Trường Đại Học Bách Khoa Tp.HCM
Hệ Đào Tạo Từ Xa
Khoa Khoa Học và Kỹ Thuật Máy Tính

HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU

Chương 6. Các kỹ thuật điều khiển tương tranh (concurrency control)

Bùi Hoài Thắng

Nội dung

* Các kỹ thuật khóa hai pha để điều khiển tương tranh
* Điều khiển tương tranh dựa vào thứ tự tem thời gian
* Các kỹ thuật điều khiển tương tranh đa phiên bản
* Các kỹ thuật điều khiển tương tranh kiểm chứng
* Độ mịn của mục tin và kỹ thuật khóa nhiều độ mịn
* Thể hiện của các kỹ thuật này trên SQL-Server

BÀI 1 – TỔNG QUAN

Mục đích của điều khiển tương tranh

* Mục đích:
  - Thi hành tính năng đơn lập (Isolation) (nhờ tính loại trừ lẫn nhau) giữa các giao tác có xung đột
  - Duy trì tính nhất quán dữ liệu nhờ việc thực thi bảo toàn tính nhất quán cho các giao tác
  - Giải quyết các xung đột read-write và write-write
* Ví dụ:
  - Trong môi trường tương tranh, nếu T1 xung đột với T2 trên mục liệu A, thì bộ điều khiển tương tranh sẽ quyết định là T1 hoặc T2 sẽ truy cập A và giao tác còn lại sẽ đợi hoặc bị quay ngược

Kỹ thuật khoá

Kỹ thuật dùng khoá (Locking Technique)

* Khoá (locking):
  - Là tác vụ đảm bảo (a) quyền đọc (permission to Read) hoặc (b) quyền ghi (permission to Write) một mục liệu cho một giao tác
  - Ví dụ: Lock(X) – mục liệu X bị khoá dành cho giao tác yêu cầu (khoá)
* Mở khoá (unlocking)
  - Là tác vụ huỷ bỏ quyền (đọc hoặc ghi) trên một mục liệu
  - Ví dụ: Unlock(X) – mục liệu X được chuyển sang trạng thái sẵn sàng để dùng cho tất cả các giao tác
* Tác vụ khoá và mở khoá là các tác vụ đơn thể

Kỹ thuật dùng khoá (tt.)

* Yêu cầu về chỉnh dạng (well-formed) :
  - CSDL đòi hỏi là tất cả các giao tác cần phải là chỉnh dạng
* Giao tác chỉnh dạng (well-formed transaction):
  - Cần phải khoá mục liệu trước khi đọc hay ghi mục liệu này
  - Không được khoá một mục liệu đã khoá (bởi chính nó) và không được mở khoá một mục liệu không bị khoá

Khoá nhị phân (Binary lock)

* Mỗi khoá có hai giá trị:
  - (a) đã khoá – locked
  - (b) không bị khoá – unlocked

Giải thuật khoá và mở khoá nhị phân

Giải thuật LOCK_ITEM(X)

```
B:  if LOCK (X) = 0 (*item is unlocked*)
    then LOCK (X) ← 1 (*lock the item*)
    else begin
         wait (until lock (X) = 0 and
         the lock manager wakes up the transaction);
    goto B
    end;
```

Giải thuật UNLOCK_ITEM(X)

```
LOCK (X) ← 0 (*unlock the item*)
if any transactions are waiting then
     wake up one of the waiting transactions;
```

Luật về khoá nhị phân

* 1. Giao tác T phải dùng tác vụ lock_item(X) trước bất kỳ lệnh read_item(X) hoặc write_item(X) nào trong T
* 2. Giao tác T phải dùng tác vụ unlock_item(X) sau tất cả các lệnh read_item(X) và write_item(X) đã hoàn thành trong T
* 3. Giao tác T không dùng lệnh lock_item(X) nếu nó đang khoá mục liệu X
* 4. Giao tác T không dùng lệnh unlock_itern(X) trừ khi nó đang khoá mục liệu X

Khoá chia sẻ/loại trừ (Shared/Exclusive lock)

* Còn gọi là khoá đọc/ghi (Read/Write lock)
* Có hai chế độ khoá (lock mode):
  - (a) chia sẻ (shared hoặc read mode): shared_lock (X)
    - Các giao tác có thể dùng share lock cùng lúc để đọc mục liệu X
    - Không thể dùng write lock (để ghi dữ liệu) trên X tại cùng thời điểm
  - (b) loại trừ (exclusive hoặc write mode): write_lock (X)
    - Chỉ có tối đa một write lock trên mục liệu X tại bất kỳ thời điểm nào
    - Không thể có thêm share lock nào khác trên X tại cùng thời điểm
  - Ma trận xung đột:

	Read	Write
Read	Y	N
Write	N	N

Quản lý khoá (lock management)

* Lock Manager: bộ quản lý các khoá trên các mục liệu
* Lock table – bảng khoá
  - Bộ quản lý khoá dùng bảng này để lưu các thông tin về các khoá: danh định giao tác khoá (transaction ID), mục liệu (data item), kiểu khoá (lock mode) và con trỏ đến mục liệu bị khoá kế tiếp (pointer to next data item).
  - Một cách đơn giản là hiện thực bằng một danh sách liên kết

Transaction ID	Data item ID	Lock mode	Ptr to next data item
T1	X1	Read	Next

Giải thuật khoá và mở khoá (*)

Giải thuật read_lock(X)

```
B: if LOCK (X) = "unlocked" then
       begin LOCK (X) ← "read-locked";
           no_of_reads (X) ← 1;
       end
   else if LOCK (X) = "read-locked" then
        no_of_reads (X) ← no_of_reads (X) +1
    else begin wait (until LOCK (X) = "unlocked" and
           the lock manager wakes up the transaction);
        go to B
   end;
```

Giải thuật khoá và mở khoá (tt.)

Giải thuật write_lock(X)

```
B: if LOCK (X) = "unlocked" then
        LOCK (X) ← "write-locked";
   else begin
        wait (until LOCK (X) = "unlocked" and
         the lock manager wakes up the transaction);
        go to B
   end;
```

Giải thuật khoá và mở khoá (tt.)

Giải thuật unlock(X)

```
if LOCK (X) = "write-locked" then begin
    LOCK (X) ← "unlocked";
    wakes up one of the transactions, if any
end
else if LOCK (X) = "read-locked" then
    begin
        no_of_reads (X) ← no_of_reads (X) -1
        if  no_of_reads (X) = 0 then
        begin
         LOCK (X) = "unlocked";
        wake up one of the transactions, if any
        end
    end;
```

Luật về khoá chia sẻ/loại trừ

* 1. Giao tác T phải dùng tác vụ read_lock(X) hoặc write_lock(X) trước bất kỳ tác vụ read_item(X) nào trong T
* 2. Giao tác T phải dùng tác vụ write_lock(X) trước bất kỳ tác vụ write_item(X) nào trong T
* 3. Giao tác T phải dùng tác vụ unlock(X) sau tất cả các tác vụ read_item(X) và write_item(X) đã hoàn thành trong T
* 4. Giao tác T không được dùng tác vụ read_lock(X) hoặc write_lock(X) nếu nó đang khoá đọc (chia sẻ - read_lock) hoặc ghi (loại trừ - write_lock) trên mục liệu X
* 5. Giao tác T không được dùng tác vụ unlock(X) trừ khi nó đang khoá đọc (chia sẻ - read_lock) hoặc ghi (loại trừ - write_lock) trên mục liệu X

Chuyển đổi khoá

* Có thể cho phép chuyển đổi khoá:
  - Đang giữ khoá đọc, nâng thành khoá ghi (upgrade)

if T_i has a read-lock (X) and T_j has no read-lock (X) (i ≠ j) then
    convert read-lock (X) to write-lock (X)
else
    force T_i to wait until T_j unlocks X

  - Đang giữ khoá ghi, chuyển xuống thành khoá đọc (downgrade)

T_i has a write-lock (X) (*no transaction can have any lock on X*)
    convert write-lock (X) to read-lock (X)

Khoá hai pha

Kỹ thuật khoá hai pha (Two-phase locking techniques)

* Hai pha (Two Phases):
  - (a) Pha khoá (Locking) hay pha giãn (Growing)
    - Giao tác tiến hành khoá (đọc hay ghi - read or write) các mục liệu cần thiết từng cái một
  - (b) Pha mở khoá (Unlocking) hay pha co (Shrinking)
    - Giao tác tiến hành mở khoá các mục liệu từng cái một
* Yêu cầu:
  - Trong một giao tác, hai pha này cần phải loại trừ lẫn nhau:
    - Trong pha khoá, không có tác vụ mở khoá nào
    - Trong pha mở khoá, không có tác vụ khoá nào
* Viết tắt: 2PL

Kỹ thuật khoá hai pha (tt.)

T1	**T2**	**Kết quả**
read_lock (Y);	read_lock (X);	Giá trị ban đầu: X=20; Y=30
read_item (Y);	read_item (X);	
unlock (Y);	unlock (X);	Nếu thực hiện tuần tự T_1 rồi T_2
write_lock (X);	write_lock (Y);	X=50, Y=80
read_item (X);	read_item (Y);	
X:=X+Y;	Y:=X+Y;	Nếu thực hiện tuần tự T_2 rồi T_1
write_item (X);	write_item (Y);	X=70, Y=50
unlock (X);	unlock (Y);	

Ví dụ dùng kỹ thuật khoá không phải hai pha

Hình 6.1

Kỹ thuật khoá hai pha (tt.)

T1	T2
read_lock (Y);	
read_item (Y);	
unlock (Y);	
	read_lock (X);
	read_item (X);
	unlock (X);
	write_lock (Y);
	read_item (Y);
	Y:=X+Y;
	write_item (Y);
	unlock (Y);
write_lock (X);	
read_item (X);	
X:=X+Y;	
write_item (X);	
unlock (X);	

Tương tranh: Kết quả X=50; Y=50
Không phải khả tuần tự hoá

Hình 6.2

Kỹ thuật khoá hai pha (tt.)

T'1	T'2
read_lock (Y);	read_lock (X);
read_item (Y);	read_item (X);
unlock (Y);	**unlock (X);**
write_lock (X);	**write_lock (Y);**
read_item (X);	read_item (Y);
X:=X+Y;	Y:=X+Y;
write_item (X);	write_item (Y);
unlock (X);	unlock (Y);

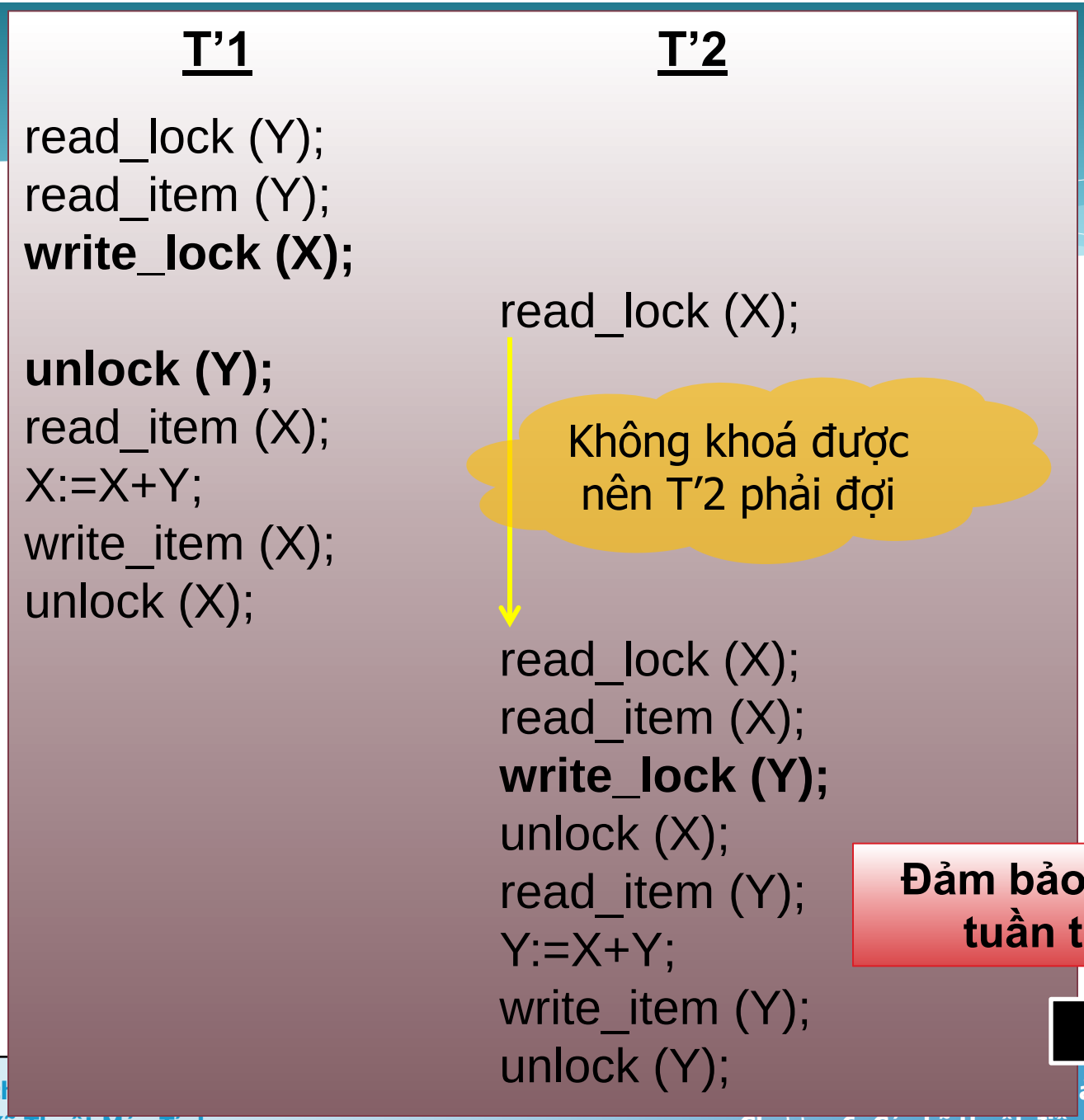
Dùng kỹ thuật khoá hai pha

Hình 6.3

Kỹ thuật khoá hai pha (tt.)

* Nhận xét:
  - Nếu tất cả các giao tác trong lịch biểu đều dùng kỹ thuật khoá hai pha, lịch biểu được đảm bảo sẽ là khả tuần tự hoá
* Chứng minh:
  - ???
* Không cần phải kiểm tra tính khả tuần tự hoá
* Giảm mức độ tương tranh:
  - T1 đang khoá X và đã dùng xong
  - T1 cần khoá thêm Y, Z, … để tiếp tục các tác vụ
  - T1 không mở khoá X vì đang ở pha khoá
  - Những giao tác khác cần truy cập X buộc phải đợi T1 chuyển qua pha mở khoá (và mở khoá X) mới có thể khoá X được

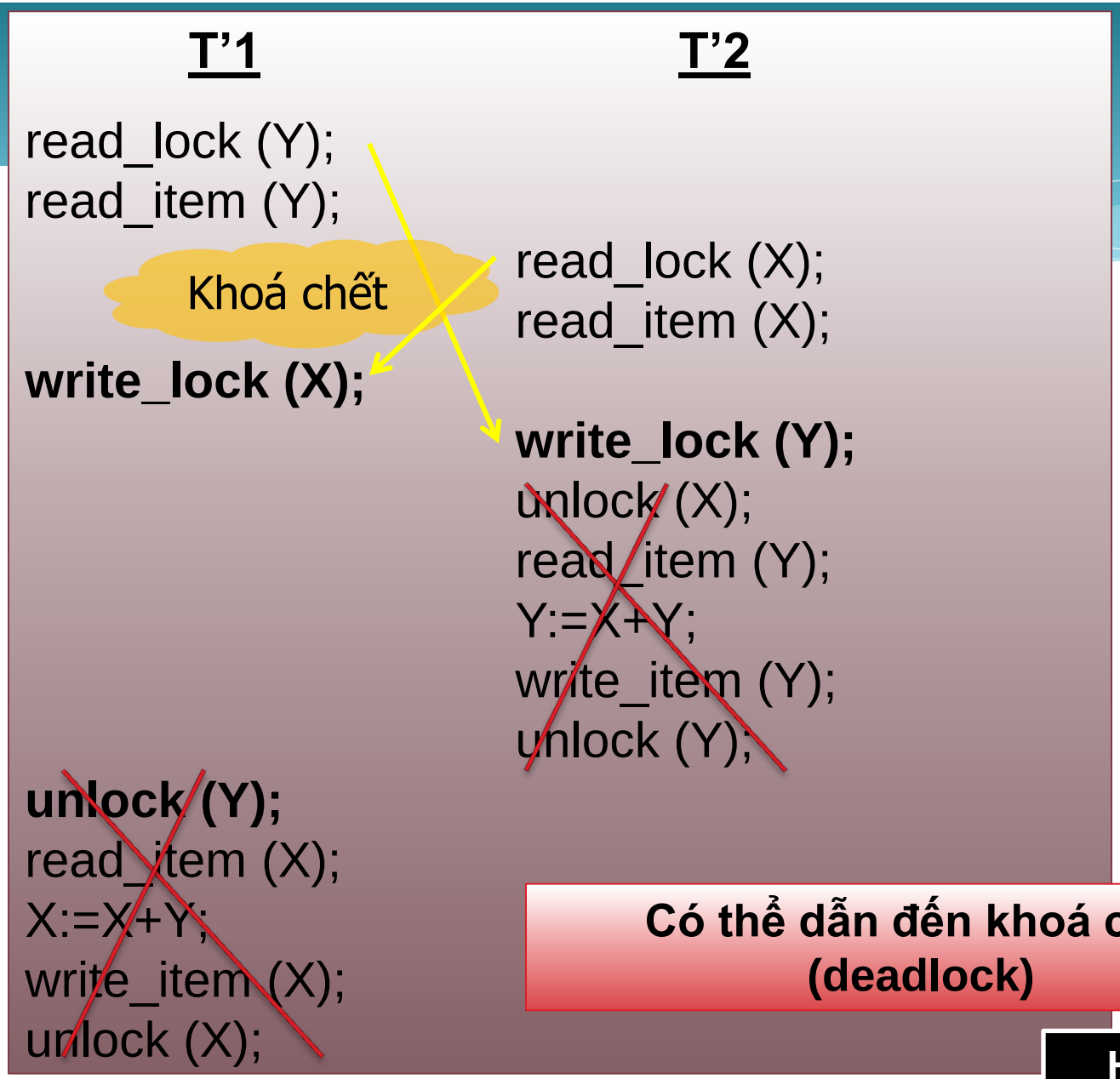
Kỹ thuật khoá hai pha (tt.)



Đảm bảo tính khả tuần tự hoá

Hình 6.4

Kỹ thuật khoá hai pha (tt.)



Hình 6.5

Giải thuật khoá hai pha bảo thủ (Conservative 2PL)

- Conservative/static 2PL – kỹ thuật khoá hai pha bảo thủ/tĩnh:
- Ngăn chặn khoá chết (deadlock) bằng cách khoá tất cả các mục liệu cần thiết trước khi bắt đầu thực hiện giao tác
- Cách thức:
  - Khai báo hai tập các mục liệu cần đọc và cần ghi cho mỗi giao tác:
  - Tiến hành khoá các mục liệu trong hai tập này
  - Nếu không khoá được bất kỳ một mục liệu nào, huỷ toàn bộ các khoá và đợi để khoá lại từ đầu
- Là giao tác phi khoá chết (deadlock-free):
  - Khoá được thì thực hiện trọn vẹn giao tác
- Không thực tế !

Giải thuật khoá hai pha nghiêm cách (Strict 2PL)

* Đòi hỏi giao tác không mở khoá ghi (write or exclusive lock) của nó cho đến khi kết thúc (bằng commit hoặc abort)
* Như vậy, không giao tác nào có thể đọc hoặc ghi các mục liệu đã ghi bởi giao tác này cho đến khi nó kết thúc
  - Thoả lịch biểu khả khôi phục nghiêm cách (strict schedule)
* Không phải là giao tác phi khoá chết (not deadlock-free)

Giải thuật khoá hai pha khắt khe (Rigorous 2PL)

* Khắt khe hơn Strict 2PL
* Đòi hỏi các giao tác chỉ mở khoá (đọc hoặc ghi) khi kết thúc (bằng commit hoặc abort)
* Thoả mãn lịch biểu nghiêm cách, nhưng dễ hiện thực hơn

Giải quyết khoá chết (Deadlock resolution)

* Mặc dù T'1 và T'2 tuân thủ khoá hai pha nhưng lại gặp vấn đề khoá chết

T'1	T'2
read_lock (Y);	
read_item (Y);	
	read_lock (X);
	read_item (X);
write_lock (X);	
(đợi T'2 mở khoá X)	
	write_lock (Y);
	(đợi T'1 mở khoá Y)

Hình 6.6

Ngăn chặn khoá chết (Deadlock prevention)

* Mỗi giao tác phải khoá tất cả các mục liệu cần thiết trước khi bắt đầu thực thi
* Phi khoá chết (deadlock-free) vì giao tác không phải đợi mục liệu nào
* Conservative 2PL dùng phương án này

Phát hiện khoá chết (Deadlock detection) để giải quyết

* Cho phép khoá chết xảy ra
* Bộ định lịch biểu (scheduler) duy trì một đồ thị đợi (wait-for-graph) để phát hiện chu trình
* Nếu có chu trình (tức có khoá chết), một giao tác nào đó (gọi là vật hy sinh - victim) trong chu trình sẽ được chọn để quay ngược (rollback)

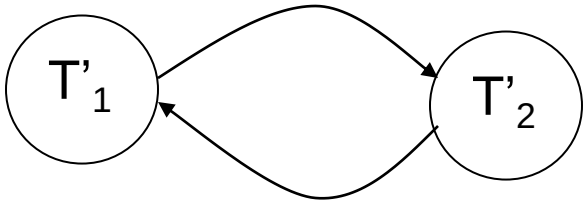
Đồ thị đợi (wait-for graph)

* Đồ thị đợi (wait-for-graph):
  - Mỗi giao tác xuất hiện trong lịch biểu là một đỉnh cùng tên
  - Khi giao tác T_i đợi để khoá mục liệu X đang khoá bởi giao tác T_j, thêm cạnh $T_i \rightarrow T_j$ vào đồ thị
  - Chu trình: $T_i \rightarrow T_j \rightarrow T_k \rightarrow \ldots \rightarrow T_i$

* Câu hỏi:
  - Khi nào nên kiểm tra chu trình trong một đồ thị đợi ?

Đồ thị đợi (wait-for graph)

T'1	T'2
read_lock (Y);	
read_item (Y);	
	read_lock (X);
	read_item (X);
write_lock (X);	
(đợi T'2 mở khoá X)	
	write_lock (Y);
	(đợi T'1 mở khoá Y)



Đồ thị đợi của T'1 và T'2

Hình 6.7

Tránh khoá chết (Deadlock avoidance)

* Không cho phép chu trình xuất hiện: khi phát hiện một giao tác có nguy cơ gây ra chu trình, quay ngược giao tác này
* Giải thuật Wound-Wait và Wait-Die sử dụng tem thời gian (timestamp) để tránh khoá chết
* Tem thời gian (timestamp):
  - Là một danh định (identifier) duy nhất gán cho mỗi giao tác
  - Thường dựa trên thứ tự xuất hiện (khởi động) của giao tác
  - Tem thời gian của T viết tắt là TS(T)
  - Nếu giao tác T1 bắt đầu trước giao tác T2, thì TS(T1) < TS(T2)
  - Giao tác vào trước (older) sẽ có TS nhỏ hơn giao tác vào sau (younger)

Tránh khoá chết (tt.)

- **Wait-die**:
  - Nếu $TS(T_i) < TS(T_j)$, tức T_i vào trước T_j, T_i sẽ đợi
  - Ngược lại, tức T_i vào sau T_j, huỷ T_i và khởi động lại sau với cùng một tem thời gian
- **Wound-wait**:
  - Nếu $TS(T_i) < TS(T_j)$, tức T_i vào trước T_j, huỷ T_j và khởi động lại sau với cùng một tem thời gian
  - Ngược lại, tức T_i vào sau T_j, T_i sẽ đợi

Tình trạng đợi quá lâu (Starvation)

* Tình trạng đợi quá lâu (starvation) xảy ra khi một giao tác cứ tiếp tục đợi hoặc bị tái khởi động và không bao giờ có cơ hội được thi hành (về sau)
* Trong một giải pháp giải quyết khoá chết, một giao tác có thể bị lựa chọn là vật hy sinh mãi để quay ngược
  - Có thể do bộ định lịch biểu dùng nguyên lý ưu tiên
* Kỹ thuật Wait-die và Wound-wait có thể giúp tránh tình trạng này

Dùng tem thời gian

Dùng tem thời gian điều khiển tương tranh

* Giải thuật điều khiển tương tranh dựa trên tem thời gian (Timestamp-based Concurrency control):
  - Dùng tem thời gian để tuần tự hoá các thực thi của các giao tác tương tranh
* Nguyên tắc tuần tự hoá:
  - Giao tác bắt đầu trước (có tem thời gian nhỏ hơn) sẽ được thi hành trước nếu thi hành tuần tự

Các tem thời gian

* Mỗi mục liệu CSDL sẽ được gán hai tem thời gian:
  - Read_TS(X) – tem thời gian đọc của mục liệu X:
    - Là tem thời gian lớn nhất của tất cả các tem thời gian của các giao tác đã đọc mục liệu X
  - Write_TS(X) – tem thời gian ghi của mục liệu X:
    - Là tem thời gian lớn nhất của tất cả các tem thời gian của các giao tác đã ghi mục liệu X
* Một cách dễ hiểu:
  - Ghi nhận giao tác trẻ nhất nào đọc (ghi) lên từng mục liệu
    - Không phải thời gian gần nhất đọc (ghi) mục liệu

Xếp thứ tự tem thời gian cơ bản (Basic Timestamp Ordering)

* Khi một giao tác T cần đọc hoặc ghi mục liệu X, giải thuật so sánh tem thời gian của T và read_TS(X) và write_TS(X) để chắc rằng thứ tự tem thời gian của giao tác đang thực thi không vi phạm (nguyên lý chung)
* Nếu nó vi phạm, giao tác T sẽ bị huỷ và sẽ được khởi động lại với *một tem thời gian mới*
* Nếu T bị huỷ và quay ngược, các giao tác T' đã dùng các giá trị ghi bởi T phải bị quay ngược.
* Tương tự các giao tác T'' đã dùng các giá trị ghi bởi T' cũng phải bị quay ngược, …
* Vấn đề này được gọi là quay ngược dắt dây

Xếp thứ tự tem thời gian cơ bản (tt.)

1. Khi giao tác *T* cần thực hiện tác vụ *write_item*(*X*):
   - Nếu *read_TS*(*X*) > *TS*(*T*) hoặc *write_TS*(*X*) > *TS*(*T*), huỷ và quay ngược *T* và từ chối tác vụ ghi này
     - nghĩa là có một giao tác trẻ hơn (vào sau) đã đọc hoặc ghi mục liệu *X*, tức *vi phạm thứ tự thực thi về tem thời gian*
   - Ngược lại, thi hành tác vụ *write_item*(*X*) của *T* và gán *write_TS*(*X*) bằng *TS*(*T*)
2. Khi giao tác *T* cần thực hiện tác vụ *read_item*(*X*):
   - Nếu *write_TS*(*X*) > *TS*(*T*), huỷ và quay ngược *T* và từ chối tác vụ đọc này
     - nghĩa là có một giáo tác trẻ hơn đã ghi mục liệu *X*, tức *vi phạm thứ tự thực thi về tem thời gian*
   - Ngược lại, thi hành tác vụ *read_item*(*X*) của *T* và gán *read_TS*(*X*) là giá trị lớn hơn giữa *TS*(*T*) và giá trị *read_TS*(*X*) hiện tại

Ví dụ xếp thứ tự tem thời gian cơ bản

T1	T2	T3	A	B	C
200	150	175	RT =0 WT=0	RT = 0 WT=0	RT=0 WT=0
r1(B) w1(B) w1(A)	 r2(A) w2(C) **abort**	 r3(C) w3(A) ???	 RT = 150 WT=200	RT = 200 WT=200	 RT=175

Hình 6.8

Xếp thứ tự tem thời gian cơ bản (tt.)

* Nhận xét:
  - Mỗi khi giải thuật phát hiện hai tác vụ có xung đột xuất hiện không đúng thứ tự (theo tem thời gian), giải thuật từ chối tác vụ đến sau bằng cách huỷ bỏ giao tác tạo ra tác vụ này
  - Như vậy, lịch biểu tạo ra bởi giải thuật này bảo đảm đặc tính khả tuần tự hoá đụng độ như trong giải thuật 2PL
  - Khoá chết không xảy ra nếu dùng giải thuật này
  - Sự chờ đợi quá lâu (starvation) có thể xảy ra khi một giao tác có thể bị huỷ bỏ và tái khởi động liên tục

Xếp thứ tự tem thời gian nghiêm cách (Strict Timestamp Ordering)

* Là giải thuật đảm bảo là lịch biểu tạo ra vừa nghiêm cách (khả khôi phục) vừa khả tuần tự hoá (xung đột)
* Khi giao tác T cần thực hiện tác vụ read_item(X) hoặc write_item(X) mà TS(T) > write_TS(X) thì sẽ đợi cho đến khi giao tác T' nào ghi lên X (TS(T') = write_TS(X)) commit hoặc abort
  - Cần phải mô phỏng khoá cho mục liệu X này cho đến khi T' kết thúc
* Giải thuật này không gây ra khoá chết, vì T đợi T' chỉ khi TS(T) > TS(T'), tức T vào sau T'

Xếp thứ tự tem thời gian nghiêm cách (tt.)

* Luật ghi của Thomas (Thomas's Write Rule)
  - Kiểm tra mỗi khi thực hiện tác vụ write_item(X)
  - Không đảm bảo khả tuần tự hoá, nhưng từ chối ít lệnh ghi hơn
* Luật:
  - (1) Nếu read_TS(X) > TS(T) thì huỷ và quay ngược T và từ chối tác vụ ghi
  - (2) Nếu write_TS(X) > TS(T), thì bỏ qua tác vụ ghi này và tiếp tục (thi hành các lệnh khác trong giao tác)
    - Vì lệnh ghi của giao tác trẻ hơn đã ghi lên kết quả của lệnh ghi này
  - Ngược lại, nếu (1) và (2) không đúng, thực thi tác vụ write_item(X) của T và gán write_TS(X) bằng TS(T)

Ví dụ xếp thứ tự tem thời gian nghiêm cách

T1	T2	T3	A	B	C
200	150	175	RT =0 WT=0	RT = 0 WT=0	RT=0 WT=0
r1(B) w1(B) w1(A)	 r2(A) w2(C) **abort**	 r3(C) w3(A) **OK**	 RT = 150 WT=200	RT = 200 WT=200	 RT=175

Hình 6.9

Tóm tắt bài 1

- Tổng quan về kỹ thuật dùng khoá
  - Khoá nhị phân
  - Khoá read/write
- Các kỹ thuật khóa hai pha để điều khiển tương tranh
  - Khoá hai pha
  - Khoá hai pha bảo thủ
  - Khoá hai pha nghiêm ngặt/khắt khe
  - Khoá chết và đợi quá lâu
- Điều khiển tương tranh dựa vào thứ tự tem thời gian
  - Tem thời gian
  - Xếp thứ tự tem thời gian cơ bản
  - Xếp thứ tự tem thời gian nghiêm cách