



# CÁC HỆ QUẢN TRỊ CSDL

**ĐẠI HỌC SƯ PHẠM TP.HCM**  
**KHOA CÔNG NGHỆ THÔNG TIN**  
**Phiên bản 2014**



# CHƯƠNG 5:

## XỬ LÝ TRUY XUẤT ĐỒNG THỜI



- ❑ Khái niệm giao tác (transaction).
- ❑ Các vấn đề xảy ra khi nhiều người cùng khai thác Cơ Sở Dữ Liệu.
- ❑ Các giải pháp cho các vấn đề trên.
- ❑ Sử dụng giao tác trong SQL Server.



- ❑ Giao tác là một dãy các thao tác cần thực hiện trên cơ sở dữ liệu dưới một đơn vị duy nhất, nghĩa là hoặc thực hiện tất cả các thao tác hoặc không thực hiện thao tác nào cả.

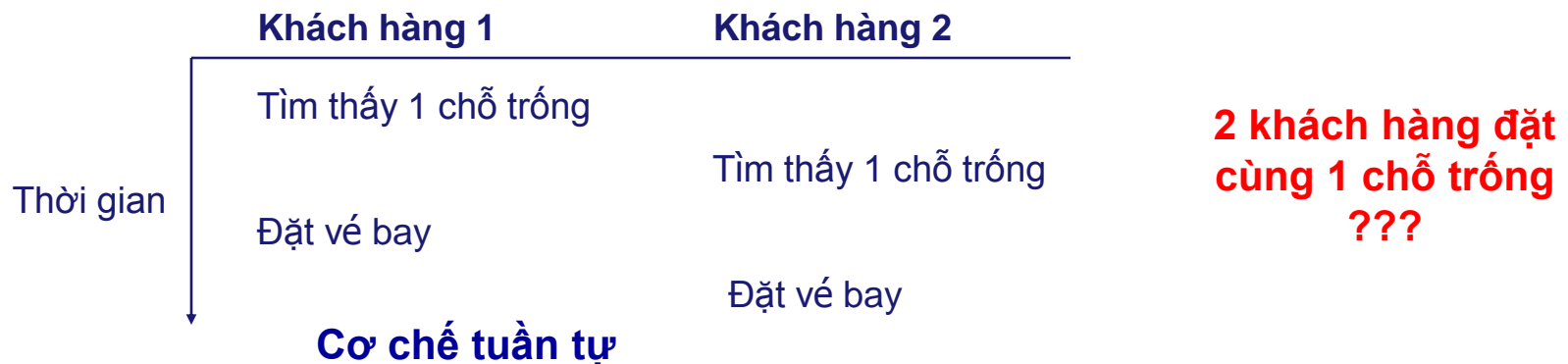


## ❖ Ví dụ

- Hệ thống giao dịch ngân hàng
- Hệ thống đặt vé bay

## ❖ DBMS là môi trường đa người dùng

- Nhiều thao tác truy xuất lên cùng một đơn vị dữ liệu
- Nhiều thao tác thi hành đồng thời



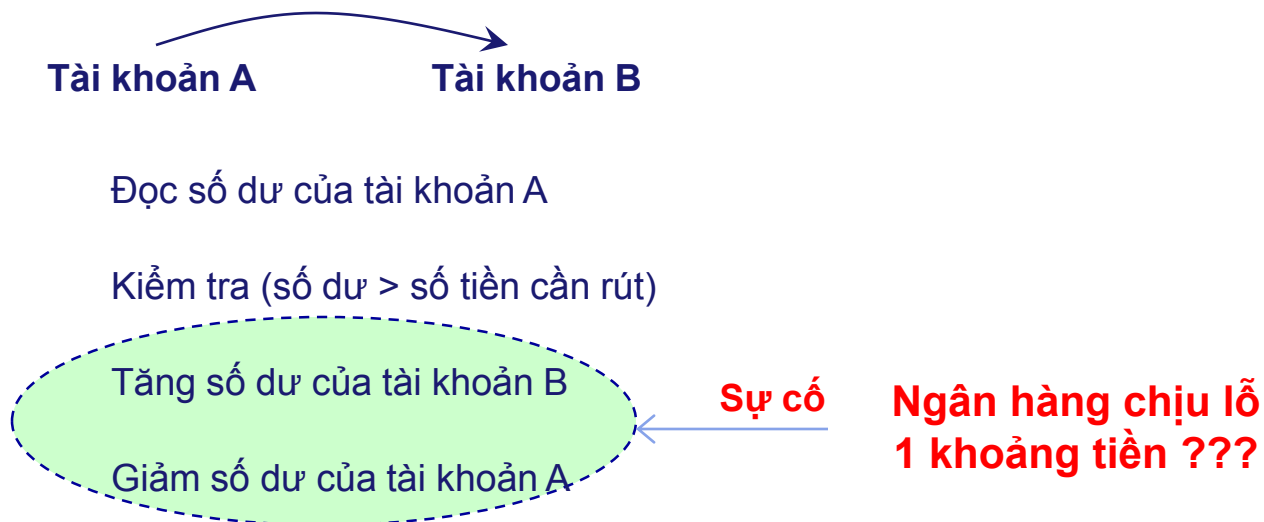


- ❑ Ví dụ: giao tác chuyển khoản từ  $A \rightarrow B$  gồm 2 thao tác sau:
  - ❑ Trừ tiền A
  - ❑ Cộng tiền B
- ❑ Chuyển khoản được thực hiện dưới dạng giao tác (transaction) nghĩa là hoặc thực hiện cả 2 việc trừ tiền A và cộng tiền B hoặc nếu có sự cố thì không làm gì cả và thông báo giao tác thất bại.



## ❖ Khi DBMS gặp sự cố

- Các thao tác có thể làm cho trạng thái CSDL không chính xác





- ❑ Vấn đề mất dữ liệu đã cập nhật
- ❑ Vấn đề không thể đọc lại
- ❑ Vấn đề dữ liệu không nhất quán



# Các vấn đề trong truy xuất đồng thời



- ❑ Vấn đề mất dữ liệu đã cập nhật
- ❑ Ví dụ: Nhà sách còn 500 quyển sách.
  - ❑ Vào lúc  $T_1$  nhân viên A nhân yêu cầu mua 400 quyển từ khách hàng X.
  - ❑ Cũng vào  $T_1$  nhân viên B nhân yêu cầu mua 300 quyển từ khách hàng Y.
  - ❑ A và B đọc dữ liệu thấy còn 500 quyển nên đều đồng ý bán

# Các vấn đề trong truy xuất đồng thời



- ❑ Vấn đề mất dữ liệu đã cập nhật (tt)
- ❑ Ví dụ: Nhà sách còn 500 quyển sách.
  - ❑ Vào lúc  $T_2$  nhân viên A sẽ thực hiện cập nhật số sách từ 500 thành 100.
  - ❑ Vào lúc  $T_3$  nhân viên B sẽ thực hiện cập nhật số sách từ 500 thành 200.
- ❑ Như vậy thao tác cập nhật của A không có tác dụng hay dữ liệu mà A cập nhật sẽ bị mất vì B cập nhật sau **???** (last in wind)

# Các vấn đề trong truy xuất đồng thời



- ❑ Vấn đề không thể đọc lại
- ❑ Ví dụ: Giả sử nhà sách còn 200 quyển sách.
  - ❑ Vào lúc  $T_1$  nhân viên A bán cho khách 150 quyển, sẽ thực hiện cập nhật số sách từ 200 thành 50. (giao dịch chưa hoàn thành chẳng hạn vì việc giao nhận tiền chưa xong)
  - ❑ Sau đó lúc  $T_2$ , B nhận được yêu cầu mua 100 quyển sách, nếu B được đọc dữ liệu chưa hoàn tất thì B sẽ từ chối bán 100 quyển sách này.

# Các vấn đề trong truy xuất đồng thời



❑ Vấn đề không thể đọc lại (tt)

❑ Ví dụ: Giả sử nhà sách còn 200 quyển sách.

- ❑ Nếu vào lúc  $T_3$  vì lý do nào đó chẳng hạn không đủ tiền khách hàng của A không mua 150 quyển sách nữa. Giao tác bán hàng của A sẽ không thể thực hiện nên quay về trạng thái số sách còn là 200.
- ❑ Nhưng B đã từ chối khách hàng.
- ❑ Nếu B không đọc được dữ liệu từ lúc  $T_1$  đến  $T_3$  thì sẽ như thế nào?



❑ Vấn đề dữ liệu không nhất quán

❑ Ví dụ: Giả sử nhân viên C cần tổng hợp 5 dòng dữ liệu 1 2 3 4 5 để làm một bản báo cáo.

- ❑  $T_1$ : C đọc và đưa các dòng 1 2 3 4 vào báo cáo

- ❑  $T_2$ : D lại xóa dòng 1 thay bằng dòng 6.

- ❑  $T_3$ : C đọc tiếp các dòng 5 6 đưa vào báo cáo

- ❑ Vậy báo cáo này xử lý cả dữ liệu cũ và mới → **SAI**

# Tính chất ACID của giao tác



## ❖ Nguyên tố (**A**tomicity)

- Hoặc là toàn bộ hoạt động của giao dịch được phản ánh đúng đắn trong CSDL hoặc không có hoạt động nào cả

## ❖ Nhất quán (**C**onsistency)

- Một giao tác được thực hiện độc lập với các giao tác khác xử lý đồng thời với nó để bảo đảm tính nhất quán cho CSDL

## ❖ Cô lập (**I**solation)

- Một giao tác không quan tâm đến các giao tác khác xử lý đồng thời với nó

## ❖ Bền vững (**D**urability)

- Mọi thay đổi mà giao tác thực hiện trên CSDL phải được ghi nhận bền vững



```
T: Read(A, t) ;  
    t:=t-50 ;  
    Write(A, t) ;  
    Read(B, t) ;  
    t:=t+50 ;  
    Write(B, t) ;
```

## ❑ Nhất quán Consistency

- Tổng  $A+B$  là không đổi
- Nếu CSDL nhất quán trước khi T được thực hiện thì sau khi T hoàn tất CSDL vẫn còn nhất quán

# Ví dụ (tt)



```
T: Read(A, t) ;  
    t:=t-50 ;  
    Write(A, t) ;  
    Read(B, t) ;  
    t:=t+50 ;  
    Write(B, t) ;
```

## □ Nguyên tố Atomicity

- $A=100, B=200$  ( $A+B=300$ )
- Tại thời điểm sau khi `write(A,t)`
  - $A=50, B=200$  ( $A+B=250$ ) - CSDL không nhất quán
- Tại thời điểm sau khi `write(B,t)`
  - $A=50, B=250$  ( $A+B=300$ ) - CSDL nhất quán
- Nếu T không bao giờ bắt đầu thực hiện hoặc T được đảm bảo phải hoàn tất thì trạng thái không nhất quán sẽ không xuất hiện





```
T: Read(A, t) ;  
   t:=t-50 ;  
   Write(A, t) ;  
   Read(B, t) ;  
   t:=t+50 ;  
   Write(B, t) ;
```

## ❑ Bền vững Durability

- Khi T kết thúc thành công
- Dữ liệu sẽ không thể nào bị mất bất chấp có sự cố hệ thống xảy ra

# Ví dụ (tt)



```

T: Read(A, t);
   t:=t-50;
   Write(A, t);
T' → Read(B, t);
      t:=t+50;
      Write(B, t);

```

## ❑ Cô lập Isolation

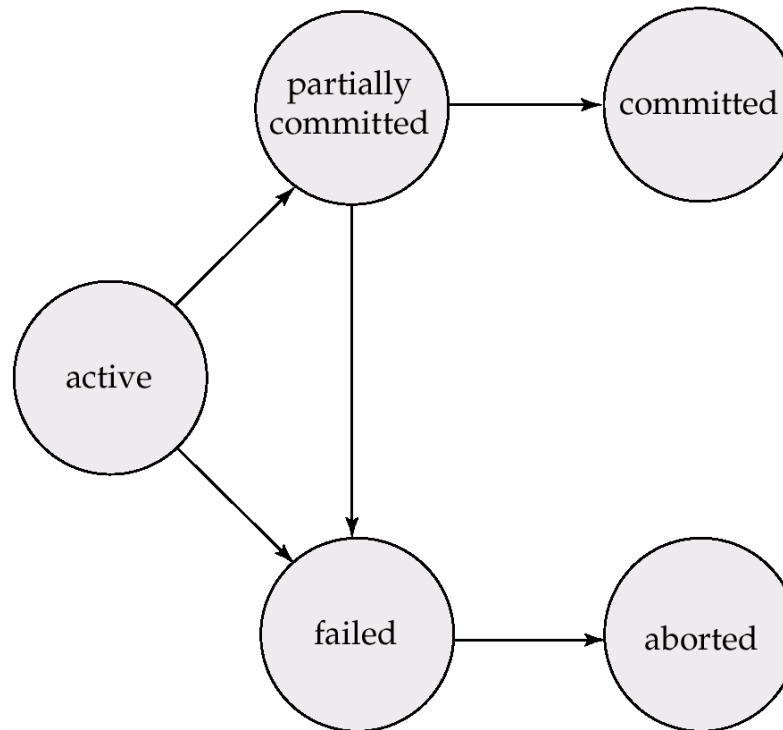
- Giả sử có 1 giao tác T' thực hiện phép toán  $A+B$  và chen vào giữa thời gian thực hiện của T
- T' kết thúc:  $A+B=50+200=250$
- T kết thúc:  $A+B=50+250=300$
- Hệ thống của các giao tác thực hiện đồng thời có trạng thái tương đương với trạng thái hệ thống của các giao tác thực hiện tuần tự theo 1 thứ tự nào đó

# Trạng thái của giao tác



- ❖ Active
  - Ngay khi bắt đầu thực hiện thao tác đọc/ghi
- ❖ Partially committed
  - Sau khi lệnh thi hành cuối cùng thực hiện
- ❖ Failed
  - Sau khi nhận ra không thể thực hiện các hành động được nữa
- ❖ Aborted
  - Sau khi giao tác được quay lui và CSDL được phục hồi về trạng thái trước trạng thái bắt đầu giao dịch
    - Bắt đầu lại giao tác (nếu có thể)
    - Hủy giao tác
- ❖ Committed
  - Sau khi mọi hành động hoàn tất thành công

# Sơ đồ trạng thái của giao tác





- ❖ Giao dịch tường minh  
(Explicit transaction)
- ❖ Giao dịch ngầm định  
(Implicit transaction)
- ❖ Giao dịch xác nhận  
(Commit transaction)



- ❖ **Khóa** (Lock) được sinh ra để giới hạn quyền truy nhập trên môi trường đa người dùng.
- ❖ Microsoft SQL Server 200X sử dụng lock để đảm bảo **tính toàn vẹn** của transaction và **tính thống nhất** của database.
- ❖ Nếu lock không được sử dụng, dữ liệu bên trong CSDL có thể bị sai về logic, và các query chạy trên đó sẽ đưa ra các kết quả không mong đợi.
- ❖ **Bản chất của lock là việc một người muốn truy nhập riêng vào một bảng**, vì vậy server sẽ lock bảng đó lại cho riêng người đó.



# Phân loại các Locks trong SQL Server

- ☐ Pessimistic Lock

- ☐ Optimistic Lock

- ☐ Shared Locks

- ☐ Exclusive Locks

- ☐ Update Locks



- ❖ **Share Lock:** khi một giao tác đang đọc dữ liệu X thì X sẽ bị share lock. Nghĩa là giao tác khác trong cùng thời điểm chỉ có quyền đọc X và không có quyền sửa X.
- ❖ **Exclusive Lock:** khi một giao tác đang cập nhật dữ liệu X thì X sẽ bị exclusive lock. Nghĩa là giao tác khác trong cùng thời điểm không thể đọc hay sửa X.

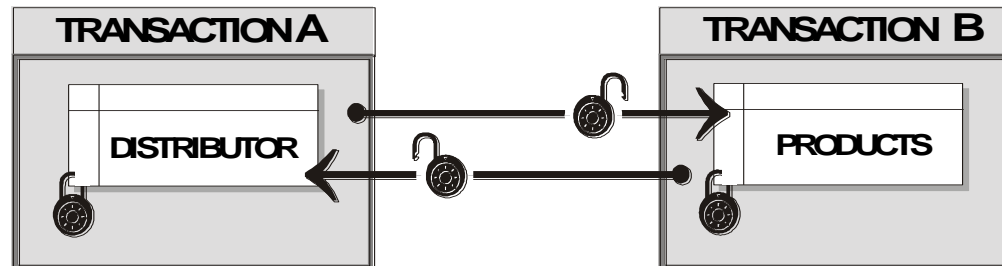




- ❖ Kỹ thuật khóa 2 giai đoạn
- ❖ Kỹ thuật khóa trên dữ liệu phân cấp
- ❖ Khóa chết (**dead lock**): là tình trạng 2 hay nhiều giao tác cùng trong trạng thái chờ giao tác giải phóng tài nguyên cần thiết để hoàn thành giao tác.



## ❖ Khóa chết (dead lock)



# Deadlock



- ❖ Một deadlock xảy ra khi có 2 người dùng (hoặc 2 phiên làm việc) đã đặt khóa trên 2 đối tượng riêng, và mỗi user muốn đặt khóa trên đối tượng của user kia. Mỗi user đều phải đợi người kia giải phóng khóa của họ ra để mình có thể đặt khóa.
- ❖ SQL Server tự động nhận ra deadlock và giải quyết bằng cách chọn một ứng dụng và bắt nó phải giải phóng khóa, trong khi đó vẫn cho ứng dụng còn lại chạy tiếp.
- ❖ Cách tốt nhất để tránh deadlock là tránh nó. Một cách để tránh nó là không chạy các transaction đồng thời.



❖ SET DEADLOCK\_PRIORITY

❖ SET LOCK\_TIMEOUT



## ❖ Phát hiện

- Cho phép trạng thái deadlock xảy ra và sau đó cố gắng khôi phục lại hệ thống
  - Chọn 1 giao tác để rollback
- Phương pháp
  - Đồ thị chờ (wait-for graph)

## ❖ Ngăn ngừa

- Quản lý các giao tác sao cho không bao giờ có deadlock
- Phương pháp
  - Sắp thứ tự tài nguyên (resource ordering)
  - Timeout
  - Wait-die
  - Wound-wait



## ❖ Đồ thị chờ

- Đỉnh là các giao tác đang giữ khóa hoặc đang chờ khóa
- Cung đi từ đỉnh T sang U khi
  - U đang giữ khóa trên đơn vị dữ liệu A
  - T đang chờ khóa trên A
  - T không thể khóa đơn vị dữ liệu A nếu U không giải phóng khóa

## ❖ Nếu đồ thị chờ không có chu trình

- Các giao tác có thể hoàn tất

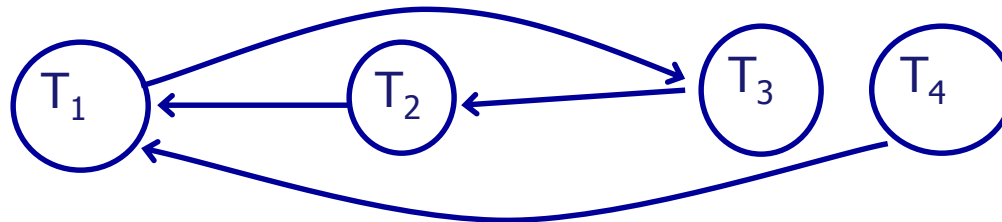
## ❖ Ngược lại

- Không một giao tác nào trong chu trình có thể tiếp tục thực hiện → deadlock

# Ví dụ



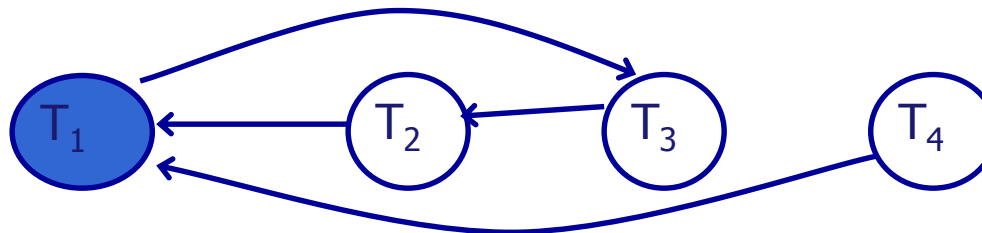
	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>
1	L(A); R(A)			
2		L(C); R(C)		
3			L(B); R(B)	
4				L(D); R(D)
5		L(A)		
6		↓ Chờ	L(C)	
7			↓ Chờ	L(A)
8	L(B)			↓ Chờ
	↓ Chờ			↓ Chờ



# Ví dụ (tt)



	$T_1$	$T_2$	$T_3$	$T_4$
1	L(A); R(A)			
2		L(C); R(C)		
3			L(B); R(B)	
4				L(D);
5		L(A)		R(D)
6			L(C)	
7				L(A)
8	L(B)			







- ❑ Áp đặt một thứ tự nào đó lên các đơn vị dữ liệu
- ❑ Nếu các giao tác thực hiện khóa những đơn vị dữ liệu theo thứ tự này
- ❑ Thì không có deadlock xảy ra trong khi chờ đợi



- ❑ Giới hạn các giao tác chỉ được thực hiện trong 1 khoảng thời gian nào đó
- ❑ Nếu giao tác vượt quá thời gian này
- ❑ Thì giao tác phải bị rollback



❑ Mỗi giao tác sẽ được gán một nhãn ghi nhận thứ tự xuất hiện, kí hiệu:  $ts(T)$

❑ Xét 2 giao tác T và U

- U đang giữ khóa trên đơn vị dữ liệu A
- T muốn khóa đơn vị dữ liệu A



- T sẽ chờ-**wait** U khi  $ts(T) < ts(U)$
- Ngược lại T sẽ bị hủy-**die** và bắt đầu làm lại ở 1 thời điểm khác



# Wound-wait



❑ Mỗi giao tác sẽ được gán một nhãn ghi nhận thứ tự xuất hiện, kí hiệu:  $ts(T)$

❑ Xét 2 giao tác T và U

- U đang giữ khóa trên đơn vị dữ liệu A
- T muốn khóa đơn vị dữ liệu A



- T buộc U rollback và trao khóa lại cho T-wound khi  $ts(T) < ts(U)$

- Ngoại lệ: nếu U đã kết thúc và giải phóng khóa, U sẽ không rollback



- Ngược lại T sẽ chờ-wait U



## □ Timeout

- Đơn giản
- Khó chọn được khoảng thời gian timeout thích hợp
- Có hiện tượng starvation
  - Giao tác lặp đi lặp lại quá trình: bắt đầu, deadlock, rollback

## □ Resource ordering

- Không thực tế
- Chờ đợi nhiều → tiềm ẩn của deadlock



## □ Wait-die và Wound-wait

- Không có starvation
- Wound-wait ít rollback các giao tác hơn wait-die
- Dễ cài đặt hơn wait-for graph
- Có thể rollback những giao tác không gây ra deadlock

## □ Wait-for graph

- Nếu đồ thị quá lớn sẽ tốn nhiều thời gian phân tích
- Rất phức tạp khi CSDL phân tán
- Giảm tối thiểu rollback các giao tác
  - Chỉ rollback 1 trong những giao tác gây ra deadlock



- ❑ BEGIN TRANSACTION
- ❑ COMMIT TRANSACTION
- ❑ ROLLBACK TRANSACTION
- ❑ SAVE TRANSACTION

## 2 loại giao tác



- ❑ Giao tác không tường minh: Mặc định các lệnh bên trong 1 lô (batch), chỉ cần 1 lệnh thực hiện không thành công thì tất cả các lệnh còn lại trong lô đó sẽ không thực hiện → không khuyến khích.
- ❑ Giao tác tường minh: Có chỉ định **BEGIN** đánh dấu bắt đầu giao tác và **COMMIT / ROLLBACK** để kết thúc giao tác.





- ❖ Transaction Recovery: 2 thao tác
  - **COMMIT**: hoàn tất giao tác thành công
  - **ROLLBACK**: giao tác thất bại quay về trạng thái trước khi thực hiện giao tác
- ❖ Transaction Log: undo the changes
  - Khi user muốn một chỉnh sửa dữ liệu, trong transaction log lưu 2 phiên bản của dòng dữ liệu đó: trước và sau khi chỉnh sửa.



- ❖ Nếu user thực hiện câu lệnh **Commit** thì end-of-transaction được ghi xuống transaction log.
- ❖ Nếu user thực hiện câu lệnh **Rollback** thì hệ thống sẽ tìm phiên bản trước khi chỉnh sửa và cập nhật lại vào CSDL.
- ❖ Điều gì xảy ra khi SQL Server khởi động và trong transaction log có một giao tác chưa Commit và cũng không Rollback?



## Chuyển giao tự động các transaction – Autocommit Transactions

- ❖ Là chế độ giao tác mặc định
- ❖ Mọi câu lệnh khi hoàn tất đều:  
được xác nhận (committed) hoặc quay lui (rolled back)
  - Nếu hoàn tất thành công → xác nhận (committed)
  - Nếu thất bại → quay lui (rolled back)
- ❖ Các lỗi biên dịch sẽ dẫn đến việc một lô xử lý (batch) và sẽ không được thực thi.



Đảm bảo tính nhất quán của dữ liệu

**Vấn đề 1:**

Một nhân viên có mã '000002' được chọn làm 'Sales Manager' (position code='0001').

- + Ta sẽ cần cập nhật trong bảng Employee
- + và cả trong bảng Position số lượng người hiện tại nắm giữ vị trí có Position Code='0001'.

- **Lệnh SQL tương ứng:**

```
UPDATE Employee  
SET cCurrentPosition = '0001'  
WHERE cEmployeeCode= '000002'
```

```
UPDATE Position  
SET iCurrentStrength=iCurrentStrength + 1  
WHERE cPositionCode='0001'
```



## Giải quyết vấn đề 1:

```
BEGIN TRANSACTION trnUpdatePosition
```

```
UPDATE Employee
```

```
SET cCurrentPosition = '0001'
```

```
WHERE cEmployeeCode= '000002'
```

```
UPDATE Position
```

```
SET iCurrentStrength =
```

```
iCurrentStrength + 1
```

```
WHERE cPositionCode = '0001'
```

```
COMMIT TRANSACTION trnUpdatePosition
```

# Cài đặt Transaction trên SQL Server



## Vấn đề 2:

- Mười ứng cử viên đã được tuyển chọn cho vị trí 0015.
- + Để phản ánh sự thay đổi này, thuộc tính siNoOfVacancy của bảng Requisition sẽ được giảm 10 cho cRequisitionCode 000004.
- + Thuộc tính iCurrentStrength của bảng Position là được tăng 10 cho cPositionCode 0015 bằng cách sử dụng lệnh sau:

```
UPDATE Requisition  
set siNoOfVacancy=siNoOfVacancy - 10  
WHERE cRequisitionCode='000004'
```

```
UPDATE Position  
set iCurrentStrength=iCurrentStrength + 10  
WHERE cPositionCode='0015'
```

**Cả hai tập lệnh trên phải bảo đảm tính Atomic,**  
**- thuộc tính iCurrentStrength tăng không quá giá trị thuộc**  
**tính iBudgetedStrength.**



## ***Giải quyết vấn đề 2:***

```
BEGIN TRANSACTION
```

```
UPDATE Requisition
```

```
SET siNoOfVacancy=siNoOfVacancy - 10
```

```
WHERE cRequisitionCode='000004'
```

```
UPDATE Position
```

```
SET iCurrentStrength=iCurrentStrength + 10
```

```
WHERE cPositionCode='0015'
```

# Cài đặt Transaction trên SQL Server



```
IF (SELECT iBudgetedStrength-iCurrentStrength FROM  
      Position          WHERE cPositionCode = '0015') <0  
  
BEGIN  
  
    PRINT 'Current strength cannot be more than budgeted  
          strength. Transaction has not been committed.'  
  
    ROLLBACK TRANSACTION  
  
END  
  
ELSE  
  
BEGIN  
  
    PRINT 'The transaction has been committed.'  
  
    COMMIT TRANSACTION  
  
END
```



# Tạo điểm dừng cho 1 TRANSACTION



❑ Lệnh `SAVE TRANSACTION` dùng để đặt 1 điểm dừng (save point) bên trong 1 transaction. Điểm dừng chia transaction thành các phần khác nhau sao cho transaction có thể quay về lại điểm dừng này nếu 1 phần của transaction bị loại bỏ có điều kiện.

❑ *Cú pháp*

*`SAVE TRAN[SACTION] {savepoint_name !  
@savepoint_variable}`*

# Cài đặt Transaction trên SQL Server



```
CREATE TABLE #TestTran(CotA INT PRIMARY KEY, CotB char(3))
GO

BEGIN TRAN

    --Vùng thứ 1
    SAVE TRAN Dong_1_2
    INSERT INTO #TestTran VALUES(1, 'aaa')
    INSERT INTO #TestTran VALUES(2, 'bbb')
    --Vùng thứ 2
    SAVE TRAN Dong_3
    INSERT INTO #TestTran VALUES(3, 'ccc')
    ROLLBACK TRAN Dong_3
COMMIT TRAN Dong_1_2
```



- ❑ Một số lỗi thường gặp sau khi thực hiện 1 câu lệnh trong giao tác:
  - Không có quyền truy cập trên 1 đối tượng (table, stored procedure,...)
  - Vi phạm ràng buộc toàn vẹn (primary key, foreign key, check, rule, các ràng buộc được kiểm tra bằng trigger,...).
  - Deadlock.
  - ...



❑ SQL Server trả giá trị lỗi về trong biến toàn cục @@ERROR.

- @@ERROR= 0: không xảy ra lỗi
- @@ERROR <> 0: xảy ra lỗi với mã lỗi là @@ERROR



- ❑ Giao tác không thể tự động **ROLLBACK** khi gặp những lỗi phát sinh trong quá trình thực hiện 1 câu lệnh thành phần trong giao tác. Vì vậy cần kiểm tra giá trị của biến **@@ERROR** sau mỗi câu lệnh thành phần trong giao tác và cần xử lý những lỗi (nếu có): yêu cầu giao tác **ROLLBACK** một cách tường minh bằng lệnh **ROLLBACK TRANSACTION**.

# Ví dụ về kiểm tra lỗi



*--insert vào bảng NhanVien*

```
insert into NhanVien values (...)
```

```
if ( @@ERROR <> 0 )
```

```
    begin
```

```
        rollback tran
```

```
        return
```

```
    end
```



## ❖ Lưu ý:

- SET XACT\_ABORT

### Cú pháp

```
SET XACT_ABORT { ON | OFF }
```

### Ý nghĩa sử dụng

- SET XACT\_ABORT ON: nếu một câu lệnh Transact-SQL phát sinh lỗi run-time, toàn bộ giao tác được kết thúc và được quay lui.
- SET XACT\_ABORT OFF: (mặc định) trong một số trường hợp chỉ những câu lệnh Transact-SQL phát sinh lỗi là bị quay lui và giao tác tiếp tục được xử lý.
- Các lỗi biên dịch, ví dụ như các lỗi về cú pháp, không bị ảnh hưởng bởi SET XACT\_ABORT.

# Các giao tác tường minh



Ví dụ: tạo bảng với ràng buộc cột i không được chứa giá trị 2

```
CREATE TABLE #t1(i INT, CONSTRAINT ck1 CHECK (i<>2) )
```

```
BEGIN TRAN
```

```
    INSERT #t1 SELECT 1
```

```
    INSERT #t1 SELECT 2 -- vi phạm ràng buộc
```

```
    INSERT #t1 SELECT 3
```

```
COMMIT
```

```
SELECT * FROM #t1
```

```
i
```

```
---
```

```
1
```

```
3
```

**(2 ROW(s) affected)**



# Các giao tác tường minh



Ví dụ: tạo bảng với ràng buộc cột i không được chứa giá trị 2

```
CREATE TABLE #t2 (i INT, CONSTRAINT ck2 CHECK (i<>2) )
```

```
SET XACT_ABORT ON
```

```
BEGIN TRAN
```

```
    INSERT #t2 SELECT 1
```

```
    INSERT #t2 SELECT 2 -- vi phạm ràng buộc
```

```
    INSERT #t2 SELECT 3
```

```
COMMIT
```

```
SELECT * FROM #t2
```

```
i
```

```
---
```

**(0 ROW(s) affected)**

# Khối lệnh TRY...CATCH



❖ Ý nghĩa: Thực hiện các lệnh trong khối TRY, nếu gặp lỗi sẽ chuyển qua xử lý bằng các lệnh trong khối CATCH

❖ Cú pháp :

```
BEGIN TRY
    <Tập_lệnh>
END TRY
BEGIN CATCH
    <Tập_lệnh>
END CATCH
```

▪ Các điểm cần lưu ý :

- TRY và CATCH phải cùng lô xử lý
- Sau khối TRY phải là khối CATCH
- Có thể lồng nhiều cấp

# Khởi lệnh TRY...CATCH



## ❖ Ví dụ:

```
BEGIN TRY

        SELECT 10/0

END TRY

BEGIN CATCH

SELECT

        ERROR_NUMBER() as ErrorNumber,

        ERROR_MESSAGE() as ErrorMessage

END CATCH
```



- ❖ Một số hàm ERROR thường dùng
  - ERROR\_NUMBER(): trả về mã số của lỗi
  - ERROR\_MESSAGE(): trả về chuỗi lỗi
  - ERROR\_SEVERITY() : trả về mức độ nghiêm trọng của lỗi
  - ERROR\_STATE(): trả về mã trạng thái của lỗi
  - ERROR\_LINE() : trả về dòng gây ra lỗi
  - ERROR\_PROCEDURE(): trả về tên thủ tục/trigger gây ra lỗi

# SQL 2005 -2008



```
SET XACT_ABORT ON
BEGIN TRAN
BEGIN TRY
-- Tập lệnh
-- ...
COMMIT
END TRY

BEGIN CATCH
    ROLLBACK
    DECLARE @ErrorMsg VARCHAR(2000)
    SELECT @ErrorMsg = 'Giao tác bị lỗi: ' + ERROR_MESSAGE()
    RAISERROR @ErrorMsg, 15, 1)
END CATCH
```



- ❑ Giao tác (Transaction) là một loạt các thao tác cần thực hiện dưới dạng một đơn vị duy nhất.
- ❑ Truy xuất đồng thời → vấn đề
  - Mất dữ liệu đã cập nhật
  - Không thể đọc lại
  - Dữ liệu không nhất quán
- ❑ Cài transaction trên SQL Server: begin transaction, commit, rollback, save transaction...





❖ Cho lược đồ CSDL sau:

**HOCSINH(MAHS, HOTEN, *MALOP*, NGAYSINH)  
LOP(MALOP, TENLOP, SOLUONG)**

❖ Giả sử bảng LOP đã có trường  
(‘12A1’, ‘Chuyên toán’, 25).

❖ Viết các câu lệnh để thêm một thêm 2 Học sinh mới vào Lớp đó.

❖ Đồng thời update lại Số lượng học sinh cho chính xác.

❖ Các câu lệnh này thực hiện dưới dạng 1 đơn vị.  
(*Dữ liệu thêm vào sinh viên tự cho*)





```
SET XACT_ABORT ON
BEGIN TRAN
DECLARE @MALOP CHAR(6) = '12A1'
INSERT INTO HOCSINH VALUES (---, ---, @MALOP, ---)
INSERT INTO HOCSINH VALUES (---, ---, @MALOP, ---)
UPDATE CLB
SET SOLUONG=(SELECT COUNT(MAHS)
              FROM HOCSINH WHERE MALOP=@MALOP)
IF @@ERROR !=0
    ROLLBACK TRAN
COMMIT TRAN
```