một ràng buộc đã tồn tại.

32

32

Trigger

Trigger: Tự động hóa tác vụ

Khái niệm: Trigger là một loại thủ tục (stored procedure) đặc biệt, được lưu trữ trên server và tự động thực thi khi có sự kiện DML (INSERT, UPDATE, DELETE) xảy ra trên bảng dữ liệu.

Ứng dụng:

- Kiểm tra tính hợp lệ của dữ liệu trước khi cho phép thay đổi.
- Duy trì tính nhất quán của dữ liệu giữa các bảng.
- Ghi log các thay đổi trên dữ liệu để phục vụ cho audit, phân tích,...
- Thực hiện các quy tắc nghiệp vụ (business rules) nào đó

33

33

Trigger

Phân loại Trigger

Nội dung:

- DML trigger: Kích hoạt bởi các sự kiện DML (INSERT, UPDATE, DELETE).
 - After trigger: Thực thi sau khi thao tác DML hoàn thành. Có thể quay lui thao tác đã thực hiện bằng lệnh rollback transaction. (Trong PL/SQL còn có Before Trigger)
 - Instead of trigger: Thay thế cho thao tác DML, thường được dùng để xử lý cập nhật trên view
- DDL trigger: Kích hoạt bởi các sự kiện DDL (CREATE, ALTER, DROP).
- Logon trigger: Kích hoạt khi người dùng đăng nhập vào SQL Server hoặc các sự kiện đặc biệt (LOGON, LOGOFF, STARTUP, SHUTDOWN)

34

34

Trigger

Row-Level và Statement-Level Trigger

Trigger có thể hoạt động ở **cấp dòng (Row-Level)** hoặc **cấp câu lệnh (Statement-Level)**, tùy vào cách nó được khai báo và sử dụng

Row-Level Trigger: được kích hoạt một lần cho mỗi dòng dữ liệu bị ảnh hưởng bởi câu lệnh DML (INSERT, UPDATE, DELETE). Trong PL/SQL (Oracle), cần sử dụng FOR EACH ROW để tạo Row-Level Trigger. Trong T-SQL (SQL Server), KHÔNG hỗ trợ Row-Level Trigger, nhưng có thể xử lý từng dòng bằng cách duyệt bảng INSERTED hoặc DELETED.

Statement-Level Trigger được kích hoạt một lần duy nhất cho toàn bộ câu lệnh DML (INSERT, UPDATE, DELETE), bất kể bao nhiêu dòng bị ảnh hưởng. PL/SQL mặc định là Statement-Level Trigger nếu không có FOR EACH ROW. T-SQL luôn sử dụng Statement-Level Trigger.

35

35

Trigger

Trigger và Constraint, **khi thêm một ràng buộc mới vào bảng đã có dữ liệu:**

- Trigger chỉ áp dụng cho dữ liệu mới. Sau khi được tạo, trigger sẽ chỉ được kích hoạt bởi các thao tác DML xảy ra **sau đó**. Nó không tự động kiểm tra lại dữ liệu cũ đã có trong bảng.
- Constraint có thể áp dụng cho dữ liệu cũ và dữ liệu mới, mặc định sẽ kiểm tra **cả dữ liệu cũ** (đã tồn tại). Ta có thể dùng tùy chọn WITH NOCHECK để nó chỉ áp dụng cho dữ liệu mới được thêm/sửa sau này.
- Trigger có thể tuân theo các quy tắc phức tạp mà Constraint không thể.

36

36

Trigger

Khi trigger được thực thi, SQL tự động tạo ra 2 bảng tạm với cùng cấu trúc với bảng mà trigger được định nghĩa trên đó.

Bảng INSERTED chứa dữ liệu mới khi thực thi câu lệnh Insert hoặc câu lệnh Update.

Bảng DELETED chứa dữ liệu bị xóa khi thực thi câu lệnh Delete hoặc chứa dữ liệu cũ (old) khi thực thi câu lệnh Update.

Hai bảng này chỉ tồn tại trong thời gian trigger xử lý và cục bộ cho mỗi trigger.

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

37

37

Trigger

Đối với thao tác insert:

Insert into HOADON values (1004, '01/09/2006', 'KH02', 180000)

HOADON			
SOHD	NGHD	MAKH	TRIGIA
1001	23/07/2006	KH01	320,000
1002	12/08/2006	KH01	840,000
1003	23/08/2006	KH02	100,000
1004	01/09/2006	KH02	180,000

INSERTED			
SOHD	NGHD	MAKH	TRIGIA
1004	01/09/2006	KH02	180,000

DELETED			
SOHD	NGHD	MAKH	TRIGIA

1004	01/09/2006	KH02	180,000
------	------------	------	---------

38

38

Trigger

Đối với thao tác delete

Delete from HOADON where sohd=1004

HOADON			
SOHD	NGHD	MAKH	TRIGIA
1001	23/07/2006	KH01	320,000
1002	12/08/2006	KH01	840,000
1003	23/08/2006	KH02	100,000
1004	01/09/2006	KH02	180,000

INSERTED			
SOHD	NGHD	MAKH	TRIGIA

DELETED			
SOHD	NGHD	MAKH	TRIGIA
1004	01/09/2006	KH02	180,000

1004	01/09/2006	KH02	180,000
------	------------	------	---------

39

39

Trigger

Đối với thao tác update:

Update HOADON set makh='kh07', trigia=300000 Where sohd=1004

HOADON			
SOHD	NGHD	MAKH	TRIGIA
1001	23/07/2006	KH01	320,000
1002	12/08/2006	KH01	840,000
1003	23/08/2006	KH02	100,000
1004	01/09/2006	KH07	300,000

INSERTED			
SOHD	NGHD	MAKH	TRIGIA
1004	01/09/2006	KH07	300,000

DELETED			
SOHD	NGHD	MAKH	TRIGIA
1004	01/09/2006	KH02	180,000

40

40

Trigger

INSERTED và DELETED là các **bảng tạm có thể chứa 0, 1 hoặc n dòng**.

Trigger trong SQL Server luôn xử lý tập dữ liệu (set-based), không chạy lặp từng dòng

Câu lệnh	Số dòng	inserted	deleted
INSERT 1 dòng	1	1 dòng	0
INSERT nhiều dòng	N	N dòng	0
UPDATE nhiều dòng	N	N dòng mới	N dòng cũ
DELETE nhiều dòng	N	0	N dòng

41

41

Trigger

Tạo trigger: cú pháp

```
CREATE [OR ALTER] TRIGGER Tên_Trigger
ON Tên_Table
AFTER (FOR) | INSTEAD OF INSERT, DELETE, UPDATE
AS
    Các_lệnh_của_Trigger
```

AFTER (FOR) | INSTEAD OF: Xác định thời điểm trigger được kích hoạt.

Xóa trigger: cú pháp

```
DROP TRIGGER Tên_Trigger
```

Trường hợp chỉ muốn ngưng kiểm tra trigger thì ta sử dụng lệnh sau:

```
ALTER TABLE Tên_bảng
DISABLE/ENABLE TRIGGER Tên_trigger (/ALL)
```

42

42

Trigger

Có 2 loại triggers: INSTEAD OF và AFTER (FOR).

INSTEAD OF:

- Trigger được gọi thực hiện **thay cho** thao tác delete/insert/update tương ứng.
- Trigger instead of thường được dùng để xử lý cập nhật trên view.

AFTER (FOR):

- Trigger được gọi thực hiện sau khi thao tác delete/ insert/ update tương ứng đã được thực hiện thành công.
- Có thể quay lui thao tác đã thực hiện bằng lệnh rollback transaction.
- Trong SQL Server, AFTER và FOR là tương đương. Nên dùng AFTER cho rõ nghĩa

43

43

Trigger

Ví dụ **Instead Of Trigger**

-- Tạo view chỉ hiển thị sản phẩm có giá lớn hơn 100

```
CREATE VIEW v_SanPhamGiaCao
```

```
AS
```

```
SELECT * FROM SanPham WHERE Gia > 100;
```

-- Tạo INSTEAD OF TRIGGER cho view

```
CREATE TRIGGER trg_v_SanPhamGiaCao_Insert
```

```
ON v_SanPhamGiaCao
```

```
INSTEAD OF INSERT
```

```
AS
```

```
BEGIN
```

44

44

Trigger

```
-- Kiểm tra giá sản phẩm trong bảng ảo inserted
IF EXISTS (SELECT 1 FROM inserted WHERE Gia <= 100)
BEGIN
    RAISERROR('Giá sản phẩm phải lớn hơn 100!', 16, 1);
    ROLLBACK TRANSACTION;
END
ELSE
BEGIN
    -- Thêm sản phẩm vào bảng SanPham
    INSERT INTO SanPham (MaSP, TenSP, Gia)
    SELECT MaSP, TenSP, Gia FROM inserted;
END
END;
```

45

45

Trigger

Trigger trg_v_SanPhamGiaCao_Insert kích hoạt thay thế cho thao tác INSERT trên view v_SanPhamGiaCao.

Trigger kiểm tra giá sản phẩm, nếu giá nhỏ hơn hoặc bằng 100, trigger sẽ báo lỗi và rollback.

Nếu giá lớn hơn 100, trigger sẽ chèn dữ liệu vào bảng gốc SanPham.

46

46

Trigger

Ví dụ **DDL TRIGGER**:

```
-- Tạo DDL TRIGGER ghi log khi có bảng được tạo
CREATE TRIGGER trg_DDL_CreateTable
ON DATABASE
FOR CREATE_TABLE
AS
BEGIN
    -- Ghi thông tin bảng được tạo vào bảng log
    INSERT INTO DDL_Log (Event, ObjectName, EventDate)
    VALUES ('CREATE_TABLE',
    EVENTDATA().value('/EVENT_INSTANCE/ObjectName)[1]',
    'VARCHAR(100)'), GETDATE());
END;
```

47

47

Trigger

Trigger trg_DDL_CreateTable kích hoạt khi có bảng được tạo trong CSDL.

Trigger ghi thông tin về sự kiện CREATE_TABLE, tên bảng, và thời gian vào bảng DDL_Log.

EVENTDATA() là một hàm hệ thống cung cấp thông tin về sự kiện DDL.

48

48

Trigger

KHACHHANG (MAKH, HOTEN, NGSINH, NGDK)

HOADON (SOHD, NGHĐ, MAKH, TRIGIA)

Ngày mua hàng (NGHĐ) của một khách hàng thành viên sẽ lớn hơn hoặc bằng ngày khách hàng đó đăng ký thành viên (NGDK).

	THÊM	XÓA	SỬA
KHACHHANG	-	-	+ (NGDK)
HOADON	+	-	+ (NGHĐ, MAKH)

49

49

Trigger

Sinh viên tìm hiểu các cấu trúc sau:

- + Khai báo biến: DECLARE
- + Xuất thông tin: PRINT, RAISERROR, THROW
- + Cấu trúc điều kiện: IF
- + Cấu trúc lặp: WHILE
- + Con trỏ: CURSOR
- + Hủy cập nhật dữ liệu vào hệ thống bộ nhớ: ROLLBACK TRANSACTION

50

50

Trigger

```
CREATE TRIGGER nghd_hoadon_insert
ON hoadon
AFTER INSERT
AS
    DECLARE @ng_muahang smalldatetime
    DECLARE @ng_dangky smalldatetime
    SELECT @ng_muahang=nghd, @ng_dangky=ngdk
    FROM khachhang, inserted
    WHERE khachhang.makh=inserted.makh
    IF @ng_muahang< @ng_dangky
    BEGIN
        rollback transaction
        print 'ngay mua hang phai lon hon ngay dang ky'
    END
```

51

51

Trigger

HOẶC:

```
CREATE TRIGGER nghd_hoadon_insert
ON hoadon
AFTER INSERT
AS
    IF (EXISTS (SELECT *
                FROM inserted i JOIN khachhang kh ON i.makh=kh.makh
                WHERE i.nghd<kh.ngdk))
    BEGIN
        rollback transaction
        print 'ngay mua hang phai lon hon ngay dang ky'
    END
```

hoặc

```
IF (SELECT i.NGHD FROM inserted i) < (SELECT kh.NGDK FROM
KHACHHANG kh JOIN inserted i ON kh.MaKH = i.MaKH)
BEGIN -- Logic xử lý lỗi END
```

52

52

Trigger

```
CREATE TRIGGER nghd_hoadon_update
ON hoadon
AFTER UPDATE
AS
IF (UPDATE (makh) OR UPDATE (nghd))
BEGIN
    DECLARE @ng_muahang smalldatetime
    DECLARE @ng_dangky smalldatetime
    SELECT @ng_muahang=nghd, @ng_dangky=ngdk
    FROM khachhang, inserted
    WHERE khachhang.makh=inserted.makh
    IF @ng_muahang< @ng_dangky
    BEGIN
        rollback transaction
        print 'ngay mua hang phai lon hon ngay dang ky'
    END
END
```



53

53

Trigger

HOẶC:

```
CREATE TRIGGER nghd_hoadon_update
ON hoadon
AFTER UPDATE
AS
IF (UPDATE (makh) OR UPDATE (nghd))
BEGIN
    IF (EXISTS (SELECT *
                FROM inserted i JOIN khachhang kh ON i.makh=kh.makh
                WHERE i.nghd<kh.ngdk))
    BEGIN
        rollback transaction
        RAISERROR('Ngày mua hàng phải lớn hơn hoặc bằng ngày
        đăng ký!', 16, 1);
    END
END
```

54

54

Trigger

```
CREATE TRIGGER nghd_khachhang_update
ON KHACHHANG
AFTER UPDATE
AS
    DECLARE @ng_dangky smalldatetime, @makhhang char(4)
    SELECT @ng_dangky=ngdk, @makhhang=makh
    FROM inserted
    IF (UPDATE (ngdk))
    BEGIN
        IF (EXISTS (SELECT *
                    FROM hoadon
                    WHERE makh=@makhhang AND nghd<@ng_dangky))
        BEGIN
            rollback transaction
        END
    END
END
```

55

55

Trigger

HOẶC:

```
CREATE TRIGGER nghd_khachhang_update
ON KHACHHANG
AFTER UPDATE
AS
    IF (UPDATE (ngdk))
    BEGIN
        IF (EXISTS (SELECT *
                    FROM hoadon hd JOIN inserted i ON hd.makh=i.makh
                    WHERE hd.nghd<i.ngdk))
        BEGIN
            rollback transaction
        END
    END
END;
```

56

56

Trigger

Biến đơn và set-based.

- Trường hợp dùng biến đơn: chỉ đúng nếu INSERT 1 dòng
- Trong thực tế, phải dùng cách (EXISTS + JOIN inserted) để xử lý đúng với nhiều bản ghi.

57

57

Trigger

UPDATE(Column) chỉ xem xét **cột đó có xuất hiện trong câu lệnh SET hay không**

Một số ví dụ về việc không thật sự update nhưng vẫn có thể kích hoạt trigger

```
UPDATE KHACHHANG  
SET TenKH = TenKH  
WHERE MaKH = 'KH01';
```

```
UPDATE KHACHHANG SET makh = makh, nghd = nghd * 1
```

```
UPDATE KHACHHANG SET makh = makh
```

Trong khi về bản chất dữ liệu thật ra không đổi gì cả.

58

58

Trigger

Để kiểm tra thay đổi thực sự: có thể thêm so sánh trước

deleted = giá trị trước khi UPDATE

inserted = giá trị sau khi UPDATE

```
IF EXISTS (
    SELECT 1
    FROM inserted i
    JOIN deleted d ON i.MaHD = d.MaHD
    WHERE i.NGHD <> d.NGHD
)
BEGIN
    PRINT 'Ngày hóa đơn thực sự thay đổi';
END
```

59

59

Trigger

```
CREATE OR ALTER TRIGGER trg_Check_NGHD
ON HOADON
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN deleted d ON i.MaHD = d.MaHD
        WHERE i.NGHD <> d.NGHD    -- giá trị thực sự thay đổi
    )
    BEGIN
        IF EXISTS (
            SELECT 1
            FROM inserted i
            JOIN KHACHHANG kh ON i.MaKH = kh.MaKH
            WHERE i.NGHD < kh.NGDK
        )
        BEGIN
            THROW 50001, N'Ngày hóa đơn không hợp lệ', 1;
        END
    END
END
```

60

60

Trigger

Trường hợp thì chỉ muốn ngưng kiểm tra trigger thì ta sử dụng lệnh sau:

```
ALTER TABLE Tên_bảng  
DISABLE/ENABLE TRIGGER Tên_trigger (/ALL)
```

Nếu muốn tắt thông báo số dòng bị ảnh hưởng bởi Trigger (Không trả về thông báo rác), đặt dòng lệnh SET NOCOUNT ON ngay sau BEGIN đầu tiên.

```
SET NOCOUNT ON;
```

61

61

Trigger

Sử dụng quá mức trigger có thể ảnh hưởng đến hiệu suất hệ thống. **Ví dụ:** Nếu một bảng có trigger thực hiện nhiều kiểm tra phức tạp sau mỗi lần **INSERT**, việc thêm dữ liệu vào bảng này sẽ chậm hơn so với bảng không có trigger.

Không nên sử dụng trigger cho các thao tác xảy ra thường xuyên với dữ liệu lớn vì nó sẽ làm giảm hiệu suất tổng thể. Khuyến khích chỉ sử dụng trigger khi cần thiết. Các giải pháp thay thế có thể là kiểm tra tính toàn vẹn của dữ liệu bằng CHECK Constraints hoặc Stored Procedures.

Lưu ý: Nguy cơ ĐỆ QUY TRIGGER và nguy cơ DEADLOCK từ Trigger. Không để trigger bảng A cập nhật bảng B và trigger B cập nhật ngược lại bảng A. Tránh viết trigger UPDATE chung (không chỉ định thuộc tính cần update) và trong thân Trigger có SQL UPDATE.

62

62

Lược đồ CSDL quản lý giáo vụ

HOCVIEN (MAHV, HO, TEN, NGSINH, GIOITINH, NOISINH, CMND, MALOP)

LOP (MALOP, TENLOP, TRGLOP, SISO, MAGVCN)

KHOA (MAKHOA, TENKHOA, NGTLAP, TRGKHOA)

MONHOC (MAMH, TENMH, TCLT, TCTH, MAKHOA)

DIEUKIEN (MAMH, MAMH_TRUOC)

GIAOVIEN (MAGV, HOTEN, HOCVI, HOCHAM, GIOITINH, NGSINH, NGVL, HESO, MUCLUONG, MAKHOA)

GIANGDAY (MALOP, MAMH, MAGV, HOCKY, NAM, TUNGAY, DENNGAY)

KETQUATHI (MAHV, MAMH, LANTHI, NGTHI, DIEM, KQUA)

63

63

Bài tập

1. Trưởng lớp của một lớp phải là học viên của lớp đó.
2. Học viên chỉ được thi lại một môn (lần thi >1) khi điểm của lần thi trước đó dưới 5.
3. Ngày thi của lần thi sau phải lớn hơn ngày thi của lần thi trước (cùng học viên, cùng môn học).
4. Mỗi năm học chỉ được giảng dạy tối đa 10 lớp
5. Mỗi học kỳ của một năm học, một lớp chỉ được dạy tối đa 3 môn.

64

64

Trigger

Row-Level Trigger: Mô phỏng trong T-SQL

```
CREATE TRIGGER trg_after_update_students
ON students
AFTER UPDATE
AS
BEGIN
    -- Duyệt từng dòng trong INSERTED và DELETED
    INSERT INTO students_log (student_id, old_name, new_name, changed_at)
    SELECT d.student_id, d.name, i.name, GETDATE()
    FROM deleted d
    JOIN inserted i ON d.student_id = i.student_id;
END;
```

65

65

Trigger

Khi nào dùng?

Row-Level Trigger:

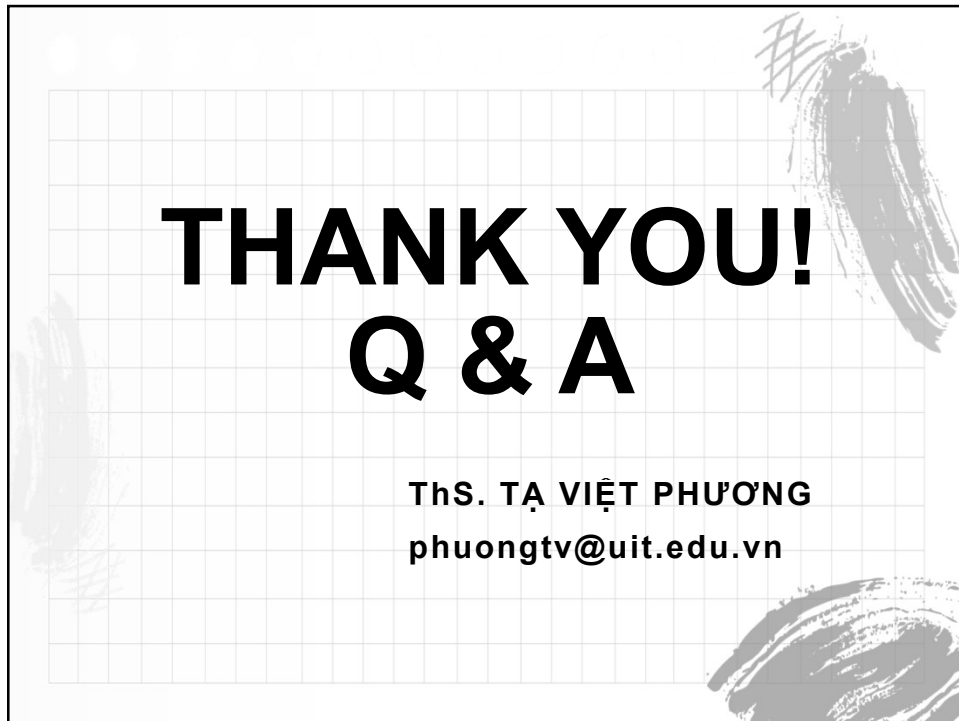
- Kiểm tra hoặc thay đổi giá trị trước khi INSERT: Dùng BEFORE INSERT
- Ghi log từng thay đổi cụ thể trên dòng dữ liệu: Dùng AFTER UPDATE
- Tạo ràng buộc nghiệp vụ trên từng dòng dữ liệu: Dùng BEFORE Trigger

◦ Statement-Level Trigger:

- Dùng AFTER INSERT/UPDATE/DELETE

66

66



67