
CHƯƠNG 6:

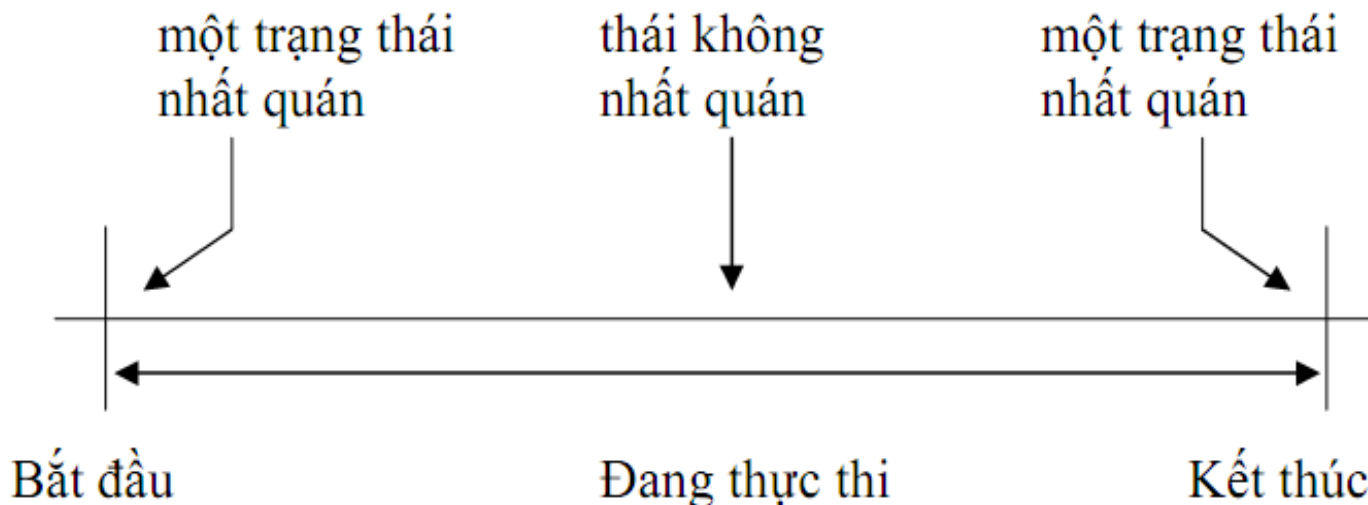
QUẢN LÝ GIAO TÁC VÀ ĐIỀU KHIỂN ĐỒNG THỜI PHÂN TÁN

MỤC ĐÍCH

- ❖ **Nhằm quản lý một số vấn đề trong quá trình truyền thông của hệ phân tán như:**
 - *Nhất quán (consistency) hoặc đáng tin cậy (reliability)*
 - *Điều khiển đồng thời (concurrency control)*
 - *Hiệu quả sử dụng các tài nguyên của hệ thống.*
- ❖ **Hiểu được sự liên quan giữa điều khiển đồng thời, cơ chế phục hồi và cấu trúc của hệ thống phân tán.**

MỤC ĐÍCH

- ❖ Phân biệt giữa nhất quán cơ sở dữ liệu (database consistency) và nhất quán giao tác (transaction consistency).
- ❖ Một **cơ sở dữ liệu ở trong một trạng thái nhất quán** (consistent state) nếu nó tuân theo tất cả các ràng buộc toàn vẹn (nhất quán) được định nghĩa trên nó.
- ❖ Cơ sở dữ liệu có thể tạm thời không nhất quán trong khi thực hiện giao tác. Điều quan trọng là cơ sở dữ liệu phải trở về trạng thái nhất quán khi quan hệ giao tác chấm dứt.



MỤC ĐÍCH

❖ Tính **nhất quán giao tác** là hành động của các giao tác đồng thời. Chúng ta mong rằng cơ sở dữ liệu vẫn nhất quán ngay cả khi có một số yêu cầu của người sử dụng đồng thời truy xuất đến cơ sở dữ liệu (đọc hoặc cập nhật).

❖ Tính chất phức tạp nảy sinh khi xét đến các **cơ sở dữ liệu có nhân bản**. Một cơ sở dữ liệu được nhân bản ở trong một trạng thái nhất quán lẫn nhau (mutually consistent state) nếu tất cả các bản sao của mỗi mục dữ liệu ở trong đó đều có giá trị giống nhau. Điều này thường được gọi là sự tương đương một bản (one copy equivalence) vì tất cả các bản đều bị buộc phải nhận cùng một trạng thái vào cuối lúc thực thi giao tác.

MỤC ĐÍCH

- ❖ **Độ tin cậy (reliability)** là khả năng tự thích ứng (resiliency) của một hệ thống đối với các loại sự cố và khả năng khôi phục lại từ những sự cố này.
- ❖ Một **hệ thống đáng tin cậy** sẽ tự thích ứng với các sự cố hệ thống và có thể tiếp tục cung cấp các dịch vụ ngay cả khi xảy ra sự cố.
- ❖ Một hệ quản trị cơ sở dữ liệu **khả hồi phục** là hệ quản trị cơ sở dữ liệu có thể chuyển sang trạng thái nhất quán (bằng cách quay trở lại trạng thái nhất quán trước đó hoặc chuyển sang một trạng thái nhất quán mới) sau khi gặp một sự cố.

NỘI DUNG

1. QUẢN LÝ GIAO TÁC PHÂN TÁN

2. ĐIỀU KHIỂN ĐỒNG THỜI PHÂN TÁN

NỘI DUNG

1. QUẢN LÝ GIAO TÁC PHÂN TÁN

1.1. KHÁI NIỆM GIAO TÁC

1.2. CÁC TÍNH CHẤT CỦA GIAO TÁC

1.3. CÁC LOẠI GIAO TÁC

1.1. KHÁI NIỆM GIAO TÁC

❖ **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

1.1. KHÁI NIỆM GIAO TÁC

Giao tác phân tán?

Giao tác là một lần thực hiện của một chương trình.

Chương trình có thể là:

- Một câu truy vấn
- Một chương trình ngôn ngữ chủ với các lời gọi được gắn vào một ngôn ngữ vấn tin.

Ví dụ:

(T1): Begin

```
read(a);  
a:=a+100;  
read(a); a:=a+2;  
write(a);
```

end

1.1. KHÁI NIỆM GIAO TÁC

Hai thao tác cơ sở:

- Đọc dữ liệu từ CSDL : `read(x)`
- Ghi dữ liệu vào CSDL: `write(x)`

Định nghĩa giao tác:

Giao tác được xem như một dãy các thao tác đọc và ghi trên cơ sở dữ liệu cùng với các bước tính toán cần thiết.

Chú ý:

- Khi đọc hoặc ghi dữ liệu vào cơ sở dữ liệu các giao tác sẽ sử dụng một không gian làm việc riêng (private workspace) để thực hiện các thao tác tính toán.
- Các thao tác tính toán này sẽ không ảnh hưởng đến cơ sở dữ liệu.

1.1. KHÁI NIỆM GIAO TÁC

Ví dụ: Xét 2 giao tác T1 và T2:

(T1) :Begin

```
    read(a) ;  
    a:=a+100 ;  
    read(a) ;  
    a:=a+2 ;  
    write(a)
```

end

(T2) :Begin

```
    read(a) ;  
    a:=a+100 ;  
    write(a) ;  
    read(a) ;  
    a:=a+2 ;  
    write(a)
```

end

Nhận xét:

- Ở giao tác T1 giá trị của biến **a** chỉ được tăng lên 2 vì lệnh **a:=a+100** được thực hiện trong không gian riêng mà không ảnh hưởng đến cơ sở dữ liệu.
- Ở giao tác T2 giá trị của biến **a** chỉ được tăng thêm 102.

1.2. CÁC TÍNH CHẤT CỦA GIAO TÁC

- Tính nguyên tử (**A**tomicity)
- Nhất quán (**C**onsistency)
- Tính cô lập (**I**solation)
- Tính bền vững (**D**urability)

1.2. CÁC TÍNH CHẤT CỦA GIAO TÁC

❖ Tính nguyên tử (Atomicity)

- Tính nguyên tử của một giao tác là sự thực hiện trọn vẹn mà không một giao tác nào được chen vào.
- Khi thực thi một giao tác thì hoặc là các hành động của giao tác đó được thực hiện hoặc là không một hành động nào được thực hiện cả.
- Tính nguyên tử đòi hỏi rằng nếu việc thực thi giao tác bị cắt ngang bởi một loại sự cố nào đó thì DBMS sẽ chịu trách nhiệm xác định những công việc của giao tác để khôi phục lại sau sự cố.
- Có 2 chiều hướng thực hiện:
 - hoặc nó sẽ được kết thúc bằng cách hoàn tất các hành động còn lại,
 - hoặc có thể kết thúc bằng cách hồi lại tất cả các hành động đã được thực hiện.

1.2. CÁC TÍNH CHẤT CỦA GIAO TÁC

02 lý do cơ bản làm cho giao tác không hoàn thành:

- Giao tác tự hủy bỏ (**transaction aborts**)
- Hệ thống bị sự cố (**system crashes**)

Tại sao giao tác tự hủy?

- Do yêu cầu của bản thân giao tác hoặc của người sử dụng nó.
- Do sự ép buộc của hệ thống:
 - Quá tải hệ thống
 - Khóa chết (deadlock).

1.2. CÁC TÍNH CHẤT CỦA GIAO TÁC

Một số khái niệm:

- **Khôi phục giao tác (transaction recovery)**: duy trì được tính nguyên tử khi có sự cố mà giao tác tự huỷ bỏ.

- **Khắc phục sự cố (crash recovery)**.

duy trì được tính nguyên tử khi có sự cố hệ thống

- **Uỷ thác (commitment)**: Sự hoàn thành một giao tác

Begin_Transaction



Commit

Begin_Transaction



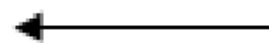
Abort

Begin_Transaction



X

System
Forces
abort



1.2. CÁC TÍNH CHẤT CỦA GIAO TÁC

❖ Tính nhất quán (Consistency)

- Tính nhất quán của một giao tác chỉ đơn giản là tính đúng đắn của nó.
- Nói cách khác, một giao tác là một chương trình đúng đắn, ánh xạ cơ sở dữ liệu từ trạng thái nhất quán này sang một trạng thái nhất quán khác.
- Dữ liệu rác (dirty data) muốn nói đến những giá trị dữ liệu đã được cập nhật bởi một giao tác trước khi nó ủy thác.

1.2. CÁC TÍNH CHẤT CỦA GIAO TÁC

❖ Tính cô lập (Isolation)

Mục đích:

Tính chất này để ngăn ngừa sự *hủy bỏ dây chuyền* (cascading abort - Còn gọi là hiệu ứng domino).

Chú ý:

- Một giao tác đang thực thi không thể đưa ra các kết quả của nó cho những giao tác khác đang cùng hoạt động trước khi nó uỷ thác.
- Nếu một giao tác cho phép những giao tác khác sử dụng những kết quả chưa hoàn tất của mình trước khi uỷ thác, rồi sau đó nó quyết định huỷ bỏ, thì mọi giao tác đã đọc những giá trị chưa hoàn tất đó cũng sẽ phải được huỷ bỏ nếu không khâu mắt xích này dễ dàng tăng nhanh và gây ra những phí tổn đáng kể cho DDBMS.

1.2. CÁC TÍNH CHẤT CỦA GIAO TÁC

Ví dụ: Xét 2 giao tác đồng thời T_1 và T_2 cùng truy xuất đến mục dữ liệu x , Giả sử giá trị của x trước khi bắt đầu thực hiện là 50.

T1: **Read(x)**
 $x := x + 1$
 Write(x)
 Commit

T2: **Read(x)**
 $x := x + 1$
 Write(x)
 Commit

Điều gì sẽ xảy ra nếu:

- a. Hai giao tác thực hiện tuần tự
- b. Hai giao tác thực hiện đồng thời

1.2. CÁC TÍNH CHẤT CỦA GIAO TÁC

a. Hai giao tác thực hiện tuần tự:

Dãy thực thi cho các hành động của 2 giao tác là:

```
T1:  Read(x)
T1:  x := x + 1
T1:  Write(x)
T1:  Commit
T2:  Read(x)
T2:  x := x + 1
T2:  Write(x)
T2:  Commit
```

Nhận xét:

- Giá trị ban đầu của x là 50
- Giá trị của x sau khi T1 uỷ thác là 51
- Giá trị của x sau khi T2 uỷ thác là 52

1.2. CÁC TÍNH CHẤT CỦA GIAO TÁC

b. Các giao tác thực thi đồng thời

Dãy hành động được thực hiện như sau có thể xảy ra:

```
T1:  Read(x)
T1:  x:=x + 1
T2:  Read(x)
T1:  Write(x)
T2:  x:=x + 1
T2:  Write(x)
T1:  Commit
T2:  Commit
```

Nhận xét:

- Giá trị ban đầu của x là 50
- Giá trị của x sau khi T1 uỷ thác là 51
- Giá trị của x sau khi T2 uỷ thác là 51

1.2. CÁC TÍNH CHẤT CỦA GIAO TÁC

Ví dụ (về sự quan trọng của tính cô lập)

Xét 02 giao tác được thực hiện lần lượt ở một hệ thống kế toán ngân hàng như sau:

1. Giao tác T1 chuyển \$1000 vào một tài khoản hiện có \$0.
2. Giao tác T2 đọc quyết toán \$1000 được ghi bởi T1 trước khi T1 hoàn tất và ghi nợ \$1000 vào cùng một tài khoản.
3. T2 hoàn tất, và tiền mặt \$1000 được chuyển cho người dùng người như đã yêu cầu thực hiện T2.
4. T1 bị huỷ bỏ vì lý do một thao tác nào đó bất hợp lệ.
5. Việc huỷ bỏ T1 yêu cầu phải huỷ bỏ T2, bởi vì thao tác thực hiện bởi T2 dựa vào các thao tác thực hiện bởi T1.
6. Tuy nhiên, việc huỷ bỏ T2 là không thể được vì hậu quả của T2 trong thế giới thực không thể được hoàn lại bởi hệ thống.

1.2. CÁC TÍNH CHẤT CỦA GIAO TÁC

❖ Tính bền vững (Durability)

Mục đích:

Để bảo đảm rằng mỗi khi giao tác uỷ thác, kết quả của nó sẽ được duy trì và không bị xoá ra khỏi CSDL.

DDBMS có trách nhiệm bảo đảm kết quả của giao tác và ghi vào CSDL.

Tính bền vững được sử dụng như là một điều kiện để **khôi phục dữ liệu (database recovery)**, nghĩa là cách khôi phục CSDL về trạng thái nhất quán mà ở đó mọi hành động đã uỷ thác đều được phản ánh.

1.3. CÁC LOẠI GIAO TÁC

1.3.1 Giao tác phẳng (flat transaction)

1.3.2 Giao tác lồng nhau (nested transaction)

1.3. CÁC LOẠI GIAO TÁC

1.3.1 Giao tác phẳng

- **Giao tác phẳng (flat transaction) có một khởi điểm duy nhất (Begin transaction) và một điểm kết thúc duy nhất (End transaction).**
- **Tất cả các ví dụ của chúng ta đã xem xét đều nằm trong nhóm này.**
- **Phần lớn các nghiên cứu về quản lý giao tác trong cơ sở dữ liệu đều tập trung vào các giao tác phẳng.**

1.3. CÁC LOẠI GIAO TÁC

1.3.2 Giao tác lồng nhau

- Đây là mô hình giao tác cho phép một giao tác chứa giao tác khác với điểm bắt đầu và ủy thác của riêng chúng. Những giao tác như thế được gọi là giao tác lồng (nested transaction).
- Những giao tác được đặt vào trong giao tác khác thường được gọi là giao tác con (subtransaction)

1.3. CÁC LOẠI GIAO TÁC

1.3.2 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

NỘI DUNG

2. ĐIỀU KHIỂN ĐỒNG THỜI PHÂN TÁN

- Một số vấn đề điều khiển đồng thời
- Một số tính chất khi thao tác trên đơn vị dữ liệu
- Lịch tuần tự và lịch khả tuần tự
- Sắp xếp các giao tác bằng nhãn thời gian
- Điều khiển đồng thời bằng cơ chế khóa

Giới thiệu

➤ **Giao tác:**

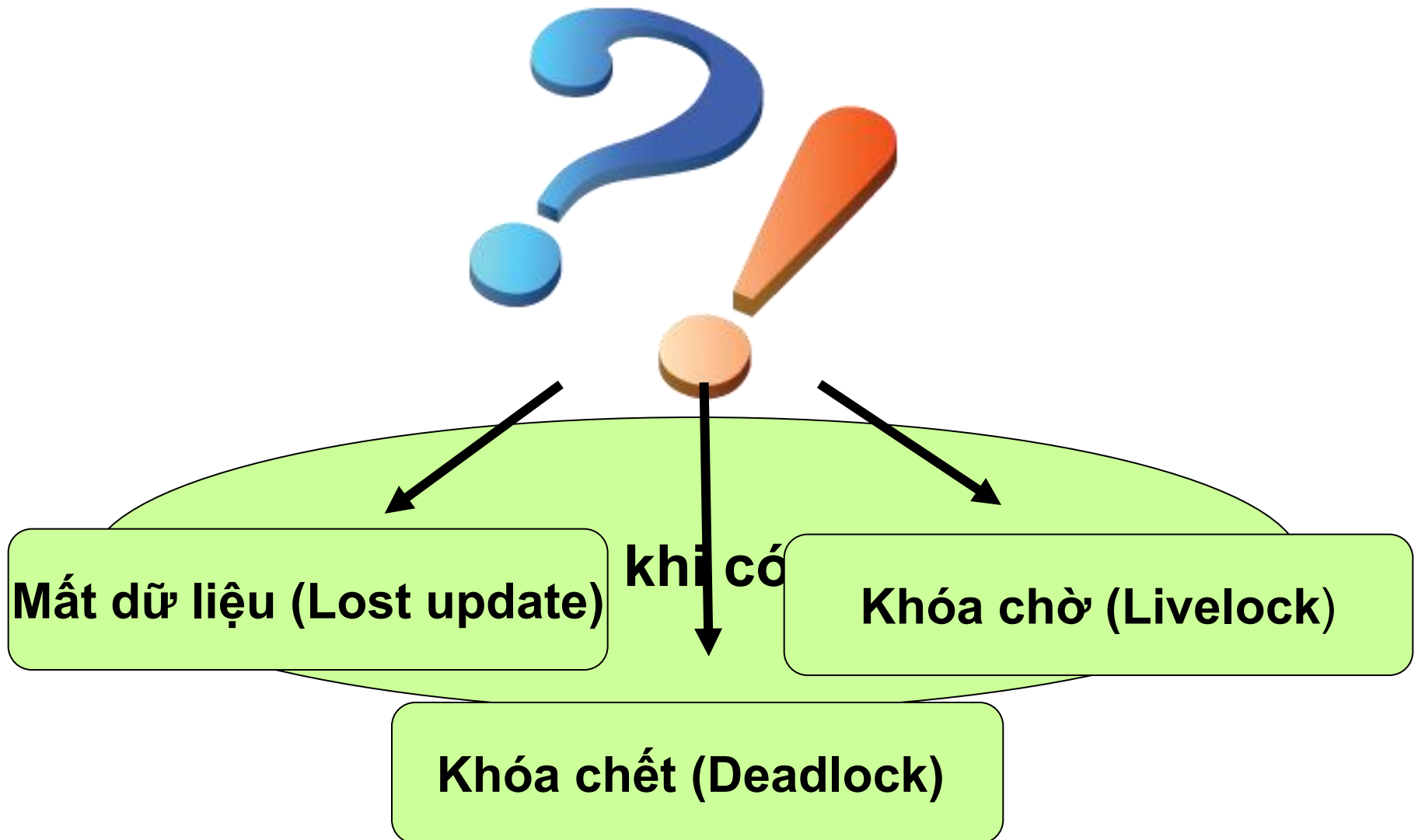
Là một tập hợp các thao tác có thứ tự truy xuất dữ liệu trên CSDL thành một đơn vị công việc logic (xem là một thao tác nguyên tố), chuyển CSDL từ trạng thái nhất quán này sang trạng thái nhất quán khác.

➤ **Nguyên lý điều khiển đồng thời:**

Là quá trình điều khiển giúp cho nhiều giao tác diễn ra đồng thời mà không xảy ra tranh chấp.

➤ **Điều khiển đồng thời phân tán** nhằm ngăn chặn việc sản sinh ra các thực hiện không khả tuần tự của các giao tác phân tán.

Giới thiệu



Giới thiệu

➤ Để ghi nhận sự hoàn tất hay không của một thao tác người ta sử dụng các lệnh sau:

<i>BEGIN TRANSACTION</i>	<i>Bắt đầu giao tác</i>
<i>COMMIT TRANSACTION</i>	<i>Kết thúc một giao tác thành công</i>
<i>ROLLBACK TRANSACTION</i>	<i>Kết thúc một giao tác không thành công, những thao tác làm ảnh hưởng CSDL trước đó được undo, CSDL được trả về tình trạng trước khi thực hiện giao tác.</i>
<i>END TRANSACTION</i>	<i>Chỉ mang ý nghĩa hình thức, thường không sử dụng</i>

1. Một số vấn đề điều khiển đồng thời

1. 1. Mất dữ liệu cập nhật (lost update)

➤ Ví dụ: Cho quan hệ HANGHOA (MaHH, Tên HH, ĐVT, SLTon)

➤ Giả sử: SLTon = 20

Transaction		Giá trị		
T ₁ (bán hàng)	T ₂ (bán hàng)	x ₁	x ₂	SLTon
Begin Transaction	Begin Transaction			20
Read(SLTon)→x ₁		20		20
	Read(SLTon)→x ₂	20	20	20
x ₁ - 10→x ₁		10	20	20
	x ₂ - 3→x ₂	10	17	20
Write x ₁ →SLTon		10	17	
	Write x ₂ →SLTon	10	17	
Commit T ₁	Commit T ₂			

➤ **Lost Update:** Thao tác cập nhật của T₁ xem như bị mất, không được ghi nhận (do những thay đổi của các giao tác khác ghi đè lên).

1. Một số vấn đề điều khiển đồng thời

1.2. Đọc dữ liệu chưa commit (uncommitted data)

Ví dụ: Cho quan hệ **HANGHOA** (MaHH, TenHH, ĐVT, SLTon)
SLton = 20

Transaction		Giá trị		
T ₁ (nhập hàng)	T ₂ (bán hàng)	x ₁	x ₂	SLton
Begin transaction				20
Read (SLton) → x ₁		20		20
x ₁ + 100 → x ₁		120		20
Write x ₁ → SLton		120		
	Begin transaction	120		
	Read (SLton) → x ₂	120		
	x ₂ - 5 → x ₂	120		
	Write x ₂ → SLton	120		
	Commit T ₂	120		
Rollback T ₁				

Transaction T₁ sửa đổi dòng X nhưng chưa commit, transaction T₂ đọc dòng X. Transaction T₁ rollback những gì thay đổi trên dòng X → dữ liệu mà Transaction T₂ đang đọc chưa hề tồn tại.

1. Một số vấn đề điều khiển đồng thời

1.3. Thao tác đọc không thể lặp lại (unrepeatable data)

Ví dụ: Xét 2 giao tác sau:

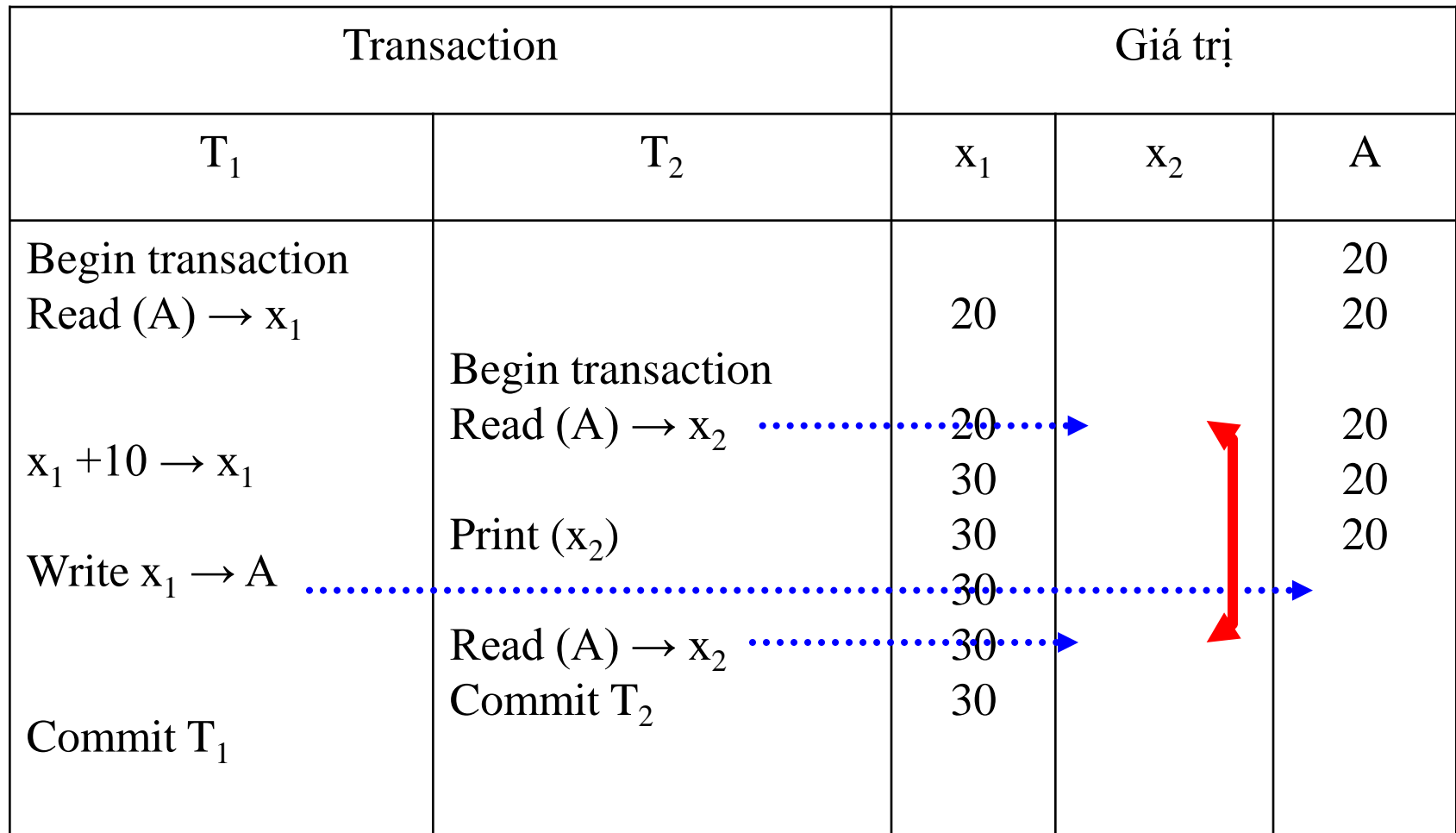
T_1	T_2
Read(A)	
	Read(A)
$A = A + 10$	Print (A)
Write (A)	Read (A)
	Print (A)

➤ Giả sử $A = 20$ và 2 giao tác này thực hiện đồng thời theo thứ tự sau:

1. Một số vấn đề điều khiển đồng thời

1.3. Thao tác đọc không thể lập lại (unrepeatable data)

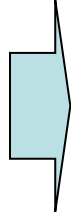
Transaction		Giá trị		
T_1	T_2	x_1	x_2	A
Begin transaction				20
Read (A) $\rightarrow x_1$		20		20
	Begin transaction			
	Read (A) $\rightarrow x_2$	20		20
$x_1 + 10 \rightarrow x_1$		30		20
	Print (x_2)	30		20
Write $x_1 \rightarrow A$		30		20
	Read (A) $\rightarrow x_2$	30		
Commit T_1	Commit T_2	30		



1. Một số vấn đề điều khiển đồng thời

1.4. Vấn đề bóng ma (phantom)

Ví dụ: T_1 thực hiện việc chuyển tiền từ tài khoản A sang tài khoản B. Khi T_1 mới chỉ thực hiện thao tác trừ số tiền ở tài khoản A (*chưa cộng số tiền vào tài khoản B*) thì T_2 muốn xem tổng số tiền ở 2 tài khoản \rightarrow Tổng số tiền không chính xác.



Transaction	
T_1	T_2
<i>Begin transaction</i> Read (A) $A = A - 50$ Write (A) Read (B) $B = B + 50$ Write (B) <i>Commit T_1</i>	<i>Begin transaction</i> Read (A) Read (B) <i>Print (A+B)</i> <i>Commit T_2</i>

1. Một số vấn đề điều khiển đồng thời

➤ Như vậy:

➤ Một tiêu chuẩn để lập lịch các giao tác là việc thực hiện đồng thời các giao tác cho kết quả như khi thực hiện tuần tự các giao tác.

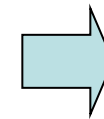
➤ Để giải quyết độ trễ thì phải có “Cơ chế điều khiển tương tranh”.

2. Một số tính chất khi thao tác trên đơn vị dữ liệu

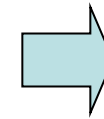
2.1. Hai thao tác tương thích

➤ Hai thao tác O_i và O_j ($O_i \in T_i$, $O_j \in T_j$) gọi là tương thích nếu và chỉ nếu kết quả của việc thực hiện đồng thời O_i và O_j giống như kết quả của việc thực hiện tuần tự O_i rồi đến O_j hoặc O_j rồi đến O_i .

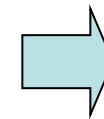
	T_1	T_2	
O_{11}	Read $A \rightarrow a_1$ $a_1 + 1 \rightarrow a_1$ Print a_1	Read $A \rightarrow a_2$ $a_2 * 2 \rightarrow a_2$ Print a_2	O_{21}
O_{12}	Read $A \rightarrow a_1$ $a_1 + 5 \rightarrow a_1$ Write $a_1 \rightarrow A$	Read $A \rightarrow a_2$ $a_2 * 2 \rightarrow a_2$ Write $a_2 \rightarrow A$	O_{22}
O_{13}	Read $A \rightarrow a_1$ $a_1 + 2 \rightarrow a_1$ Write $a_1 \rightarrow A$	Read $A \rightarrow a_2$ $a_2 + 7 \rightarrow a_2$ Write $a_2 \rightarrow A$	O_{23}
O_{14}	Read $B \rightarrow b_1$ $b_1 + 1 \rightarrow b_1$ Write $b_1 \rightarrow B$		



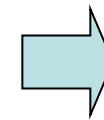
O_{11} và O_{21} là tương thích



O_{12} và O_{22} không tương thích



O_{13} và O_{23} không tương thích



O_{14} tương thích với các thao tác còn lại

2. Một số tính chất khi thao tác trên đơn vị dữ liệu

2.2. Hai thao tác khả hoán vị

➤ Hai thao tác O_i và O_j (O_i thuộc T_i , O_j thuộc T_j) là khả hoán vị nếu kết quả thực hiện O_i, O_j hay O_j, O_i là như nhau.

	T_1	T_2	
O_{11}	Read $A \rightarrow a_1$ $a_1 + 1 \rightarrow a_1$ Print a_1	Read $A \rightarrow a_2$ $a_2 * 2 \rightarrow a_2$ Print a_2	O_{21}
O_{12}	Read $A \rightarrow a_1$ $a_1 + 5 \rightarrow a_1$ Write $a_1 \rightarrow A$	Read $A \rightarrow a_2$ $a_2 * 2 \rightarrow a_2$ Write $a_2 \rightarrow A$	O_{22}
O_{13}	Read $A \rightarrow a_1$ $a_1 + 2 \rightarrow a_1$ Write $a_1 \rightarrow A$	Read $A \rightarrow a_2$ $a_2 + 7 \rightarrow a_2$ Write $a_2 \rightarrow A$	O_{23}
O_{14}	Read $B \rightarrow b_1$ $b_1 + 1 \rightarrow b_1$ Write $b_1 \rightarrow B$		

➤ O_{11} và O_{21} là khả hoán vị

➤ O_{12} và O_{22} không khả hoán vị

➤ O_{13} và O_{23} là khả hoán vị

➤ O_{14} khả hoán vị với các thao tác còn lại

2. Một số tính chất khi thao tác trên đơn vị dữ liệu

Nhận xét:

➤ Các thao tác truy xuất các đơn vị dữ liệu khác nhau là tương thích và khả hoán vị

➤ Các thao tác truy xuất trên cùng đơn vị dữ liệu:

- Nếu có liên quan đến phép cộng, trừ thì khả hoán vị

- Read – Read → khả hoán vị

- Write – Write → không có tính khả hoán vị

- Read – Write → không có tính khả hoán vị

- Write – Read → không có tính khả hoán vị

Chú ý: Hai thao tác không khả hoán vị thì gọi là xung đột

3. Lịch tuần tự và lịch khả tuần tự

3.1. Lịch tuần tự (serial schedule)

➤ Một lịch S được lập từ n giao tác T_1, T_2, \dots, T_n xử lý đồng thời gọi là lịch tuần tự nếu các thao tác của từng giao tác được thực hiện liên tiếp nhau.

T_i	T_j	T_k
R(x) W(x) R(y)	R(x) W(y)	R(y) W(x)

Tuần tự



T_i	T_j	T_k
R(x) W(y)	R(x) W(x)	R(y)

Không tuần tự

3. Lịch tuần tự và lịch khả tuần tự

3.2. Lịch khả tuần tự (serializable schedule)

➤ Một lịch S được lập từ n giao tác T_1, T_2, \dots, T_n xử lí đồng thời gọi là lịch khả tuần tự nếu nó cho cùng kết quả với một lịch tuần tự được lập từ n giao tác trên.

3. Lịch tuần tự và lịch khả tuần tự

3.2. Lịch khả tuần tự (serializable schedule)

➤ Ví dụ 1:

Lịch 1 (A = 1, B = 2)		Lịch 2 tuần tự (A = 1, B = 2)	
T ₁	T ₂	T ₁	T ₂
Read A → a ₁ a ₁ + 1 → a ₁ Write a ₁ → A	Read A → a ₂ a ₂ * 2 → a ₂ Write a ₂ → A	Read A → a ₁ a ₁ + 1 → a ₁ Write a ₁ → A	
Read B → b ₁ b ₁ + 1 → b ₁ Write b ₁ → B	Read B → b ₂ b ₂ * 2 → b ₂ Write b ₂ → B	Read B → b ₁ b ₁ + 1 → b ₁ Write b ₁ → B	Read A → a ₂ a ₂ * 2 → a ₂ Write a ₂ → A
			Read B → b ₂ b ₂ * 2 → b ₂ Write b ₂ → B

Kết quả Li**Lịch 1 có tính khả tuần tự** 2 (A = 4, B = 6)

3. Lịch tuần tự và lịch khả tuần tự

3.2. Lịch khả tuần tự (serializable schedule)

➤ Ví dụ 2:

Lịch 1 (A = 1, B = 2)		Lịch 2 (A = 1, B = 2)	
T ₁	T ₂	T ₁	T ₂
Read A → a ₁ a ₁ + 1 → a ₁ Write a ₁ → A Read B → b ₁ b ₁ + 1 → b ₁ Write b ₁ → B	Read A → a ₂ a ₂ * 2 → a ₂ Write a ₂ → A Read B → b ₂ b ₂ * 2 → b ₂ Write b ₂ → B	Read A → a ₁ a ₁ + 1 → a ₁ Write a ₁ → A Read B → b ₁ b ₁ + 1 → b ₁ Write b ₁ → B	Read A → a ₂ a ₂ * 2 → a ₂ Write a ₂ → A Read B → b ₂ b ₂ * 2 → b ₂ Write b ₂ → B

Kết quả: **Lịch 2 không có tính khả tuần tự** (A = 1, B = 4)

3. Lịch tuần tự và lịch khả tuần tự

Như vậy

➤ Tính khả tuần tự của các giao tác là điều kiện đủ để tránh đụng độ trong việc truy xuất đồng thời (Một lịch nếu khả tuần tự thì không đụng độ, nếu không có khả tuần tự thì chưa chắc đụng độ)

➤ Bộ lập lịch (schedule): Là một bộ phận của DBMS chịu trách nhiệm lập lịch khả tuần tự từ n giao tác xử lý đồng thời, sẽ tiến hành lập lịch các thao tác (thao tác nào sẽ được thực hiện trước, thao tác nào sẽ được thực hiện sau).

4. Sắp xếp các giao tác bằng nhãn thời gian

4.1. Khái niệm nhãn thời gian (timestamp)

➤ Là 1 con số được phát sinh bởi bộ lập lịch, được gán cho mỗi giao tác để chỉ định thời điểm bắt đầu thực hiện giao tác. **Nhãn thời gian có tính chất duy nhất và tăng dần** ($T_i < T_j \leftrightarrow t_{Ti} < t_{Tj}$)

➤ Nhãn thời gian của đơn vị dữ liệu: là nhãn thời gian của giao tác cuối cùng có truy cập đến đơn vị dữ liệu đó thành công, hay là nhãn thời gian cao nhất trong số các giao tác có truy cập thành công đến đơn vị dữ liệu đó.

T_1	T_2	T_3	t_A
Read A	Read A	Read A	$t_A = t_{T1}$ $t_A = t_{T2}$ $t_A = t_{T3}$

4. Sắp xếp các giao tác bằng nhãn thời gian

4.2. Thuật toán sắp xếp toàn phần

Procedure Read (T_i, A)

Begin

If $t_A \leq t_{T_i}$ then

Thực hiện thao tác đọc

$t_A := t_{T_i}$

Else

Nhận xét

Rollback T_i và bắt đầu lại

➤ **Thuật toán chỉ sắp xếp thứ tự các giao tác (thời gian bắt đầu) không nhằm sắp xếp trình tự thực hiện các thao tác trong mỗi giao tác.**

If $t_A \leq t_{T_i}$ then

Thực hiện thao tác ghi

$t_A := t_{T_i}$

Else

Rollback T_i và bắt đầu lại với nhãn thời gian mới

End Proc

Ghi chú:

- t_A : nhãn thời gian của đơn vị dữ liệu A
- t_{T_i} : nhãn thời gian của giao tác T_i
- Ban đầu $t_A = 0$ khi chưa có giao tác truy cập

➤ Ví dụ 1: $t_{T_1} = 100$, $t_{T_2} = 120$, $t_A = 0$

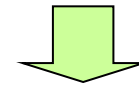
T_1	T_2	t_A
Read A	Read A	$t_A = 100$
$A = A + 1$		$t_A = 120$
	$A = A + 1$	
Write A	Write A	$t_A = 120$
		$t_A > t_{T_1}$ nên T_1 phải rollback và bắt đầu lại với timestamp mới

4. Sắp xếp các giao tác bằng nhãn thời gian

4.2. Thuật toán sắp xếp toàn phần

➤ Ví dụ 2: $t_{T_1} = 100$, $t_{T_2} = 120$, $t_A = 0$

Nhận xét



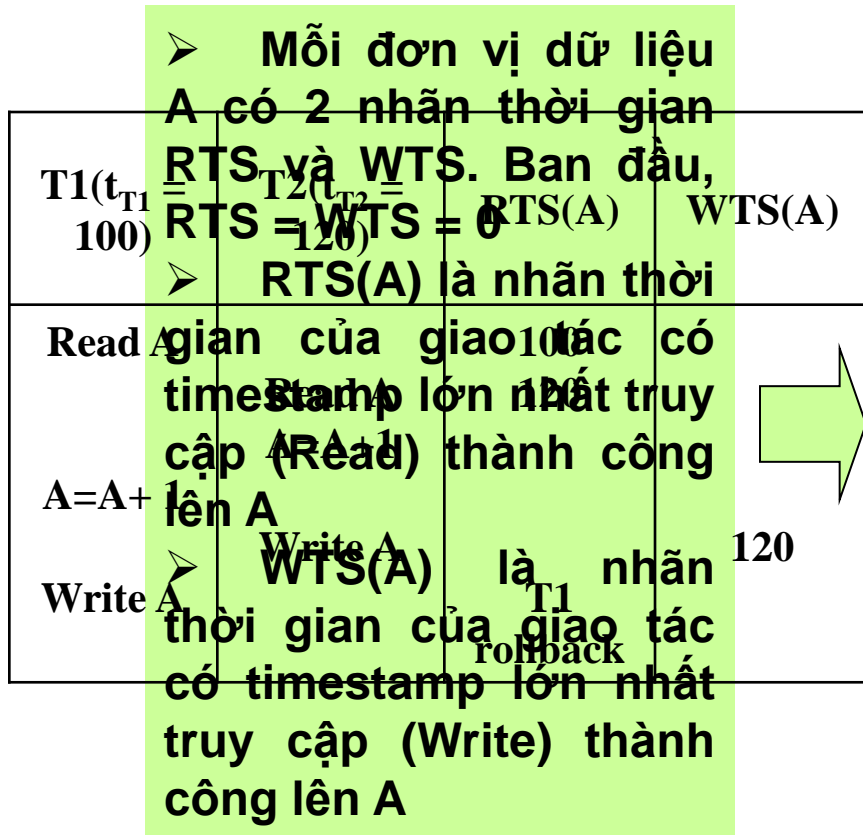
T_1	T_2	t_A
Read A	Read A	$t_A = 100$
	Read A	$t_A = 120$
Read A	Read A	$t_A = 120$
		$t_A > t_{T_1}$ nên T_1 phải rollback và bắt đầu lại với timestamp mới

Như vậy: Thuật toán sắp xếp toàn phần : không quan tâm đến tính chất của thao tác dữ liệu (Read/Write) nên chỉ có 1 nhãn thời gian duy nhất cho 1 đơn vị dữ liệu. Nếu quan tâm đến tính chất của thao tác dữ liệu thì cần 2 nhãn thời gian cho 1 đơn vị dữ liệu tương ứng với thao tác đọc và ghi trên đơn vị dữ liệu đó.

➤ Trong trường hợp này rõ ràng không xảy ra đụng độ T_1 không cần phải rollback và bắt đầu lại với timestamp mới. Tuy nhiên do thuật toán sắp xếp toàn phần không phân biệt tính chất của thao tác dữ liệu là Read hay Write nên T_1 vẫn bị rollback và bắt đầu lại.

4. Sắp xếp các giao tác bằng nhãn thời gian

4.3. Thuật toán sắp xếp từng phần



Procedure Read (T_i, A)

Begin

If $WTS(A) \leq t_{T_i}$ then

Thực hiện thao tác đọc

$RTS(A) := \text{Max}(RTS(A), t_{T_i})$

Else

Rollback T_i và bắt đầu lại với nhãn thời gian mới

End Proc

Procedure Write (T_i, A)

Begin

If $(RTS(A) \leq t_{T_i})$ and $(WTS(A) \leq t_{T_i})$ then

Thực hiện thao tác ghi

$WTS(A) := t_{T_i}$

Else

Rollback T_i và bắt đầu lại với nhãn thời gian mới

End Proc

4. Sắp xếp các giao tác bằng nhãn thời gian

4.3. Thuật toán sắp xếp từng phần

➤ Nhận xét : Trong thuật toán sắp xếp từng phần, số lượng giao tác bị rollback lại ít hơn trong thuật toán sắp xếp toàn phần (do nếu 2 giao tác chỉ thực hiện «Đọc» thì không gây ra ðụng ðộ)

T_1	T_2	Nhận xét
(1) Read A (3) Read A	(2) Read A	Thuật toán sắp xếp toàn phần : T_1 bị rollback ở (3) Thuật toán sắp xếp từng phần : không có rollback

T_1	T_2	Nhận xét
(1) Read A (3) Read A	(2) Write A	Thuật toán sắp xếp toàn phần : T_1 bị rollback ở (3) Thuật toán sắp xếp từng phần : T_1 bị rollback ở (3)

5. Điều khiển đồng thời bằng cơ chế khóa

5.1. Khái niệm khóa (lock)

➤ Phương pháp thông dụng nhất để điều khiển việc truy xuất các đơn vị dữ liệu là sử dụng khóa (lock).

➤ Lock là một đặc quyền truy xuất (access privilege) lên các đơn vị dữ liệu của các giao tác mà bộ quản lý khóa có thể trao cho một giao tác hay thu hồi lại. Khi 1 giao tác đã khóa (lock) trên 1 đơn vị dữ liệu nào đó thì các giao tác khác không được phép truy cập đến đơn vị dữ liệu đó cho đến khi nó nhả khóa (unlock).

➤ Khi một giao tác T thực hiện được việc lock đơn vị dữ liệu A, ta nói, T đang giữ lock A

➤ Tại mỗi thời điểm, chỉ có một tập con các đơn vị dữ liệu bị khóa, vì vậy bộ quản lý khóa có thể lưu các khóa hiện hành trong một bảng khóa (lock table) với các mẫu tin có dạng sau: (A, L, T) (giao tác T có một khóa kiểu L trên đơn vị dữ liệu A).


5. Điều khiển đồng thời bằng cơ chế khóa

5.2. Kỹ thuật khóa đơn giản

➤ Một giao tác khi có yêu cầu truy xuất đến đơn vị dữ liệu thì phải phát ra yêu cầu xin khóa (lock) trên đơn vị dữ liệu đó, nếu yêu cầu này được chấp thuận thì được quyền thao tác và như vậy các giao tác khác sẽ không được phép truy cập đến đơn vị dữ liệu đó cho đến khi giao tác giữ khóa được unlock

Không khóa

T ₁	T ₂	A (5)
Read A		5
	Read A	5
A=A+1		5
	A=A+1	5
	Write A	6
Write A		6

T ₁	T ₂	Nhận xét
Lock A Read A		
	Lock A	
A=A+1	Read A	
	A=A+1	
Write A	Write A	
Unlock A		
	Unlock A	
		T ₂ chờ T ₁ Unlock ↓ T ₁ sẽ hoàn tất trước khi T ₂ bắt đầu. Khi đó A=7

Kỹ thuật khóa

5. Điều khiển đồng thời bằng cơ chế khóa

5.2. Kỹ thuật khóa đơn giản

➤ Ví dụ

T_1	T_2
Lock A Read A $A=A+1$	
Write A Lock B Read B $B=B+1$ Write B Unlock B Unlock A	Lock A Read A Unlock A

➤ Nhận xét

✓ Nếu không phân biệt khóa cho thao tác đọc hay viết thì rõ ràng sẽ có nhiều giao tác phải chờ để được quyền khóa trên 1 đơn vị dữ liệu.

✓ Thực tế là nhiều khi một giao tác chỉ cần lấy giá trị của 1 đơn vị dữ liệu nhưng không thay đổi giá trị đó.

✓ Vì vậy để giảm bớt tình huống phải chờ khi các giao tác cùng đọc dữ liệu, người ta đề nghị tách yêu cầu khóa thành 2 yêu cầu khóa riêng biệt.

5. Điều khiển đồng thời bằng cơ chế khóa

5.3. Kỹ thuật khóa đọc/viết (Readlock/Writelock)

* Khóa để đọc (hay Shared lock): một giao tác T chỉ muốn đọc 1 đơn vị dữ liệu A sẽ thực hiện lệnh RLOCK(A), ngăn không cho bất kì giao tác khác ghi giá trị mới của A trong khi T đã khóa A. Tuy nhiên các giao tác khác vẫn có thể giữ 1 khóa đọc trên A cùng lúc với T.

* Khóa để ghi (Exclusive lock): một giao tác T muốn thay đổi giá trị của 1 đơn vị dữ liệu A đầu tiên sẽ lấy khóa ghi bằng cách thực hiện lệnh WLOCK(A). Khi 1 giao tác đang giữ 1 khóa ghi trên 1 đơn vị dữ liệu, các giao tác khác không thể lấy được khóa đọc hay khóa ghi trên A cùng một lúc với T.

➤ Cả hai khóa đọc và khóa ghi đều được loại bỏ bằng lệnh UNLOCK

5. Điều khiển đồng thời bằng cơ chế khóa

5.3. Kỹ thuật khóa đọc/viết (Readlock/Writelock)

Điều kiện để xin khóa đọc/viết

- Một yêu cầu xin RLOCK(A) chỉ được chấp thuận nếu A chưa bị khóa bởi 1 WLOCK trước đó.
- Một yêu cầu xin WLOCK(A) chỉ được chấp thuận nếu A được tự do.

Bảng tương thích các loại khóa

		Lock đang được giữ	
		R-lock	W-lock
Giao tác yêu cầu lock	R-lock	Yes	No
	W-lock	No	No

5. Điều khiển đồng thời bằng cơ chế khóa

5.3. Kỹ thuật khóa đọc/viết (Readlock/Writelock)

Ví dụ (Với đơn vị dữ liệu $B=2$)

Nhận xét

T_1	T_2	Ghi chú
Rlock B Read $B \rightarrow a_1$ Unlock B		$a_1=B=2$
	Rlock B Read $B \rightarrow a_2$ Unlock B	$a_2=B=2$
	$a_2 + 1 \rightarrow a_2$	$a_2=3$
	Wlock B Write $a_2 \rightarrow B$ Unlock B	$B=a_2=3$
$a_1 + 1 \rightarrow a_1$ Wlock B Write $a_1 \rightarrow B$ Unlock B		$a_1=3$ $B=a_1=3$

➤ Lịch thao tác không khả tuần tự nên xảy ra lost update.

➤ Việc sử dụng cơ chế lock không đủ để bảo đảm tính khả tuần tự cho lịch thao tác

➤ Để giải quyết tính không khả tuần tự. Dùng chiến lược lock 2 pha, hoặc yêu cầu các giao tác lock các đơn vị dữ liệu theo 1 thứ tự cố định nào đó.

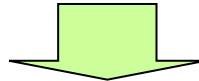
5. Điều khiển đồng thời bằng cơ chế khóa

5.4. Các vấn đề trong kỹ thuật khóa

Livelock

➤ Giả sử khi T_1 giải phóng khóa trên A, khóa này được trao lại cho T_2 . Điều gì sẽ xảy ra nếu như trong khi T_2 đang đợi nhận khóa, một giao tác T_3 khác cũng xin một khóa trên A, và T_3 lại được trao khóa này trước T_2 . Rồi sau khi T_3 được trao khóa trên A thì lại có 1 giao tác T_4 xin khóa trên A,... Và rất có thể T_2 phải đợi mãi và chẳng bao giờ nhận được khóa trong khi luôn có 1 giao tác khác giữ khóa trên A, dù rằng có một số lần T_2 có cơ hội nhận được khóa trên A.

Như vậy, *Livelock là trường hợp 1 giao tác chờ được cấp quyền lock trên 1 đơn vị dữ liệu nào đó mà không xác định được thời điểm được đáp ứng yêu cầu.*



Yêu cầu hệ thống khi trao khóa phải ghi nhận tất cả các thỉnh cầu chưa được đáp ứng, và khi đơn vị dữ liệu A được mở khóa thì trao cho các giao tác đã xin đầu tiên trong số những giao tác đang đợi khóa A. Và khi đó bộ lập lịch (schedulers) sẽ tiến hành thực hiện cơ chế: giao tác nào yêu cầu trước thì được đáp ứng trước (FIFO).

5. Điều khiển đồng thời bằng cơ chế khóa

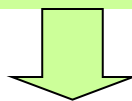
5.4. Các vấn đề trong kỹ thuật khóa

Deadlock

T_1	T_2	Nhận xét
Lock A	Lock B	- T_1 chờ T_2 unlock B
Lock B	Lock A	- T_2 chờ T_1 unlock A → Deadlock

➤ T_1 yêu cầu và được trao khóa trên A, còn T_2 yêu cầu và được trao khóa trên B. Do đó khi T_1 yêu cầu khóa trên B, nó sẽ phải đợi vì T_2 đã khóa B. Tương tự khi T_2 yêu cầu khóa trên A, nó cũng buộc phải đợi vì T_1 đã khóa A. Kết quả là không một giao tác nào tiếp tục hoạt động được: mỗi giao tác đều phải đợi cho giao tác kia mở khóa, và chúng đều phải đợi nhưng chẳng bao giờ nhận được khóa yêu cầu.

Deadlock là tình trạng trong đó những giao tác có liên quan không thể thực hiện tiếp các thao tác của nó mà phải chờ nhau mãi



Bỏ qua (Hủy, làm lại)

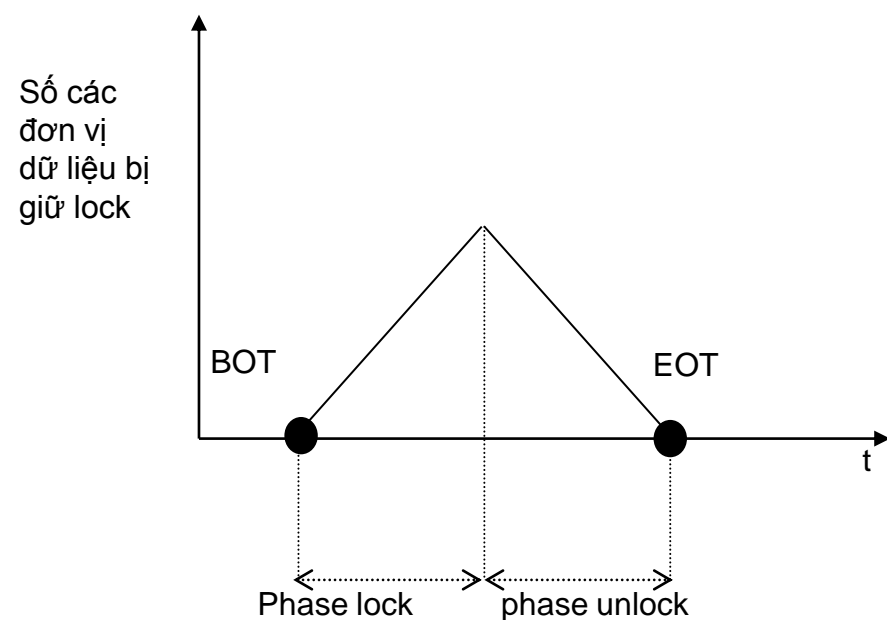
Ngăn ngừa (Chờ theo chiều nhất định)

Phát hiện (Đồ thị chờ)

5. Điều khiển đồng thời bằng cơ chế khóa

5.4. Nghi thức lock 2 giai đoạn (2 phase)

➤ Một giao tác thực hiện cơ chế lock 2 phase là một giao tác không thực hiện một lock nào nữa sau khi đã unlock (*thực hiện xong hết tất cả các yêu cầu lock rồi mới thực hiện unlock*).



Thời gian	T ₁	T ₂	T ₃
		Lock(A)	
		Unlock(A)	
			Lock(A)
			Unlock(A)
	Lock(B)		
	Unlock(B)		
		Lock(B)	
		Unlock(B)	

Trong các giao tác trên, T₁ và T₃ là các giao tác 2 pha, còn T₂ thì không

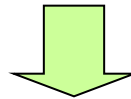
5. Điều khiển đồng thời bằng cơ chế khóa

5.4. Nghi thức lock 2 giai đoạn (2 phase)

Nhận xét

➤ Một lịch S được lập từ n giao tác thỏa nghi thức khóa 2 pha thì khả tuần tự.

➤ Sử dụng nghi thức lock 2 giai đoạn chỉ có thể đảm bảo tính khả tuần tự cho lịch thao tác nhưng không thể bảo đảm không xảy ra vấn đề deadlock.



T_1	T_2	Ghi chú
Lock A	Lock B	
Lock B	Lock A	
Unlock A	Unlock B	
Unlock B	Unlock A	

Note: The table contains a diagram showing a cycle of dependencies between T1 and T2. T1 locks A then B, and unlocks A then B. T2 locks B then A, and unlocks B then A. Arrows indicate that T1's Lock B depends on T2's Unlock B, and T2's Lock A depends on T1's Unlock A, leading to a deadlock.