NGÔN NGỮ TRUY VẤN DỮ LIỆU

Giảng viên: ThS. Trần Ngọc Thăng

BÀI 6
BẪY LỖI VÀ GIAO TÁC

Giảng viên: ThS. Trần Ngọc Thăng

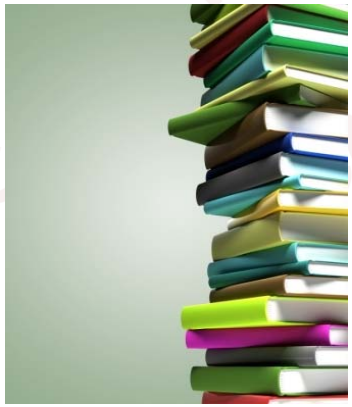
MỤC TIÊU BÀI HỌC

- Trình bày được khái niệm, đặc điểm và phân loại Trigger;
- Tạo mới, quản lý và sử dụng Trigger;
- Trình bày được khái niệm, đặc điểm và phân loại Transaction;
- Tạo mới, quản lý và sử dụng Transaction.

CÁC KIẾN THỨC CẦN CÓ

Để hiểu rõ bài này, yêu cầu học viên cần có các kiến thức cơ bản liên quan đến các môn học sau:

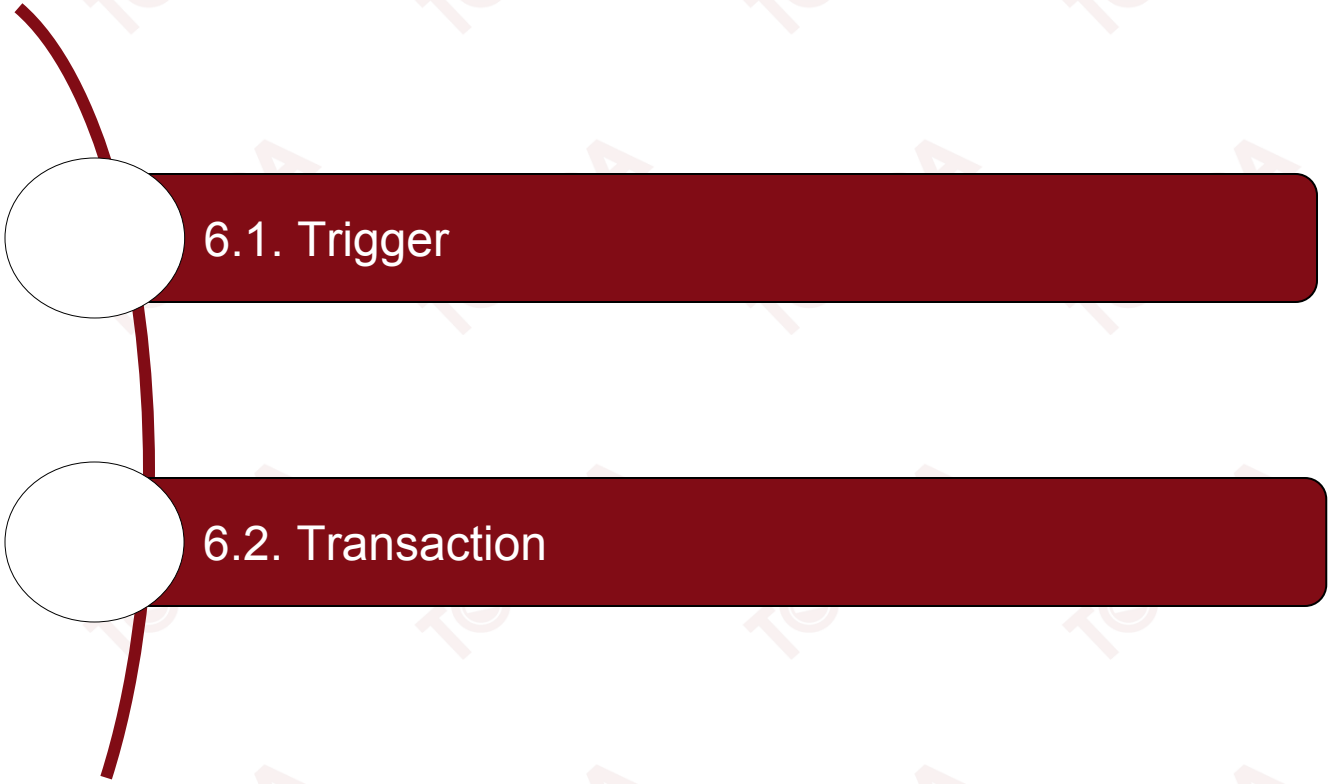
- Tin học cơ bản;
- Nhập môn lập trình;
- Cơ sở dữ liệu;
- Hệ quản trị cơ sở dữ liệu SQL Server.



HƯỚNG DẪN HỌC

- Đọc tài liệu và tóm tắt những nội dung chính của từng bài;
- Luôn liên hệ và lấy ví dụ thực tế khi học đến từng vấn đề và khái niệm;
- Thực hiện các thao tác trên hệ quản trị cơ sở dữ liệu SQL Server;
- Làm bài tập và luyện thi trắc nghiệm theo yêu cầu từng bài.

CẤU TRÚC NỘI DUNG



6.1. TRIGGER

6.1.1. Giới thiệu về Trigger

6.1.2. Tạo và quản lý Trigger

6.1.3. Các tình huống sử dụng trigger

6.1.1 GIỚI THIỆU VỀ TRIGGER

- **Trigger là gì?**
  - Trigger là một loại stored procedure đặc biệt.
  - Tự động thực thi khi có một câu lệnh Update, Insert, hoặc Delete được thực hiện trên bảng hoặc bảng ảo View.
  - Không thể gọi một trigger thi hành một cách trực tiếp.
  - Tương tự như việc cài đặt sự kiện trong lập trình sự kiện với ngôn ngữ C#.
- **Trigger dùng để làm gì?**
  - Tạo ràng buộc toàn vẹn dữ liệu cho phù hợp với mô hình quan hệ CSDL.
  - Kiểm soát dữ liệu hiện tại khi có thay đổi đến giá trị trong mẫu tin trong bảng.
  - Kiểm tra dữ liệu nhập vào phù hợp với mối quan hệ dữ liệu giữa các bảng.
  - Định nghĩa thông báo lỗi của người dùng.
  - So sánh trạng thái của dữ liệu trước và sau hiệu chỉnh.
- Ví dụ: Cài đặt mã tự tăng theo định dạng.

6.1.1 GIỚI THIỆU VỀ TRIGGER (tiếp theo)

- **Các dạng ràng buộc toàn vẹn (RBTV) nên dùng trigger**
  - Ngoại trừ các RB khóa chính, khóa ngoại, miền giá trị (check, rule), default, unique. Còn lại các RB khác chúng ta có thể dùng trigger.
  - Chúng ta có thể thấy rõ nhất việc ứng dụng trigger trong RB liên thuộc tính - liên bộ - liên quan hệ.
  - Ngoài ra đôi khi chính những RB đơn giản như rule hay khóa ngoại, … Ta vẫn có thể dùng trigger vì tính "Thân thiện hóa" câu báo lỗi trong RB.
- **Các đặc điểm của trigger:**
  - Dùng trigger cho việc cài đặt những RBTV phức tạp.
  - Có thể dùng trigger trên view.
  - Một trigger có thể có nhiều hành động.
  - Trigger làm tăng khả năng RB và ẩn dấu nhiều giao tác.

6.1.1 GIỚI THIỆU VỀ TRIGGER (tiếp theo)

- **Các hạn chế:**
  - Trigger không được tạo trên bảng tạm #<tên bảng>.
  - Trigger Instead of Delete và Update không thể tạo trên bảng chứa khóa ngoại.
- **Cơ chế hoạt động của trigger:**
  - Trigger tương ứng với hành động Insert, phát sinh bảng Inserted.
  - Trigger tương ứng với hành động Delete, phát sinh bảng Deleted.
  - Trigger tương ứng với hành động Update, phát sinh bảng Inserted và Deleted.
- **Chú ý:**
  - Bảng Inserted và Deleted có cấu trúc giống như bảng gắn trigger.
  - Bảng Inserted chứa bộ thêm vào, bảng Deleted chứa bộ hủy.
  - Đối với lệnh Update: Inserted chứa bộ mới, Deleted chứa bộ cũ.

6.1.1 GIỚI THIỆU VỀ TRIGGER (tiếp theo)

- **Chú ý:**
  - Trigger loại Instead of sẽ bỏ qua hành động kích hoạt lệnh Insert, Update, Delete thay vào đó là thực hiện các lệnh bên trong trigger. Nhưng dữ liệu trong các bảng Inserted, Deleted vẫn có.
  - Trigger dạng Instead of có thể định nghĩa trên table hay view.
  - Trigger loại After (mặc định), nó chỉ được thi hành sau khi hành động trigger kích hoạt, rồi tới các RB toàn vẹn loại mô tả (Là những RB khóa chính, khóa ngoại, miền giá trị, default, Unique) rồi mới tới lệnh trong trigger.

6.1.2 TẠO VÀ QUẢN LÝ TRIGGER

Một số gợi ý trước khi tạo trigger:

- Xác định tên của trigger.
- Chỉ định table hoặc view gắn trigger.
- Chỉ định trigger loại Instead of hoặc After.
- Chỉ định biến cố Insert, Update, Delete.
- Các câu lệnh tương ứng với nhiệm vụ mà trigger sẽ thực hiện.
- Các tùy chọn nếu cần:
  - Replication: Nhân bản;
  - Encrytion: Mã hóa. Câu lệnh CREATE TRIGGER được sử dụng để định nghĩa trigger và có cú pháp:

```
CREATE TRIGGER tên_trigger ON tên_bảng
FOR {[INSERT][,][UPDATE][,][DELETE]}
 AS
  [IF UPDATE(tên_cột)
   [AND UPDATE(tên_cột)|OR UPDATE(tên_cột)...]
  Các_câu_lệnh_của_trigger
```

6.1.2 TẠO VÀ QUẢN LÝ TRIGGER (tiếp theo)

- Ví dụ: Tạo trigger trg_nhatkybanhang_insert có chức năng tự động giảm số lượng hàng hiện có khi một mặt hàng nào đó được bán.

```
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
```

- Khi thực hiện câu lệnh INSERT trên bảng NHATKYBANHANG, dữ liệu trong bảng MATHANG sẽ thay đổi như sau:

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

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

6.1.2 TẠO VÀ QUẢN LÝ TRIGGER (tiếp theo)

Ví dụ: Tạo một trigger cho thao tác Insert, để kiểm tra ngày lập hoá đơn thì luôn luôn lớn hơn ngày giao với bảng Hóa Đơn có cấu trúc như sau:

Hoadon (MaHD, NgayLapHD, NgayGiao).

```
CREATE TRIGGER Trg_NgayLap_NgayGiaoHD
ON Hoadon AFTER INSERT
AS
DECLARE @NgayLapHD DateTime, @NgayGiao DateTime
SELECT @NgayLapHD=hd.NGayLapHD,NgayGiao=hd.NgayGiaoNhan
FROM HoaDon hd INNER JOIN Inserted i ON hd.MaHD=i.Mahd
If @NgayGiao<@NgayLapHD
      BEGIN
        RAISERROR(500103,10,1)
        ROLLBACK TRANSACTION
      END
```

6.1.2 TẠO VÀ QUẢN LÝ TRIGGER (tiếp theo)

- Đổi tên trigger:

```
EXEC sp_rename [@objname=]'<Tên đối tượng>'
                    [@newname=]'<Tên mới>'
                    [,[@objtype=]'<loại đối tượng>']
```

```
EXEC sp_rename      'tr_cau1_baitap1',
                    'tr_themsua_DC_HB_on_NV','Object'
```

- Thay đổi nội dung câu lệnh tạo trigger:

```
ALTER TRIGGER <tên trigger>
ON <tên table>|<tên view>
FOR | INSTEAD OF <biến cố kích hoạt trigger>
AS
      <Câu lệnh thực thi trigger>
```

6.1.2 TẠO VÀ QUẢN LÝ TRIGGER (tiếp theo)

- Xem thông tin các trigger:

```
--Hiển thị các đặc tính của trigger trên một bảng
Exec sp_helptrigger <tên bảng>
--Xem mã lệnh của một trigger
Exec sp_helptext <tên trigger>
```

- Xóa trigger:

```
DROP TRIGGER <tên trigger>
```

- Vô hiệu hóa trigger:

```
ALTER TABLE <tên bảng> DISABLE TRIGGER <tên trigger>|ALL
```

- Làm cho trigger có hiệu lực trở lại:

```
ALTER TABLE <tên bảng> ENABLE TRIGGER <tên trigger>|ALL
```

6.1.3 CÁC TÌNH HUỐNG SỬ DỤNG TRIGGER

Sử dụng mệnh đề IF UPDATE trong trigger

- Thay vì chỉ định một trigger được kích hoạt trên toàn bảng, ta có thể chỉ định trigger được kích hoạt và thực hiện những thao tác cụ thể khi việc thay đổi dữ liệu chỉ liên quan đến một số cột nhất định nào đó của cột.
- IF UPDATE không sử dụng được đối với câu lệnh DELETE.
- Ví dụ: Xét hai bảng MATHANG và NHATKYBANHANG, tạo trigger được kích hoạt khi ta tiến hành cập nhật cột SOLUONG cho một bản ghi của bảng NHATKYBANHANG.

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

6.1.3 CÁC TÌNH HUỐNG SỬ DỤNG TRIGGER (tiếp theo)

```
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
```

6.1.3 CÁC TÌNH HUỐNG SỬ DỤNG TRIGGER (tiếp theo)

- Với trigger *trg_nhatkybanhang_update_soluong* câu lệnh

```
UPDATE nhatkybanhang SET soluong=soluong+20 WHERE stt=1
```

sẽ được kích hoạt, còn câu lệnh

```
UPDATE nhatkybanhang SET nguoimua='Mai Hữu Toàn' WHERE stt=3
```

không được kích hoạt.

- 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.

```
CREATE TRIGGER trg_R_test
ON R
FOR UPDATE
AS
 IF UPDATE(A)
  Print 'A updated'
 IF UPDATE(C)
  Print 'C updated'
```

6.1.3 CÁC TÌNH HUỐNG SỬ DỤNG TRIGGER (tiếp theo)

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.
- Để huỷ bỏ tác dụng của câu lệnh làm kích hoạt trigger, ta sử dụng câu lệnh:

```
ROLLBACK TRANSACTION
```

- Ví dụ: Tạo trigger không cho phép xóa dữ liệu trong bảng MATHANG.

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

6.1.3 CÁC TÌNH HUỐNG SỬ DỤNG TRIGGER (tiếp theo)

Sử dụng trigger với biến con trỏ

- Sử dụng trong trường hợp câu lệnh INSERT, UPDATE và DELETE có tác động đến nhiều dòng dữ liệu.
- Sử dụng con trỏ để duyệt qua các dòng dữ liệu và kiểm tra trên từng dòng.
  - Khai báo theo cú pháp:

```
DECLARE tên_con_trỏ CURSOR
FOR câu_lệnh_SELECT
```

  - Mở một biến con trỏ:

```
OPEN tên_con_trỏ
```

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

```
FETCH [[NEXT|PRIOR|FIST|LAST] FROM] tên_con_trỏ
[INTO danh_sách_biến]
```

6.1.3 CÁC TÌNH HUỐNG SỬ DỤNG TRIGGER (tiếp theo)

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 control   INTO @mahang,@tenhang,@soluong
 END
CLOSE contro DEALLOCATE contro
```

6.2. TRANSACTION

6.2.1. Giới thiệu về Transaction

6.2.2. Quản lý giao tác trong SQL

6.2.3. Transaction lồng nhau

6.2.4. Các tình huống sử dụng transaction

6.2.1. GIỚI THIỆU VỀ TRANSACTION

Khái niệm

- Giao tác (Transaction) là một chuỗi các thao tác thực thi cùng với nhau như là một khối thống nhất của công việc (unit of work);
- Một transaction hoặc là sẽ thực hiện thành công hoàn toàn hoặc là sẽ không thực hiện bất cứ lệnh nào trong khối lệnh (all or nothing).
- Ví dụ: Để chuyển một số tiền 5 tỉ đồng từ tài khoản A sang tài khoản B cần làm các bước sau:
  - Trừ $500 từ account A;
  - Cộng $500 vào account B.

6.2.1. GIỚI THIỆU VỀ TRANSACTION (tiếp theo)

Các đặc điểm Transaction: Để tạo thành một khối lệnh thống nhất, Transaction phải đảm bảo 4 đặc điểm:

- Tính nguyên tố (Atomicity): Cả khối lệnh được thực hiện hoặc không lệnh nào trong khối được thực hiện;
- Tính nhất quán (Consistency): Tất cả dữ liệu ở trong tình trạng nhất quán khi transaction được hoàn tất;
- Tính riêng biệt (Isolation): Thực hiện các thao tác độc lập với transaction đồng thời khác;
- Tín bền vững (Durability): Hệ thống ổn định ngay cả khi hệ thống phát sinh lỗi. Đặc điểm này được đảm bảo bởi transaction log.

6.2.1. GIỚI THIỆU VỀ TRANSACTION (tiếp theo)

Phân loại Transaction:

- Giao dịch tường minh (Explicit transaction): Là một transaction mà chúng ta phải định nghĩa bắt đầu một transaction (Begin transaction) và kết thúc một transaction (Commit Transaction).
- Giao dịch ngầm định (Implicit transaction): Tự động khởi tạo ngay khi ở chế độ ON và SQL Server chạy, chỉ cần điều khiển commit hay rollback transaction này. SQL Server tự động bắt đầu một transaction khi nó thực thi bất kỳ các lệnh sau: Alter Table, Creat, Delete, Drop, Fetch, Grant, Insert, Open, Revoke, Select, Truncate Table, Update.
- Giao dịch phân tán (Distributed Transaction): Là giao tác liên quan nhiều server.

6.2.1. GIỚI THIỆU VỀ TRANSACTION (tiếp theo)

Sử dụng Transaction Log:

- Nhật ký giao dịch (Transaction log) là một File riêng biệt (hoặc lưu ở vùng đĩa riêng) ở trong database server. Nó lưu trữ dấu vết thực hiện của các thao tác;
- Bằng cách lưu trữ dấu vết thực hiện trong tệp nhật ký (file log), database server dễ dàng khôi phục lại dữ liệu khi gặp sự cố;
- Transaction log đảm bảo tính hoàn chỉnh và tính bền vững;
- SQL Server viết tất cả các thay đổi trên CSDL xuống transaction log, vì vậy nếu transaction đã bắt đầu nhưng chưa chạy xong, chúng ta vẫn có thể phục hồi lại tất cả các thay đổi từ file log.

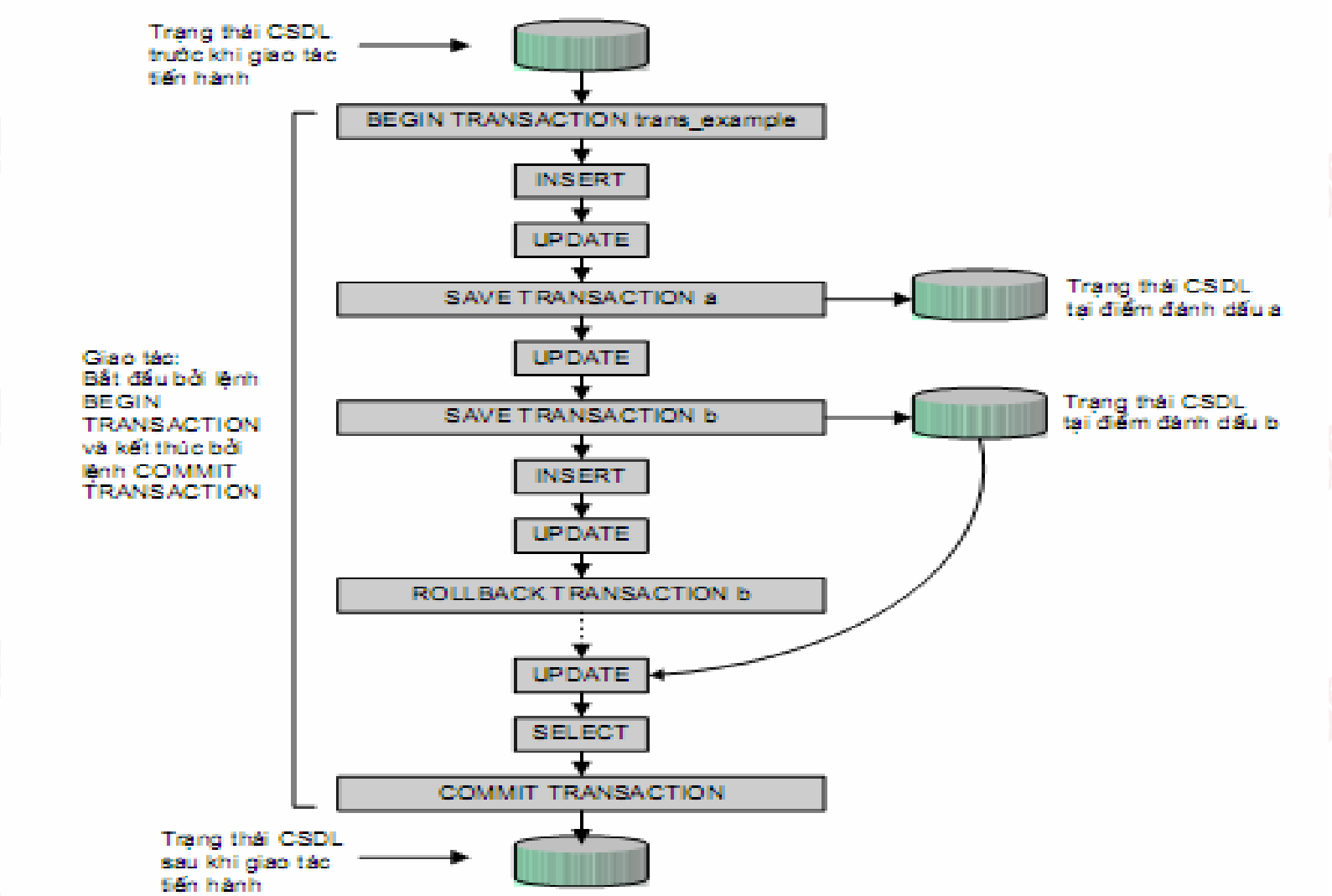
6.2.1. GIỚI THIỆU VỀ TRANSACTION (tiếp theo)

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

- BEGIN TRANSACTION: Bắt đầu một giao tác;
- SAVE TRANSACTION: Đánh dấu một vị trí trong giao tác (gọi là điểm đánh dấu).
- ROLLBACK TRANSACTION: Quay lui trở lại đầu giao tác hoặc một điểm đánh dấu trước đó trong giao tác.
- COMMIT TRANSACTION: Đánh dấu điểm kết thúc một giao tác. Khi câu lệnh này thực thi cũng có nghĩa là giao tác đã thực hiện thành công.
- ROLLBACK [WORK]: Quay lui trở lại đầu giao tác.
- COMMIT [WORK]: Đánh dấu kết thúc giao tác.

6.2.1. GIỚI THIỆU VỀ TRANSACTION (tiếp theo)

Hoạt động của một giao tác



6.2.2. QUẢN LÝ GIAO TÁC TRONG SQL

- Bắt đầu một transaction:

```
BEGIN TRAN [ SACTION ] [ transaction_name |
   @tran_name_variable]
```

- Hoàn Tất Transaction:

```
COMMIT [ TRAN [ SACTION ] [ transaction_name
|@tran_name_variable ] ]
```

- Lưu vị trí Transaction:

```
SAVE   TRAN   [   SACTION   ]   {   savepoint_name   |
   @savepoint_variable }
```

- Hủy một Transaction:

```
ROLLBACK [ TRAN [ SACTION ] [ transaction_name |
@tran_name_variable | savepoint_name |
@savepoint_variable ] ]
```

6.2.2. QUẢN LÝ GIAO TÁC TRONG SQL (tiếp theo)

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

```
BEGIN TRANSACTION giaotac1
UPDATE monhoc SET sodvht=4 WHERE sodvht=3
UPDATE diemthi SET diemlan2=0 WHERE diemlan2 IS NULL
ROLLBACK TRANSACTION giaotac1
```

- Ví dụ: Giao tác dưới đây kết thúc bởi lệnh COMMIT và thực hiện thành công việc cập nhật dữ liệu trên các bảng MONHOC và DIEMTHI.

```
BEGIN TRANSACTION giaotac2
UPDATE monhoc SET sodvht=4 WHERE sodvht=3
UPDATE diemthi SET diemlan2=0 WHERE diemlan2 IS NULL
COMMIT TRANSACTION giaotac2
```

6.2.3. TRANSACTION LỒNG NHAU

- Các giao tác trong SQL có thể được lồng vào nhau theo từng cấp.
- Điều này thường gặp đối với các giao tác trong các thủ tục lưu trữ được gọi hoặc từ một tiến trình trong một giao tác khác.
- Khi các giao tác SQL được lồng vào nhau, giao tác ngoài cùng nhất là giao tác có vai trò quyết định.
- Ví dụ: Xét bảng CREATE TABLE T (A INT PRIMARY KEY, B INT) và thủ tục.

```
CREATE PROC sp_TranEx(@a INT,@b INT)
AS
BEGIN
  BEGIN TRANSACTION T1
  IF NOT EXISTS (SELECT * FROM T WHERE A=@A )
    INSERT INTO T VALUES(@A,@B)
  IF NOT EXISTS (SELECT * FROM T WHERE A=@A+1)
    INSERT INTO T VALUES(@A+1,@B+1)
  COMMIT TRANSACTION T1
END
```

6.2.3. TRANSACTION LỒNG NHAU (tiếp theo)

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

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

A	B
20	40
21	41
40	80
41	81

- Sau khi giao tác trên thực hiện xong, dữ liệu trong bảng T sẽ là:
- Như vậy, tác dụng của lời gọi thủ tục sp_tranex 30, 60 trong giao tác đã bị huỷ bỏ bởi câu lệnh ROLLBACK TRANSACTION trong giao tác.

6.2.4. CÁC TÌNH HUỐNG SỬ DỤNG TRANSACTION

Transaction và xử lý lỗi

- BEGIN TRY để mở ra khối try block. Khối try block sẽ chứa các lệnh cần thực hiện trong transaction. END TRY để kết thúc khối try block
- BEGIN CATCH. Đây là phần chứa đoạn lệnh sẽ được thực hiện khi có lỗi trong phần try block. Trong phần catch lệnh đầu tiên là ROLLBACK để quay lui transaction.
- RAISERROR để báo cho ứng dụng biết thủ tục đã gây ra lỗi và truyền thông báo lỗi cho ứng dụng.

```
BEGIN TRAN
BEGIN TRY
-- lệnh1
-- lệnh2
-- ...
COMMIT
END TRY
BEGIN CATCH
ROLLBACK
DECLARE @ErrorMessage VARCHAR(2000)
SELECT @ErrorMessage = 'Lỗi: ' + ERROR_MESSAGE()
RAISERROR(@ErrorMessage, 16, 1) END CATCH
```

6.2.4. CÁC TÌNH HUỐNG SỬ DỤNG TRANSACTION (tiếp theo)

Transaction và biến con trỏ

Trong một số trường hợp transaction được sử dụng kết hợp với biến con trỏ để đảm bảo sự nhất quán trong thực thi thao tác.

```
DECLARE MyCursor CURSOR FOR
SELECT c.CustomerID,c.Companyname,c.contactname,
 o.OrderID,o.OrderDate
FROM Customers c, Orders o WHERE c.CustomerID =
o.CustomerID
FOR UPDATE
OPEN MyCursor
DECLARE @cid VARCHAR( 8), @c VARCHAR( 80), @o INT,
 @od DATETIME, @cn VARCHAR( 80)
FETCH NEXT FROM MyCursor INTO @cid, @c, @cn, @o, @od
SELECT @cid
BEGIN TRANSACTION
UPDATE Customers SET CompanyName = 'q'
WHERE CURRENT OF Mycursor
DEALLOCATE MyCursor
SELECT *  FROM Customers
ROLLBACK TRANSACTION
```

TÓM LƯỢC CUỐI BÀI

Trong bài này chúng ta đã xem xét các nội dung sau:

- Trình bày được khái niệm, đặc điểm và phân loại Trigger;
- Tạo mới, quản lý và sử dụng Trigger;
- Trình bày được khái niệm, đặc điểm và phân loại Transaction;
- Tạo mới, quản lý và sử dụng Transaction.