PHẦN 2. ĐIỀU KHIỂN TƯƠNG TRANH (Concurrency Control)

Tài liệu tham khảo

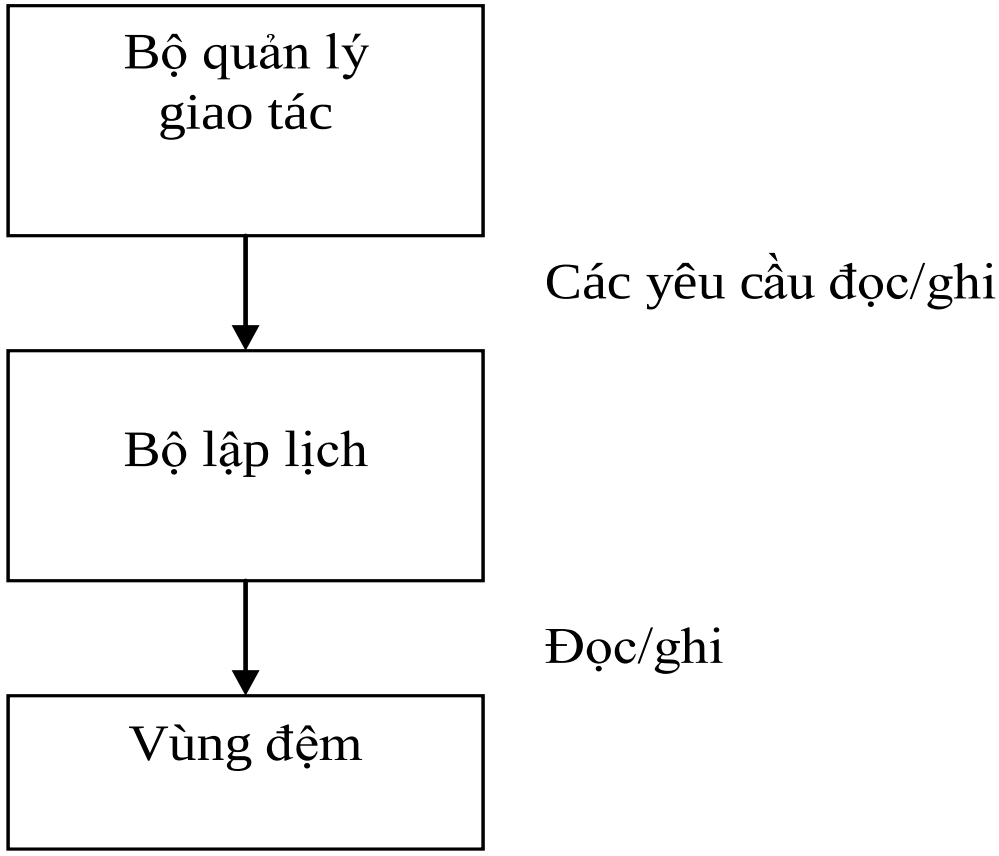
- [TH] Hồ Thuần, Hồ Cẩm Hà, *Các hệ cơ sở dữ liệu*, Lý thuyết và thực hành, NXB GD, 2005., Chương 8.
- [JU] Jeffrey.D. Ullman, *Principles of Database and Knowledge-base systems*, tập 1&2, Computer Science Press, 1988. Bản dịch: Trần Đức Quang, *Nguyên lý các hệ CSDL và cơ sở tri thức*, NXB thống kê, 1999, Chương 9.
- [HJJ] Hector Garcia–Molina, Jeffrey D. Ullman, Jennifer Widom, *Database Systems – The complete book*, Prentice Hall, 2002.
- [MM] Michael V. Mannino, *Database Design, Application Development, and Administration*, Mc Graw–Hill, 2004.

NỘI DUNG

1. Mục đích của điều khiển tương tranh
2. Vấn đề đụng độ
3. Lịch và tính khả tuần tự của lịch
4. Điều khiển tương tranh dựa trên khoá
5. Điều khiển tương tranh dựa trên nhãn thời gian

1. Mục đích của điều khiển tương tranh

- Nhiều người dùng đồng thời truy cập vào CSDL
- Đảm bảo duy trì tính toàn vẹn dữ liệu



2. Vấn đề đụng độ (1)

- Mục dữ liệu: Mục dữ liệu là một khối dữ liệu có được trong một lần đọc/ghi.
- Ví dụ: có 3 mục dữ liệu A, B, C;
- Giao tác T1: C: = C –1; A: =A+1
- Giao tác T2: B: = B -3; A: =A+3

2. Vấn đề đụng độ (2)

- Giả sử ban đầu A= B =C =10

T1	T2
READ(C, t) t:= t − 1 WRITE(C, t) READ(A, t) t := t +1 WRITE(A,t)	READ(B, s) s:= s− 3 WRITE(B, s) READ(A, s) s := s +3 WRITE(A, s)

- Nếu thực hiện tuần tự T1;T2; kết quả là A=14; B =7; C =9

2. Vấn đề đụng độ (3)

- Nếu thực hiện đồng thời T1;T2 như sau:

T1	T2	A	B	C
READ(C, t)				10
t := t − 1				
WRITE(C, t)				9
	READ(B, s)		10	
	s := s −3			
	WRITE(B, s)		7	
READ(A, t)		10		
t := t +1				
	READ(A, s)	10		
	s := s + 3			
	WRITE(A,s)	13		
WRITE(A, t)		11		

3. Lịch và tính khả tuần tự của lịch

Khái niệm

- Lịch S được lập từ các giao tác T1, T2,…,Tn là một dãy các thao tác của một tập các giao tác tương tranh {Ti} mà trong đó thứ tự của các thao tác trong mỗi giao tác được bảo toàn.
- Lịch tuần tự là lịch thực hiện tuần tự hoàn chỉnh từng giao tác một.
- Lịch từ n giao tác T1, T2,…,Tn là khả tuần tự (serializable) nếu việc thực hiện lịch tương đương với việc thực hiện một lịch tuần tự nào đó
- Việc tìm ra một lịch tuần tự tương đương với một lịch nào đó được gọi là tuần tự hoá.

3. Lịch và tính khả tuần tự của lịch

- Ví dụ:

T1	T2
READ(A, t) t := t +100 WRITE(A, t) READ(B,t) t := t +100 WRITE(B,t)	READ(A,s) s := s *2 WRITE(A, s) READ(B,s) s := s*2 WRITE(B,s)

3. Lịch và tính khả tuần tự của lịch

Ví dụ, lịch tuần tự S1 thực hiện T1, T2:

T1	T2	A	B
READ(A,t)		25	25
t := t +100			
WRITE(A, t)		125	
READ(B, t)			25
t := t +100			
WRITE(B,t)			125
	READ(A,s)	125	
	s := s * 2		
	WRITE(A, s)	250	
	READ(B, s)		125
	s := s * 2		
	WRITE(B,s)		250

3. Lịch và tính khả tuần tự của lịch

- Lịch S2 thực hiện tuần tự T2; T1

T1	T2	A	B
	READ(A,s)	25	25
	s := s * 2		
	WRITE(A, s)	50	
	READ(B, s)		25
	s := s * 2		
	WRITE(B,s)		50
READ(A,t)		50	
t := t +100			
WRITE(A, t)		150	
READ(B, t)			50
t := t+100			
WRITE(B,t)			150

3. Lịch và tính khả tuần tự của lịch

- Lịch S3 khả tuần tự vì tương đương với lịch S1

T1	T2	A	B
READ(A,t)		25	25
t := t +100			
WRITE(A, t)		125	
	READ(A,s)	125	
	s := s * 2		
	WRITE(A, s)	250	
READ(B, t)			25
t := t+100			
WRITE(B,t)			125
	READ(B, s)		125
	s := s * 2		
	WRITE(B,s)		250

3. Lịch và tính khả tuần tự của lịch

- Lịch S4 không khả tuần tự

T1	T2	A	B
READ(A,t)		25	25
t := t +100			
WRITE(A, t)		125	
	READ(A,s)	125	
	s := s * 2		
	WRITE(A, s)	250	
	READ(B, s)		25
	s := s * 2		
	WRITE(B,s)		50
READ(B, t)			50
t := t+100			
WRITE(B,t)			150

3. Lịch và tính khả tuần tự của lịch

- Kí hiệu giao tác và lịch biểu:
  - Giả thiết: Không quan tâm đến các thao tác tính toán cục bộ của riêng giao tác.
- Kí hiệu:
  - $r_T(X)$, $w_T(X)$: giao tác T đọc/ghi mục dữ liệu X
  - $r_{Ti}(X)$ hay $r_i(X)$, $w_{Ti}(X)$ hay $w_i(X)$: giao tác T_i đọc/ ghi mục dữ liệu X.
- Ví dụ:
  - T1: $r_1(A)$; $w_1(A)$; $r_1(B)$; $w_1(B)$;
  - T2: $r_2(A)$; $w_2(A)$; $r_2(B)$; $w_2(B)$;
- Ví dụ một lịch S được lập từ T1, T2 như sau:

S: $r_1(A)$; $w_1(A)$; $r_2(A)$; $w_2(A)$; $r_1(B)$; $w_1(B)$; $r_2(B)$; $w_2(B)$;

3. Lịch và tính khả tuần tự của lịch
Khả tuần tự xung đột

- xung đột (conflict): cặp hai thao tác trong một lịch biểu, mà nếu như thay đổi thứ tự của chúng thì hành vi (kết quả) của ít nhất một trong những giao tác liên quan có thể bị thay đổi.
- Nhận xét: giả sử T_i và T_j là hai giao tác khác nhau.
- Các tình huống có thể tráo đổi thứ tự 2 thao tác:
  - $R_i(X)$ và $R_j(Y)$ không bao giờ xung đột nhau
  - $R_i(X)$ và $W_j(Y)$ không xung đột nếu $X \neq Y$
  - $W_i(X)$ và $R_j(Y)$ không xung đột nếu $X \neq Y$.
  - $W_i(X)$ và $W_j(Y)$ không xung đột nếu $X \neq Y$.

3. Lịch và tính khả tuần tự của lịch
Khả tuần tự xung đột

- Ba tình huống không chấp nhận tráo đổi thứ tự hai thao tác:
  - Hai thao tác của cùng một giao tác, ví dụ $R_i(X)$ và $W_i(Y)$.
  - Hai thao tác ghi trên cùng một mục dữ liệu bởi các giao tác khác nhau: $W_i(X)$ và $W_j(X)$.
  - Một thao tác đọc và một thao tác ghi thuộc 2 giao tác khác nhau và cùng thực hiện trên cùng một mục dữ liệu sẽ là xung đột nhau:
    - $R_i(X)$; $W_j(X)$; giao tác T_i đọc giá trị của X mà chưa bị ảnh hưởng bởi thao tác ghi trên X của giao tác T_j .
    - $W_j(X)$; $R_i(X)$; sau khi T_j thực hiện ghi X, giá trị của X sẽ thay đổi và giao tác T_i đọc được giá trị mới này.

3. Lịch và tính khả tuần tự của lịch
Khả tuần tự xung đột

- Khái niệm "xung đột":
  - Hai thao tác kế tiếp trong một lịch gọi là xung đột nếu chúng thuộc hai giao tác khác nhau, cùng tác động trên một mục dữ liệu và ít nhất một trong hai thao tác là thao tác ghi.
- S và S' là tương đương xung đột: Nếu lịch S biến đổi được về lịch S' nhờ một dãy các tráo đổi thứ tự các thao tác không xung đột.
- Tuần tự hoá bằng cách tráo đổi các thao tác không xung đột gọi là tuần tự hoá xung đột (conflict serializability).
- Lịch S tương đương xung đột với một lịch tuần tự được gọi là khả tuần tự xung đột.

3. Lịch và tính khả tuần tự của lịch
Khả tuần tự xung đột

- Ví dụ: Chứng minh lịch sau là khả tuần tự xung đột

  S3: $R_1(A)$; $W_1(A)$; $R_2(A)$; $W_2(A)$; $R_1(B)$; $W_1(B)$; $R_2(B)$; $W_2(B)$;

- Biến đổi S3 bởi một dãy các tráo đổi thứ tự các cặp thao tác không xung đột:

  $R_1(A)$; $W_1(A)$; $R_2(A)$; $\underline{W_2(A); R_1(B);}$ $W_1(B)$; $R_2(B)$; $W_2(B)$;

  $R_1(A)$; $W_1(A)$; $\underline{R_2(A); R_1(B)}$; $W_2(A)$; $W_1(B)$; $R_2(B)$; $W_2(B)$;

  $R_1(A)$; $W_1(A)$; $R_1(B)$; $R_2(A)$; $\underline{W_2(A); W_1(B)}$; $R_2(B)$; $W_2(B)$;

  $R_1(A)$; $W_1(A)$; $R_1(B)$; $\underline{R_2(A); W_1(B);}$ $W_2(A)$; $R_2(B)$; $W_2(B)$;

  $R_1(A)$; $W_1(A)$; $R_1(B)$; $W_1(B)$; $R_2(A)$; $W_2(A)$; $R_2(B)$; $W_2(B)$;

  - Kết quả, S3 chuyển thành một lịch tuần tự T_1; T_2.
  - S3 là lịch khả tuần tự xung đột.

- **Thuật toán:** Kiểm tra tính khả tuần tự xung đột

Input: Lịch S của n giao tác $T_1, T_2, \ldots, T_n$

Output: – True, nếu S khả tuần tự xung đột

(và đưa ra lịch biểu tuần tự tương đương)

– False, nếu ngược lại

Method:

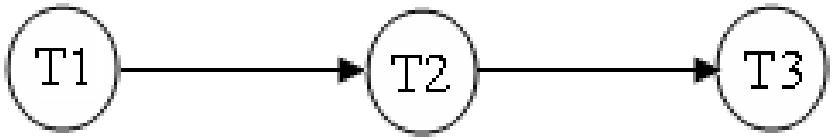
1. Lập đồ thị có hướng G(V,E) , với V = $\{T_1, T_2, \ldots, T_n\}$
   - E = $\{T_i \rightarrow T_j\}$. Có cung $T_i \rightarrow T_j$ nếu, với A là mục dữ liệu bất kì:

     + tại bước k: $R_i(A)$ thì có bước k + m: $W_j(A)$

     + hoặc tại bước k: $W_i(A)$ thì bước k + m là $W_j(A)$ hoặc $R_j(A)$.
2. Kiểm tra xem G có chu trình không,
   - + Nếu có, return False;
   - + Nếu không, return True

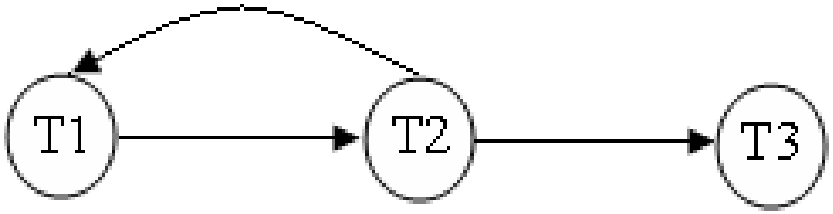
3. Lịch và tính khả tuần tự của lịch
Khả tuần tự xung đột

- Ví dụ: $R_2(A)$; $R_1(B)$; $W_2(A)$; $R_3(A)$; $W_1(B)$; $W_3(A)$; $R_2(B)$; $W_2(B)$;



  - Lịch khả tuần tự xung đột.

- Ví dụ: $R_2(A)$; $R_1(B)$; $W_2(A)$; $R_2(B)$; $R_3(A)$; $W_1(B)$; $W_3(A)$; $W_2(B)$;



  - Lịch không khả tuần tự xung đột.

3. Lịch và tính khả tuần tự của lịch
Khả tuần tự trực quan

- Có những lịch không khả tuần tự xung đột nhưng vẫn khả tuần tự (khả tuần tự trực quan)
- Ví dụ: Xét lịch S tuần tự: T_1; T_2; T_3;
  - S: $W_1(Y)$; $W_1(X)$; $W_2(Y)$; $W_2(X)$; $W_3(X)$;
  - S1: $W_1(Y)$; $W_2(Y)$; $W_2(X)$; $W_1(X)$; $W_3(X)$;

  Trong đó các thao tác $W_3(X)$ ghi X nhưng không thực hiện đọc X (thao tác ghi "nhắm mắt").
- Một lịch khả tuần tự xung đột cũng là khả tuần tự trực quan.
- Mọi lịch khả tuần tự trực quan mà không khả tuần tự xung đột đều chứa một hay nhiều thao tác ghi "nhắm mắt"

3. Lịch và tính khả tuần tự của lịch
Khả tuần tự trực quan

- Khái niệm tuần tự trực quan (View Serializability)

  Hai lịch S1 và S2 được lập từ n giao tác T_1, T_2, ..., T_n là tương đương trực quan nếu:

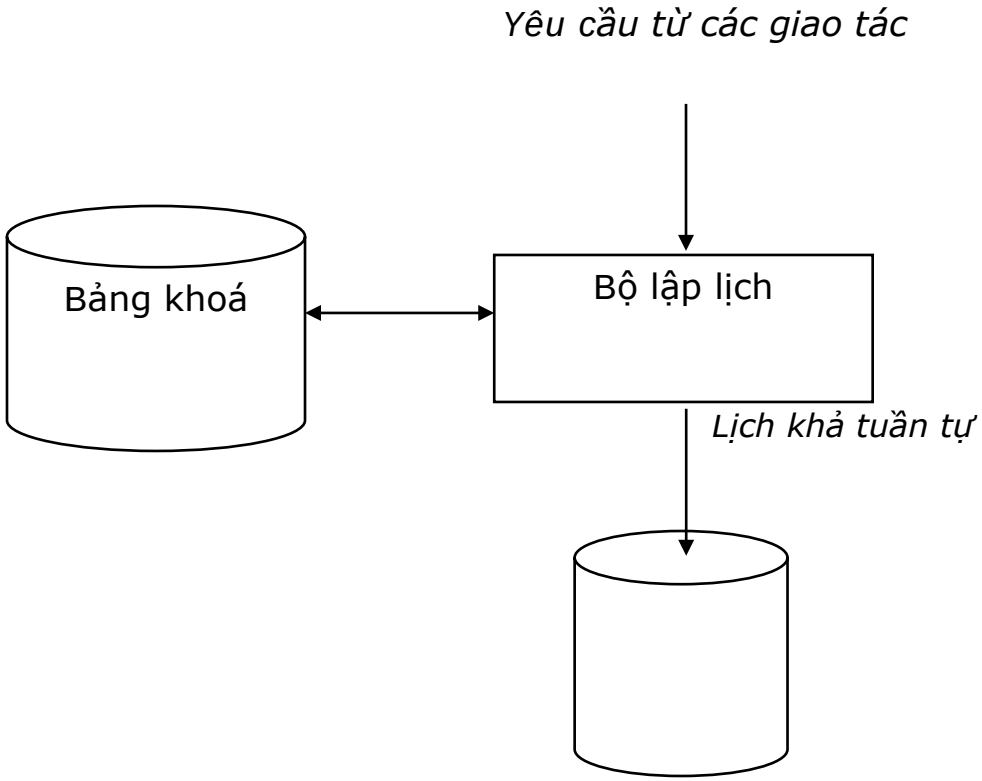
  - 1– Với mỗi mục dữ liệu x, nếu giao tác T_i đọc giá trị lần đầu của x trong lịch S1 thì trong lịch S2, giao tác T_i cũng phải đọc giá trị lần đầu của x.
  - 2– Với mỗi thao tác đọc mục dữ liệu x của giao tác T_i trong S1, nếu giá trị đọc được là do giao tác T_j ghi thì trong lịch S2, giao tác T_i cũng đọc giá trị của x được ghi bởi giao tác T_j.
  - 3– Với mỗi mục dữ liệu x, nếu trong lịch S1, thao tác ghi lần cuối được thực hiện bởi giao tác T_i thì trong lịch S2 cũng vậy.

4. Điều khiển tương tranh dựa trên khóa
Khái niệm Khóa

- Khoá: là một biến gắn với một mục dữ liệu trong CSDL để biểu diễn trạng thái của mục dữ liệu này trong mối liên hệ với một thao tác thực hiện trên đó (đọc/ghi).
- Các mô hình khoá:
  - Khoá chia sẻ–khoá độc quyền
  - Khoá chia sẻ–khoá độc quyền– Khoá cập nhật
  - Khoá chia sẻ–khoá độc quyền–Khoá tăng trị
  - …..

4. Điều khiển tương tranh dựa trên khóa

- Yêu cầu khoá do bộ quản lý điều khiển tương tranh (concurrency-control manager) đưa ra. Giao tác chỉ có thể tiếp tục sau khi yêu cầu khoá được chấp nhận.



4. Điều khiển tương tranh dựa trên khóa
Các mô hình Khóa

- Mô hình Khoá chia sẻ–khoá độc quyền:
  - Mục dữ liệu bị khoá theo 2 thể thức:

    1. *Khoá độc quyền hay khoá ghi (exclusive lock|write lock)*. Mục dữ liệu có thể vừa được đọc và ghi.

       Chỉ thị **XL(X)**: khoá mục dữ liệu X để đọc và ghi.

    2. *Khoá chia sẻ hay khoá đọc (shared lock| read lock)*. Mục dữ liệu chỉ có thể được đọc.

       Chỉ thị **SL(X)**: khoá mục dữ liệu X để đọc.

    Kí hiệu: $U_i(X)$: Giao tác T_i giải phóng khoá mà nó đã có trên mục dữ liệu X.

4. Điều khiển tương tranh dựa trên khóa
Các mô hình Khóa

- Ma trận tương thích khoá (Lock-compatibility matrix)

		Các thể thức khoá được yêu cầu	
		S	X
Các thể thức khoá đang giữ trên mục dữ liệu	S	True	False
	X	False	False

  - Nếu một mục dữ liệu chưa bị khoá thì khoá sẽ được cấp phát theo đúng yêu cầu của giao tác.
  - Một giao tác được cấp một khoá trên một mục dữ liệu nếu yêu cầu khoá tương thích với các khoá đang được các giao tác khác giữ trên mục dữ liệu đó .
  - Nhiều giao tác có thể cùng giữ khoá đọc trên một mục dữ liệu, nhưng khi có một giao tác đang giữ khoá ghi trên một mục dữ liệu thì không một giao tác nào khác được giữ khoá trên mục dữ liệu đó.
  - Nếu yêu cầu khoá không được chấp nhận, giao tác phải đợi cho đến khi tất cả các khoá không tương thích do các giao tác khác đang giữ được giải phóng.

4. Điều khiển tương tranh dựa trên khóa

- Ví dụ:
  - T1: SL1(A); R1(A); XL1(B); R1(B); W1(B); U1(A); U1(B);
  - T2: SL2(A); R2(A); SL2(B); R2(B); U2(A); U2(B);

T1	T2
$SL_1(A)$; $R_1(A)$;	
$XL_1(B)$: Denied	$SL_2(A)$; $R_2(A)$; $SL_2(B)$; $R_2(B)$;
	$U_2(A)$; $U_2(B)$;
$XL_1(B)$; $R_1(B)$; $W_1(B)$;	
$U_1(A)$; $U_1(B)$;	

4. Điều khiển tương tranh dựa trên khóa
Nguyên tắc Khóa

- Nguyên tắc khoá dữ liệu:
  - Tính nhất quán
  - Tính hợp lệ
  - Giao thức khoá hai pha

4. Điều khiển tương tranh dựa trên khóa

- (1) Tính nhất quán của các giao tác:
  - Một giao tác bất kì muốn truy cập (đọc/ghi) một mục dữ liệu thì phải có khoá phù hợp trên mục dữ liệu đó, cụ thể:
    - Để có thao tác $R_i(X)$ thì trước đó phải có yêu cầu $SL_i(X)$ hoặc $XL_i(X)$ và không có chỉ thị $U_i(X)$ nào nằm giữa.
    - Để có thao tác $W_i(X)$ thì trước đó phải có yêu cầu $XL_i(X)$ và không có chỉ thị $U_i(X)$ nào nằm giữa.
  - Mọi khoá cần phải có chỉ thị giải phóng khoá trên cùng một mục dữ liệu.

4. Điều khiển tương tranh dựa trên khóa

- (2)Tính hợp lệ của lịch:

  Một mục dữ liệu hoặc là bị chiếm giữ bởi duy nhất một khoá độc quyền của một giao tác, hoặc là bởi một vài khoá chia sẻ, nhưng nhất thiết không thể xảy ra cả hai trường hợp trên:

  - Khi có $XL_i(X)$ xuất hiện trong lịch thì sau đó không thể có $XL_j(X)$ hay $SL_j(X)$ nào của giao tác T_j khác T_i, nếu như giữa chúng không có thao tác mở khoá $U_i(X)$.
  - Khi có $SL_i(X)$ xuất hiện trong lịch thì sau đó không thể có $XL_j(X)$ của giao tác T_j khác T_i, nếu như giữa chúng không có thao tác mở khoá $u_i(X)$.

4. Điều khiển tương tranh dựa trên khóa

- (3) Giao thức khoá hai pha (two–phase locking):

  Mọi chỉ thị yêu cầu khoá phải đứng trước mọi chỉ thị giải phóng khoá. Mỗi giao tác đều được chia làm hai pha:

  - Pha mở rộng: các giao tác yêu cầu tất cả các khoá cần thiết và không được giải phóng khoá nào
  - Pha rút gọn: Giao tác giải phóng các khoá của nó nhưng không yêu cầu thêm bất kì khoá nào.

Giao thức khoá 2 pha

- đảm bảo tính khả tuần tự xung đột.
- giúp loại bỏ khả năng gây ra tình huống “hoài vọng” (Starvation hay live lock): một giao tác đợi vô hạn mà không được cấp đủ các khóa trên các mục dữ liệu theo yêu cầu.
- không xử lý được tình huống “nghẽn khoá” (deadlock)

4. Điều khiển tương tranh dựa trên khóa

- Nghẽn khoá (Deadlock): tình huống có ít nhất hai giao tác, trong đó giao tác này buộc phải chờ đợi vô hạn để được cấp một khoá mà đang bị chiếm giữ bởi giao tác kia.
- Ví dụ:

T1	T2	A	B
$XL_1(A)$; $R_1(A)$;		25	25
	$XL_2(B)$; $R_2(B)$;		
A := A +100;			
	B := B * 2;		
$W_1(A)$;		125	
	$W_2(B)$;		50
XL(B): Denied	XL(A): Denied		

4. Điều khiển tương tranh dựa trên khóa

- Nâng cấp khoá: Giao tác yêu cầu khoá độc quyền trên mục dữ liệu mà chính nó đang giữ khóa chia sẻ.
- ưu điểm: Tăng cường các thao tác song song, nâng cao hiệu năng của hệ thống.
- nhược điểm: tăng khả năng gây ra tình trạng nghẽn khoá
  - phòng tránh: dùng mô hình 3 thể thức khoá: khoá độc quyền, khoá chia sẻ, khoá cập nhật

4. Điều khiển tương tranh dựa trên khóa

- Khoá cập nhật (update lock)
  - $SL_i(X)$:
  - $XL_i(X)$:
  - $UL_i(X)$: Giao tác T_i chỉ có quyền đọc mà không ghi mục dữ liệu X. Chỉ khoá cập nhật mới được nâng cấp thành khoá độc quyền, còn khoá chia sẻ thì không.

		Các thể thức khoá được yêu cầu		
		S	X	U
Các thể thức khoá đang giữ trên mục dữ liệu	S	True	False	True
	X	False	False	False
	U	False	False	False

5. Giao thức nhãn thời gian

- Mỗi giao tác được cấp một nhãn thời gian (timestamp) khi đưa vào hệ thống. Nếu đã có một giao tác T_i có nhãn thời gian TS(T_i), một giao tác mới T_j sẽ được gán nhãn TS(T_j) sao cho TS(T_i) < TS(T_j).
- Giao thức điều khiển thi hành tương tranh sao cho các nhãn thời gian đảm bảo theo thứ tự khả tuần tự.
- Giao thức duy trì 2 giá trị nhãn thời gian trên mỗi mục dữ liệu X
  - **nhãn đọc X**
  - **nhãn ghi X**

5. Giao thức nhãn thời gian

- **Nhãn ghi X, kí hiệu: W-timestamp**(*X*), là giá trị lớn nhất trong các nhãn thời gian của các giao tác đã ghi X thành công.
- **Nhãn đọc X, kí hiệu: R-timestamp**(*X*), là giá trị lớn nhất trong các nhãn thời gian của các giao tác đọc X thành công

- Giao thức yêu cầu cấp nhãn thời gian cần đảm bảo bất kì cặp thao tác đọc/ghi xung đột nào cũng được thi hành theo đúng thứ tự nhãn thời gian.

5. Giao thức nhãn thời gian

- Khi giao tác T_i đưa ra yêu cầu đọc X: **read**(X)

1. Nếu TS(T_i) < **W**-timestamp(X), khi đó T_i cần đọc mục dữ liệu X mà gía trị này đã bị cập nhật. Vì vậy, thao tác đọc phải bị huỷ bỏ, T_i phục hồi (rolled back) và sẽ bắt đầu lại với nhãn thời gian mới.

2. Nếu TS(T_i)> **W**-timestamp(X), thao tác đọc được thi hành và nhãn đọc R-timestamp(X) gán bằng Max{R-timestamp(X); TS(T_i)}

5. Giao thức nhãn thời gian

- Khi giao tác T_i đưa ra yêu cầu **write**(X).

1. Nếu TS(T_i) < R-timestamp(X), giá trị của X mà T_i sẽ cập nhật đã "quá hạn" (đáng ra cần cập nhật từ trước). Khi đó, thao tác **write** bị huỷ bỏ, và T_i phải quay lui (rolled back)
2. Nếu TS(T_i) < W-timestamp(X), T_i đang cố ghi một giá trị đã "lỗi thời" của X. Thao tác **write** bị huỷ bỏ, và T_i phải quay lui
3. Ngược lại, thao tác **write** được thi hành và nhãn ghi W-timestamp(X) được đặt lại là TS(T_i).

5. Giao thức nhãn thời gian

Ví dụ: Một lịch với các giao tác được gán nhãn thời gian 1, 2, 3, 4, 5

T_1	T_2	T_3	T_4	T_5
				read(X)
	read(Y)			
read(Y)				
		write(Y)		
		write(Z)		
				read(Z)
	write(X)			
	abort			
read(X)				
			write(Z)	
			abort	
				write(Y)
				write(Z)

5. Giao thức nhãn thời gian

Tính đúng đắn của giao thức gán nhãn thời gian

- Giao thức gán nhãn thời gian đảm bảo tính khả tuần tự vì tất cả các cung trong đồ thị tuần tự hoá (precedence graph) đều có dạng:

Vì vậy, sẽ không có chu trình trong đồ thị

- Giao thức nhãn thời gian đảm bảo tránh được nghẽn khoá, vì không có giao tác nào phải đợi
- Có thể xảy ra hiện tượng quay lui lan truyền và sinh ra lịch không có khả năng phục hồi.

5. Giao thức nhãn thời gian

- Vấn đề với giao thức gán nhãn thời gian:
  - Giả sử T_i bị huỷ bỏ (abort), nhưng T_j đã đọc mục dữ liệu được ghi bởi T_i
  - Khi đó, T_j phải huỷ bỏ; nếu T_j đã được phép chuyển giao (commit) sớm hơn, thì lịch không thể phục hồi.
  - Hơn nữa, bất kì giao tác nào đã đọc mục dữ liệu được ghi bởi T_j cũng phải huỷ bỏ (abort)
  - Điều này dẫn đến quay lui lan truyền (cascading rollback)
- Giải pháp:
  - Một giao tác được cấu trúc sao cho các thao tác ghi của nó đều được thực hiện tại thời điểm cuối cùng
  - Tất cả các thao tác ghi của một giao tác được nhóm lại thành một thao tác nguyên tố; không một giao tác nào được thi hành trong khi một giao tác đang ghi;
  - Một giao tác bị huỷ bỏ sẽ khởi động lại với nhãn thời gian mới

5. Giao thức nhãn thời gian
Luật ghi Thomas (Thomas' Write Rule)

- Sửa đổi giao thức gán nhãn thời gian: các thao tác ghi quá hạn có thể được bỏ qua trong một số tình huống nhất định.
- Khi T_i cố ghi mục dữ liệu X, nếu TS(T_i) < W-timestamp(X), thì T_i đang cố ghi một giá trị đã quá hạn (lỗi thời) của X. Vì thế, thay vì quay lui T_i, thao tác ghi này có thể bỏ qua.
- Trong các trường hợp khác, giao thức gán nhãn thời gian không thay đổi.
- Luật ghi Thomas đem lại khả năng tương tranh cao hơn; cho phép một số lịch khả tuần tự trực quan nhưng không khả tuần tự xung đột.

5. Giao thức nhãn thời gian
Luật ghi Thomas (Thomas' Write Rule)

- Khi giao tác T_i đưa ra yêu cầu **write**(X).

1. Nếu TS(T_i) < R-timestamp(X), giá trị mà T_i cần ghi vào mục dữ liệu X đã được đọc bởi một giao tác có nhãn thời gian lớn hơn nên thao tác **write** bị huỷ bỏ, và T_i phải quay lui (rolled back), bắt đầu lại với nhãn thời gian mới
2. Nếu TS(T_i) < W-timestamp(X) T_i đang cố ghi một giá trị đã quá hạn vào X. Nhưng trong tình huống TS(T_i)≥ R-timestamp(X), thao tác **write** sẽ bị bỏ qua (loại ra khỏi giao tác)
3. Ngược lại, thao tác **write** được thi hành và nhãn ghi W-timestamp(X) được đặt lại là TS(T_i).