# CHƯƠNG V
# ĐIỀU KHIỂN CẠNH TRANH
# (Concurrency Control)

**MỤC ĐÍCH**

Một trong các tính chất cơ bản của một giao dịch là *tính cô lập*. Khi một vài giao dịch thực hiện một cách cạnh tranh trong CSDL, tính cô lập có thể không được bảo tồn. Đối với hệ thống, cần phải điều khiển sự trao đổi giữa các giao dịch cạnh tranh; sự điều khiển này được thực hiện thông qua một trong tập hợp đa dạng các cơ chế được gọi là *sơ đồ điều khiển cạnh tranh*.

Các sơ đồ điều khiển cạnh tranh được xét trong chương này được dựa trên tính khả tuần tự. Trong chương này ta cũng xét sự quản trị các giao dịch thực hiện cạnh canh nhưng không xét đến sự cố hỏng hóc.

**YÊU CẦU**

Hiểu các khái niệm

Hiểu các kỹ thuật điều khiển cạnh tranh:

- Các kỹ thuật dựa trên chốt (lock)
- Các kỹ thuật dựa trên tem thởi gian
- Các kỹ thuật hỗp hợp

Hiểu nguyên lý của các kỹ thuật này

Hiểu các kỹ thuật điều khiển deadlock

# V.1. GIAO THỨC DỰA TRÊN CHỐT

Một phương pháp để đảm bảo tính khả tuần tự là yêu cầu việc truy xuất đến hạng mục dữ liệu được tiến hành theo kiểu loại trừ tương hỗ; có nghĩa là trong khi một giao dịch đang truy xuất một hạng mục dữ liệu, không một giao dịch nào khác có thể sửa đổi hạng mục này. Phương pháp chung nhất được dùng để thực thi yêu cầu này là cho phép một giao dịch truy xuất một hạng mục dữ liệu chỉ nếu nó đang giữ chốt trên hạng mục dữ liệu này.

## V.1.1. CHỐT (Lock)

Có nhiều phương thức chốt hạng mục dữ liệu. Ta hạn chế việc nghiên cứu trên hai phương thức:

1. **Shared.** Nếu một giao dịch $T_i$ nhận được một chốt ở phương thức shared (ký hiệu là S) trên hạng mục Q, khi đó $T_i$ có thể đọc, nhưng không được viết Q.
2. **Exclusive.** Nếu một giao dịch $T_i$ nhận được một chốt ở phương thức Exclusive (ký hiệu là X), khi đó $T_i$ có thể cả đọc lẫn viết Q.

Ta yêu cầu là mỗi giao dịch đòi hỏi một chốt ở một phương thức thích hợp trên hạng mục dữ liệu Q, phụ thuộc vào kiểu hoạt động mà nó sẽ thực hiện trên Q. Giả sử một giao dịch $T_i$ đồi hỏi một chốt phương thức A trên hạng mục Q mà trên nó giao dịch $T_j$ ($T_j \neq T_i$) hiện đang giữ một chốt phương thức B. Nếu giao dịch $T_i$ có thể được cấp một chốt trên Q ngay, bất chấp sự hiện diện của chốt phương thức B, khi đó ta nói phương thức A tương thích với phương thức B. Một hàm như vậy có thể được biểu diễn bởi một ma trận. Quan hệ tương thích giữa hai phương thức chốt được cho bởi ma trận **comp** sau:

| | S | X |
|---|---|---|
| S | True | False |
| X | False | False |

Comp(A, B)= true có nghĩa là các phương thức A và B tương thích.

***figure V- 1***

Các chốt phương thức **shared** có thể được giữ đồng thời trên một hạng mục dữ liệu. Một chốt **exclusive** đến sau phải chờ đến tận khi tất cả các chốt phương thức **shared** đến trước được tháo ra.

Một giao dịch yêu cầu một chốt **shared** trên hạng mục dữ liệu Q bằng cách thực hiện chỉ thị **lock-S(Q)**, yêu cầu một chốt **exclusive** thông qua chỉ thị **lock-X(Q)**. Một hạng mục dữ liệu Q có thể được tháo chốt thông qua chỉ thị **unlock(Q)**.

Để truy xuất một hạng mục dữ liệu, giao dịch $T_i$ đầu tiên phải chốt hạng mục này. Nếu hạng mục này đã bị chốt bởi một giao dịch khác ở phương thức không tương thích, bộ điều khiển cạnh tranh sẽ không cấp chốt cho đến tận khi tất cả các chốt không tương thích bị giữ bởi các giao dịch khác được tháo. Như vậy $T_i$ phải chờ đến tận khi tất cả các chốt không tương thích bị giữ bởi các giao dịch khác được giải phóng.

Giao dịch $T_i$ có thể tháo chốt một hạng mục dữ liệu mà nó đã chốt trước đây. Một giao dịch cần thiết phải giữ một chốt trên một hạng mục dữ liệu chừng nào mà nó còn truy xuất hạng mục này. Hơn nữa, đối với một giao dịch việc tháo chốt ngay sau truy xuất cuối cùng đến hạng mục dữ liệu không luôn luôn là điều mong muốn vì như vậy tính khả tuần tự có thể không được đảm bảo. Để minh hoạ cho tình huống này, ta xét ví dụ sau: A và B là hai tài khoản có thể được

truy xuất bởi các giao dịch $T_1$ và $T_2$ . Giao dịch $T_1$ chuyển 50$ từ tài khoản B sang tài khoản A và đươch xác định như sau:

$T_1$ : **Lock-X(B);**
**Read(B);**
**B:=B-50;**
**Write(B);**
**Unlock(B);**
**Lock-X(A);**
**Read(A);**
**A:=A+50;**
**Write(A);**
**Unlock(A);**

***figure V- 2***

Giao dịch $T_2$ hiển thị tổng số lượng tiền trong các tài khoản A và B (A + B) và được xác định như sau;

$T_2$ : **Lock-S(A);**
**Read(A);**
**Unlock(A);**
**Lock-S(B);**
**Read(B);**
**Unlock(B);**
**Display(A+B);**

***figure V- 3***

Giả sử giá trị của tài khoản A và B tương ứng là 100$ và 200$. Nếu hai giao dịch này thực hiện tuần tự, hoặc theo thứ tự $T_1$, $T_2$ hoặc theo thứ tự $T_2$ , $T_1$ , và khi dó $T_2$ sẽ hiển thị giá trị 300$. Tuy nhiên nếu các giao dịch này thực hiện cạnh tranh, giả sử theo lịch trình schedule-1, trong trường hợp như vậy giao dịch $T_2$ sẽ hiển thị giá trị 250$ --- một kết quả không đúng. Lý do của sai lầm này là do giao dịch $T_1$ đã tháo chốt hạng mục B quá sớm và $T_2$ đã tham khảo một trạng thái không nhất quán !!!

Lịch trình schedule 1 bày tỏ các hành động được thực hiện bởi các giao dịch cũng như các thời điểm khi các chốt được cấp bởi bộ quản trị điều khiển cạnh tranh. Giao dịch đưa ra một yêu cầu chốt không thể thực hiện hành động kế của mình đến tận khi chốt được cấp bởi bộ quản trị điều khiển cạnh tranh; do đó, chốt phải được cấp trong khoảng thời gian giữa hoạt động yêu cầu chốt và hành động sau của giao dịch. Sau này ta sẽ luôn giả thiết chốt được cấp cho giao dịch ngay trước hành động kế và như vậy ta có thể bỏ qua cột bộ quản trị điều khiển cạnh tranh trong bảng

| $T_1$ | $T_2$ | Bộ quản trị điều khiển cạnh tranh |
|---|---|---|
| **Lock-X(B)** | | |
| | | **Grant-X(B,$T_1$)** |
| **Read(B)** | | |
| **B:=B-50** | | |
| **Write(B)** | | |
| **Unlock(B)** | | |
| | **Lock-S(A)** | |
| | | **Grant-S(A,$T_2$)** |
| | **Read(A)** | |
| | **Unlock(A)** | |
| | **Lock-S(B)** | |
| | | **Grant-S(B,$T_2$)** |
| | **Read(B)** | |
| | **Unlock(B)** | |
| | **Display(A+B)** | |
| **Lock-X(A)** | | |
| | | **Grant-X(A,$T_1$)** |
| **Read(A)** | | |
| **A:=A+50** | | |
| **Write(A)** | | |
| **Unlock(A)** | | |

***Schedule-1***

***figure V- 4***

Bây giờ giả sử rằng tháo chốt bị làm trễ đến cuối giao dịch. Giao dịch $T_3$ tương ứng với $T_1$ với tháo chốt bị làm trễ được định nghĩa như sau:

$T_3$ : **Lock-X(B);**
**Read(B);**
**B:=B-50;**
**Write(B);**
**Lock-X(A);**
**Read(A);**
**A:=A+50;**
**Write(A);**
**Unlock(B);**
**Unlock(A);**

***figure V- 5***

Giao dịch $T_4$ tương ứng với $T_2$ với tháo chốt bị làm trễ được xác định như sau:

$T_4$ : **Lock-S(A);**
**Read(A);**
**Lock-S(B);**
**Read(B);**
**Display(A+B);**
**Unlock(A);**
**Unlock(B);**

***figure V- 6***

Các lịch trình có thể trên $T_3$ và $T_4$ không để cho $T_4$ hiển thị trạng thái không nhất quán.

Tuy nhiên, sử dụng chốt có thể dẫn đến một tình huống không mong đợi. Ta hãy xét lịch trình bộ phận schedule-2 trên $T_3$ và $T_4$ sau:

| $T_3$ | $T_4$ |
|---|---|
| **Lock-X(B)** | |
| **Read(B)** | |
| **B:=B-50** | |
| **Write(B)** | |
| | **Lock-S(A)** |
| | **Read(A)** |
| | **Lock-S(B)** |
| **Lock-X(A)** | |

***Schedule-2***

***figure V- 7***

Do $T_3$ giữ một chốt phương thức Exclusive trên B, nên yêu cầu một chốt phương thức shared của $T_4$ trên B phải chờ đến khi $T_3$ tháo chốt. Cũng vậy, $T_3$ yêu cầu một chốt Exclusive trên A trong khi $T_4$ đang giữ một chốt shared trên nó và như vậy phải chờ. Ta gặp phải tình huống trong đó $T_3$ chờ đợi $T_4$ đồng thời $T_4$ chờ đợi $T_3$ -- một sự chờ đợi vòng tròn -- và như vậy không giao dịch nào có thể tiến triển. Tình huống này được gọi là deadlock (khoá chết). Khi tình huống khoá chết xảy ra hệ thống buộc phải cuộn lại một trong các giao dịch. Mỗi khi một giao dịch bị cuộn lại, các hạng mục dữ liệu bị chốt bởi giao dịch phải được tháo chốt và nó trở nên sẵn có cho giao dịch khác, như vậy các giao dịch này có thể tiếp tục được sự thực hiện của nó.

Nếu ta không sử dụng chốt hoặc tháo chốt hạng mục dữ liệu ngay khi có thể sau đọc hoặc viết hạng mục, ta có thể rơi vào trạng thái không nhất quán. Mặt khác, nếu ta không tháo chốt một hạng mục dữ liệu trước khi yêu cầu một chốt trên một hạng mục khác, dealock có thể xảy ra. Có các phương pháp tránh dealock trong một số tình huống, tuy nhiên nói chung dealock là khó tránh khi sử dụng chốt nếu ta muốn tránh trạng thái không nhất quán. Dealock được ưa thích hơn trạng thái không nhất quán vì chúng có thể điều khiển được bằng cách cuộn lại các giao dịch trong khi

đó trạng thái không nhất quán có thể dẫn đến các vấn đề thực tế mà hệ CSDL không thể điều khiển.

Ta sẽ yêu cầu mỗi giao dịch trong hệ thống tuân theo một tập các quy tắc , được gọi là giao thức chốt (locking protocol), chỉ định khi một giao dịch có thể chốt và tháo chốt mỗi một trong các hạng mục dự liệu. Giao thức chốt hạn chế số các lịch trình có thể. Tập các lịch trình như vậy là một tập con thực sự của tập tất cả các lịch trình khả tuần tự có thể.

Xét { $T_0$ , $T_1$ , ..., $T_n$ } một tập các giao dịch tham gia vào lịch trình S. Ta nói $T_i$ đi trước $T_j$ trong S, và được viết là $T_i \rightarrow T_j$ , nếu tồn tại một hạng mục dữ liệu Q sao cho $T_i$ giữ chốt phương thức A trên Q , $T_j$ giữ chốt phương thức B trên Q muộn hơn và **comp(A,B)** = false. Nếu $T_i \rightarrow T_j$ , thì $T_i$ sẽ xuất hiện trước $T_j$ trong bất kỳ lịch trình tuần tự nào.

Ta nói một lịch trình S là hợp lệ dưới một giao thức chốt nếu S là một lịch trình tuân thủ các quy tắc của giao thức chốt đó. Ta nói rằng một giao thức chốt đảm bảo tính khả tuần tự xung đột nếu và chỉ nếu đối với tất cả các lịch trình hợp lệ, quan hệ $\rightarrow$ kết hợp là phi chu trình.

## V.1.2. CẤP CHỐT

Khi một giao dịch yêu cầu một chốt trên một hạng mục dữ liệu ở một phương thức và không có một giao dịch nào khác giữ một chốt trên cùng hạng mục này ở một phương thức xung đột, chốt có thể được cấp. Tuy nhiên, phải thận trọng để tránh kịch bản sau: giả sử $T_2$ giữ một chốt phương thức shared trên một hạng mục dữ liệu, một giao dịch khác $T_1$ yêu cầu một chốt phương thức exclusive cũng trên hạng mục này, rõ ràng $T_1$ phải chờ $T_2$ tháo chốt. Trong khi đó một giao dịch khác $T_3$ yêu cầu một chốt phương thức shared, do yêu cầu chốt này tương thích với phương thức chốt được giữ bởi $T_1$ nên nó được cấp cho $T_3$. Tại thời điểm $T_2$ tháo chốt, $T_1$ vẫn phải chờ sự tháo chốt của $T_3$, nhưng bây giờ lại có một giao dịch $T_4$ yêu cầu một chốt phương thức shared và nó lại được cấp do tính tương thích và cứ như vậy, có thể $T_1$ sẽ không bao giờ được cấp chốt mà nó yêu cầu trên hạng mục dữ liệu. Ta gọi hiện tượng này là bị chết đói (starved).

Để tránh sự chết đói của các giao dịch, việc cấp chốt được tiến hành như sau: Khi một giao dịch $T_i$ yêu cầu một chốt trên một hạng mục dữ liệu Q ở phương thức M, chốt sẽ được cấp nếu các điều kiện sau được thoả mãn:

1. Không có giao dịch khác đang giữ một chốt trên Q ở phương thức xung đột với M
2. Không có một giao dịch nào đang chờ được cấp một chốt trên M và đã đưa ra yêu cầu về chốt trước $T_i$

## V.1.3. GIAO THỨC CHỐT HAI KỲ (Two-phase locking protocol)

Giao thức chốt hai kỳ là một giao thức đảm bảo tính khả tuần tự. Giao thức này yêu cầu mỗi một giao dịch phát ra yêu cầu chốt và tháo chốt thành hai kỳ:

1. **Kỳ xin chốt (Growing phase).** Một giao dịch có thể nhận được các chốt, nhưng có không thể tháo bất kỳ chốt nào
2. **Kỳ tháo chốt (Shrinking phase).** Một giao dịch có thể tháo các chốt nhưng không thể nhận được một chốt mới nào.

Khởi đầu, một giao dịch ở kỳ xin chốt. Giao dịch tậu được nhiều chốt như cần thiết. Mỗi khi giao dịch tháo một chốt, nó đi vào kỳ tháo chốt và nó không thể phát ra bất kỳ một yêu cầu chốt nào nữa. Các giao dich $T_3$ và $T_4$ là hai kỳ. Các giao dịch $T_1$ và $T_2$ không là hai kỳ. Người ta có thể chứng minh được giao thức chốt hai kỳ đảm bảo tính khả tuần tự xung đột, nhưng không đảm bảo tránh được dealock và việc cuộn lại hàng loạt. Cuộn lại hàng loạt có thể tránh được bởi một sự sửa đổi chốt hai kỳ được gọi là giao thức chốt hai kỳ nghiêm ngặt. Chốt hai kỳ nghiêm

ngặt đòi hỏi thêm tất cả các chốt phương thức exclusive phải được giữ đến tận khi giao dịch bàn giao. Yêu cầu này đảm bảo rằng bất kỳ dữ liệu nào được viết bởi một giao dịch chưa bàn giao bị chốt trong phương thức exclusive đến tận khi giao dịch bàn giao, điều đó ngăn ngừa bất kỳ giao dịch khác đọc dữ liệu này.

Một biến thể khác của chốt hai kỳ là giao thức chốt hai kỳ nghiêm khắc. Nó đòi hỏi tất cả các chốt được giữ đến tận khi giao dịch bàn giao. Hầu hết các hệ CSDL thực hiện chốt hai kỳ nghiêm ngặt hoặc nghiêm khắc.

Một sự tinh chế giao thức chốt hai kỳ cơ sở dựa trên việc cho phép chuyển đổi chốt: nâng cấp một chốt shared sang exclusive và hạ cấp một chốt exclusive thành chốt shared. Chuyển đổi chốt không thể cho phép một cách tuỳ tiện, nâng cấp chỉ được phép diễn ra trong kỳ xin chốt, còn hạ cấp chỉ được diễn ra trong kỳ tháo chốt. Một giao dịch thử nâng cấp một chốt trên một hạng mục dữ liệu Q có thể phải chờ. Giao thức chốt hai kỳ với chuyển đổi chốt cho phép chỉ sinh ra các lịch trình khả tuần tự xung đột. Nếu các chốt exclusive được giữ đến tận khi bàn giao, các lịch trình sẽ là cascadeless.

Ta xét một ví dụ: Các giao dịch $T_8$ và $T_9$ được nêu trong ví dụ chỉ được trình bày bởi các hoạt động ý nghĩa là **Read** và **Write.**

$T_8$ : **Read($A_1$);**
**Read($A_2$);**
**...**
**Read($A_n$);**
**Write($A_1$).**

$T_9$ : **Read($A_1$);**
**Read($A_2$);**
**Display($A_1 + A_2$).**

***figure V- 8***

Nếu ta sử dụng giao thức chốt hai kỳ, khi đó $T_8$ phải chốt $A_1$ ở phương thức exclusive. Bởi vậy, sự thực hiện cạnh tranh của hai giao dịch rút cuộc trở thành thực hiện tuần tự. Ta thấy rằng $T_8$ cần một chốt exclusive trên $A_1$ chỉ ở cuối sự thực hiện của nó, khi nó write($A_1$). Như vậy, $T_8$ có thể khởi động chốt $A_1$ ở phương thức shared, và đổi chốt này sang phương thức exclusive sau này. Như vậy ta có thể nhận được tính cạnh tranh cao hơn, vì như vậy $T_8$ và $T_9$ có thể truy xuất đến $A_1$ và $A_2$ đồng thời.

Ta biểu thị sự chuyển đổi từ phương thức shared sang phương thức exclusive bởi upgrade và từ phương thức exclusive sang phương thức shared bởi downgrade. Upgrade chỉ được phép xảy ra trong kỳ xin chốt và downgrade chỉ được phép xảy ra trong kỳ tháo chốt. Lịch trình chưa hoàn tất dưới đây cho ta một minh hoạ về giao thức chốt hai kỳ với chuyển đổi chốt.

Chú ý rằng một giao dịch thử cập nhật một chốt trên một hạng mục dữ liệu Q có thể buộc phải chờ. Việc chờ bắt buộc này xảy ra khi Q đang bị chốt bởi giao dịch khác ở phương thức shared.

*Giao thức chốt hai kỳ với chuyển đổi chốt chỉ sinh ra các lịch trình khả tuần tự xung đột, các giao dịch có thể được tuần tự hoá bởi các điểm chốt của chúng. Hơn nữa, nếu các chốt exclusive được giữ đến tận khi kết thúc giao dịch, lịch trình sẽ là cascadeless*.

| $T_8$ | $T_9$ |
|---|---|
| **Lock-S($A_1$)** | |
| | **Lock-S($A_2$)** |
| **Lock-S($A_2$)** | |
| | **Lock-S($A_2$)** |
| **Lock-S($A_3$)** | |
| **…** | |
| | **Unlock($A_1$)** |
| | **Unlock($A_2$)** |
| **UpGrade($A_1$)** | |

***figure V- 9***

Ta mô tả một sơ đồ đơn giản nhưng dược sử dụng rộng rãi để sinh tự động các chỉ thị chốt và tháo chốt thích hợp cho một giao dịch: Mỗi khi giao dich T xuất ra một chỉ thị **Read(Q),** hệ thống sẽ xuất ra một chỉ thị **Lock-S(Q)** ngay trước chỉ thị **Read(Q)**. Mỗi khi giao dịch T xuất ra một hoạt động **Write(Q),** hệ thống sẽ kiểm tra xem T đã giữ một chốt shared nào trên Q hay chưa, nếu đã, nó xuất ra một chỉ thị **Upgrade(Q)** ngay trước chỉ thị **Write(Q),** nếu chưa, nó xuất ra chỉ thị **Lock-X(Q)** ngay trước **Write(Q).** Tất cả các chốt giao dịch nhận được sẽ được tháo chốt sau khi giao dịch bàn giao hay bỏ dở.

## V.1.4. GIAO THỨC DỰA TRÊN ĐỒ THỊ (Graph-Based Protocol)

Ta đã biết, trong trường hợp thiếu vắng các thông tin liên quan đến cách thức các hạng mục dữ liệu được truy xuất, giao thức chốt hai kỳ là cần và đủ để đảm bảo tính khả tuần tự. Nếu ta muốn phát triển các giao thức không là hai kỳ, ta cần các thông tin bổ xung trên cách thức mỗi giao dịch truy xuất CSDL. Có nhiều mô hình khác nhau về lượng thông tin được cung cấp. Mô hình đơn giản nhất đòi hỏi ta phải biết trước thứ tự trong đó các hạng mục dữ liệu sẽ được truy xuất. Với các thông tin như vậy, có thể xây dựng các giao thức chốt không là hai kỳ nhưng vẫn đảm bảo tính khả tuần tự xung đột.

Để có được hiểu biết trước như vậy, ta áp đặt một thứ tự bộ phận, ký hiệu $\rightarrow$, trên tập tất cả các hạng mục dữ liệu D =$\{ d_1 , d_2 , ..., d_n \}$. Nếu $d_i \rightarrow d_j$ , bất kỳ giao dịch nào truy xuất cả $d_i$ và $d_j$ phải truy xuất $d_i$ trước khi truy xuất $d_j$ . Thứ tự bộ phận này cho phép xem D như một đồ thị định hướng phi chu trình, được gọi là đồ thị CSDL (DataBase Graph). Trong phần này, để đơn giản, ta hạn chế chỉ xét các đồ thị là các cây và ta sẽ đưa ra một giao thức đơn giản, được gọi là giao thức cây (tree protocol), giao thức này hạn chế chỉ dùng các chốt exclusive.

Trong giao thức cây, chỉ cho phép chỉ thị chốt **Lock-X**, mỗi giao dịch T có thể chốt một hạng mục dữ liệu nhiều nhất một lần và phải tuân theo các quy tắc sau:

1. Chốt đầu tiên bởi T có thể trên bất kỳ hạng mục dữ liệu nào
2. Sau đó, một hạng mục dữ liệu Q có thể bị chốt bởi T chỉ nếu cha của Q hiện đang bị chốt bởi T
3. Các hạng mục dữ liệu có thể được tháo chốt bất kỳ lúc nào

4. Một hạng mục dữ liệu đã bị chốt và được tháo chốt bởi T, không thể bị T chốt lại lần nữa.

*Các lịch trình hợp lệ trong giao thức cây là khả tuần tự xung đột*.

Ví dụ: Cây CSDL là:

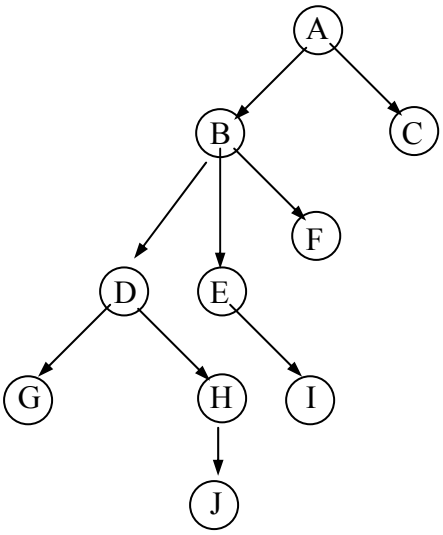



***figure V- 10***

Chỉ các chỉ thị chốt và tháo chốt của các giao dịch được trình bày:

$T_{10}$ : **Lock-X(B); Lock-X(E); Lock-X(D); Unlock(B); Unlock(E); Lock-X(G); Unlock(D); Unlock(G).**

$T_{11}$ : **Lock-X(D); Lock-X(H); Unlock(D); Unlock(H).**

$T_{12}$ : **Lock-X(B); Lock-X(E); Unlock(B); Unlock(E).**

$T_{13}$ : **Lock-X(D); Lock-X(H); Unlock(D); Unlock(H).**

***figure V- 11***

Một lịch trình tuân theo giao thức cây chứa tất cả bốn giao dịch trên được cho trong hình bên dưới. Ta nhận thấy, các lịch trình tuân thủ giao thức cây không chỉ là khả tuần tự xung đột mà còn đảm bảo không có dealock. Giao thức cây có mặt thuận lợi so với giao thức hai kỳ là tháo chốt có thể xảy ra sớm hơn. Việc tháo chốt sớm có thể dẫn đến rút ngắn thời gian chờ đợi và tăng tính cạnh tranh. Hơn nữa, do giao thức là không dealock, nên không có cuộn lại. Tuy nhiên giao thức cây có điểm bất lợi là, trong một vài trường hợp, một giao dịch có thể phải chốt những hạng mục dữ liệu mà nó không truy xuất. Chẳng hạn, một giao dịch cần truy xuất các hạng mục dữ liệu A và J trong đồ thị CSDL trên, phải chốt không chỉ A và J mà phải chốt cả các hạng mục B, D, H. Việc chốt bổ xung này có thể gây ra việc tăng tổng phí chốt, tăng thời gian chờ đợi và giảm tính cạnh tranh. Hơn nữa, nếu không biết trước các hạng mục dữ liệu nào sẽ cần thiết phải chốt, các giao dịch sẽ phải chốt gốc của cây mà điều này làm giảm mạnh tính cạnh tranh.

Đối với một tập các giao dịch, có thể có các lịch trình khả tuần tự xung đột không thể nhận được từ việc tuân theo giao thức cây. Có các lịch trình được sinh ra bởi tuân theo giao thức chốt hai kỳ nhưng không thể được sinh ra bởi tuân theo giao thức cây và ngược lại.

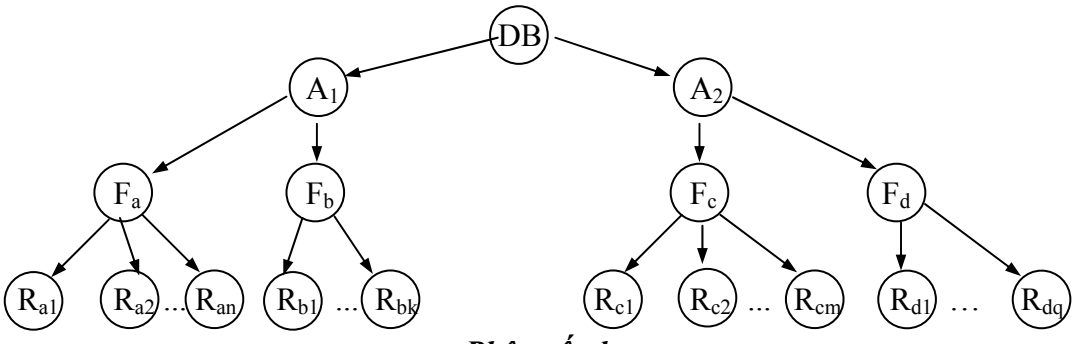
| $T_{10}$ | $T_{11}$ | $T_{12}$ | $T_{13}$ |
|---|---|---|---|
| **Lock-X(B)** | | | |
| | **Lock-X(D)** | | |
| | **Lock-X(H)** | | |
| | **Unlock(D)** | | |
| **Lock-X(E)** | | | |
| **Lock-X(D)** | | | |
| **Unlock(B)** | | | |
| **Unlock(E)** | | | |
| | | **Lock-X(B)** | |
| | | **Lock-X(E)** | |
| | **Unlock(H)** | | |
| **Lock-X(G)** | | | |
| **Unlock(D)** | | | |
| | | | **Lock-X(D)** |
| | | | **Lock-X(H)** |
| | | | **Unlock(D)** |
| | | | **Unlock(H)** |
| | | **Unlock(E)** | |
| | | **Unlock(B)** | |
| **Unlock(G)** | | | |

***Lịch trình khả tuần tự dưới giao thức cây***

***figure V- 12***

## V.1.5. ĐA HẠT (Multiple Granularity)

Trong các sơ đồ điều khiển cạnh tranh được mô tả trước đây, ta đã sử dụng hạng mục dữ liệu như đơn vị trên nó sự đồng bộ hoá được thực hiện. Tuy nhiên, có các hoàn cảnh trong đó việc nhóm một vài hạng mục dữ liệu và xử lý chúng như một đơn vị đồng bộ hoá mang lại nhiều lợi ích. Nừu một giao dich $T_i$ phải truy xuất toàn bộ CSDL và giao thức chốt được sử dụng, khi đó $T_i$ phải chốt mỗi hạng mục dữ liệu trong CSDL. Như vậy việc thực hiện các chốt này sẽ tiêu tốn một thời gian đáng kể. Sẽ hiệu quả hơn nếu $T_i$ chỉ cần một yêu cầu chốt để chốt toàn bộ CSDL. Mặt khác, nếu $T_i$ cần truy xuất chỉ một vài hạng mục dữ liệu, nó không cần thiết phải chốt toàn bộ CSDL vì như vậy sẽ giảm tính cạnh tranh. Như vậy, cái mà ta cần là một cơ chế cho phép hệ thống xác định nhiều mức hạt. Một cơ chế như vậy là cho phép các hạng mục dữ liệu có kích cỡ khác nhau và xác định một sự phân cấp các hạt dữ liệu, trong đó các hạt nhỏ được ẩn náu bên trong các hạt lớn. Sự phân cấp như vậy có thể được biểu diễn đồ thị như một cây. Một nút không là lá của cây đa hạt biểu diễn dữ liệu được kết hợp với con cháu của nó. Như một ví dụ minh hoạ, ta xét cây sau:




***Phân cấp hạt***

***figure V- 13***

Nó gồm bốn mức nút. Mức cao nhất là toàn bộ CSDL. Thấp hơn là các nút kiểu vùng: CSDL bao gồm các vùng này. Mỗi vùng lại có các nút kiểu file như các con của nó, mỗi vùng chứa đúng các file này và không file nào nằm trong nhiều hơn một vùng. Cuối cùng, mỗi file có các nút con kiểu mẩu tin, không mẩu tin nào hiện diện trong hơn một file.

Mỗi nút trong cây có thể được chốt một các cá nhân. Như đã làm trong giao thức chốt hai kỳ, ta sẽ sử dụng các phương thức chốt **shared** và **exclusive** . Khi một giao dịch chốt một nút, trong phương thức **shared** hoặc **exclusive**, giao dịch cũng chốt tất cả các nút con cháu của nút này ở cùng phương thức. Ví dụ $T_i$ chốt tường minh file $F_b$ ở phương thức exclusive, nó đã chốt ẩn tất cả các mẩu tin của $F_b$ cũng trong phương thức exclusive.

Giả sử giao dịch $T_j$ muốn chốt mẩu tin $R_{b6}$ của file $F_b$. vì giao dịch $T_i$ đã chốt tường minh file $F_b$, mẩu tin $R_{b6}$ cũng bị chốt ẩn. Song làm thế nào để hệ thống biết được $T_j$ có thể chốt $R_{b6}$ hay không: $T_j$ phải duyệt cây từ gốc đến mẩu tin $R_{b6}$ , nếu có một nút bất kỳ tren đường dẫn bị chốt ở phương thức không tương thích, $T_j$ phải chờ. Bây giờ, nếu $T_k$ muốn chốt toàn bộ CSDL, nó phải chốt nút gốc. Tuy nhiên, do $T_i$ hiện đang giữ một chốt trên $F_b$, một bộ phận của cây, nên $T_k$ sẽ không thành công. Vậy làm thế nào để hệ có thể chốt được nút gốc: Một khả năng là tìm kiếm trên toàn bộ cây, giải pháp này phá huỷ hoàn toàn sơ đồ mục đích của sơ đồ chốt đa hạt. Một giải pháp hiệu quả hơn là đưa vào một lớp mới các phương thức chốt, được gọi là phương thức chốt tăng cường (intension lock mode). Nếu một nút bị chốt ở phương thức tăng cường, chốt tường minh được tiến hành ở mức thấp hơn của cây (hạt min hơn). Chốt tăng cường được được đặt trên tất cả các tổ tiên của một nút trước khi nút đó được chốt tường minh. Như vậy một giao dịch không cần thiết phải tìm kiếm toàn bộ cây để xác định nó có thể chốt một nút thành công hay không. Một giao dịch muốn chốt một nút, chẳng hạn Q, phải duyệt một đường dẫn từ gốc đến Q, trong khi duyệt cây, giao dịch chốt các nút trên đường đi ở phương thức tăng cường.

Có một phương thức tăng cường kết hợp với phương thức **shared** và một với phương thức **exclusive**. Nếu một nút bị chốt ở phương thức tăng cường shared (IS), chốt tường minh được tiến hành ở mức thấp hơn trong cây, nhưng chỉ là một các chốt phương thức shared. Tương tự, nếu một nút bị chốt ở phương thức tăng cường exclusive (IX), chốt tường minh được tiến hành ở mức thấp hơn với các chốt exclusive hoặc shared. Nếu một nút bị chốt ở phương thức shared và phương thức tăng cường exclusive (SIX), cây con có gốc là nút này bị chốt tường minh ở phương thức shared và chốt tường minh được tiến hành ở mức thấp hơn với các chốt exclusive. Hàm tính tương thích đối với các phương thức chốt này được cho bởi ma trận:

| | IS | IX | S | SIX | X |
|---|---|---|---|---|---|
| IS | True | True | True | True | False |
| IX | True | True | False | False | False |
| S | True | False | True | False | False |
| SIX | True | False | False | False | False |
| X | False | False | False | False | False |

***figure V- 14***

Giao thức chốt đa hạt dưới đây đảm bảo tính khả tuần tự. Mỗi giao dịch T có thể chốt một nút Q theo các quy tắc sau:

1. Hàm tương thích chốt phải được kiểm chứng
2. Gốc của cây phải được chốt đầu tiên, và có thể được chốt ở bất kỳ phương thức nào
3. Một nút Q có thể được chốt bởi T ở phương thức S hoặc IS chỉ nếu cha của Q hiện đang bị chốt bởi T ở hoặc phương thức IX hoặc phương thức IS.
4. Một nút Q có thể được chốt bởi T ở phương thức X, SIX hoặc IX chỉ nếu cha của Q hiện đang bị chốt ở hoặc phương thức IX hoặc phương thức SIX
5. T có thể chốt một nút chỉ nếu trước đó nó chưa tháo chốt một nút nào.
6. T có thể tháo chốt một nút Q chỉ nếu không con nào của Q hiện đang bị chốt bởi T

*Ta thấy rằng giao thức đa hạt yêu cầu các chốt được tậu theo thứ tự Top-Down, được tháo theo thứ tự Bottom-Up*.

Ví dụ: Xét cây phân cấp hạt như trên và các giao dịch sau:

- Giả sử giao dịch $T_{18}$ đọc mẩu tin $R_{a2}$ của file $F_a$. Khi đó $T_{18}$ cần phải chốt DB, vùng $A_1$ và $F_a$ ở phương thức IS và $R_{a2}$ ở phương thức S: **Lock-IS(DB); Lock-IS($A_1$); Lock-IS($F_a$); lock-S($R_{a2}$)**
- Giả sử giao dịch $T_{19}$ sửa đổi mẩu tin $R_{a9}$ trong file $F_a$ , khi đó $T_{19}$ cần phải chốt CSDL, vùng $A_1$ và file $F_a$ ở phương thức IX và $R_{a9}$ ở phương thức X: **Lock-IX(DB); Lock-IX($A_1$); lock-IX($F_a$); lock-X($R_{a9}$)**
- Giả sử giao dịch $T_{20}$ đọc tất cả các mẩu tin của file $F_a$ , khi đó $T_{20}$ cần phải chốt CSDL, và vùng $A_1$ ở phương thức IS và chốt $F_a$ ở phương thức S: **Lock-IS(DB); Lock-IS($A_1$); Lock-S($F_a$)**
- Giả sử giao dịch $T_{21}$ đọc toàn bộ CSDL, nó có thể làm điều đó sau khi chốt CSDL ở phương thức S: **Lock-S(DB)**

Chú ý rằng $T_{18}$, $T_{20}$ và $T_{21}$ có thể truy xuất đồng thời CSDL, giao dịch $T_{19}$ có thể thực hiện cạnh tranh với $T_{18}$ nhưng không với $T_{20}$ hoặc $T_{21}$

# V.2. GIAO THỨC DỰA TRÊN TEM THỜI GIAN (Timestamp-based protocol)

## V.2.1. TEM THỜI GIAN (Timestamp)

Ta kết hợp với mỗi giao dịch $T_i$ trong hệ thống một tem thời gian cố định duy nhất, được biểu thị bởi $TS(T_i)$. Tem thời gian này được gán bởi hệ CSDL trước khi giao dịch $T_i$ bắt đầu thực

hiện. Nếu một giao dịch $T_i$ đã được gán tem thời gian $TS(T_i)$ và một giao dịch mới $T_j$ đi vào hệ thống, khi đó $TS(T_i) < TS(T_j)$. Có hai phương pháp đơn giản để thực hiện sơ đồ này:

1. Sử dụng giá trị của đồng hồ hệ thống như tem thời gian: Một tem thời gian của một giao dịch bằng giá trị của đồng hồ khi giao dịch đi vào hệ thống.
2. Sử dụng bộ đếm logic: bộ đếm được tăng lên mỗi khi một tem thời gian đã được gán, tem thời gian của một giao dịch bằng với giá trị của bộ đếm khi giao dich đi vào hệ thống.

Tem thời gian của các giao dịch xác định thứ tự khả tuần tự. Như vậy, *nếu $TS(T_i) < TS(T_j)$, hệ thống phải đảm bảo rằng lịch trình được sinh ra là tương đương với một lịch trình tuần tự trong đó $T_i$ xuất hiện trước $T_j$* .

Để thực hiện sơ đồ này, ta kết hợp với mỗi hạng mục dữ liệu Q hai giá trị tem thời gian:

- **W-timestamp(Q)** biểu thị tem thời gian lớn nhất của giao dịch bất kỳ đã thực hiện **Write(Q)** thành công
- **R-timestamp(Q)** biểu thị tem thời gian lớn nhất của giao dịch bất kỳ đã thực hiện **Read(Q)** thành công

Các tem thời gian này được cập nhật mỗi khi một **Write** hoặc một **Read** mới được thực hiện.

## V.2.2. GIAO THỨC THỨ TỰ TEM THỜI GIAN (Timestamp-Ordering Protocol)

Giao thức thứ tự tem thời gian đảm bảo rằng các **Write** và **Read** xung đột bất kỳ được thực hiện theo thứ tự tem thời gian. Giao thức này hoạt động như sau:

1. Giả sử giao dịch $T_i$ phát ra **Read(Q)**.
   a. Nếu $TS(T_i) <$ W-Timestamp(Q), $T_i$ cần đọc một giá trị của Q đã được viết rồi. Do đó, hoạt động **Read** bị vứt bỏ và $T_i$ bị cuộn lại.
   b. Nếu $TS(T_i) \geq$ W-Timestamp(Q), hoạt động **Read** được thực hiện và R-Timestamp được đặt bằng giá trị lớn nhất trong hai giá trị R-Timestamp và $TS(T_i)$.
2. Giả sử giao dịch $T_i$ phát ra **Write(Q)**.
   a. Nếu $TS(T_i) <$ R-Timestamp(Q), Giá trị của Q mà $T_i$ đang sinh ra được giả thiết là để được dùng cho các giao dịch đi sau nó (theo trình tự thời gian), nhưng nay không cần đến nữa. Do vậy, hoạt động **Write** này bị vứt bỏ và $T_i$ bị cuộn lại
   b. Nếu $TS(T_i) <$ W-Timestamp(Q), $T_i$ đang thử viết một giá trị đã quá hạn của Q, Từ đó, hoạt động **Write** bị vứt bỏ và $T_i$ bị cuộn lại
   c. Ngoài ra, hoạt động **Write** được thực hiện và W-Timestamp(Q) được đặt là $TS(T_i)$

Một giao dịch $T_i$ bị cuộn lại bởi sơ đồ điều khiển cạnh tranh như kết quả của hoạt động **Read** hoặc **Write** đang được phát ra, được gán với một tem thời gian mới và được tái khởi động lại (được xem như một giao dịch mới tham gia vào hệ thống)

Ta xét các giao dịch $T_{14}$ và $T_{15}$ được xác định như dưới đây:

$T_{14}$ : **Read(B);**
**Read(A);**
**Display(A+B);.**

$T_{15}$ : **Read(B);**
**B:=B-50;**
**Write(B);**
**Read(A);**
**A:=A+50;**
**Write(A);**
**Display(A+B).**

***figure V- 15***

Ta giả thiết rằng một giao dịch được gán cho một tem thời gian ngay trước chỉ thị đầu tiên của nó. Như vậy, lịch trình schedule-3 dưới đây có $TS(T_{14}) < TS(T_{15})$, và là một lịch trình hợp lệ dưới giao thức tem thời gian:

| $T_{14}$ | $T_{15}$ |
|---|---|
| **Read(B)** | |
| | **Read(B)** |
| | **B:=B-50** |
| | **Write(B)** |
| **Read(A)** | |
| | **Read(A)** |
| **Display(A+B)** | |
| | **A:=A+50** |
| | **Write(A)** |
| | **Display(A+B)** |

***Schedule-3***

***figure V- 16***

Giao thức thứ tự tem thời gian đảm bảo tính khả tuần tự xung đột và không dealock.

## V.2.3. QUY TẮC VIẾT THOMAS (Thomas' Write rule)

Một biến thể của giao thức tem thời gian cho phép tính cạnh tranh cao hơn giao thức thứ tự tem thời gian. Trước hết ta xét lịch trình schedule-4 sau:

| $T_{16}$ | $T_{17}$ |
|---|---|
| **Read(Q)** | |
| | **Write(Q)** |
| **Write(Q)** | |

***Schedule-4***

***figure V- 17***

Nếu áp dụng giao thức thứ tự tem thời gian, ta có $TS(T_{16}) < TS(T_{17})$. Hoạt dộng **Read(Q)** của $T_{16}$ và **Write(Q)** của $T_{17}$ thành công, khi $T_{16}$ toan thực hiện hoạt động **Write(Q)** của nó, vì $TS(T16) < TS(T_{17}) = W\text{-timestamp}(Q)$, nên **Write(Q)** của $T_{16}$ bị vứt bỏ và giao dịch $T_{16}$ bị cuộn lại. Sự cuộn lại này là không cần thiết. Nhận xét này cho ta một sửa đổi phiên bản giao thức thứ tự tem thời gian:

Các quy tắc giao thức đối với **Read** không thay đổi, các quy tắc đối với **Write** được thay đổi chút ít như sau:

Giả sử giao dịch $T_i$ phát ra **Write(Q).**

1. Nếu $TS(T_i) < R\text{-timestamp}(Q)$, Giá trị của Q mà $T_i$ đang sinh ra được giả thiết là để được dùng cho các giao dịch đi sau nó (theo trình tự thời gian), nhưng nay không cần đến nữa. Do vậy, hoạt động **Write** này bị vứt bỏ và $T_i$ bị cuộn lại.
2. Nếu $TS(T_i) < W\text{-timestamp}(Q)$, $T_i$ đang thử viết một giá trị lỗi thời của Q. Do vậy, hoạt động **Write** này có thể bị bỏ lơ (không được thực hiện, nhưng $T_i$ không bị cuộn lại).
3. Ngoài ra, hoạt động **Write** được thực hiện và $W\text{-timestamp}(Q)$ được đặt là $TS(T_i)$.

Sự sửa đổi đối với giao thức thứ tự tem thời gian này được gọi là quy tắc viết Thomas. Quy tắc viết Thomas cho khả năng sinh các lịch trình khả tuần tự mà các giao thức thức trước đây không thể.

## V.2.4. GIAO THỨC DỰA TRÊN TÍNH HỢP LỆ

Trong trường hợp đa số các giao dịch trong hệ thống là các giao dịch chỉ đọc (read-only), tỷ suất xung đột giữa các giao dịch là thấp. Như vậy nhiều giao dịch trong chúng thực hiện thiếu sự giám sát của sơ đồ điều khiển cạnh tranh cũng vẫn giữ cho hệ thống ở trạng thái nhất quán. Hơn nữa, một sơ đồ điều khiển cạnh tranh đưa vào một tổng phí đáng kể (cho thực hiện mã lệnh, thời gian chờ của giao dịch ...). Việc tìm một sơ đồ với tổng phí nhỏ là một mục tiêu. Nhưng khó khăn là ta phải biết trước những giao dịch sẽ bị dính líu vào một xung đột. Để có được các hiểu bét đó, ta cần một sơ đồ để giám sát hệ thống.

Ta giả thiết rằng mỗi giao dịch $T_i$ , trong thời gian sống của nó, thực hiện trong hai hoặc ba kỳ khác nhau, phụ thuộc vào nó là một giao dịch chỉ đọc hay là một giao dịch cập nhật. Các kỳ này, theo thứ tự, là như sau:

1. **Kỳ đọc.** Trong kỳ này, các giá trị của các hạng mục dữ liệu khác nhau được đọc vào các biến cục bộ của $T_i$ . Tất cả các hoạt động **Write** được thực hiện trên các biến cục bộ tạm, không cập nhật CSDL hiện hành.
2. **Kỳ hợp lệ.** Giao dịch $T_i$ thực hiện một phép kiểm thử sự hợp lệ để xác định xem nó có thể sao chép đến CSDL các biến cục bộ tạm chứa các kết quả của các hoạt **Write** mà không vi phạm tính khả tuần tự xung đột hay không.

3. **Kỳ viết.** Nếu $T_i$ thành công trong kỳ hợp lệ, các cập nhật hiện hành được áp dụng vào CSDL, nếu không $T_i$ bị cuộn lại.

Mỗi giao dịch phải trải qua ba kỳ theo thứ tự trên, tuy nhiên, ba kỳ của các giao dịch đang thực hiện cạnh tranh có thể đan xen nhau.

Các kỳ đọc và kỳ viết tự nó đã rõ ràng. Chỉ có kỳ hợp lệ là cần thảo luận thêm. Để thực hiện kiểm thử sự hợp lệ, ta cần biết khi nào các kỳ khác nhau của giao dịch $T_i$ xảy ra. Do vậy, ta sẽ kết hợp ba tem thời gian với giao dịch $T_i$ :

1. **Start($T_i$).** Thời gian khi $T_i$ bắt đầu sự thực hiện.
2. **Validation($T_i$).** Thời gian khi $T_i$ kết thúc kỳ đọc và khởi động kỳ hợp lệ.
3. **Finish($T_i$).** Thời gian khi $T_i$ kết thúc kỳ viết.

Ta xác định thứ tự khả tuần tự bởi kỹ thuật thứ tự tem thời gian sử dụng giá trị tem thời gian **Validation($T_i$).** Như vậy, giá trị $TS(T_i)$ = Validation($T_i$) và nếu $TS(T_j) < TS(T_k)$ thì bất kỳ lịch trình nào được sinh ra phải tương đương với một lịch trình tuần tự trong đó giao dịch $T_i$ xuất hiện trước giao dịch $T_k$ . Lý dota chọn Validation($T_i$) như tem thời gian của $T_i$ , mà không chọn Start($T_i$), là vì ta hy vọng thời gian trả lời sẽ nhanh hơn.

Phép kiểm thử hợp lệ đối với $T_j$ đòi hỏi rằng, đối với mỗi giao dịch $T_i$ với $TS(T_i) < TS(T_j)$, một trong các điều kiện sau phải được thoả mãn:

1. **Finish($T_i$) < Start($T_j$).** Do $T_i$ hoàn tất sự thực hiện của nó trước khi $T_j$ bắt đầu, thứ tự khả tuần tự được duy trì.
2. **Tập các hạng mục dữ liệu được viết bởi $T_i$ không giao với tập các hạng mục dữ liệu được đọc bởi $T_j$ và $T_i$ hoàn tất kỳ viết của nó trước khi $T_j$ bắt đầu kỳ hợp lệ (Start($T_j$) < Finish($T_i$) < Validation($T_j$)).** Điều kiện này đảm bảo rằng các viết của $T_i$ và $T_j$ là không chồng chéo. Do các viết của $T_i$ không ảnh hưởng tới đọc của $T_j$ và do $T_j$ không thể ảnh hưởng tới đọc của $T_i$ , thứ tự khả tuần tự được duy trì.

Lịch trình schedule-5 cho ta một minh hoạ về giao thức dựa trên tính hợp lệ:

| $T_{14}$ | $T_{15}$ |
|---|---|
| **Read(B)** | |
| | **Read(B)** |
| | **B:=B-50** |
| | **Read(A)** |
| | **A:=A+50** |
| **Read(A)** | |
| ***Xác nhận tính hợp lệ*** | |
| **Display(A+B)** | |
| | ***Xác nhận tính hợp lệ*** |
| | **Write(B)** |
| | **Write(A)** |

***figure V- 18***

Sơ đồ hợp lệ tự động canh chừng việc cuộn lại hàng loạt, do các **Write** hiện tại xảy ra chỉ sau khi giao dịch phát ra **Write** đã bàn giao.

# V.3. CÁC SƠ ĐỒ ĐA PHIÊN BẢN (Multiversion Schemes)

Các sơ đồ điều khiển cạnh tranh được thảo luận trước đây đảm bảo tính khả tuần tự hoặc bởi làm trễ một hoạt động hoặc bỏ dở giao dịc đã phát ra hoạt động đó. Chẳng hạn, một hoạt động **Read** có thể bị làm trễ vì giá trị thích hợp còn chưa được viết hoặc nó có thể bị vứt bỏ vì giá trị mà nó muốn đọc đã bị viết đè rồi. Các khó khăn này có thể được che đi nếu bản sao cũ của mỗi hạng mục dữ liệu được giữ trong một hệ thống.

Trong các hệ CSDL đa phiên bản, mỗi hoạt động **Write(Q)** tạo ra một bản mới của Q. Khi một hoạt động **Read(Q)** được phát ra, hệ thống chọn lựa một trong các phiên bản của Q để đọc. Sơ đồ điều khiển cạnh tranh phải đảm bảo rằng việc chọn lựa này được tiến hành sao cho tính khả tuần tự được đảm bảo. Do lý do hiệu năng, một giao dịch phải có khả năng xác định dễ dàng và mau chóng phiên bản dhạng mục dữ liệu sẽ đọc.

## V.3.1. THỨ TỰ TEM THỜI GIAN ĐA PHIÊN BẢN

Kỹ thuật chung được dùng trong các sơ đồ đa phiên bản là tem thời gian. Ta kết hợp với một giao dịch một tem thời gian tĩnh duy nhất, ký hiệu $TS(T_i)$. Tem thời gian này được gán trước khi khi giao dịch bắt đầu sự thực hiện. Mỗi hạng mục dữ liệu Q kết hợp với một dãy $<Q_1, Q_2, ... , Q_m>$ mỗi phiên bản $Q_k$ chứa ba trường dữ liệu:

- **Content** là giá trị của phiên bản $Q_i$
- **W-timestamp($Q_k$)** là tem thời gian của giao dịch đã tạo ra phiên bản $Q_k$
- **R-timestamp($Q_k$)** là tem thời gian lớn nhất của giao dịch đã đọc thành công phiên bản $Q_k$

Một giao dịch, gọi là $T_i$ , tạo ra phiên bản $Q_k$ của hạng mục dữ liệu Q bằng cách phát ra một hoạt động **Write(Q)**. Trường **Content** của phiên bản chứa giá trị được viết bởi $T_i$ . **W-timestamp** và **R-timestamp** được khởi động là $TS(T_i)$. Giá trị **R-timestamp** được cập nhật mỗi khi một giao dịch $T_j$ đọc nội dung của $Q_k$ và **R-timestamp($Q_k$)** < $TS(T_j)$.

Sơ đồ tem thời gian đa phiên bản dưới đây sẽ đảm bảo tính khả tuần tự. Sơ đồ hoạt động như sau: giả sử $T_j$ phát ra một hoạt động **Read(Q)** hoặc **Write(Q).** $Q_k$ ký hiệu phiên bản của Q, tem thời gian viết của nó là tem thời gian viết lớn nhất nhỏ hơn hoặc bằng $TS(T_j)$.

1. Nếu giao dịch $T_j$ phát ra một **Read(Q)**, khi đó giá trị trả lại là nội dung của phiên bản $Q_k$
2. Nếu $T_j$ phát ra một **Write(Q)** và nếu $TS(T_j)$ < **R-timestamp($Q_k$)** khi đó giao dịch $T_j$ bị cuộn lại. Nếu không, nếu $TS(T_j)$ = **W-timestamp($Q_k$)** nội dung của $Q_k$ bị viết đè, khác đi một phiên bản mới của Q được tạo.

Các phiên bản không còn được dùng đến nữa bị xoá đi dựa trên quy tắc sau: Giả sử có hai phiên bản $Q_i$ và $Q_j$ của một hạng mục dữ liệu và cả hai phiên bản này cùng có W-timestamp nhỏ hơn tem thời gian của giao dịch già nhất trong hệ thống, khi đó phiên bản già hơn trong hai phiên bản $Q_i$ và $Q_j$ sẽ không còn được dùng nữa và bị xoá đi.

Sơ đồ thứ tự tem thời gian đa phiên bản có tính chất hay đó là một yêu cầu **Read** không bao giờ thất bại và không phải chờ đợi. Trong một hệ thống mà hoạt động **Read** xảy ra nhiều hơn **Write** cái lợi này là đáng kể. Tuy nhiên có vài điều bất lợi của sơ đồ này là: thứ nhất đọc một

hạng mục dữ liệu cũng đòi hỏi cập nhật trường **R-timestamp**, thứ hai là xung đột giữa các giao dịch được giải quyết bằng cuộn lại.

### V.3.2. CHỐT HAI KỲ ĐA PHIÊN BẢN

Giao thức chốt hai kỳ đa phiên bản cố gắng tổ hợp những ưu điểm của điều khiển cạnh tranh với các ưu điểm của chốt hai kỳ. Giao thức này phân biệt các giao dịch ***chỉ đọc*** và các giao dịch ***cập nhật***.

Các giao dịch ***cập nhật*** thực hiện chốt hai kỳ nghiêm khắc (các chốt được giữ đến tận khi kết thúc giao dịch). Mỗi hạng mục dữ liệu có một tem thời gian. Tem thời gian trong trường hợp này không là tem thời gian dựa trên đồng hồ thực mà là một bộ đếm, sẽ được gọi là TS-counter.

Các giao dịch ***chỉ đọc*** được gán tem thời gian là giá trị hiện hành của TS-counter trước khi chúng bắt đầu sự thực hiện: chúng tuân theo giao thức thứ tự tem thời gian đa phiên bản để thực hiện đọc. Như vậy, khi một giao dịch chỉ đọc $T_i$ phát ra một **Read(Q),** giá trị trả lại là nội dung của phiên bản mà tem thời gian của nó là tem thời gian lớn nhất nhỏ hơn $TS(T_i)$.

Khi một giao dịch ***cập nhật*** đọc một hạng mục, nó tậu một chốt **shared** trên hạng mục, rồi đọc phiên bản mới nhất của hạng mục (đối với nó). Khi một giao dịch cập nhật muốn viết một hạng mục, đầu tiên nó tậu một chốt exclusive trên hạng mục này, rồi tạo ra một phiên bản mới cho hạng mục. **Write** được thực hiện trên phiên bản mới này và tem thời gian của phiên bản mới được khởi động là $+\infty$.

Khi một giao dịch cập nhật $T_i$ hoàn tất các hoạt động của nó, nó thực hiện xử lý bàn giao như sau: Đầu tiên, $T_i$ đặt tem thời gian trên mỗi phiên bản nó đã tạo là TS-counter+1; sau đó $T_i$ tăng TS-counter lên 1. Chỉ một giao dịch cập nhật được phép thực hiện xử lý bàn giao tại một thời điểm.

Các phiên bản bị xoá cùng kiểu cách với thứ tự tem thời gian đa phiên bản.

## V.4. QUẢN TRỊ DEADLOCK

Một hệ thống ở trạnh thái deadlock nếu tồn tại một tập hợp các giao dịch sao cho mỗi giao dịch trong tập hợp đang chờ một giao dịch khác trong tạp hợp. Chính xác hơn, tồn tại một tập các giao dịch $\{ T_0 , T_2 , ..., T_n \}$ sao cho $T_0$ đang chờ một hạng mục dữ liệu được giữ bởi $T_1$ , $T_1$ đang chờ một hạng mục dữ liệu đang bị chiếm bởi $T_2$ , ..., $T_{n-1}$ đang chờ một hạng mục dữ liệu được giữ bởi $T_n$ và $T_n$ đang chờ một hạng mục $T_0$ đang chiếm. Không một giao dịch nào có thể tiến triển được trong tình huống như vậy. Một cách chữa trị là viện dẫn một hành động tẩy rửa, chẳng hạn cuộn lại một vài giao dịch tham gia vào deadlock.

Có hai phương pháp chính giải quyết vấn đề deadlock: Ngăn ngừa deadlock, phát hiện deadlock và khôi phục. Giao thức ngăn ngừa deadlock đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock. Sơ đồ phát hiện deadlock và khôi phục (deadlock-detection and deadlock-recovery scheme) cho phép hệ thống đi vào trạng thái deadlock và sau đó cố gắng khôi phục. Cả hai phương pháp đều có thể dẫn đến việc cuộn lại giao dịch. Phòng ngừa deadlock thường được sử dụng nếu xác suất hệ thống đi vào deadlock cao, phát hiện và khôi phục hiệu quả hơn trong các trường hợp còn lại.

### V.4.1. PHÒNG NGỪA DEADLOCK (Deadlock prevention)

Có hai cách tiếp cận phòng ngừa deadlock: Một đảm bảo không có chờ đợi vòng tròn xảy ra bằng cách sắp thứ tự các yêu cầu chốt hoặc đòi hỏi tất cả các chốt được tậu cùng nhau. Một

cách tiếp cận khác gần hơn với khắc phục deadlock và thực hiện cuộn lại thay vì chờ đợi một chốt. Chờ đợi là tiềm ẩn của deadlock.

Sơ đồ đơn giản nhất dưới cách tiếp cận thứ nhất đòi hỏi mỗi giao dịch chốt tất cả các hạng mục dữ liệu trước khi nó bắt đầu thực hiện. Hơn nữa, hoặc tất cả được chốt trong một bước hoặc không hạng mục nào được chốt. Giao thức này có hai bất lợi chính: một là khó dự đoán, trước khi giao dịch bắt đầu, các hạng mục dữ liệu nào cần được chốt, hai là hiệu suất sử dụng hạng mục dữ liệu rất thấp do nhiều hạng mục có thể bị chốt nhưng không được sử dụng trong một thời gian dài.

Sơ đồ phòng ngừa deadlock khác là áp đặt một thứ tự bộ phận trên tất cả các hạng mục dữ liệu và yêu cầu một giao dịch chốt một hạng mục dữ liệu theo thứ tự được xác định bởi thứ tự bộ phận này.

Cách tiếp cận thứ hai để phòng ngừa deadlock là sử dụng ưu tiên và cuộn lại quá trình. Với ưu tiên, một giao dịch $T_2$ yêu cầu một chốt bị giữ bởi giao dịch $T_1$ , chốt đã cấp cho $T_1$ có thể bị lấy lại và cấp chgo $T_2$ , $T_1$ bị cuộn lại. Để điều khiển ưu tiên, ta gán một tem thời gian duy nhất cho mỗi giao dịch. Hệ thống sử dụng các tem thời gian này để quyết định một giao dịch phải chờ hay cuộn lại. Việc chốt vẫn được sử dụng để điều khiển cạnh tranh. Nếu một giao dịch bị cuộn lại, nó vẫn giữ tem thời gian cũ của nó khi tái khởi động. Hai sơ đồ phòng ngừa deadlock sử dụng tem thời gian khác nhau được đề nghị:

1. Sơ đồ **Wait-Die** dựa trên kỹ thuật không ưu tiên. Khi giao dịch $T_i$ yêu cầu một hạng mục dữ liệu bị chiếm bởi $T_j$ , $T_i$ được phép chờ chỉ nếu nó có tem thời gian nhỏ hơn của $T_j$ nếu không $T_i$ bị cuộn lại (die).
2. Sơ đồ **Wound-Wait** dựa trên kỹ thuật ưu tiên. Khi giao dịch $T_i$ yêu cầu một hạng mục dữ liệu hiện đang bị giữ bởi $T_j$ , $T_i$ được phép chờ chỉ nếu nó có tem thời gian lớn hơn của $T_j$ , nếu không $T_j$ bị cuộn lại (Wounded).

Một điều quan trọng là phải đảm bảo rằng, mỗi khi giao dịch bị cuộn lại, nó không bị chết đói (starvation) có nghĩa là nó sẽ không bị cuộn lại lần nữa và được phép tiến triển.

Cả hai sơ đồ **Wound-Wait** và **Wait-Die** đều tránh được sự chết đói: tại một thời điểm, có một giao dịch với tem thời gian nhỏ nhất. Giao dịch này không thể bị yêu cầu cuộn lại trong cả hai sơ đồ. Do tem thời gian luôn tăng và do các giao dịch không được gán tem thời gian mới khi chúng bị cuộn lại, một giao dịch bị cuộn lại sẽ có tem thời gian nhỏ nhất (vào thời gian sau) và sẽ không bị cuộn lại lần nữa.

Tuy nhiên, có những khác nhau lớn trong cách thức hoạt động của hai sơ đồ:

- Trong sơ đồ **Wait-Die**, một giao dịch già hơn phải chờ một giao dịch trẻ hơn giải phóng hạng mục dữ liệu. Như vậy, giao dịch già hơn có xu hướng bị chờ nhiều hơn. Ngược lại, trong sơ đồ **Wound-Wait,** một giao dịch già hơn không bao giờ phải chờ một giao dịch trẻ hơn.
- Trong sơ đồ **Wait-Die,** nếu một giao dịch $T_i$ chết và bị cuộn lại vì nó đòi hỏi một hạng mục dữ liệu bị giữ bởi giao dịch $T_j$ , khi đó $T_i$ có thể phải tái phát ra cùng dãy các yêu cầu khi nó khởi động lại. Nếu hạng mục dữ liệu vẫn bị chiếm bởi $T_j$ , $T_i$ bị chết lần nữa. Như vậy, $T_i$ có thể bị chết vài lần trước khi tậu được hạng mục dữ liệu cần thiết. Trong sơ đồ **Wound-Wait,** Giao dịch $T_i$ bị thương và bị cuộn lại do $T_j$ yêu cầu hạng mục dữ liệu nó chiếm giữ. Khi $T_i$ khởi động lại, và yêu cầu hạng mục dữ liệu, bây giờ, đang bị $T_j$ giữ, $T_i$ chờ. Như vậy, có ít cuộn lại hơn trong sơ đồ **Wound-Wait.**

Một vấn đề nổi trội đối với cả hai sơ đồ là có những cuộn lại không cần thiết vẫn xảy ra.

## V.4.2. SƠ ĐỒ DỰA TRÊN TIMEOUT

Một cách tiếp cận khác để quản lý deadlock được dựa trên lock timeout. Trong cách tiếp cận này, một giao dịch đã yêu cầu một chốt phải chờ nhiều nhất một khoảng thời gian xác định. Nếu chốt không được cấp trong khoảng thời gian này, giao dịch được gọi là mãn kỳ (time out), giao dịch tự cuộn lại và khởi động lại. Nếu có một deadlock, một hoặc một vài giao dịch dính líu đến deadlock sẽ time out và cuộn lại, để các giao dịch khác tiến triển. Sơ đồ này nằm trung gian giữa phòng ngừa deadlock và phát hiện và khôi phục deadlock.

Sơ đồ timeout dễ thực thi và hoạt động tốt nếu giao dịch ngắn và nếu sự chờ đợi lâu là do deadlock. Tuy nhiên, khó quyết định được khoảng thời gian timeout. Sơ đồ này cũng có thể đưa đến sự chết đói.

## V.4.3. PHÁT HIỆN DEADLOCK VÀ KHÔI PHỤC

Nếu một hệ thống không dùng giao thức phòng ngừa deadlock, khi đó sơ đồ phát hiện và khôi phục phải được sử dụng. Một giải thuật kiểm tra trạng thái của hệ thống được gọi theo một chu kỳ để xác định xem deadlock có xẩy ra hay không. Nếu có, hệ thống phải khôi phục lại từ deadlock, muốn vậy hệ thống phải:

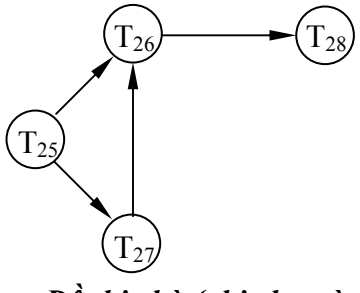
- Duy trì thông tin về sự cấp phát hiện hành các hạng mục dữ liệu cho các giao dịch cũng như các yêu cầu hạng mục dữ liệu chưa được chưa được giải quyết.
- Cung cấp một thuật toán sử dụng các thông tin này để xác định hệ thống đã đi vào trạng thái deadlock chưa.
- Phục hồi từ deadlock khi phát hiện được deadlock đã xảy ra.

### V.4.3.1 PHÁT HIỆN DEADLOCK

Deadlock có thể mô tả chính xác bằng đồ thị định hướng được gọi là đồ thị chờ (wait for graph). Đồ thị này gồm một cặp G = < V, E >, trong đó V là tập các đỉnh và E là tập các cung. Tập các đỉnh gồm tất cả các giao dịch trong hệ thống. Mỗi phần tử của E là một cặp $T_i \rightarrow T_j$ , nó chỉ ra rằng $T_i$ chờ $T_j$ giải phóng một hạng mục dữ liệu nó cần.

Khi giao dịch $T_i$ yêu cầu một hạng mục dữ liệu đang bị giữ bởi giao dịch $T_j$ khi đó cung $T_i \rightarrow T_j$ được xen vào đồ thị. Cạnh này bị xoá đi chỉ khi giao dịch $T_j$ không còn giữ hạng mục dữ liệu nào mà $T_i$ cần.

Deadlock tồn tại trong hệ thống nếu và chỉ nếu đồ thị chờ chứa một chu trình. Mỗi giao dịch tham gia vào chu trình này được gọi là bị deadlock. Để phát hiện deadlock, hệ thống phải duy trì đồ thị chờ và gọi theo một chu kỳ thủ tục tìm kiếm chu trình. Ta xét ví dụ sau:




***Đồ thị chờ (phi chu trình)***

***figure V- 19***

Do đồ thị không có chu trình nên hệ thông không trong trạng thái deadlock. Bây giờ, giả sử $T_{28}$ yêu cầu một hạng mục dữ liệu được giữ bởi $T_{27}$ , cung $T_{28} \rightarrow T_{27}$ được xen vào đồ thị, điều này dẫn đến tồn tại một chu trình $T_{26} \rightarrow T_{27} \rightarrow T_{28} \rightarrow T_{26}$ có nghĩa là hệ thống rơi vào tình trạng deadlock và $T_{26}$ , $T_{27}$ , $T_{28}$ bị deadlock.

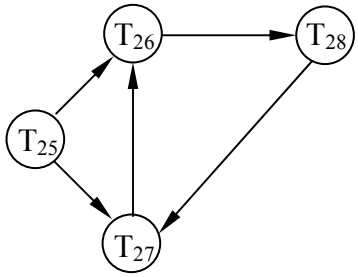



***figure V- 20***

Vấn đề đặt ra là khi nào thì chạy thủ tục phát hiện ? câu trả lời phụ thuộc hai yêu tố sau:

1. Deadlock thường xảy ra hay không ?
2. Bao nhiêu giao dịch sẽ bị ảnh hưởng bởi deadlock

Nếu deadlock thường xảy ra, việc chạy thủ tục phát hiện diễn ra thường xuyên hơn. Các hạng mục dữ liệu được cấp cho các giao dịch bị deadlock sẽ không sẵn dùng cho các giao dịch khác đến khi deadlock bị phá vỡ. Hơn nữa, số chu trình trong đồ thị có thể tăng lên. Trong trường hợp xấu nhất, ta phải gọi thủ tục phát hiện mỗi khi có một yêu cầu cấp phát không được cấp ngay.

### V.4.3.2 KHÔI PHỤC TỪ DEADLOCK

Khi thuật toán phát hiện xác định được sự tồn tại của deadlock, hệ thống phải khôi phục từ deadlock. Giải pháp chung nhất là cuộn lại một vài giao dịch để phá vỡ deadlock. Ba việc cần phải làm là:

1. **Chọn nạn nhân**. Đã cho một tập các giao dịch bị deadlock, ta phải xác định giao dịch nào phải cuộn lại để phá vỡ deadlock. Ta sẽ cuộn lại các giao dịch sao cho giá phải trả là tối thiểu. Nhiều nhân tố xác định giá của cuộn lại:
    a. Giao dịch đã tính toán được bao lâu và bao lâu nữa.
    b. Giao dịch đã sử dụng bao nhiêu hạng mục dữ liệu
    c. Giao dịch cần bao nhiêu hạng mục dữ liệu nữa để hoàn tất.
    d. Bao nhiêu giao dịch bị cuộn lại.
2. **Cuộn lại (Rollback).** Mỗi khi ta đã quyết định được giao dịch nào phải bị cuộn lại, ta phải xác định giao dịch này bị cuộn lại bao xa. Giải pháp đơn giản nhất là cuộn lại toàn bộ: bỏ dở giao dịch và bắt đầu lại nó. Tuy nhiên, sẽ là hiệu quả hơn nếu chỉ cuộn lại giao dịch đủ xa như cần thiết để phá vỡ deadlock. Nhưng phương pháp này đòi hỏi hệ thống phải duy trì các thông tin bổ xung về trạng thái của tất cả các giao dịch đang chạy.
3. **Sự chết đói (Starvation).** Trong một hệ thống trong đó việc chọn nạn nhân dựa trên các nhân tố giá, có thể xảy ra là một giao dịch luôn là nạn nhân của việc chọn này và kết quả là giao dịch này không bao giờ có thể hoàn thành. Tình huống này được gọi là sự chết đói. Phải đảm bảo việc chọn nạn nhân không đưa đến chết đói. Một giải pháp xem số lần bị cuộn lại của một giao dịch như một nhân tố về giá.

# V.5. BÀI TẬP CHƯƠNG V

**V.1** Chứng tỏ rằng giao thức chốt hai kỳ đảm bảo tính khả tuần tự xung đột và các giao dịch có thể được làm tuần tự tương ứng với các điểm chốt của chúng.

**V.2** Xét hai giao dịch sau:

$T_{31}$ : **Read(A);**
**Read(B);**
**If A=0 then B:=B+1;**
**Write(B);**

$T_{32}$ : **Read(B);**
**Read(A);**
**If B=0 then A:=A+1;**
**Write(A);**

Thêm các chỉ thị chốt và tháo chốt vào hai giao dịch $T_{31}$ và $T_{32}$ sao cho chúng tuân theo giao thức chốt hai kỳ. Sự thực hiện các giao dịch này có thể dẫn đến deadlock ?

**V.3** Nêu các ưu điểm và nhược điểm của giao thức chốt hai kỳ nghiêm ngặt.

**V.4** Nêu các ưu điểm và nhược điểm của giao thức chốt hai kỳ nghiêm khắc.

**V.5** Bộ quản trị điều khiển cạnh tranh của một hệ CSDL phải quyết định cấp cho một đòi hỏi chốt hoặc bắt giao dịch yêu cầu phải chờ. Hãy thiết kế một cấu trúc dữ liệu (tiết kiệm không gian) cho phép bộ quản trị điều khiển cạnh tranh cho ra quyết định mau chóng.

**V.6** Xét sự mở rộng thành giao thức cây-chốt sau. Nó cho phép cả chốt shared và exclusive:

1. Một giao dịch chỉ đọc chỉ có thể yêu cầu các chốt shared. Một giao dịch cập nhật chỉ có thể yêu cầu các chốt exclusive
2. Mỗi giao dịch phải tuân theo các quy tắc của giao thức cây-chốt. Các giao dịch chỉ đọc đầu tiên có thể chốt bất kỳ hạng mục dữ liệu nào, các giao dịch cập nhật đầu tiên phải chốt gốc.

   Chứng tỏ giao thức đảm bảo tính khả tuần tự và không có deadlock

**V.7** Cho ra ví dụ về các lịch trình có thể dưới giao thức cây nhưng không thể dưới giao thức chốt hai kỳ và ngược lại.

**V.8** Xét một biến thể của giao thức cây, được gọi là giao thức rừng. CSDL được tổ chức như một rừng. Mỗi giao dịch T phải tuân theo các quy tắc sau:

1. Chốt đầu tiên trong mỗi cây có thể trên bất kỳ hạng mục dữ liệu nào
2. Chốt từ thứ hai trở về sau có thể được yêu cầu chỉ nếu cha của nút được yêu cầu hiện đang bị chốt
3. Các hạng mục dữ liệu có thể được tháo chốt bất kỳ lúc nào
4. Một hạng mục dữ liệu không được chốt lại bởi T sau khi T đã tháo chốt cho nó

   Chứng tỏ giao thức rừng không đảm bảo tính khả tuần tự

**V.9** Khi một giao dịch bị cuộn lại dưới thứ tự tem thời gian, nó được gán một tem thời gian mới. Tại sao không đơn giản để nó giữ lại tem thời gian cũ ?

**V.10** Trong chốt đa hạt, sự khác nhau giữa chốt tường minh và chốt ẩn là gì ?

**V.11** phương thức chốt SIX là hữu dụng trong chốt đa hạt. Phương thức exclusive và shared tăng cường không được sử dụng. Tại sao nó lại vô dụng ?

**V.12** Đối với mỗi một trong các giao thức sau, mô tả sắc thái ứng dụng thực tế gợi ý sử dụng giao thức và sắc thái gợi ý không nên sử dụng giao thức:

1. Chốt hai kỳ
2. Chốt hai kỳ với chốt đa hạt
3. Giao thức cây
4. Thứ tự tem thời gian
5. Hợp lệ
6. Thứ tự tem thời gian đa phiên bản
7. Chốt hai kỳ đa phiên bản

**V.13** Chứng tỏ rằng có những lịch trình là có thể dưới giao thức chốt hai kỳ nhưng không thể dưới giao thức tem thời gian và ngược lại.

**V.14** Với điều kiện nào tránh deadlock ít đắt giá hơn cho phép deadlock rồi phát hiện?