

# **Quản lý giao tác**

(Transaction Management)

# Nội dung

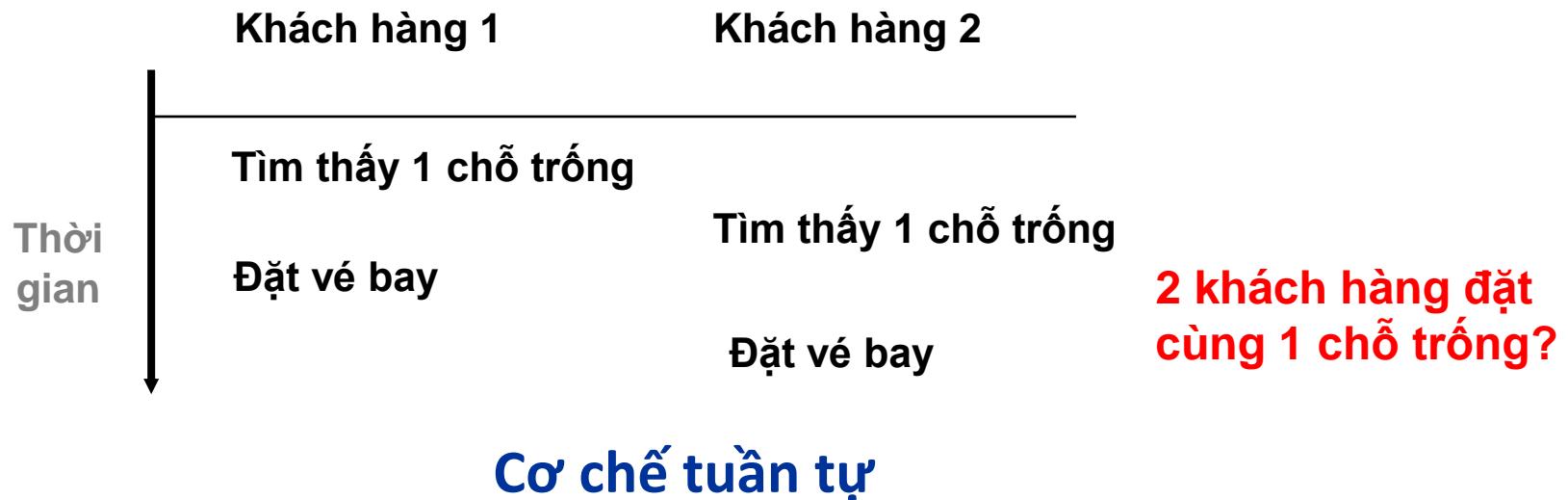
- Giới thiệu
- Giao tác
- Tính chất ACID của giao tác
- Các thao tác của giao tác
- Trạng thái của giao tác

# Nội dung

- Giới thiệu
- Giao tác
- Tính chất ACID của giao tác
- Các thao tác của giao tác
- Trạng thái của giao tác

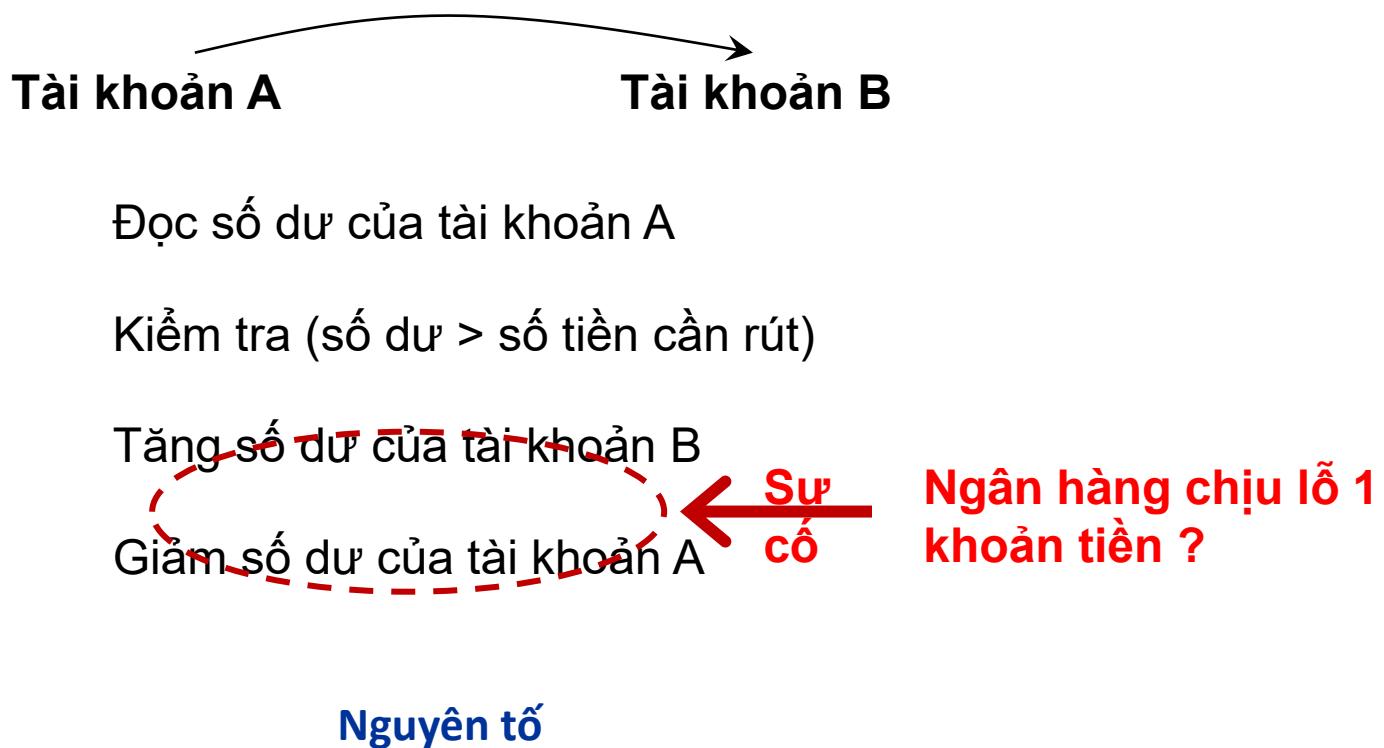
# Giới thiệu

- DBMS là môi trường đa người dùng
  - Nhiều thao tác truy xuất lên cùng 1 đơn vị dữ liệu
  - Nhiều thao tác thi hành đồng thời
- Ví dụ: Hệ thống đặt vé bay



# Giới thiệu

- ❑ Khi DBMS gặp sự cố, các thao tác có thể làm cho trạng thái CSDL không chính xác
- ❑ Ví dụ: Hệ thống giao dịch ngân hàng



# Nội dung

- Giới thiệu
- Giao tác
- Tính chất ACID của giao tác
- Các thao tác của giao tác
- Trạng thái của giao tác

# Giao tác (Transaction)

- ❑ Giải pháp cho vấn đề tuần tự (serial) và nguyên tố (atomic) là gom các nhóm thao tác phải thực hiện với nhau trong cùng 1 giao tác.
- ❑ **Định nghĩa:** Giao tác là một dãy các thao tác cần thực hiện trên cơ sở dữ liệu dưới một đơn vị duy nhất
  - hoặc tất cả các thao tác được thực hiện
  - hoặc không thực hiện thao tác nào cả

# Giao tác

- ❑ Ví dụ: giao tác chuyển khoản từ A → B gồm 2 thao tác
  - Trừ tiền A
  - Cộng tiền B
- ❑ Chuyển khoản được thực hiện dưới dạng giao tác, nghĩa là
  - hoặc thực hiện cả 2 thao tác trừ tiền A và cộng tiền B (giao tác thành công)
  - hoặc nếu có sự cố thì không thực hiện thao tác nào cả (giao tác thất bại)

# Nội dung

- Giới thiệu
- Giao tác
- Tính chất ACID của giao tác
- Các thao tác của giao tác
- Trạng thái của giao tác

# Tính chất của giao tác

- Để đảm bảo tính toàn vẹn của dữ liệu, ta yêu cầu hệ CSDL duy trì các tính chất sau của giao tác:
  - Nguyên tố (Atomicity)
  - Nhất quán (Consistency)
  - Cô lập (Isolation)
  - Bền vững (Durability)

**ACID**

# Tính chất ACID của giao tác

## ❑ Nguyên tố (Atomicity)

- Hoặc là toàn bộ hoạt động của giao dịch được phản ánh đúng đắn trong CSDL hoặc không có hoạt động nào cả.
- Đảm bảo bởi thành phần quản lý giao tác

## ❑ Nhất quán (Consistency)

- 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.
- Đảm bảo bởi người lập trình ứng dụng hay người viết ra giao tác

# Tính chất ACID của giao tác

## ❑ Cô lập (Isolation)

- Một giao tác không cần quan tâm đến các giao tác khác đang thực hiện đồng thời trong hệ thống.
- Đảm bảo bởi thành phần quản lý truy xuất đồng thời

## ❑ Tính bền vững (Durability)

- Mọi thay đổi mà giao tác thực hiện trên CSDL phải được ghi nhận bền vững.
- Đảm bảo bởi thành phần quản lý phục hồi

# Tính chất ACID của giao tác

- Ví dụ: T là một giao dịch chuyển 50\$ từ tài khoản A sang tài khoản B.
  - Giao dịch này có thể được xác định như sau:

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

# Ví dụ

## ❑ 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)` ;

# Ví dụ

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

## □ 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

## ❑ 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

$T: \text{Read}(A, t);$   
 $t := t - 50;$   
 $T' \xrightarrow{\text{Write}(A, t);}$   
 $\text{Read}(B, t);$   
 $t := t + 50;$   
 $\text{Write}(B, t);$

# Ví dụ

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

## □ Durability

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

# Nội dung

- Giới thiệu
- Giao tác
- Tính chất ACID của giao tác
- Các thao tác của giao tác
- Trạng thái của giao tác

# Các thao tác của giao tác

- ❑ Giả sử CSDL gồm nhiều đơn vị dữ liệu
- ❑ Một đơn vị dữ liệu:
  - Có một giá trị
  - Được truy xuất và sửa đổi bởi các giao tác

# Các thao tác của giao tác

Các truy xuất CSDL được thực hiện bởi hai hoạt động sau:

## □ READ(X)

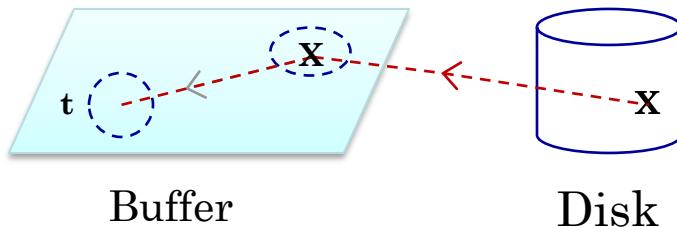
- chuyển hàng mục dữ liệu X từ CSDL đến buffer của giao dịch thực hiện hoạt động READ này

## □ WRITE(X)

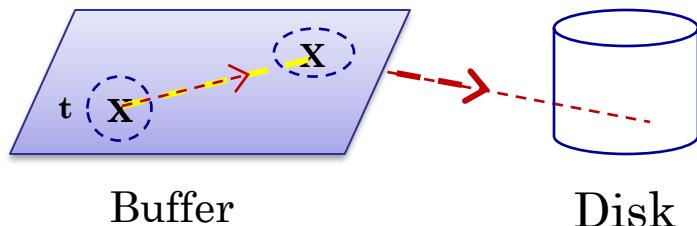
- chuyển hàng mục dữ liệu X từ buffer của giao dịch thực hiện WRITE đến CSDL

# Các thao tác của giao tác

- Input(X)
- Read(X, t)



- Write(X, t)
- Output(X)



## ❑ Buffer manager

- Input
- Output

## ❑ Transaction

- Read
- Write

# Ví dụ

- Giả sử CSDL có 2 đơn vị dữ liệu A và B với ràng buộc  $A=B$  trong mọi trạng thái nhất quán
- Giao tác T thực hiện 2 bước
  - $A:=A*2$
  - $B:=B*2$
- Biểu diễn T
  - Read(A,t) ;  $t=t*2$ ; Write(A,t);
  - Read(B,t) ;  $t=t*2$ ; Write(B,t);

# Ví dụ

Hành động	t	Mem A	Mem B	Disk A	Disk B
Read(A,t)	8	8		8	8
$t := t * 2$	16	8		8	8
Write(A,t)	16	16		8	8
Read(B,t)	8	16	8	8	8
$t := t * 2$	16	16	8	8	8
Write(B,t)	16	16	16	8	8
Output(A)	16	16	16	16	8
Output(B)	16	16	16	16	16

# Bài tập

- ❑ A = 500, B = 200
- ❑ Giao tác T thực hiện
  - A:=A-100
  - B:=B+100
- ❑ Biểu diễn quá trình xảy ra trên bộ nhớ khi thực hiện T

# Nội dung

- Giới thiệu
- Giao tác
- Tính chất ACID của giao tác
- Các thao tác của giao tác
- Trạng thái của giao tác

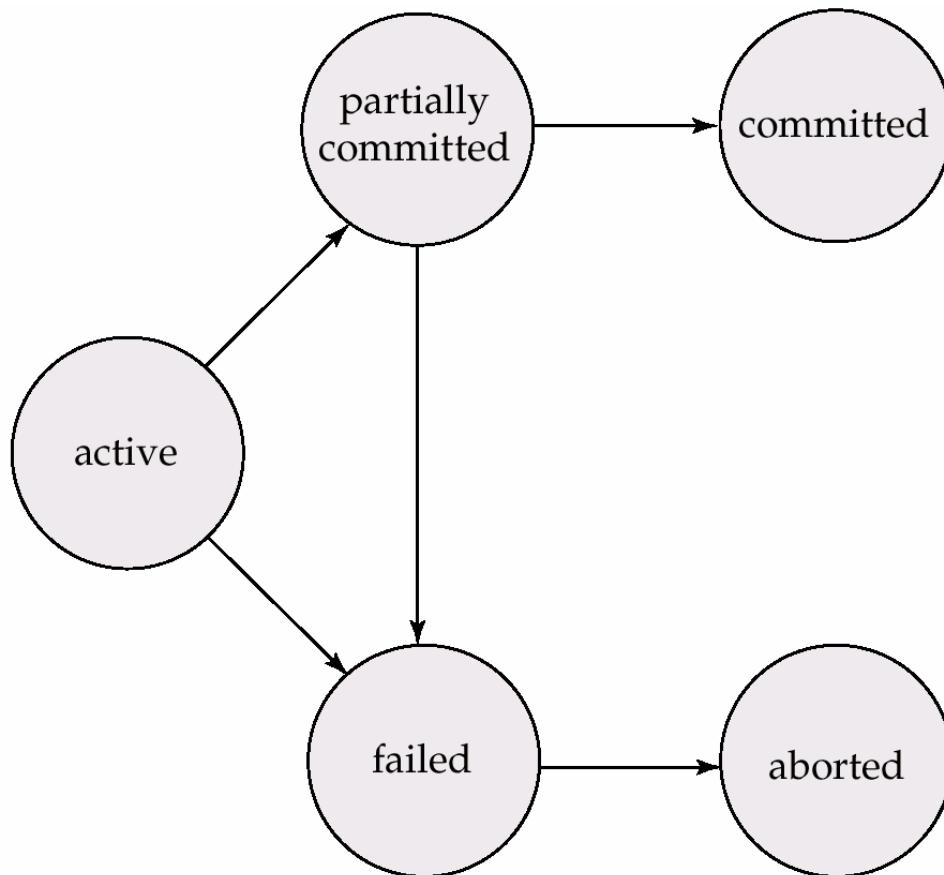
# Các trạng thái của giao tác

Một giao tác phải ở trong một trong các trạng thái sau:

- Hoạt động (Active)**
  - Ngay khi bắt đầu thực hiện thao tác đọc/ghi
- Được bàn giao bộ phận (Partially committed)**
  - Sau khi lệnh thi hành cuối cùng được thực hiện
- Thất bại (Failed)**
  - Sau khi phát hiện ra sự thực hiện không thể tiếp tục được nữa
- Bỏ dở (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
- Được bàn giao (Committed)**
  - Sau khi mọi hành động hoàn tất thành công

# Các trạng thái của giao tác

## ❑ Sơ đồ trạng thái



# **Quản lý truy xuất đồng thời**

# Nội dung

- ❑ Giới thiệu
- ❑ Lịch thao tác (schedule)
  - Lịch tuần tự (serial schedule)
  - Lịch khả tuần tự (serilizable schedule)
    - Conflict-Serializable
    - View-Serializable

# Nội dung

- ❑ Giới thiệu
- ❑ Lịch thao tác (schedule)
  - Lịch tuần tự (serial schedule)
  - Lịch khả tuần tự (serilizable schedule)

# Giới thiệu

- ❑ Thực hiện tuần tự
  - Tại một thời điểm, một giao tác chỉ có thể bắt đầu khi giao tác trước nó hoàn tất
- ❑ Thực hiện đồng thời
  - Cho phép nhiều giao tác cùng truy xuất dữ liệu
  - Gây ra nhiều phức tạp về nhất quán dữ liệu

# Lý do thực hiện đồng thời

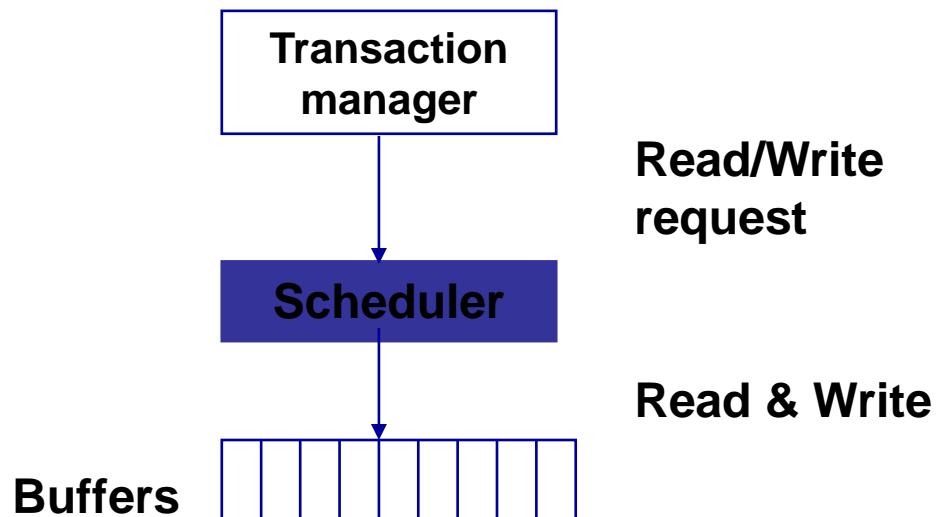
- ❑ Tận dụng tài nguyên và thông lượng
  - Trong khi 1 giao tác đang thực hiện đọc/ghi trên đĩa, 1 giao tác khác đang xử lý tính toán trên CPU
- ❑ Giảm thời gian chờ
  - Các giao tác ngắn phải chờ đợi các giao tác dài
  - Chia sẻ chu kỳ CPU và truy cập đĩa để làm giảm sự trì hoãn trong khi các giao tác thực thi

# Giới thiệu

- ❑ Khi có nhiều giao tác thực hiện đồng thời, tính nhất quán CSDL có thể bị phá vỡ mặc dù cá nhân mỗi giao tác vẫn thực hiện đúng đắn.
- ❑ Vì vậy, cần có **lịch thao tác** (schedule) để xác định chuỗi các thao tác của nhiều giao tác cạnh tranh mà vẫn đảm bảo tính nhất quán.
- ❑ Bộ phận quản lý các lịch thao tác này gọi là **Bộ lập lịch** (scheduler).

# Bộ lập lịch (scheduler)

- ❑ Là một thành phần của DBMS, có nhiệm vụ lập lịch để thực hiện nhiều giao tác xử lý đồng thời



# Nội dung

- ❑ Giới thiệu
- ❑ Lịch thao tác (schedule)
  - Lịch tuần tự (serial schedule)
  - Lịch khả tuần tự (serilizable schedule)

# Lịch thao tác (schedule)

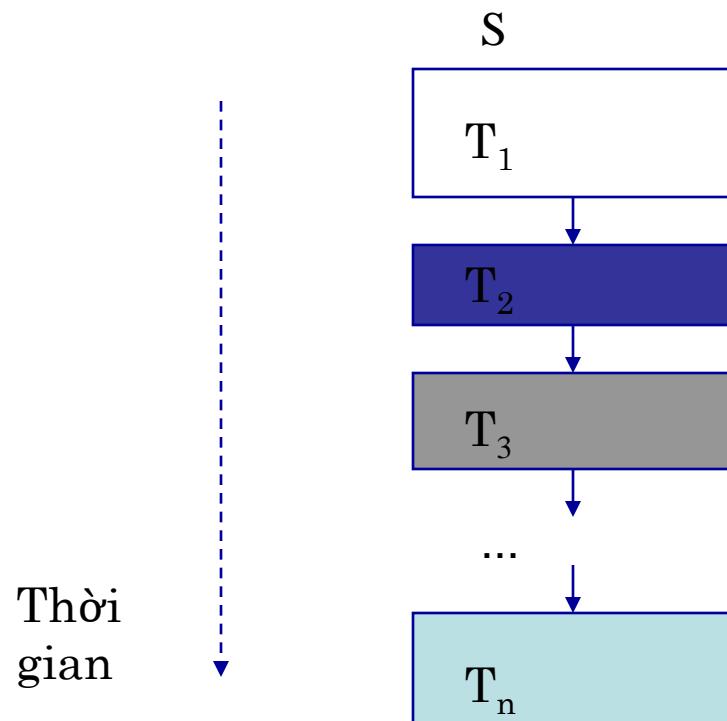
- Lịch  $S$  của  $n$  giao tác  $T_1, T_2, \dots, T_n$  là *dãy có thứ tự* các thao tác trong  $n$  giao tác này
- Thứ tự xuất hiện của các thao tác trong lịch phải giống với thứ tự xuất hiện trong giao tác
- Gồm có:
  - Lịch tuần tự (Serial)
  - Lịch khả tuần tự (Serializable)
    - Conflict-Serializability
    - View-Serializability

# Nội dung

- ❑ Giới thiệu
- ❑ Lịch thao tác (schedule)
  - Lịch tuần tự (serial schedule)
  - Lịch khả tuần tự (serilizable schedule)

# Lịch tuần tự (Serial schedule)

- ❑ Một lịch S được gọi là tuần tự nếu các hành động của các giao tác  $T_i$  ( $i=1..n$ ) được thực hiện liên tiếp nhau



# Lịch tuần tự

## ❑ Ví dụ

$T_1$	$T_2$
Read(A,t)	Read(A,s)
$t := t + 100$	$s := s * 2$
Write(A,t)	Write(A,s)
Read(B,t)	Read(B,s)
$t := t + 100$	$s := s * 2$
Write(B,t)	Write(B,s)

- ❑ Giả sử ràng buộc nhất quán trên CSDL là **A=B**
- ❑ Từng giao tác thực hiện riêng lẻ thì tính nhất quán sẽ được bảo toàn

# Lịch tuần tự

## ❑ Ví dụ

$S_1$	$T_1$	$T_2$	A	B
			25	25
	Read(A,t)			
	$t := t + 100$			
	Write(A,t)		125	
	Read(B,t)			
	$t := t + 100$			
	Write(B,t)		125	
		Read(A,s)		
		$s := s * 2$		
		Write(A,s)	250	
		Read(B,s)		
		$s := s * 2$		
		Write(B,s)	250	

# Lịch tuần tự

## ❑ Ví dụ

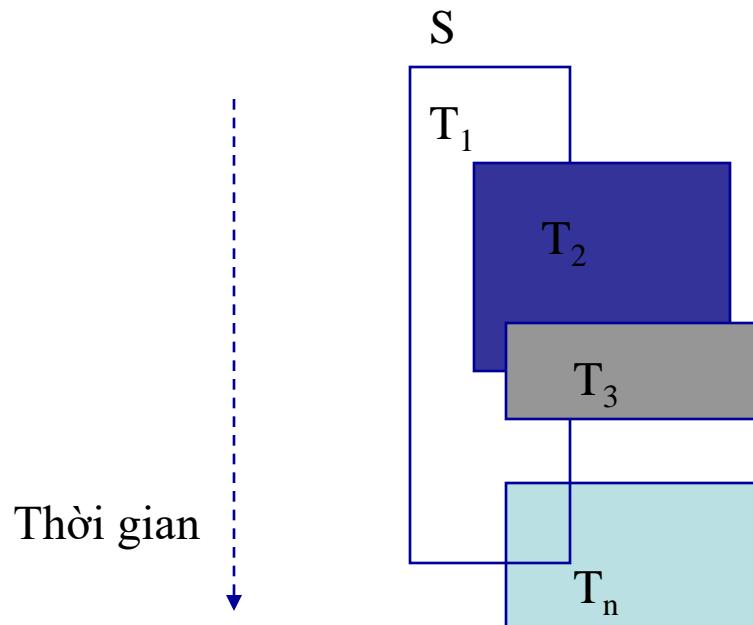
S <sub>2</sub>	T <sub>1</sub>	T <sub>2</sub>	A	B
			25	25
		Read(A,s) s:=s*2		
		Write(A,s)	50	
		Read(B,s) s:=s*2		
		Write(B,s)		50
	Read(A,t) t:=t+100			
	Write(A,t)		150	
	Read(B,t) t:=t+100			
	Write(B,t)			150

# Nội dung

- ❑ Giới thiệu
- ❑ Lịch thao tác (schedule)
  - Lịch tuần tự (serial schedule)
  - Lịch khả tuần tự (serilizable schedule)

# Lịch khả tuần tự (Serializable schedule)

- ☐ Một lịch S được lập từ n giao tác  $T_1, T_2, \dots, T_n$  xử lý đồng thời được gọi là khả tuần tự nếu nó cho cùng kết quả với 1 lịch tuần tự nào đó được lập từ n giao tác này



# Lịch khả tuần tự

S <sub>3</sub>	T <sub>1</sub>	T <sub>2</sub>	A	B
Read(A,t) t:=t+100 Write(A,t)			25	25
Read(B,t) t:=t+100 Write(B,t)		Read(A,s) s:=s*2 Write(A,s)	125	
		Read(B,s) s:=s*2 Write(B,s)	125 250	

- Trước khi S3 thực hiện
    - A=B=c (c là hằng số)
  - Sau khi S3 kết thúc
    - A=2\*(c+100)
    - B=2\*(c+100)
  - Trạng thái CSDL nhất quán
- S3 khả tuần tự

# Lịch khả tuần tự

S <sub>4</sub>	T <sub>1</sub>	T <sub>2</sub>	A	B
			25	25
Read(A,t)				
t:=t+100				
Write(A,t)			125	
		Read(A,s)		
		s:=s*2		
		Write(A,s)	250	
		Read(B,s)		
		s:=s*2		
		Write(B,s)	50	
Read(B,t)				
t:=t+100				
Write(B,t)			150	

- Trước S4 khi thực hiện
    - A=B=c (c là hằng số)
  - Sau khi S4 kết thúc
    - A = 2\*(c+100)
    - B = 2\*c + 100
  - Trạng thái CSDL không nhất quán
- S4 không khả tuần tự

# Lịch khả tuần tự

S <sub>5</sub>	T <sub>1</sub>	T <sub>2</sub>	A	B
			25	25
Read(A,t) t:=t+100 Write(A,t)				
		Read(A,s) s:=s*1 Write(A,s)	125	
		Read(B,s) s:=s*1 Write(B,s)	125	
Read(B,t) t:=t+100 Write(B,t)			25	
				125

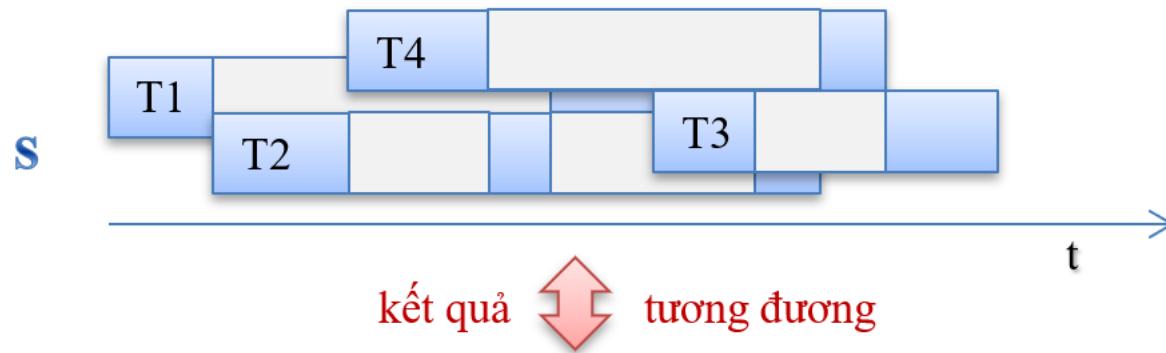
- Khi S5 kết thúc
    - A và B bằng nhau
    - Trạng thái cuối cùng nhất quán
- S5 khả tuần tự, có kết quả giống với lịch tuần tự

# Tóm tắt

## ❑ Lịch tuần tự



## ❑ Lịch khả tuần tự



# Biểu diễn lịch

## ❑ Cách 1 (đầy đủ)

S	T1	T2
	Read(A, t)	
	$t = t + 100$	
	Write(A, t)	
		Read(A, s)
		$s = s * 2$
		Write(A, s)
	Read(B, t)	
	$t = t + 100$	
	Write(B, t)	
		Read(B, s)
		$s = s * 2$
		Write(B, s)

# Biểu diễn lịch

- ❑ Cách 2 (giản lược các thao tác không đọc ghi)

S	T1	T2
	Read(A)	
	Write(A)	
		Read(A)
		Write(A)
	Read(B)	
	Write(B)	
		Read(B)
		Write(B)

# Biểu diễn lịch

## ❑ Cách 3

**S:** R1(A); W1(A); R2(A); W2(A); R1(B);  
W1(B); R2(B); W2(B);

# **Các loại lịch khả tuần tự**

# Lịch khả tuần tự

- **Conflict Serializable:** Dựa trên ý tưởng hoán vị các hành động không xung đột để chuyển một lịch đồng thời S về một lịch tuần tự S'. Nếu có một cách biến đổi như vậy thì S là một lịch Conflict Serializable.
- **View Serializable:** Dựa trên ý tưởng lịch đồng thời S và lịch tuần tự S' đọc và ghi những giá trị dữ liệu giống nhau. Nếu có một lịch S' như vậy thì S là một lịch View Serializable.

# Lịch Conflict Serializability

- Xét 2 hành động liên tiếp nhau của 2 giao tác khác nhau trong một lịch giao tác, khi 2 hành động đó được đảo thứ tự có thể dẫn đến 1 trong 2 hệ quả:
  - Hoạt động của cả hai giao tác chứa 2 hành động đó không bị ảnh hưởng gì: 2 hành động đó không xung đột với nhau.
  - Hoạt động của ít nhất một trong 2 giao tác chứa 2 hành động đó bị ảnh hưởng: 2 hành động xung đột.

# Lịch Conflict Serializability

Cho lịch S có 2 giao tác  $T_i$  và  $T_j$ , xét các trường hợp:

- $R_i(X); R_j(Y)$ 
  - Không bao giờ có xung đột, ngay cả khi  $X=Y$ .
- $R_i(X); W_j(Y)$ 
  - Không xung đột khi  $X \neq Y$  vì  $T_j$  không làm thay đổi dữ liệu đọc của  $T_i$ ,  $T_i$  không sử dụng dữ liệu ghi của  $T_j$ .
  - Xung đột khi  $X=Y$ .
- $W_i(X); R_j(Y)$ 
  - Không xung đột khi  $X \neq Y$ .
  - Xung đột khi  $X=Y$ .
- $W_i(X); W_j(Y)$ 
  - Không xung đột khi  $X \neq Y$ .
  - Xung đột khi  $X=Y$ .

# Lịch Conflict Serializable

- ❑ Hai thao tác trong lịch S có xung đột nếu chúng:
  - thuộc 2 giao tác khác nhau,
  - truy xuất đến cùng 1 đơn vị dữ liệu,
  - và trong chúng có ít nhất một hành động ghi (write)
- ❑ Hai hành động xung đột thì không thể nào đảo thứ tự của chúng trong một lịch giao tác

# Lịch Conflict Serializable

Ví dụ:

- Các thao tác xung đột:
  - R1(X), W2(X)
  - W1(X), R2(X)
  - W1(X), W2(X)
- Các thao tác không xung đột:
  - R1(X), R2(X)
  - R1(X), W1(X)

# Lịch Conflict Serializable

## Định nghĩa

- $S$  và  $S'$  là những lịch giao tác conflict equivalent nếu  $S$  có thể chuyển được thành  $S'$  thông qua một chuỗi các hoán vị những thao tác không xung đột.
- Một lịch giao tác  $S$  là conflict serializable nếu  $S$  là conflict equivalent với một lịch giao tác tuần tự nào đó.
- $S$  là conflict serializable thì  $S$  là khả tuần tự. Nhưng  $S$  là khả tuần tự thì không chắc  $S$  conflict serializable.

# Lịch Conflict Serializable

- ❑ **Ví dụ:** S và S' có conflict equivalent?

S	T1	T2
Read(A)		
Write(A)		
	Read(A)	
	Write(A)	
Read(B)		
Write(B)		
	Read(B)	
	Write(B)	

S'	T1	T2
Read(A)		
Write(A)		
	Read(B)	
	Write(B)	
		Read(A)
		Write(A)
		Read(B)
		Write(B)

S và S' là những lịch giao tác conflict equivalent.

# VD Conflict Serializable

S	T1	T2
	Read(A)	
	Write(A)	
		Read(A)
		Write(A)
	Read(B)	
	Write(B)	
		Read(B)
		Write(B)

S'	T1	T2
	Read(A)	
	Write(A)	
	Read(B)	
	Write(B)	
		Read(A)
		Write(A)
		Read(B)
		Write(B)

- Do các thao tác không xung đột nên có thể hoán chuyển chúng

S	T1	T2
	Read(A)	
	Write(A)	
		Read(A)
		<b>Write(A)</b>
	<b>Read(B)</b>	
	Write(B)	
		Read(B)
		Write(B)



S	T1	T2
	Read(A)	
	Write(A)	
		Read(A)
		<b>Read(B)</b>
		Write(A)
		Write(B)
		Read(B)
		Write(B)

# VD Conflict Serializable

S	T1	T2
	Read(A)	
	Write(A)	
		Read(A)
		Write(A)
	Read(B)	
	Write(B)	
		Read(B)
		Write(B)

S'	T1	T2
	Read(A)	
	Write(A)	
	Read(B)	
	Write(B)	
		Read(A)
		Write(A)
		Read(B)
		Write(B)

S	T1	T2
	Read(A)	
	Write(A)	
		<b>Read(A)</b>
	<b>Read(B)</b>	
		Write(A)
	Write(B)	
		Read(B)
		Write(B)



S	T1	T2
	Read(A)	
	Write(A)	
	Read(B)	
		Read(A)
		Write(A)
	Write(B)	
		Read(B)
		Write(B)

# VD Conflict Serializable

S	T1	T2
	Read(A)	
	Write(A)	
	Read(B)	
		Read(A)
		Write(A)
		Read(B)
		Write(B)

S'	T1	T2
	Read(A)	
	Write(A)	
	Read(B)	
	Write(B)	
		Read(A)
		Write(A)
		Read(B)
		Write(B)

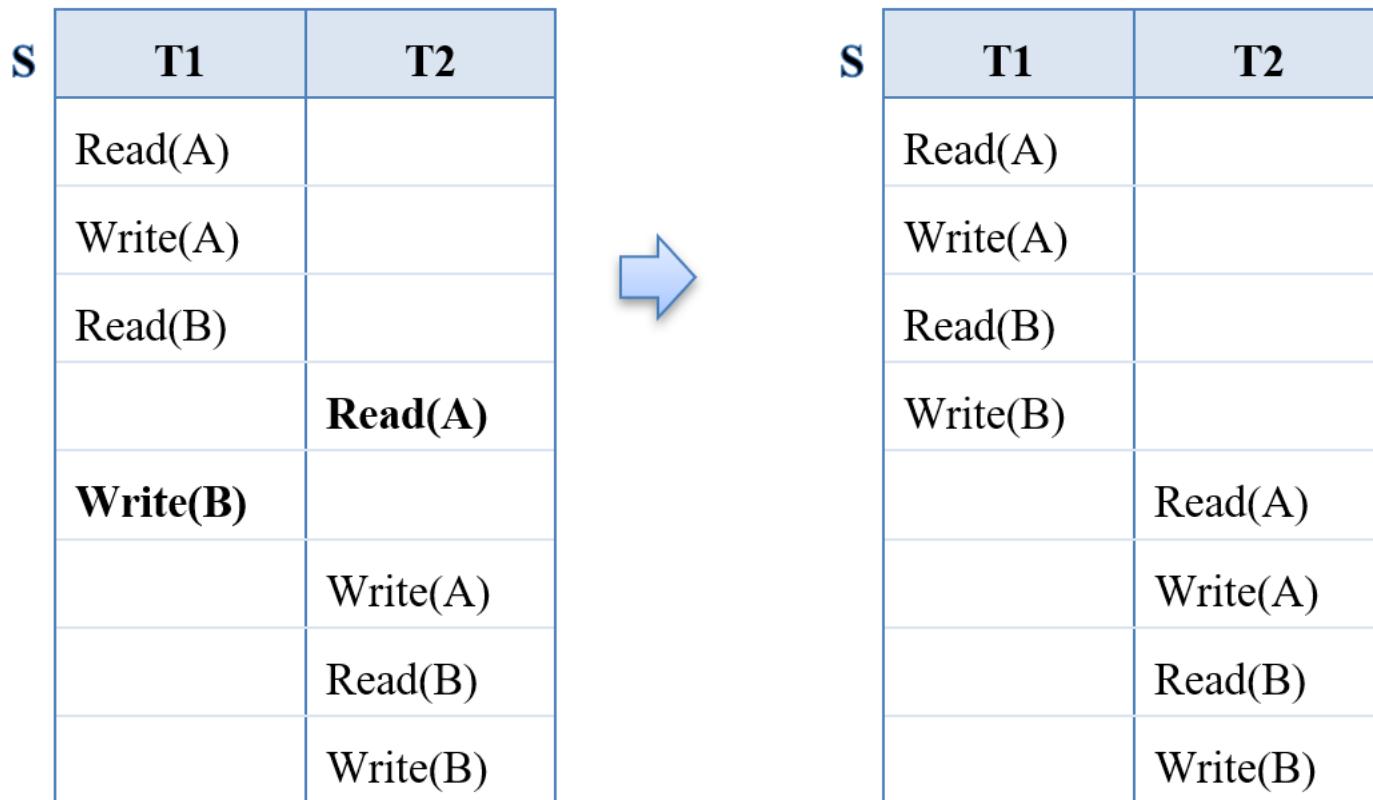
S	T1	T2
	Read(A)	
	Write(A)	
	Read(B)	
		Read(A)
		<b>Write(A)</b>
	<b>Write(B)</b>	
		Read(B)
		Write(B)



S	T1	T2
	Read(A)	
	Write(A)	
	Read(B)	
		Read(A)
		Write(B)
		Write(A)
		Read(B)
		Write(B)

# VD Conflict Serializable

S	T1	T2
	Read(A)	
	Write(A)	
	Read(B)	Read(A)
	Write(B)	Write(A)
		Read(B)
		Write(B)
		Read(A)
		Write(A)
		Read(B)
		Write(B)



# Kiểm tra lịch Conflict Serializability

- Ý tưởng: Các hành động xung đột trong lịch S được thực hiện theo thứ tự nào thì các giao tác thực hiện chúng trong S' (kết quả sau khi hoán vị) cũng theo thứ tự đó.

# Kiểm tra lịch Conflict Serializability

- Cho lịch S có 2 giao tác T1 và T2:
  - T1 thực hiện hành động A1
  - T2 thực hiện hành động A2
- Ta nói T1 thực hiện trước T2 trong lịch S, ký hiệu  $T1 <_s T2$  khi:
  - A1 thực hiện trước A2 trong S (không nhất thiết liên tiếp)
  - A1 và A2 là 2 hành động xung đột (A1 và A2 cùng thao tác lên 1 đơn vị dữ liệu và có ít nhất 1 hành động là ghi trong A1 và A2)

# Phương pháp đồ thị trình tự (Precedence Graph)

- Cho lịch  $S$  bao gồm các thao tác  $T_1, T_2, \dots, T_n$ .  
Đồ thị trình tự của  $S$ , ký hiệu  $P(S)$  có
  - đỉnh là các giao tác  $T_i$
  - cung đi từ  $T_i$  đến  $T_j$  nếu  $T_i <_S T_j$ .
- $S$  conflict serializable khi và chỉ khi  $P(S)$  không có chu trình.
  - Thứ tự hình học các đỉnh là thứ tự của các giao tác trong lịch tuần tự tương đương với  $S$ .

# Phương pháp đồ thị trình tự (Precedence Graph)

## □ Bổ đề:

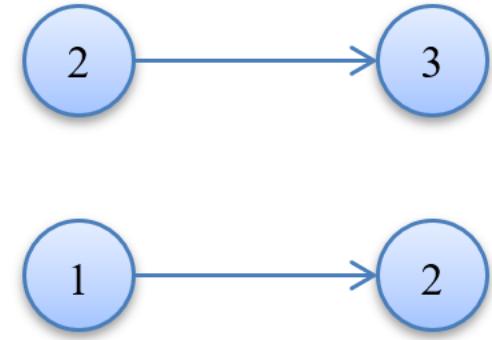
- Với 2 lịch  $S$  và  $S'$  được lập từ cùng các giao tác,
- $S$  và  $S'$  conflict equivalent  $\Rightarrow P(S) = P(S')$ .

# Đồ thị trình tự - ví dụ

S1	T1	T2	T3
		Read(A)	
	Read(B)		
		Write(A)	
			Read(A)
	Write(B)		
			Write(A)
		Read(B)	
			Write(B)

# Đồ thị trình tự - ví dụ

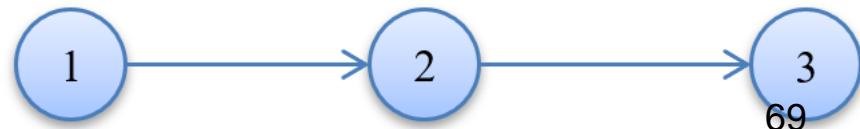
S1	T1	T2	T3
		Read(A)	
Read(B)			
	Write(A)		
		Read(A)	
Write(B)			
		Write(A)	
	Read(B)		
	Write(B)		



# Đồ thị trình tự - ví dụ

S1	T1	T2	T3
		Read(A)	
Read(B)			
	Write(A)		
		Read(A)	
Write(B)			
		Write(A)	
	Read(B)		
	Write(B)		

- P(S1) không có chu trình
- S1 conflict serializable theo thứ tự T1, T2, T3



## Đồ thị trình tự - Ví dụ 2

- Cho S2: R2(A) R1(B) W2(A) R2(B) R3(A) W1(B)  
W3(A) W2(B).
- S2 có Conflict Serializability hay không?

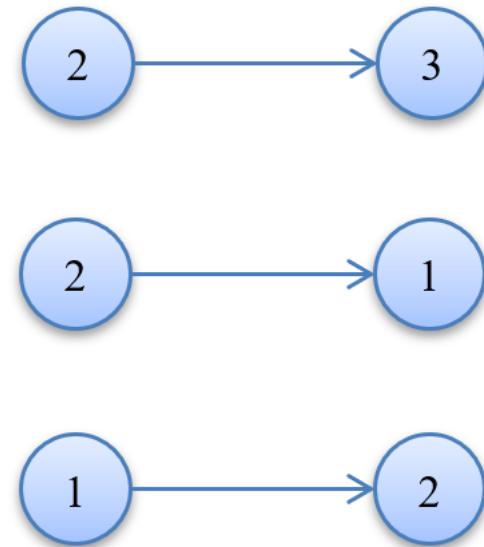
# Đồ thị trình tự - Ví dụ 2

- R2(A) R1(B) W2(A) R2(B) R3(A) W1(B) W3(A) W2(B)

S2	T1	T2	T3
		Read(A)	
Read(B)			
	Write(A)		
	Read(B)		
		Read(A)	
Write(B)			
		Write(A)	
	Read(B)		

# Đồ thị trình tự - Ví dụ 2

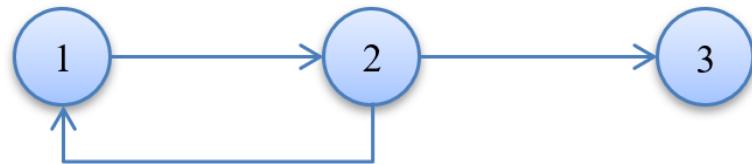
S2	T1	T2	T3
		Read(A)	
Read(B)			
		Write(A)	
		Read(B)	
			Read(A)
Write(B)			
			Write(A)
		Read(B)	



# Đồ thị trình tự - Ví dụ 2

S2	T1	T2	T3
		Read(A)	
Read(B)			
	Write(A)		
	Read(B)		
		Read(A)	
Write(B)			
		Write(A)	
	Read(B)		

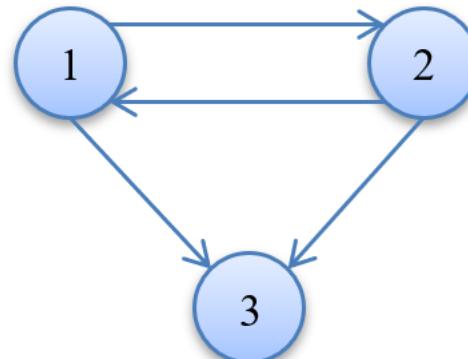
- P(S2) có chu trình, nên S2 không conflict serializable



# Lịch View Serializability

## ❑ Xét lịch S

S	T1	T2	T3
Read(A)			
		Write(A)	
Write(A)			
			Write(A)



- ❑ Đồ thị trình tự P(S) có chu trình, S không conflict  
serializable, S có khả tuần tự hay không?

# Lịch View Serializability

*Lịch S không conflict serializable*

S	T1	T2	T3
Read(A)			
	Write(A)		
Write(A)			
		Write(A)	

*Lịch S' serial*

S'	T1	T2	T3
Read(A)			
Write(A)			
		Write(A)	
			Write(A)

- So sánh lịch S và 1 lịch tuần tự S': Giả sử
  - trước khi lịch S thực hiện, có giao tác Tb thực hiện việc ghi A
  - và sau khi S thực hiện có giao tác Tf thực hiện việc đọc A.

# Lịch View Serializability

*Lịch S không conflict serializable*

S	T1	T2	T3
Read(A)			
	Write(A)		
Write(A)			
		Write(A)	

*Lịch S' serial*

S'	T1	T2	T3
Read(A)			
Write(A)			
		Write(A)	
			Write(A)

## ☐ Nhận xét lịch S và S':

- Đều có T1 thực hiện Read(A) từ giao tác Tb → Kết quả đọc luôn giống nhau.
- Đều có T3 thực hiện việc ghi cuối cùng lên A. T2, T3 không có lệnh đọc A → Dù S hay S' được thực hiện thì kết quả đọc A của Tf luôn giống nhau.

# Lịch View Serializability

*Lịch S không conflict serializable*

S	T1	T2	T3
Read(A)			
	Write(A)		
Write(A)			
		Write(A)	

*Lịch S' serial*

S'	T1	T2	T3
Read(A)			
Write(A)			
		Write(A)	
			Write(A)

- → Kết quả của S và S' giống nhau  
→ S vẫn khả tuần tự

# Lịch View Serializability

- Một lịch S được gọi là khả tuần tự view nếu tồn tại một lịch tuần tự S' được tạo từ các giao tác của S sao cho S và S' đọc và ghi những giá trị giống nhau.
- Lịch S được gọi là khả tuần tự view khi và chỉ khi nó tương đương view (view-equivalent) với một lịch tuần tự S'.

# Lịch View Serializability

**Ví dụ:** Cho lịch S và S' như sau

- S: R2(B) W2(A) R1(A) R3(A) W1(B) W2(B) W3(B)
- S': R2(B) W2(A) W2(B) R1(A) W1(B) R3(A) W3(B)

# Lịch View Serializability

S	T1	T2	T3
		Read(B)	
	Read(A)	Write(A)	
	Write(B)		Read(A)
		Write(B)	
			Write(B)

# Lịch View Serializability

S'	T1	T2	T3
		Read(B)	
		Write(A)	
		Write(B)	
	Read(A)		
	Write(B)		
			Read(A)
			Write(B)

➔ S khả tuần tự view

# View equivalent

- $S$  và  $S'$  là những lịch tương đương view (view equivalent) nếu thỏa các điều kiện sau:
  - Nếu trong  $S$  có  $T_i$  đọc giá trị ban đầu của  $A$  thì nó cũng đọc giá trị ban đầu của  $A$  trong  $S$ .
  - Nếu  $T_i$  đọc giá trị của  $A$  được ghi bởi  $T_j$  trong  $S$  thì  $T_i$  cũng phải đọc giá trị của  $A$  được ghi bởi  $T_j$  trong  $S'$ .
  - Với mỗi đơn vị dữ liệu  $A$ , giao tác thực hiện lệnh ghi cuối cùng lên  $A$  (nếu có) trong  $S$  thì giao tác đó cũng phải thực hiện lệnh ghi cuối cùng lên  $A$  trong  $S'$ .

# View equivalent

- ❑ Một lịch giao tác S là view serializable nếu S là view equivalent với một lịch giao tác tuần tự S' nào đó.
- ❑ Nếu S là conflict serializable thì S view serializable (suy chiềng ngược lại thì không đúng).

# Kiểm tra lịch View Serializability

- Cho một lịch giao tác S. Thêm 1 giao tác cuối Tf vào trong S sao cho Tf thực hiện việc **đọc hết tất cả** đơn vị dữ liệu ở trong S:

S: ... W1(A) ..... W2(A) **Rf(A)**

- Thêm 1 giao tác đầu tiên Tb vào trong S sao cho Tb thực hiện việc **ghi các giá trị ban đầu** cho tất cả đơn vị dữ liệu ở trong S:

S: **Wb(A)** ... W1(A) ..... W2(A) ...

# Kiểm tra lịch View Serializability

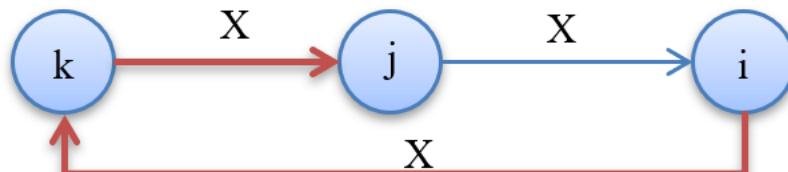
Đồ thị phức (PolyGraph) cho S, ký hiệu G(S), với

- Đỉnh là các giao tác Ti (bao gồm cả Tb và Tf), và
- Cung đi từ Tj đến Ti nếu giá trị mà Ri(X) đọc được là do Tj ghi (Ri(X) có gốc là Tj)

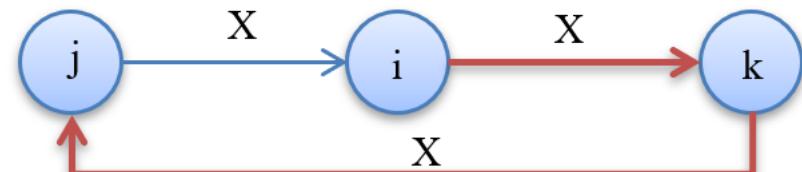
# Kiểm tra lịch View Serializability

- ❑ Với mỗi  $Wj(X) \dots Ri(X)$ , xét  $Wk(X)$  khác  $Tb$  sao cho  $Tk$  không chèn vào giữa  $Tj$  và  $Ti$ :
  - nếu  $Tj \neq Tb$  và  $Ti \neq Tf$  thì vẽ cung  $Tk \rightarrow Tj$  và  $Ti \rightarrow Tk$

Tk	Tj	Ti
Write(X)	Write(X)	Read(X)



Tj	Ti	Tk
Write(X)	Read(X)	Write(X)



(Tk có thể nằm trước Tj hoặc sau Ti)

# Kiểm tra lịch View Serializability

- nếu  $T_j = T_b$  thì vẽ cung  $T_i \rightarrow T_k$

$T_j = T_b$	$T_i$	$T_k$
Write(X)		
	Read(X)	
		<b>Write(X)</b>



- nếu  $T_i = T_f$  thì vẽ cung  $T_k \rightarrow T_j$

$T_k$	$T_j$	$T_i = T_f$
<b>Write(X)</b>		
	Write(X)	
		Read(X)



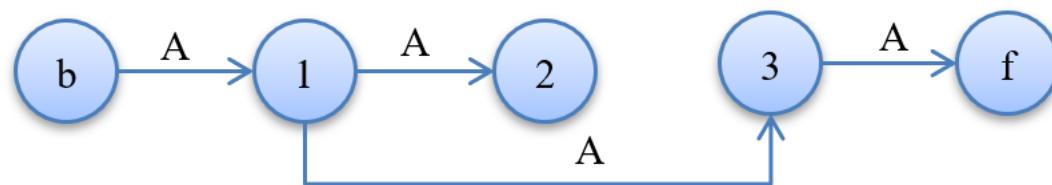
# Ví dụ 1

S	T1	T2	T3
Read(A)			
	Write(A)		
Write(A)			
		Write(A)	

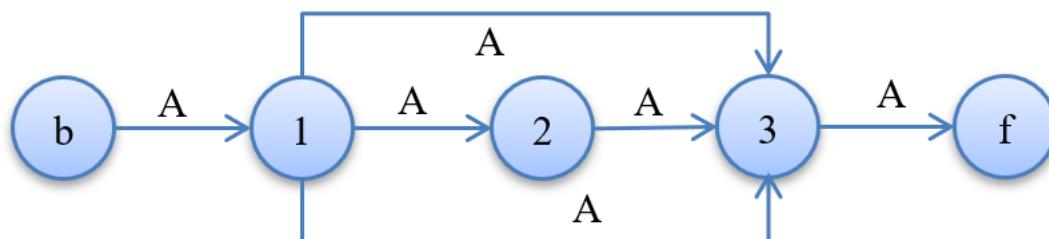
<b>T<sub>b</sub></b>	<b>T<sub>1</sub></b>	<b>T<sub>2</sub></b>	<b>T<sub>3</sub></b>	<b>T<sub>f</sub></b>
	<i>Write(A)</i>			
	<i>Read(A)</i>			
		Write(A)		
	Write(A)			
			<i>Write(A)</i>	
			<i>Read(A)</i>	



<b>Tb</b>	<b>T1</b>	<b>T2</b>	<b>T3</b>	<b>Tf</b>
<i>Write(A)</i>				
	<i>Read(A)</i>			
		<i>Write(A)</i>		
	<i>Write(A)</i>		<i>Write(A)</i>	
				<i>Read(A)</i>



<b>T<sub>b</sub></b>	<b>T<sub>1</sub></b>	<b>T<sub>2</sub></b>	<b>T<sub>3</sub></b>	<b>T<sub>f</sub></b>
Write(A)				
	Read(A)			
		Write(A)		
	Write(A)			
			Write(A)	
				Read(A)



G(S) không có chu trình

S view serializable theo thứ tự T<sub>b</sub>, T<sub>1</sub>, T<sub>2</sub>, T<sub>3</sub>, T<sub>f</sub><sup>81</sup>

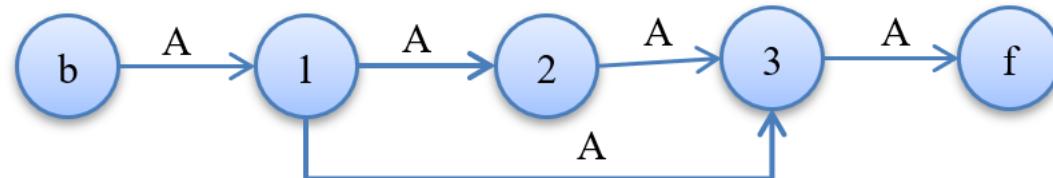
# Ví dụ 2

S	T1	T2	T3
	Read(A)		
		Write(A)	
			Read(A)
	Write(A)		
			Write(A)

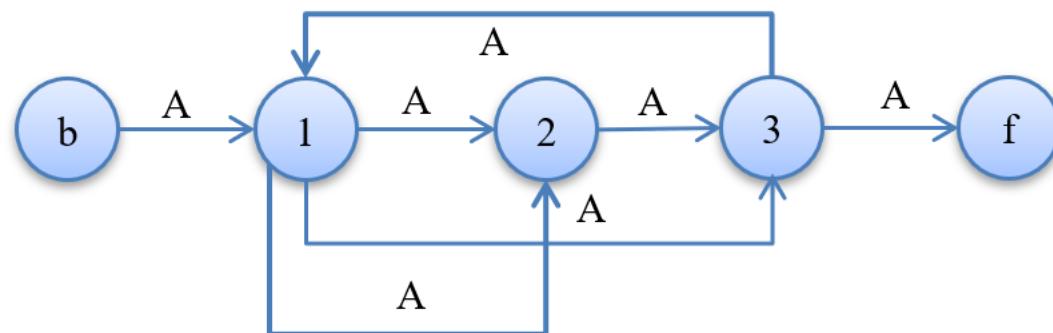
<b>S'</b>	<b>T<sub>b</sub></b>	<b>T<sub>1</sub></b>	<b>T<sub>2</sub></b>	<b>T<sub>3</sub></b>	<b>T<sub>f</sub></b>
	<i>Write(A)</i>				
		<i>Read(A)</i>			
			<i>Write(A)</i>		
				<i>Read(A)</i>	
		<i>Write(A)</i>			
				<i>Write(A)</i>	
					<i>Read(A)</i>



<b>T<sub>b</sub></b>	<b>T<sub>1</sub></b>	<b>T<sub>2</sub></b>	<b>T<sub>3</sub></b>	<b>T<sub>f</sub></b>
<i>Write(A)</i>				
	<i>Read(A)</i>			
		<i>Write(A)</i>		
			<i>Read(A)</i>	
	<i>Write(A)</i>			
			<i>Write(A)</i>	
				<i>Read(A)</i>

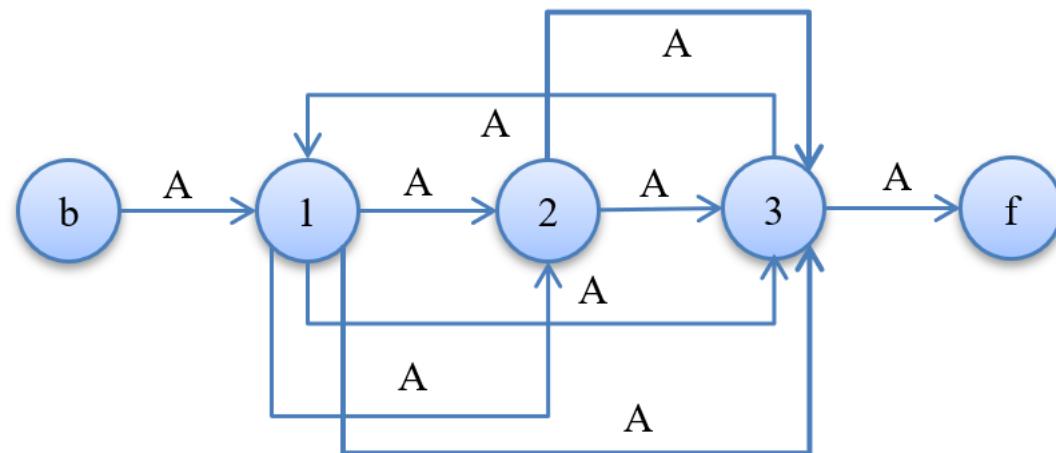


<b>T<sub>b</sub></b>	<b>T<sub>1</sub></b>	<b>T<sub>2</sub></b>	<b>T<sub>3</sub></b>	<b>T<sub>f</sub></b>
Write(A)				
	Read(A)			
		Write(A)		
			Read(A)	
	Write(A)			
			Write(A)	
				Read(A)

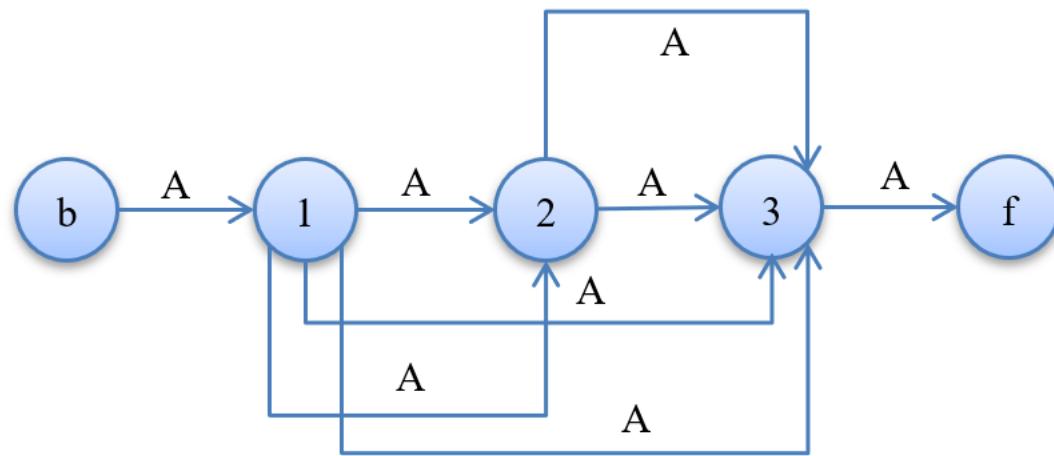


Chọn 1 trong 2 cung sao cho không có chu trình

S'	Tb	T1	T2	T3	Tf
Write(A)					
	Read(A)				
		Write(A)			
			Read(A)		
	Write(A)				
			Write(A)		
				Read(A)	



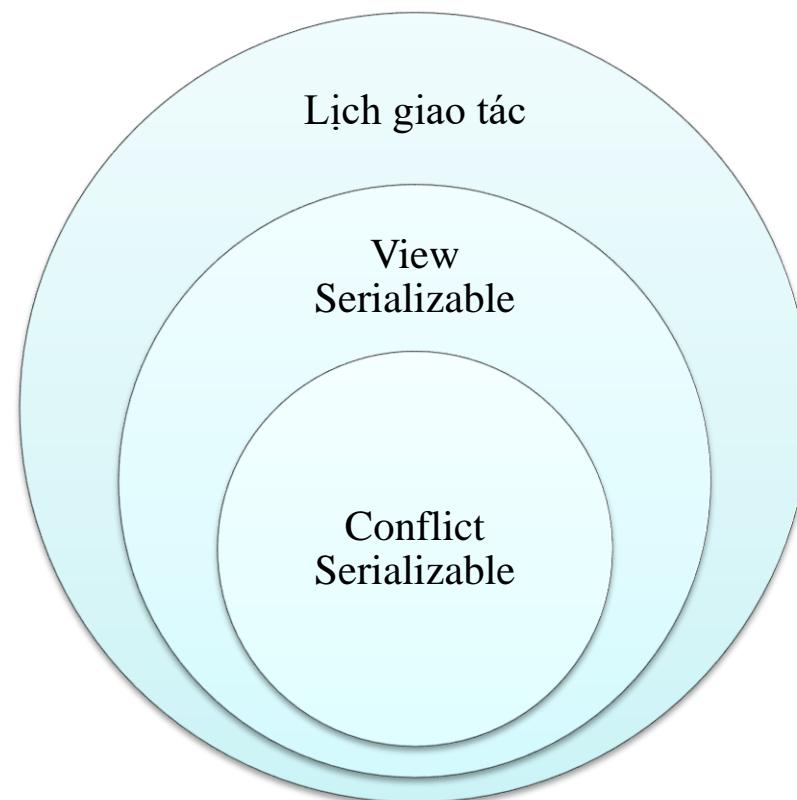
S'	T <sub>b</sub>	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>f</sub>
	Write(A)				
		Read(A)			
			Write(A)		
				Read(A)	
		Write(A)			
				Write(A)	
					Read(A)



G(S)

không có chu trình sau khi bỏ cung  $3 \rightarrow 1$

S view  
serializable.



# Bài tập Precedence graph

- Cho các lịch S sau:
  1. w1(A) r2(A) r3(A) w4 (A)
  2. w3(A) w2(C) r1(A) w1(B) r1(C) w2(A) r4(A) w4(D)
  3. r1(A) w2(A) w1(A) w3(A)
  4. r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)
- Vẽ P(S)
- S có conflict-serializable không? Nếu có thì S tương đương với lịch tuần tự nào?

# Bài tập Precedence graph

- Cho các lịch S sau:
  5. w1(X) w2(Y) w2(X) w1(X) w3(X)
  6. r2(A) r1(B) w2(A) r3(A) w1(B) r2(B) w2(B)
  7. r2(A) r1(B) w2(A) r2(B) r3(A) w1(B) w3(A) w2(B)
- Vẽ P(S)
- S có conflict-serializable không? Nếu có thì S tương đương với lịch tuần tự nào?

# Bài tập Poly graph

❑ Cho lịch S:

1. r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)
2. w1(A) r3(A) r2(A) w2(A) r1(A) w3(A)
3. r2(A) r1(A) w1(C) r3(C) w1(B) r4(B) w3(A) r4(C)  
w2(D) r2(B) w4(A) w4(B)
4. w1(A) r2(A) w2(A) r1(A)

❑ Vẽ  $\tilde{G}(S)$

❑ S có view-serializable hay không ?

# Bài tập Poly graph

❑ Cho lịch S:

5. r1(A) r3(D) w1(B) r2(B) w3(B) r4(B) w2(C) r5(C)  
w4(E) r5(E) w5(B)
6. w1(A) r2(A) w3(A) r4(A) w5(A) r6(A)
7. r1(X) r2(X) w1(X) w2(X)

❑ Vẽ G(S)

❑ S có view-serializable hay không ?

# Nhận xét

- ❑ Các lịch trình **tuần tự** kém hiệu quả do không cho phép giao tác thi hành xen kẽ
- ❑ Các lịch trình **không khả tuần tự** tiềm tàng khả năng gây các vấn đề bất thường
- ❑ Các lịch trình **khả tuần tự** cho phép giao tác thi hành xen kẽ và cho kết quả đúng

# Nhận xét

Tuy nhiên, việc kiểm tra tính khả tuần tự của một lịch trình thực tế rất khó khăn, do các nguyên nhân:

- ❑ Các giao tác, khi thi hành thường thể hiện dưới dạng những tiến trình của HĐH, thường được bản thân HĐH điều phối. HQTCSDL thực tế không điều phối được các thao tác trong giao tác.
- ❑ Các giao tác được gửi tới HQTCSDL liên tục, khó xác định điểm bắt đầu và kết thúc giao tác.  
→ Các HQTCSDL thường sử dụng những quy tắc để đảm bảo tính khả tuần tự của lịch trình.

# Câu hỏi

- Xét 2 lịch sau có tương đương về mặt kết quả không?
- Giả sử  $X = 20, Y = 30$

T1	T2	T1	T2
$R1(X)$ $X = X-10$	$R2(X)$ $X = X+5$	$R1(X)$ $X = X-10$ $W1(X)$	$R2(X)$ $X = X+5$ $W2(X)$

# Bài tập:

1. Lịch nào sau đây là khả tuần tự? Với mỗi lịch là không khả tuần tự, xác định 1 lịch tuần tự tương ứng

- a) R1(X),R3(X), W1(X), R2(X),W3(X)
- b) R1(X),R3(X), W3(X), W1(X),R2(X)
- c) R3(X),R2(X),W3(X),R1(X),W1(X)
- d) R3(X),R2(X),R1(X),W3(X),W1(X)

# Bài tập

2. Cho:

S1: R1(X), R2(Z), R1(Z), R3(X), R3(Y), W1(X),  
W3(Y), R2(Y), W2(Z), W2(Y)

S2: R1(X), R2(Z), R3(x), R1(Z), R2(Y), R3(Y),  
W1(X), W2(Z), W3(Y), W2(Y)

S1, S2 có khả tuần tự không? Viết 1 lịch tuần tự.

# **HƯỚNG DẪN THỰC HÀNH**

# Giao tác

- ❑ Một giao tác thường là kết quả của việc thực hiện một chương trình người dùng được viết trong một ngôn ngữ cấp cao hay ngôn ngữ SQL và được phân cách bởi các câu lệnh có dạng:

**begin transaction**

...

**end transaction**

## Phân loại

- ❑ Giao tác ngầm định (Implicit transaction)
- ❑ Giao tác tường minh (Explicit transaction)
- ❑ Giao tác xác nhận (Commit transaction)

# Giao tác tường minh (Explicit)

## Giao tác

Begin tran [tên\_giao\_tác]

lệnh | khối\_lệnh

{ Commit tran | Rollback tran } [tên\_giao\_tác]

## Tạo điểm lưu

save tran tên\_điểm\_lưu

Hủy những gì sau điểm lưu nếu rollback.

# Điểm lưu (save point)

begin tran t1

lệnh | khối\_lệnh

save tran s1

lệnh | khối\_lệnh

rollback tran s1

=> chưa chấm dứt t1

lệnh | khối\_lệnh

commit tran t1

=> Rollback tran s1 chỉ hủy bỏ kết quả sau lệnh save tran s1 và tiếp tục làm tiếp (giao tác t1 vẫn còn).

# GT không tường minh (implicit)

- ❑ Bắt đầu giao tác với các lệnh

ALTER TABLE, DROP, TRUNCATE TABLE,

CREATE, OPEN, FETCH, REVOKE, GRANT

DELETE, INSERT, SELECT, UPDATE

- ❑ Kết thúc bằng lệnh : commit | rollback tran

- ❑ Khi kết thúc cũng là lúc bắt đầu một giao tác mới.

- ❑ Thiết lập thông số chấp nhận

SET IMPLICIT\_TRANSACTIONS ON|OFF

# Giao tác tự động (autocommit)

- Cơ chế tự động xác nhận được thực thi khi trong giao tác xuất hiện lỗi lúc chạy hay lỗi cú pháp.
  - Lỗi cú pháp == giao tác bị hủy (rollback)
  - Lỗi lúc chạy (khóa chính, sai dữ liệu,...)  
== giao tác được chấp nhận đến thời điểm đó.

# Giao tác lồng nhau

- ❑ Cho phép các giao tác lồng với nhau.
- ❑ Lệnh `commit` chỉ có tác dụng cho giao tác cấp ‘con’ gần nhất.
- ❑ Lệnh `rollback tran` có tác dụng hủy tất cả và trở về điểm ban đầu của giao tác cấp ‘cha’ nhất.
- ❑ Biến `@@trancount` chỉ xem vào thời điểm hiện tại có bao nhiêu giao tác đang tồn tại.

# Giao tác lồng nhau

Begin tran t1

.....

begin tran t2

.....

print @@trancount => kết quả là 2

commit tran t2

.....

print @@trancount => kết quả là 1

commit tran t1