GIAO DỊCH

***

* Mục tiêu

* Kiến thức cần có để học chương này

* Tài liệu tham khảo liên quan đến chương

* Nội dung

* Vấn đề nghiên cứu của chương kế tiếp

IV.1 KHÁI NIỆM:

Một giao dịch là một đơn vị thực hiện chương trình truy xuất và có thể cập nhật nhiều hạng mục dữ liệu. Một giao dịch thường là kết quả của sự thực hiện một chương trình người dùng được viết trong một ngôn ngữ thao tác dữ liệu mức cao hoặc một ngôn ngữ lập trình (SQL, COBOL, PASCAL ...), và được phân cách bởi các cau lệnh (hoặc các lời gọi hàm) có dạng begin transaction và end transaction. Giao dịch bao gồm tất cả các hoạt động được thực hiện giữa begin và end transaction.

Để đảm bảo tính toàn vẹn của dữ liệu, ta yêu cầu hệ CSDL duy trì các tính chất sau của giao dịch:

- Tính nguyên tử (Atomicity). Hoặc toàn bộ các hoạt động của giao dịch được phản ánh đúng đắn trong CSDL hoặc không có gì cả.

- **Tính nhất quán (consistency). Sự thực hiện của một giao dịch là cô lập (Không** có giao dịch khác thực hiện đồng thời) để bảo tồn tính nhất quán của CSDL.
- **Tính cô lập (Isolation). Cho dù nhiều giao dịch có thể thực hiện đồng thời, hệ** thống phải đảm bảo rằng đối với mỗi cặp giao dịch Ti, Tj , hoặc Tj kết thúc thực hiện trước khi Ti khởi động hoặc Tj bắt đầu sự thực hiện sau khi Ti kết thúc. Như vậy mỗi giao dịch không cần biết đến các giao dịch khác đang thực hiện đồng thời trong hệ thống.
- **Tính bền vững (Durability). Sau một giao dịch hoàn thành thành công, các thay** đổi đã được tạo ra đối với CSDL vẫn còn ngay cả khi xảy ra sự cố hệ thống.

Các tính chất này thường được gọi là các tính chất ACID (Các chữ cái đầu của bốn tính chất). Ta xét một ví dụ: Một hệ thống nhà băng gồm một số tài khoản và một tập các giao dịch truy xuất và cập nhật các tài khoản. Tại thời điểm hiện tại, ta giả thiết rằng CSDL nằm trên đĩa, nhưng một vài phần của nó đang nằm tạm thời trong bộ nhớ. Các truy xuất CSDL được thực hiện bởi hai hoạt động sau:

- READ(X). chuyển hạng mục dữ liệu X từ CSDL đến buffer của giao dịch thực hiện hoạt động READ này.
- WRITE(X). chuyển hạng mục dữ liệu X từ buffer của giao dịch thực hiện WRITE đến CSDL.

Trong hệ CSDL thực, hoạt động WRITE không nhất thiết dẫn đến sự cập nhật trực tiếp dữ liệu trên đĩa; hoạt động WRITE có thể được lưu tạm thời trong bộ nhớ và được thực hiện trên đĩa muộn hơn. Trong ví dụ, ta giả thiết hoạt động WRITE cập nhật trực tiếp CSDL.

Ti là một giao dịch chuyển 50 từ tài khoản A sang tài khoản B. Giao dịch này có thể được xác định như sau: **Ti : READ(A);**

A:=A - 50;
WRITE(A)
READ(B);
B:=B + 50;
WRITE(B);

Ta xem xét mỗi một trong các yêu cầu ACID

- **Tính nhất quán: Đòi hỏi nhất quán ở đây là tổng của A và B là không thay đổi bởi** sự thực hiện giao dịch. Nếu không có yêu cầu nhất quán, tiền có thể được tạo ra hay bị phá huỷ bởi giao dịch. Dễ dàng kiểm nghiệm rằng nếu CSDL nhất quán trước một thực hiện giao dịch, nó vẫn nhất quán sau khi thực hiện giao dịch. Đảm bảo tính nhất quán cho một giao dịch là trách nhiệm của người lập trình ứng dụng người đã viết ra giao dịch. Nhiệm vụ này có thể được làm cho dễ dàng bởi kiểm thử tự động các ràng buộc toàn vẹn.
- **Tính nguyên tử: Giả sử rằng ngay trước khi thực hiện giao dịch Ti, giá trị của các** tài khoản A và B tương ứng là 1000 và 2000. Giả sử rằng trong khi thực hiện giao dịch Ti, một sự cố xảy ra cản trở Ti hoàn tất thành công sự thực hiện của nó. Ta cũng giả sử rằng sự cố xảy ra sau khi hoạt động WRITE(A) đã được thực hiện, nhưng trước khi hoạt động WRITE(B) được thực hiện. Trong trường hợp này giá trị của tài khoản A và B là 950 và 2000. Ta đã phá huỷ 50$. Tổng A+B không còn được bảo tồn.

Như vậy, kết quả của sự cố là trạng thái của hệ thống không còn phản ánh trạng thái của thế giới mà CSDL được giả thiết nắm giữ. Ta sẽ gọi trạng thái như vậy là trạng thái không nhất quán. Ta phải đảm bảo rằng tính bất nhất này không xuất hiện trong một hệ CSDL. Chú ý rằng, cho dù thế nào tại một vài thời điểm, hệ thống cũng phải ở trong trạng thái không nhất quán. Ngay cả khi giao dịch Ti , trong quá trình thực hiện cũng tồn tại thời điểm tại đó giá trị của tài khoản A là 950 và tài khoản B là 2000 – một trạng thái không nhất quán. Trạng thái này được thay thế bởi trạng thái nhất quán khi giao dịch đã hoàn tất. Như vậy, nếu giao dịch không bao giờ khởi động hoặc được đảm bảo sẽ hoàn tất, trạng thái không nhất quán sẽ không bao giờ xảy ra. Đó chính là lý do có yêu cầu về tính nguyên tử: Nếu tính chất nguyên tử được cung cấp, tất cả ***các hành động của giao dịch được phản ánh trong CSDL hoặc không có gì cả. ý*** tưởng cơ sở để đảm bảo tính nguyên tử là như sau: hệ CSDL lưu vết (trên đĩa) các giá trị cũ của bất kỳ dữ liệu nào trên đó giao dịch đang thực hiện viết, nếu giao dịch không hoàn tất, giá trị cũ được khôi phục để đặt trạng thái của hệ thống trở lại trạng thái trước khi giao dịch diễn ra. Đảm bảo tính nguyên tử là trách nhiệm của hệ CSDL, và được quản lý bởi một thành phần được gọi là thành phần quản trị giao dịch (transaction-management component).

- **Tính bền vững: Tính chất bền vững đảm bảo rằng mỗi khi một giao dịch hoàn tất,** tất cả các cập nhật đã thực hiện trên cơ sở dữ liệu vẫn còn đó, ngay cả khi xảy ra sự có hệ thống sau khi giao dịch đã hoàn tất. Ta giả sử một sự cố hệ thống có thể gây ra việc mất dữ liệu trong bộ nhớ chính, nhưng dữ liệu trên đĩa thì không mất. Có thể đảm bảo tính bền vững bởi việc đảm bảo hoặc các ***cập nhật được thực hiện bởi giao dịch đã được viết lên đĩa trước khi giao dịch kết thúc hoặc thông tin về sự cập nhật được thực hiện bởi giao dịch và được viết lên đĩa đủ cho phép CSDL xây dựng lại các cập nhật khi hệ CSDL được khởi động lại sau sự cố. Đảm bảo tính bền vững là trách*** nhiệm của một thành phần của hệ CSDL được gọi là thành phần quản trị phục hồi (recovery-management component). Hai thành phần quản trị giao dịch và quản trị phục hồi quan hệ mật thiết với nhau.

- **Tính cô lập: Ngay cả khi tính nhất quán và tính nguyên tử được đảm bảo cho mỗi** giao dịch, trạng thái không nhất quán vẫn có thể xảy ra nếu trong hệ thống có một số giao dịch được thực hiện đồng thời và các hoạt động của chúng đan xen theo một cách không mong muốn. Ví dụ, CSDL là không nhất quán tạm thời trong khi giao dịch chuyển khoản từ A sang B đang thực hiện, nếu một giao dịch khác thực hiện đồng thời đọc A và B tại thời điểm trung gian nàyvà tính A+B, nó đã tham khảo một giá trị không nhất quán, sau đó nó thực hiện cập nhật A và B dựa trên các giá trị không nhất quán này, như vậy CSDL có thể ở trạng thái không nhất quán ngay cả khi cả hai giao dịch hoàn tất thành công. Một giải pháp cho vấn đề các giao dịch thực hiện đồng thời là thực ***hiện tuần tự các giao dịch, tuy nhiên giải pháp này làm giả hiệu năng của hệ*** thống. Các giải pháp khác cho phép nhiều giao dịch thực hiện cạnh tranh đã được phát triển ta sẽ thảo luận về chúng sau này. Tính cô lập của một giao dịch đảm bảo rằng sự thực hiện đồng thời các giao dịch dẫn đến một trạng thái hệ thống tương đương với một trạng thái có thể nhận được bởi thực hiện các giao dịch này một tại một thời điểm theo một thứ nào đó. Đảm bảo tính cô lập là trách nhiệm của một thành phần của hệ CSDL được gọi là thành phần quản trị cạnh tranh (concurrency-control component).

IV.2 TRẠNG THÁI GIAO DỊCH:

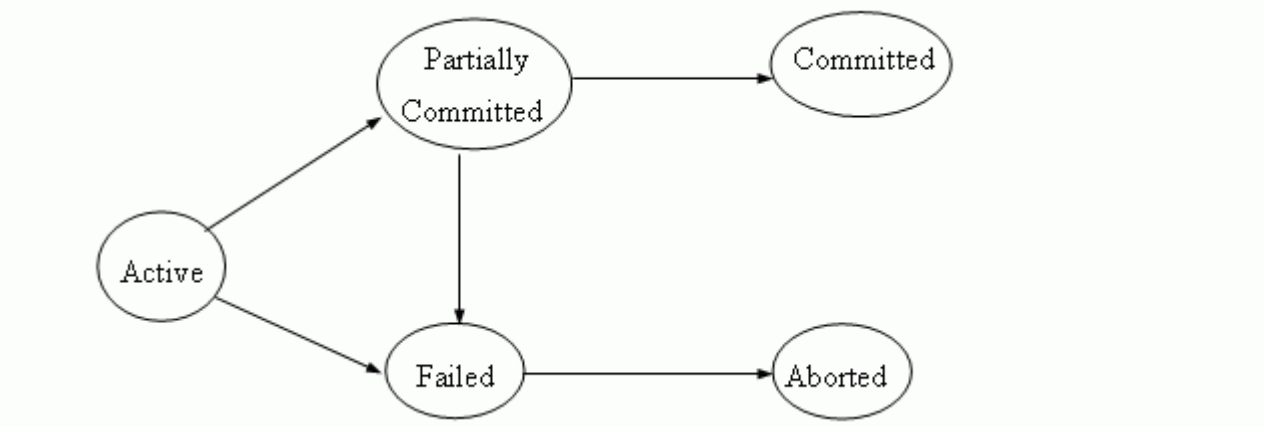
Nếu không có sự cố, tất cả các giao dịch đều hoàn tất thành công. Tuy nhiên, một giao dịch trong thực tế có thể không thể hoàn tất sự thực hiện của nó. Giao dịch như vậy được gọi là bị bỏ dở. Nếu ta đảm bảo được tính nguyên tử, một giao dịch bị bỏ dở không được phép làm ảnh hưởng tới trạng thái của CSDL. Như vậy, bất kỳ thay đổi nào mà giao dịch bị bỏ dở này phải bị huỷ bỏ. Mỗi khi các thay đổi do giao dịch bị bỏ dở bị huỷ bỏ, ta nói rằng giao dịch bị cuộn lại (rolled back). Việc này là trách nhiệm của sơ đồ khôi phục nhằm quản trị các giao dịch bị bỏ dở. Một giao dịch hoàn tất thành công sự thực hiện của nó được gọi là được bàn giao (committed). Một giao dịch được bàn giao (committed), thực hiện các cập nhật sẽ biến đổi CSDL sang một trạng thái nhất quán mới và nó là bền vững ngay cả khi có sự cố. Mỗi khi một giao dịch là được bàn giao (committed), ta không thể huỷ bỏ các hiệu quả của nó bằng các bỏ dở nó. Cách duy nhất để huỷ bỏ các hiệu quả của một giao dịch được bàn giao (committed) là thực hiện một giao dịch bù (compensating transaction); nhưng không phải luôn luôn có thể tạo ra một giao dịch bù. Do vậy trách nhiệm viết và thực hiện một giao dịch bù thuộc về người sử dụng và không được quản lý bởi hệ CSDL.

Một giao dịch phải ở trong một trong các thạng thái sau:

- Tích cực (Active). Trạng thái khởi đầu; giao dịch giữ trong trạng thái này trong khi nó đang thực hiện.
- được bàn giao bộ phận (Partially Committed). Sau khi lệnh cuối cùng được thực hiện.
- Thất bại (Failed). Sau khi phát hiện rằng sự thực hiện không thể tiếp tục được nữa.
- Bỏ dở (Aborted). Sau khi giao dịch đã bị cuộn lại và CSDL đã phục hồi lại trạng thái của nó trước khi khởi động giao dịch.
- được bàn giao (Committed). Sau khi hoàn thành thành công giao dịch.

Ta nói một giao dịch đã được bàn giao (committed) chỉ nếu nó đã di vào trạng thái Committed, tương tự, một giao dịch bị bỏ dở nếu nó đã đi vào trạng thái Aborted. Một giao dịch được gọi là kết thúc nếu nó hoặc là committed hoặc là Aborted. Một giao dịch khởi đầu bởi trạng thái Active. Khi nó kết thúc lệnh sau cùng của nó, nó chuyển sang trạng thái partially committed. Tại thời điểm này, giao dịch đã hoàn thành sự thực hiện của nó, nhưng nó vẫn có thể bị bỏ dở do đầu ra hiện tại vẫn có thể trú tạm thời trong bộ nhớ chính và như thế một sự cố phần cứng vẫn có thể ngăn cản sự hoàn tất của giao dịch. Hệ CSDL khi đó đã kịp viết lên đĩa đầy đủ thông tin giúp việc tái tạo các cập nhật đã được thực hiện trong quá trình thực hiện giao dịch, khi hệ thống tái khởi động sau sự cố. Sau hhi các thông tin sau cùng này được viết lên đĩa, giao dịch chuyển sang trạng thái committed.

Biểu đồ trạng thái tương ứng với một giao dịch như sau:



Với giả thiết sự cố hệ thống không gây ra sự mất dữ liệu trên đĩa, Một giao dịch đi vào trạng thái Failed sau khi hệ thống xác định rằng giao dịch không thể tiến triển bình thường được nữa (do lỗi phần cứng hoặc phần mềm). Như vậy, giao dịch phải được cuộn lại rồi chuyển sang trạng thái bỏ dở. Tại diểm này, hệ thống có hai lựa chọn:

- ***Khởi động lại giao dịch, nhưng chỉ nếu giao dịch bị bở dở là do lỗi phần cứng hoặc*** phần mềm nào đó không liên quan đến logic bên trong của giao dịch. Giao dịch được khởi động lại được xem là một giao dịch mới.
- ***Giết giao dịch thường được tiến hành hoặc do lỗi logic bên trong giao dịch, lỗi này*** cần được chỉnh sửa bởi viết lại chương trình ứng dụng hoặc do đầu vào xấu hoặc do dữ liệu mong muốn không tìm thấy trong CSDL.

Ta phải thận trọng khi thực hiện viết ngoài khả quan sát (observable external Write - như viết ra terminal hay máy in). Mỗi khi một viết như vậy xẩy ra, nó không thể bị xoá do nó có thể phải giao tiếp với bên ngoài hệ CSDL. Hầu hết các hệ thống cho phép các viết như thế xẩy ra chỉ khi giao dịch đã di vào trạng thái committed. Một cách để thực thi một sơ đồ như vậy là cho hệ CSDL lưu trữ tạm thời bất kỳ giá trị nào kết hợp với các viết ngoài như vậy trong lưu trữ không hay thay đổi và thực hiện các viết hiện tại chỉ sau khi giao dịch đã đi vào trạng thái committed. Nếu hệ thống thất bại sau khi giao dịch đi vào trạng thái committed nhưng trước khi hoàn tất các viết ngoài, hệ CSDL sẽ làm các viết ngoài này (sử dụng dữ liệu trong lưu trữ không hay thay đổi) khi hệ thống khởi động lại.

Trong một số ứng dụng, có thể muốn cho phép giao dịch tích cực trình bày dữ liệu cho người sử dụng, đặc biệt là các giao dịch kéo dài trong vài phút hay vài giờ. Ta không thể cho phép xuất ra dữ liệu khả quan sát như vậy trừ phi ta buộc phải làm tổn hại tính nguyên tử giao dịch. Hầu hết các hệ thống giao dịch hiện hành đảm bảo tính nguyên tử và do vậy cấm dạng trao đổi với người dùng này.

IV.3 THỰC THI TÍNH NGUYÊN TỬ VÀ TÍNH BỀN VỮNG:

Thành phần quản trị phục hồi của một hệ CSDL hỗ trợ tính nguyên tử và tính bền vững. Trước tiên ta xét một sơ đồ đơn giản (song cực kỳ thiếu hiệu quả). Sơ đồ này giả thiết rằng chỉ một giao dịch là tích cực tại một thời điểm và được dựa trên tạo bản sao của CSDL được gọi là các bản sao bóng (shadow copies). Sơ đồ giả thiết rằng CSDL chỉ là

một file trên đĩa. Một con trỏ được gọi là db_pointer được duy trì trên đĩa; nó trỏ tới bản sao hiện hành của CSDL.

Trong sơ đồ CSDL bóng (shadow-database), một giao dịch muốn cập nhật CSDL, đầu tiên tạo ra một bản sao đầy đủ của CSDL. Tất cả các cập nhật được làm trên bản sao này, không đụng chạm tới bản gốc (bản sao bóng). Nếu tại một thời điểm bất kỳ giao dịch bị bỏ dở, bản sao mới bị xoá. Bản sao cũ của CSDL không bị ảnh hưởng. Nếu giao dịch hoàn tất, nó được được bàn giao (committed) như sau. Đầu tiên, Hỏi hệ điều hành để đảm bảo rằng tất cả các trang của bản sao mới đã được viết lên đĩa (flush). Sau khi flush con trỏ db_pointer được cập nhật để trỏ đến bản sao mới; bản sao mới trở thành bản sao hiện hành của CSDL. Bản sao cũ bị xoá đi. Giao dịch được gọi là đã được được bàn giao (committed) tại thời điểm sự cập nhật con trỏ db_pointer được ghi lên đĩa. Ta xét kỹ thuật này quản lý sự cố giao dịch và sự cố hệ thống ra sao? Trước tiên, ta xét sự cố giao dịch. Nếu giao dịch thất bại tại thời điểm bất kỳ trước khi con trỏ db_pointer được cập nhật, nội dung cũ của CSDL không bị ảnh hưởng. Ta có thể bỏ dở giao dịch bởi xoá bản sao mới. Mỗi khi giao dịch được được bàn giao (committed), tất cả các cập nhật mà nó đã thực hiện là ở trong CSDL được trỏ bởi db_pointer. Như vậy, hoặc tất cả các cập nhật của giao dịch đã được phản ánh hoặc không hiệu quả nào được phản ánh, bất chấp tới sự cố giao dịch. Bây giờ ta xét sự cố hệ thống. Giả sử sự cố hệ thống xảy ra tại thời điểm bất kỳ trước khi db_pointer đã được cập nhật được viết lên đĩa. Khi đó, khi hệ thống khởi động lại, nó sẽ đọc db_pointer và như vậy sẽ thấy nội dung gốc của CSDL – không hiệu quả nào của giao dịch được nhìn thấy trên CSDL. Bây giờ lại giả sử rằng sự cố hệ thống xảy ra sau khi db_pointer đã được cập nhật lên đĩa. Trước khi con trỏ được cập nhật, tất cả các trang được cập nhật của bản sao mới đã được viết lên đĩa. Từ giả thiết file trên đĩa không bị hư hại do sự cố hệ thống. Do vậy, khi hệ thống khởi động lại, nó sẽ đọc db_pointer và sẽ thấy nội dung của CSDL sau tất cả các cập nhật đã thực hiện bởi giao dịch. Sự thực thi này phụ thuộc vào việc viết lên db_pointer, việc viết này phải là nguyên tử, có nghĩa là hoặc tất cả các byte của nó được viết hoặc không byte nào được viết. Nếu chỉ một số byte của con trỏ được cập nhật bởi việc viết nhưng các byte khác thì không thì con trỏ trở thành vô nghĩa và cả bản cũ lẫn bản mới của CSDL có thể tìm thấy khi hệ thống khởi động lại. May mắn thay, hệ thống đĩa cung cấp các cập nhật nguyên tử toàn bộ khối đĩa hoặc ít nhất là một sector đĩa. Như vậy hệ thống đĩa đảm bảo việc cập nhật con trỏ db_pointer là nguyên tử. Tính nguyên tử và tính bền vững của giao dịch được đảm bảo bởi việc thực thi bản sao bóng của thành phần quản trị phục hồi. Sự thực thi này cực kỳ thiếu hiệu quả trong ngữ cảnh CSDL lớn, do sự thực hiện một giao dịch đòi hỏi phải sao toàn bộ CSDL. Hơn nữa sự thực thi này không cho phép các giao dịch thực hiện đồng thời với các giao dịch khác. Phương pháp thực thi tính nguyên tử và tính lâu bền mạnh hơn và đỡ tốn kém hơn được trình bày trong chương hệ thống phục hồi.

IV.4 CÁC THỰC HIỆN CẠNH TRANH:

Hệ thống xử lý giao dịch thường cho phép nhiều giao dịch thực hiện đồng thời. Việc cho phép nhiều giao dịch cập nhật dữ liệu đồng thời gây ra những khó khăn trong việc bảo đảm sự nhất quán dữ liệu. Bảo đảm sự nhất quán dữ liệu mà không đếm xỉa tới sự thực hiện cạnh tranh các giao dịch sẽ cần thêm các công việc phụ. Một phương pháp dễ tiến hành là cho các giao dịch thực hiện tuần tự: đảm bảo rằng một giao dịch khởi

động chỉ sau khi giao dịch trước đã hoàn tất. Tuy nhiên có hai lý do hợp lý để thực hiện cạnh tranh là:

- Một giao dịch gồm nhiều bước. Một vài bước liên quan tới hoạt động I/O; các bước khác liên quan đến hoạt động CPU. CPU và các đĩa trong một hệ thống có thể hoạt động song song. Do vậy hoạt động I/O có thể được tiến hành song song với xử lý tại CPU. Sự song song của hệ thống CPU và I/O có thể được khai thác để chạy nhiều giao dịch song song. Trong khi một giao dịch tiến hành một hoạt động đọc/viết trên một đĩa, một giao dịch khác có thể đang chạy trong CPU, một giao dịch thứ ba có thể thực hiện đọc/viết trên một đĩa khác ... như vậy sẽ tăng lượng đầu vào hệ thống có nghĩa là tăng số lượng giao dịch có thể được thực hiện trong một lượng thời gian đã cho, cũng có nghĩa là hiệu suất sử dụng bộ xử lý và đĩa tăng lên.
- Có thể có sự trộn lẫn các giao dịch đang chạy trong hệ thống, cái thì dài cái thì ngắn. Nếu thực hiện tuần tự, một quá trình ngắn có thể phải chờ một quá trình dài đến trước hoàn tất, mà điều đó dẫn đến một sự trì hoãn không lường trước được trong việc chạy một giao dịch. Nếu các giao dịch đang hoạt động trên các phần khác nhau của CSDL, sẽ tốt hơn nếu ta cho chúng chạy đồng thời, chia sẻ các chu kỳ CPU và truy xuất đĩa giữa chúng. Thực hiện cạnh tranh làm giảm sự trì hoãn không lường trước trong việc chạy các giao dịch, đồng thời làm giảm thời gian đáp ứng trung bình: Thời gian để một giao dịch được *hoàn tất sau khi đã được đệ trình.*

Động cơ để sử dụng thực hiện cạnh tranh trong CSDL cũng giống như động cơ để thực hiện đa chương trong hệ điều hành. Khi một vài giao dịch chạy đồng thời, tính nhất quán CSDL có thể bị phá huỷ cho dù mỗi giao dịch là đúng. Một giải pháp để giải quyết vấn đề này là sử dụng định thời. Hệ CSDL phải điều khiển sự trao đổi giữa các giao dịch cạnh tranh để ngăn ngừa chúng phá huỷ sự nhất quán của CSDL. Các cơ chế cho điều đó được gọi là sơ đồ điều khiển cạnh tranh (concurrency-control scheme).

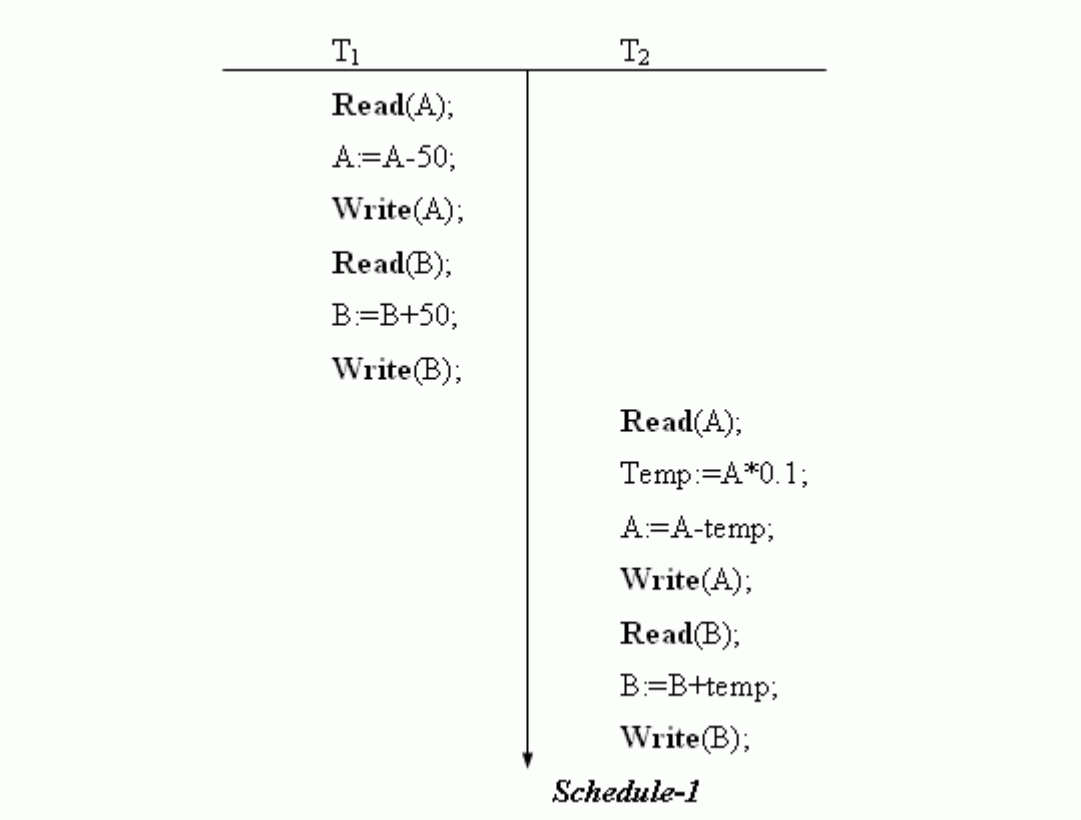
Xét hệ thống nhà băng đơn giản, nó có một số tài khoản và có một tập hợp các giao dịch, chúng truy xuất, cập nhật các tài khoản này. Giả sử T1 và T2 là hai giao dịc chuyển khoản từ một tài khoản sang một tài khoản khác. Giao dịch T1 chuyển 50$ từ tài khoản A sang tài khoản B và được xác định như sau:

T_1 : **Read**(A);
A:=A-50;
Write(A);
Read(B);
B:=B+50;
Write(B);

Giao dịch T2 chuyển 10% số dư từ tài khoản A sang tài khoản B, và được xác định như sau:

T_2 : **Read**(A);
Temp:=A*0.1;
A:=A-temp;
Write(A);
Read(B);
B:=B+temp;
Write(B);

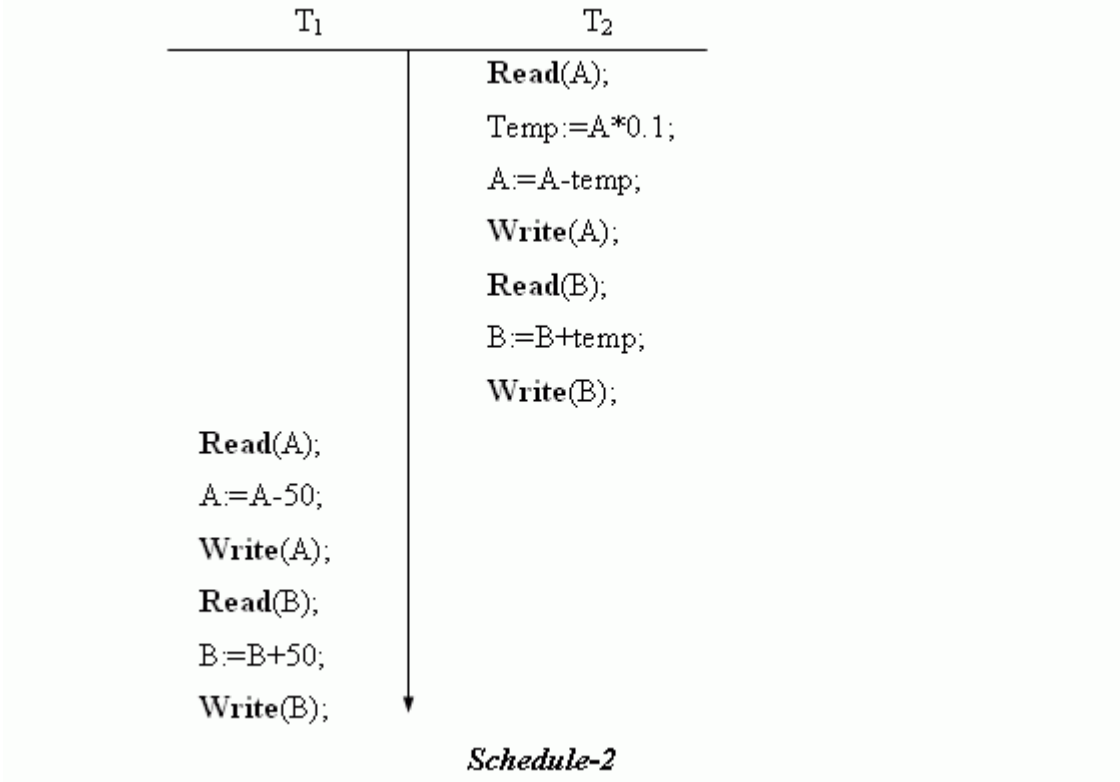
Giả sử giá trị hiện tại của A và B tương ứng là 1000$ và 2000$. Giả sử rằng hai giao dịch này được thực hiện mỗi một tại một thời điểm theo thứ tự T1 rồi tới T2. Như vậy, dãy thực hiện này là như hình bên dưới, trong đó dãy các bước chỉ thị ở trong thứ tự thời gian từ đỉnh xuống đáy, các chỉ thị của T1 nằm ở cột trái còn các chỉ thị của T2 nằm ở cột phải:



Schedule-1

Giá trị sau cùng của các tài khoản A và B, sau khi thực hiện dãy các chỉ thị theo trình tự này là 855$ và 2145$ tương ứng. Như vậy, tổng giá trị của hai tài khoản này (A + B) được bảo tồn sau khi thực hiện cả hai giao dịch.

Tương tự, nếu hai giao dịch được thực hiện mỗi một tại một thời điểm song theo trình tự T2 rồi đến T1 , khi đó dãy thực hiện sẽ là:

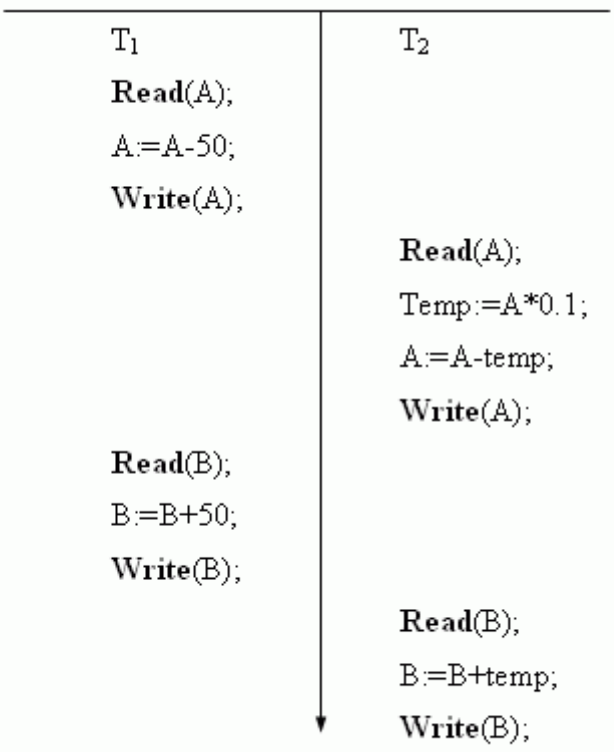


Schedule-2

Và kết quả là các giá trị cuối cùng của tài khoản A và B tương ứng sẽ là 850$ và 2150$.

Các dãy thực hiện vừa được mô tả trên được gọi là các lịch trình (schedules). Chúng biểu diễn trình tự thời gian các chỉ thị được thực hiện trong hệ thống. Một lịch trình đối với một tập các giao dịch phải bao gồm tất cả các chỉ thị của các giao dich này và phải bảo tồn thứ tự các chỉ thị xuất hiện trong mỗi một giao dịch. Ví dụ, đối với giao dịch T1 , chỉ thị Write(A) phải xuất hiện trước chỉ thị Read(B), trong bất kỳ lịch trình hợp lệ nào. Các lịch trình schedule-1 và schedule-2 là tuần tự. Mỗi lịch trình tuần tự gồm một dãy các chỉ thị từ các giao dịch, trong đó các chỉ thị thuộc về một giao dịch xuất hiện cùng nhau trong lịch trình. Như vậy, đối với một tập n giao dịch, có n! lịch trình tuần tự hợp lệ khác nhau. Khi một số giao dịch được thực hiện đồng thời, lịc trình tương ứng không nhất thiết là tuần tự. Nếu hai giao dịch đang chạy đồng thời, hệ điều hành có thể thực hiện một giao dịch trong một khoảng ngắn thời gian, sau đó chuyển đổi ngữ cảnh, thực hiện giao dịch thứ hai một khoảng thời gian sau đó lại chuyển sang thực hiện giao dịch thứ nhất một khoảng và cứ như vậy (hệ thống chia sẻ thời gian).

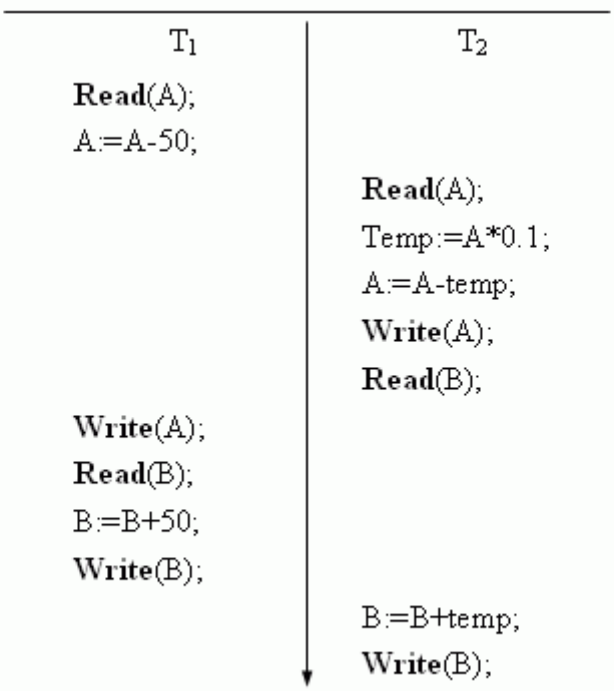
Có thể có một vài dãy thực hiện, vì nhiều chỉ thị của các giao dịch có thể đan xen nhau. Nói chung, không thể dự đoán chính xác những chỉ thị nào của một giao dịch sẽ được thực hiện trước khi CPU chuyển cho giao dịch khác. Do vậy, số các lịch trình có thể đối với một tập n giao dịch lớn hơn n! nhiều.



Schedule-3 — một lịch trình cạnh tranh tương đương schedule-1

Không phải tất cả các thực hiện cạnh tranh cho ra một trạng thái đúng. Ví dụ schedule-4 sau cho ta một minh hoạ về nhận định này:

Sau khi thực hiện giao dịch này, ta đạt tới trạng thái trong đó giá trị cuối của A và B tương ứng là 950$ và 2100$. Trạng thái này là một trạng thái không nhất quán (A+B trước khi thực hiện giao dịch là 3000$ nhưng sau khi giao dịch là 3050$). Như vậy, nếu giao phó việc điều khiển thực hiện cạnh tranh cho hệ điều hành, sẽ có thể dẫn tới các trạng thái không nhất quán. Nhiệm vụ của hệ CSDL là đảm bảo rằng một lịch trình được phép thực hiện sẽ đưa CSDL sang một trạng thái nhất quán. Thành phần của hệ CSDL thực hiện nhiệm vụ này được gọi là thành phần điều khiển cạnh tranh (concurrency-control component). Ta có thể đảm bảo sự nhất quán của CSDL với thực hiện cạnh tranh bằng cách nắm chắc rằng một lịch trình được thực hiện có cùng hiệu quả như một lịch trình tuần tự.



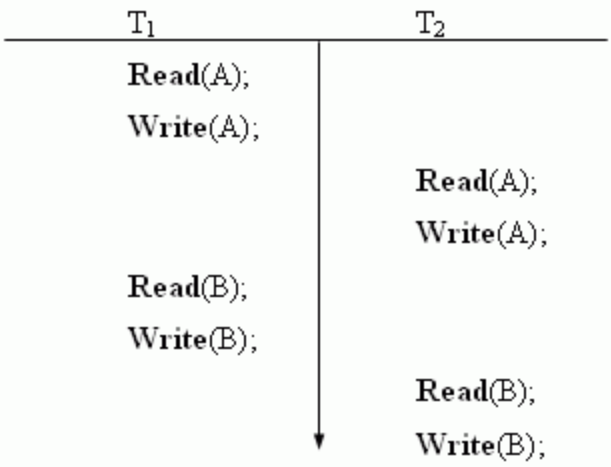
Schedule-4 — một lịch trình cạnh tranh

IV.5 TÍNH KHẢ TUẦN TỰ: (Serializability)

IV.5.1 Tuần tự xung đột

IV.5.2 Tuần tự view

Hệ CSDL phải điều khiển sự thực hiện cạnh tranh các giao dịch để đảm bảo rằng trạng thái CSDL giữ nguyên ở trạng thái nhất quán. Trước khi ta kiểm tra hệ CSDL có thể thực hiện nhiệm vụ này như thế nào, đầu tiên ta phải hiểu các lịch trình nào sẽ đảm bảo tính nhất quán và các lịch trình nào không. Vì các giao dịch là các chương trình, nên thật khó xác định các hoạt động chính xác được thực hiện bởi một giao dịch là hoạt động gì và những hoạt động nào của các giao dịch tác động lẫn nhau. Vì lý do này, ta sẽ không giải thích kiểu hoạt động mà một giao dịch có thể thực hiện trên một hạng mục dữ liệu. Thay vào đó, ta chỉ xét hai hoạt động: Read và Write. Ta cũng giả thiết rằng giữa một chỉ thị Read(Q) và một chỉ thị Write(Q) trên một hạng mục dữ liệu Q, một giao dịch có thể thực hiện một dãy tuỳ ý các hoạt động trên bản sao của Q được lưu trú trong buffer cục bộ của giao dịch. Vì vậy ta sẽ chỉ nêu các chỉ thị Read và Write trong lịch trình, như trong biểu diễn với quy ước như vậy của schedule-3 dưới đây:



Schedule-3 (viết dưới dạng thoả thuận)

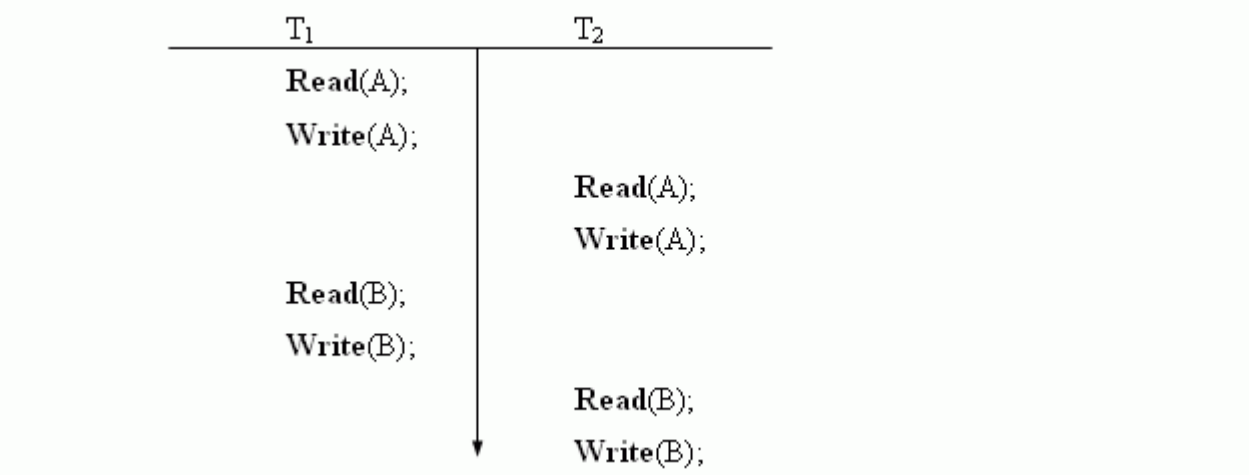
IV.5.1 Tuần tự xung đột

Xét lịch trình S trong đó có hai chỉ thị liên tiếp Ii và Ij của các giao dịch Ti , Tj tương ứng (i (j). Nếu Ii và Ij tham khảo đến các hạng mục dữ liệu khác nhau, ta có thể đổi chỗ Ii và Ij mà không làm ảnh hưởng đến kết quả của bất kỳ chỉ thị nào trong lịch trình. Tuy nhiên, nếu Ii và Ij tham khảo cùng một hạng mục dữ liệu Q, khi đó thứ tự của hai bước này có thể rất quan trọng. Do ta đang thực hiện chỉ các chỉ thị Read và Write, nên ta có bốn trường hợp cần phải xét sau:

1. Ii = Read(Q); Ij = Read(Q): Thứ tự của Ii và Ij không gây ra vấn đề nào, do Ti và Tj đọc cùng một giá trị Q bất kể đến thứ tự giữa Ii và Ij.
2. Ii = Read(Q); Ij = Write(Q): Nếu Ii thực hiện trước Ij, Khi đó Ti không đọc giá trị được viết bởi Tj bởi chỉ thị Ij . Nếu Ij thực hiện trước Ii , Ti sẽ đọc giá trị của Q được viết bởi Ij , như vậy thứ tự của Ii và Ij là quan trọng.
3. Ii = Write(Q); Ij = Read(Q): Thứ tự của Ii và Ij là quan trọng do cùng lý do trong trường hợp trước.
4. Ii = Write(Q); Ij = Write(Q): Cả hai chỉ thị là hoạt động Write, thứ tự của hai chỉ thị này không ảnh hưởng đến cả hai giao dịch Ti và Tj . Tuy nhiên, giá trị nhận được bởi chỉ thị Read kế trong S sẽ bị ảnh hưởng do kết quả phụ thuộc vào chỉ thị Write được thực hiện sau cùng trong hai chỉ thị Write này. Nếu không còn chỉ thị Write nào sau Ii và Ij trong S, thứ tự của Ii vf Ij sẽ ảnh hưởng trực tiếp đến giá trị cuối của Q trong trạng thái CSDL kết quả (của lịch trình S).

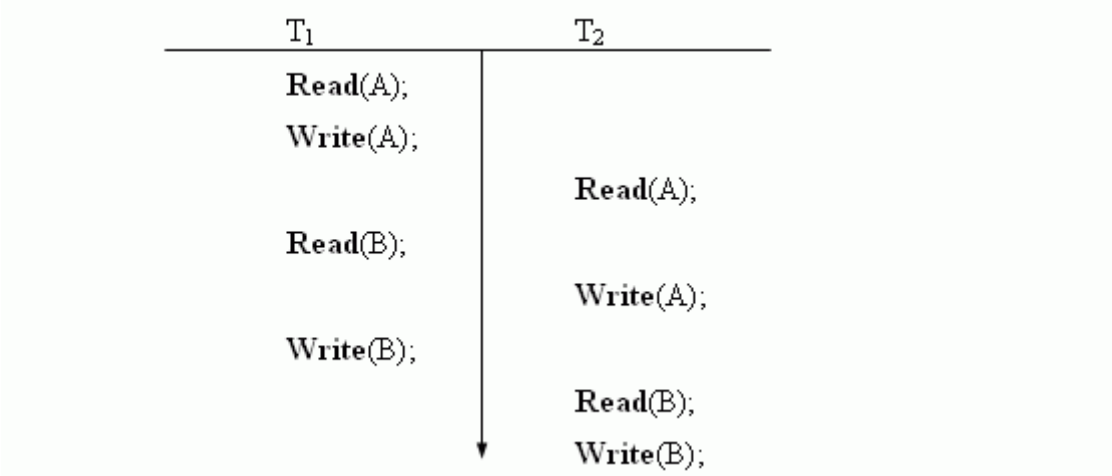
Như vậy chỉ trong trường hợp cả Ii và Ij là các chỉ thị Read, thứ tự thực hiện của hai chỉ thị này (trong S) là không gây ra vấn đề.

Ta nói Ii và Ij xung đột nếu các hoạt động này nằm trong các giao dịch khác nhau, tiến hành trên cùng một hạng mục dữ liệu và có ít nhất một hoạt động là Write. Ta xét lịch trình schedule-3 như ví dụ minh hoạ cho các chỉ thị xung đột.



Chỉ thị Write(A) trong T1 xung đột với Read(A) trong T2. Tuy nhiên, chỉ thị Write(A) trong T2 không xung đột với chỉ thị Read(B) trong T1 do các chỉ thị này truy xuất các hạng mục dữ liệu khác nhau.

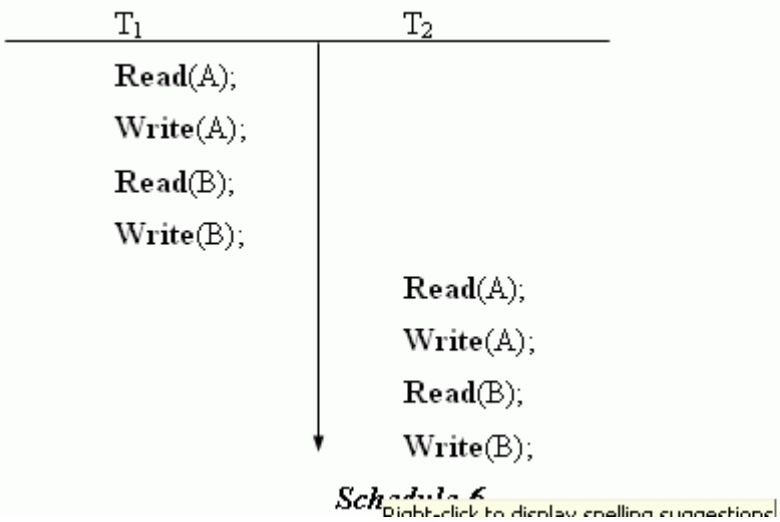
Ii và Ij là hai chỉ thị liên tiếp trong lịch trình S. Nếu Ii và Ij là các chỉ thị của các giao dịch khác nhau và không xung đột, khi đó ta có thể đổi thứ tự của chúng mà không làm ảnh hưởng gì đến kết quả xử lý và như vậy ta nhận được một lịch trình mới S' tương đương với S. Do chỉ thị Write(A) của T2 không xung đột với chỉ thị Read(B) của T1, ta có thể đổi chỗ các chỉ thị này để được một lịch trình tương đương – schedule-5 dưới đây



Ta tiếp tục đổi chỗ các chỉ thị không xung đột như sau:

- Đổi chỗ chỉ thị Read(B) của T1 với chỉ thị Read(A) của T2
- Đổi chỗ chỉ thị Write(B) của T1 với chỉ thị Write(A) của T2
- Đổi chỗ chỉ thị Write(B) của T1 với chỉ thị Read(A) của T2

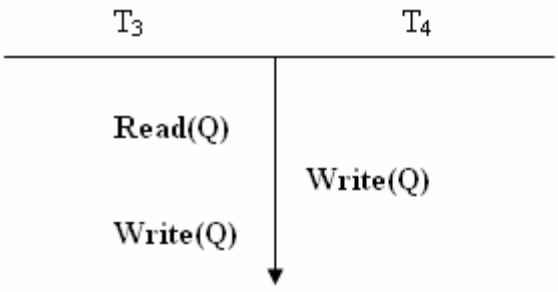
Kết quả cuối cùng của các bước đổi chỗ này là một lịch trình mới (schedule-6 –lịch trình tuần tự) tương đương với lịch trình ban đầu (schedule-3):



Sự tương đương này cho ta thấy: bất chấp trạng thái hệ thống ban đầu, schedule-3 sẽ sinh ra cùng trạng thái cuối như một lịch trình tuần tự nào đó.

Nếu một lịch trình S có thể biến đổi thành một lịch trình S' bởi một dãy các đổi chỗ các chỉ thị không xung đột, ta nói S và S' là tương đương xung đột (conflict equivalent). Trong các schedule đã được nêu ở trên, ta thấy schedule-1 tương đương xung đột với schedule-3.

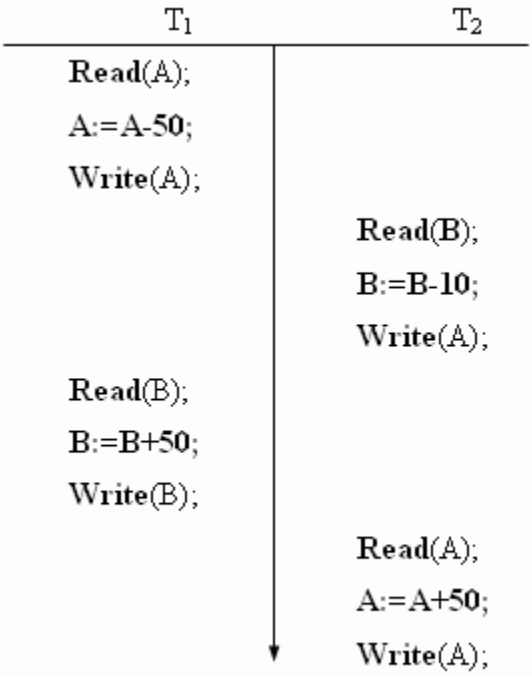
Khái niệm tương đương xung đột dẫn đến khái niệm tuần tự xung đột. Ta nói một lịch trình S là khả tuần tự xung đột (conflict serializable) nếu nó tương đương xung đột với một lịch trình tuần tự. Như vậy, schedule-3 là khả tuần tự xung đột. Như một ví dụ, lịch trình schedule-7 dưới đây không tương đương xung đột với một lịch trình tuần tự nào do vậy nó không là khả tuần tự xung đột:



Schedule-7

Có thể có hai lịch trình sinh ra cùng kết quả, nhưng không tương tương xung đột. Ví dụ, giao dịch T5 chuyển 10$ từ tài khoản B sang tài khoản A. Ta xét lịch trình schedule-8 như dưới đây, lịch trình này không tương đương xung đột với lịch trình tuần tự < T1, T5 > do trong lịch trình schedule-8 chỉ thị Write(B) của T5 xung đột với chỉ thị **Read(B) của T1 như vậy ta không thể di chuyển tất cả các chỉ thị của T1 về trước các chỉ thị** của T5 bởi việc hoán đổi liên tiếp các chỉ thị không xung đột. Tuy nhiên, các giá trị sau cùng của tài khoản A và B sau khi thực hiện lịch schedule-8 hoặc sau khi thực hiện lịch

trình tuần tự <T1, T5 > là như nhau--- là 960 và 2040 tương ứng. Qua ví dụ này ta thấy cần thiết phải phân tích cả sự tính toán được thực hiện bởi các giao dịch mà không chỉ các hoạt động Read và Write. Thuy nhiên sự phân tích như vậy sẽ nặng nề và phải trả một giá tính toán cao hơn.



Schedule-8

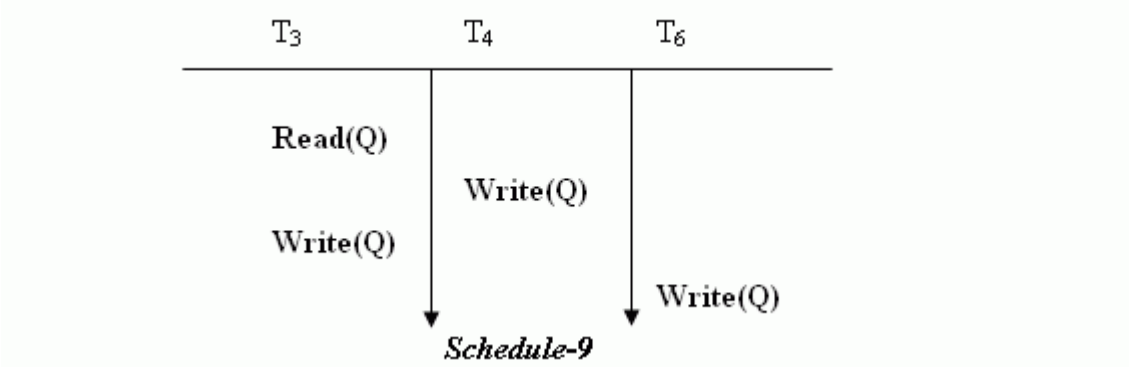
IV.5.2 Tuần tự view

Xét hai lịch trình S và S', trong đó cùng một tập hợp các giao dịch tham gia vào cả hai lịch trình. Các lịch trình S và S' được gọi là tương đương view nếu ba điều kiện sau được thoả mãn:

1. Đối với mỗi hạng mục dữ liệu Q, nếu giao dịch Ti đọc giá trị khởi đầu của Q trong lịch trình S, thì giao dịch Ti phải cũng đọc giá trị khởi đầu của Q trong lịch trình S'.
2. Đối với mỗi hạng mục dữ liệu Q, nếu giao dịch Ti thực hiện Read(Q) trong lịch trình S và giá trị đó được sản sinh ra bởi giao dịch Tj thì Ti cũng phải đọc giá trị của Q được sinh ra bởi giao dịch Tj trong S'.
3. Đối với mỗi hạng mục dữ liệu Q, giao dịch thực hiện hoạt động Write(Q) sau cùng trong lịch trình S, phải thực hiện hoạt động Write(Q) sau cùng trong lịch trình S'.

Điều kiện 1 và 2 đảm bảo mỗi giao dịch đọc cùng các giá trị trong cả hai lịch trình và do vậy thực hiện cùng tính toán. Điều kiện 3 đi cặp với các điều kiện 1 và 2 đảm bảo cả hai lịch trình cho ra kết quả là trạng thái cuối cùng của hệ thống như nhau. Trong các ví dụ trước, schedule-1 là không tương tương view với lịch trình 2 do, trong schedule-1, giá trị của tài khoản A được đọc bởi giao dịch T2 được sinh ra bởi T1 , trong khi điều này không xảy ra trong schedule-2. Schedule-1 tương đương view với schedule-3 vì các giá trị của các tài khoản A và B được đọc bởi T2 được sinh ra bởi T1 trong cả hai lịch trình.

Quan niệm tương đương view đưa đến quan niệm tuần tự view. Ta nói lịch trình S là khả tuần tự view (view serializable) nếu nó tương dương view với một lịch trình tuần tự. Ta xét lịch trình sau:



Schedule-9

Nó tương đương view với lịch trình tuần tự < T3, T4, T6 > do chỉ thị Read(Q) đọc giá trị khởi đầu của Q trong cả hai lịch trình và T6 thực hiện Write sau cùng trong cả hai lịch trình như vậy schedule-9 khả tuần tự view.

Mỗi lịch trình khả tuần tự xung đột là khả tuần tự view, nhưng có những lịch trình khả tuần tự view không khả tuần tự xung đột (ví dụ schedule-9).

Trong schedule-9 các giao dịch T4 và T6 thực hiện các hoạt động Write(Q) mà không thực hiện hoạt động Read(Q), Các Write dạng này được gọi là các Write mù (blind write). Các Write mù xuất hiện trong bất kỳ lịch trình khả tuần tự view không khả tuần tự xung đột.

IV.6 TÍNH KHẢ PHỤC HỒI (Recoverability)

IV.6.1 Lịch trình khả phục hồi

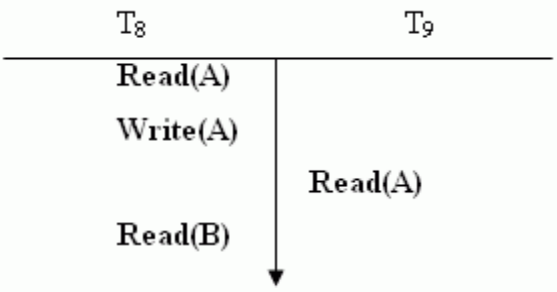
IV.6.2 Lịch trình cascadeless

Ta đã nghiên cứu các lịch trình có thể chấp nhận dưới quan điểm sự nhát quán của CSDL với giả thiết không có giao dịch nào thất bại. Ta sẽ xét hiệu quả của thất bại giao dịch trong thực hiện cạnh tranh.

Nếu giao dịch Ti thất bại vì lý do nào đó, ta cần huỷ bỏ hiệu quả của giao dịch này để đảm bảo tính nguyên tử của giao dịch. Trong hệ thống cho phép thực hiện cạnh tranh, cũng cần thiết đảm bảo rằng bất kỳ giao dịch nào phụ thuộc vào Ti cũng phải bị bỏ. Để thực hiện sự chắc chắn này, ta cần bố trí các hạn chế trên kiểu lịch trình được phép trong hệ thống.

IV.6.1 Lịch trình khả phục hồi (Recoverable Schedule)

Xét lịch trình schedule-10 trong đó T9 là một giao dịch chỉ thực hiện một chỉ thị Read(A). Giả sử hệ thống cho phép T9 bàn giao (commit) ngay sau khi thực hiện chỉ thị Read(A). Như vậy T9 bàn giao trước T8 . Giả sử T8 thất bại trước khi bàn giao, vì T9 vì T9 đã đọc giá trị của hạng mục giữ liệu A được viết bởi T8 , ta phải bỏ dở T9 để đảm bảo tính nguyên tử giao dịch. Song T9 đã được bàn giao và không thể bỏ dở được. Ta có tình huống trong đó không thể khôi phục đúng sau thất bại của T8 .

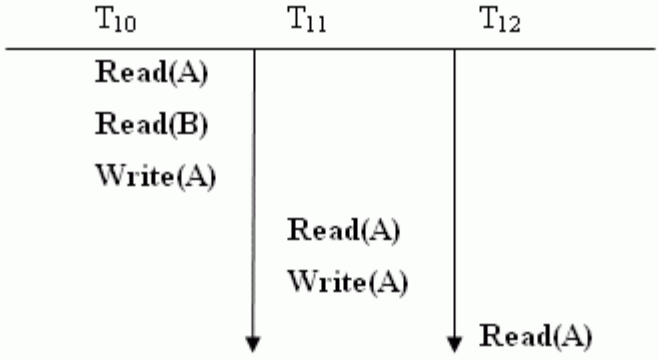


Schedule-10

Lịch trình schedule-10 là một ví dụ về lịch trình không phục hồi được và không dược phép. Hầu hết các hệ CSDL dòi hỏi tất cả các lịch trình phải phục hồi được. Một lịch *trình khả phục hồi là lịch trình trong đó, đối với mỗi cặp giao dịch Ti , Tj , nếu Tj đọc hạng mục dữ liệu được viết bởi Ti thì hoạt động bàn giao của Tj phải xảy ra sau hoạt động bàn giao của Ti* .

IV.6.2 Lịch trình cascadeless

Ngay cả khi lịch trình là khả phục hồi, để phục hồi đúng sau thất bại của một giao dịch Ti ta phải cuộn lại một vài giao dịch. Tình huống như thế xảy ra khi các giao dịch đọc dữ liệu được viết bởi Ti . Ta xét lich trình schedule-11 sau



Schedule-11

Giao dịch T10 viết một giá trị được đọc bởi T11 . Giao dịch T12 đọc một giá trị được viết bởi T11 . Giả sử rằng tại điểm này T10 thất bại. T10 phải cuộn lại, do T11 phụ thuộc vào T10 nên T11 cũng phải cuộn lại và cũng như vậy với T12. Hiện tượng trong đó một giao dịch thất bại kéo theo một dãy các giao dịch phải cuộn lại được gọi là sự cuộn lại hàng loạt (cascading rollback).

cuộn lại hàng loạt dẫn đến việc huỷ bỏ một khối lượng công việc đáng kể. Phải hạn chế các lịch trình để việc cuộn lại hàng loạt không thể xảy ra. Các lịch trình như vậy được gọi là các lịch trình cascadeless. Một lịch trình cascadeless là một lịch trình trong đó mỗi *cặp giao dịch Ti , Tj nếu Tj đọc một hạng mục dữ liệu được viết trước đó bởi Ti, hoạt động bàn giao của Ti phải xuất hiện trước hoạt động đọc của TJ . Một lịch trình* cascadeless là khả phục hồi.

IV.7 THỰC THI CÔ LẬP (Implementation of Isolation)

Có nhiều sơ đồ điều khiển cạnh tranh có thể được sử dụng để đảm bảo các tính chất một lịch trình phải có (nhằm giữ CSDL ở trạng thái nhất quán, cho phép quản lý các giao dịch ...), ngay cả khi nhiều giao dịch thực hiện cạnh tranh, chỉ các lịch trình có thể chấp nhận được sinh ra, bất kể hệ điều hành chia sẻ thời gian tài nguyên như thế nào giữa các giao dịch.

Như một ví dụ, ta xét một sơ đồ điều khiển cạnh tranh sau: Một giao dịch tậu một chốt (lock) trên toàn bộ CSDL trước khi nó khởi động và tháo chốt khi nó đã bàn giao. Trong khi giao dịch giữ chốt không giao dịch nào khác được phép tậu chốt và như vậy phải chờ đến tận khi chốt được tháo. Trong đối sách chốt, chỉ một giao dịch được thực hiện tại một thời điểm và như vậy chỉ lịch trình tuần tự được sinh ra. Sơ đồ điều khiển cạnh tranh này cho ra một hiệu năng cạnh tranh nghèo nàn. Ta nói nó cung cấp một bậc cạnh tranh nghèo (poor degree of concurrency).

Mục đích của các sơ đồ điều khiển cạnh tranh là cung cấp một bậc canh tranh cao trong khi vẫn đảm bảo các lịch trình được sinh ra là khả tuần tự xung đột hoặc khả tuần tự view và cascadeless.

IV.8 ĐỊNH NGHĨA GIAO DỊCH TRONG SQL

Chuẩn SQL đặc tả sự bắt đầu một giao dịch một cánh không tường minh. Các giao dịch được kết thúc bởi một trong hai lệnh SQL sau:

- Commit work bàn giao giao dịch hiện hành và bắt đầu một giao dịch mới
- Rollback work gây ra sự huỷ bỏ giao dịch hiện hành

Từ khoá work là chọn lựa trong cả hai lệnh. Nếu một chương trình kết thúc thiếu cả hai lệnh này, các cập nhật hoặc được bàn giao hoặc bị cuộn lại là các sự thực hiện phụ thuộc.

Chuẩn cũng đặc tả hệ thống phải đảm bảo cả tính khả tuần tự và tính tự do từ việc cuộn lại hàng loạt. Định nghĩa tính khả tuần tự được ding bởi chuẩn là một lịch trình phải có cùng hiệu quả như một lịch trình tuần tự như vậy tính khả tuần tự xung đột và view đều được chấp nhận.

Chuẩn SQL-92 cũng cho phép một giao dịch đặc tả nó có thể được thực hiện theo một cách mà có thể làm cho nó trở nên không khả tuần tự với sự tôn trọng các giao dịch khác. Ví dụ, một giao dịch có thể hoạt động ở mức Read uncommitted, cho phép giao dịch đọc các mẩu tin them chí nếu chúng không được bàn giao. Đặc điểm này được cung cấp cho các giao dich dài các kết quả của chúng không nhất thiết phải chính xác. Ví du, thông tin xấp xỉ thường là đủ cho các thống kê được dùng cho tối ưu hoá vấn tin.

Các mức nhất quán được đặc tả trong SQL-92 là:

- Serializable : mặc nhiên
- **Repeatable read : chỉ cho phép đọc các record đã được bàn giao, hơn nữa yêu** cầu giữa hai Read trên một record bởi một giao dịch không một giao dịch nào khác được phép cập nhật record này. Tuy nhiên, giao dịch có thể không khả tuần tự với sự tôn trọng các giao dịch khác. Ví dụ, khi tìm kiếm các record thoả mãn các điều kiện nào đó, một giao dịch có thể tìm thấy một vài record được xen bởi một giao dịch đã bàn giao,

- **Read committed: Chỉ cho phép đọc các record đã được bàn giao, nhưng** không có yêu cầu thêm trên các Read khả lặp. Ví dụ, giữa hai Read của một record bởi một giao dịch, các mẩu tin có thể được cập nhật bởi các giao dịch đã bàn giao khác.
- Read uncommitted: Cho phép đọc cả các record chưa được bàn giao. Đây là mức nhất quán thấp nhất được phép trong SQL-92.

IV.9 KIỂM THỬ TÍNH KHẢ TUẦN TỰ

IV.9.1 Kiểm thử tính khả tuần tự xung đột

IV.9.2 Kiểm thử tính khả tuần tự view

Khi thiết kế các sơ đồ điều khiển cạnh tranh, ta phải chứng tỏ rằng các lịch trình được sinh ra bởi sơ đồ là khả tuần tự. Để làm điều đó, trước tiên ta phải biết làm thế nào để xác định, với một lịch trình cụ thể đã cho, có là khả tuần tự hay không.

IV.9.1 Kiểm thử tính khả tuần tự xung đột

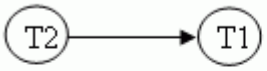
Giả sử S là một lịch trình. Ta xây dựng một đồ thị định hướng, được gọi là đồ thị trình tự (precedence graph), từ S. Đồ thị gồm một cặp (V, E) trong đó V là tập các đỉnh và E là tập các cung. Tập các đỉnh bao gồm tất cả các giao dịch tham gia vào lịch trình. Tập các cung bao gồm tất cả các cung dạng Ti (Tj sao cho một trong các điều kiện sau được thoả mãn:

1. Ti thực hiện Write(Q) trước Tj thực hiện Read(Q).
2. Ti thực hiện Read(Q) trước khi Tj thực hiện Write(Q).
3. Ti thực hiện Write(Q) trước khi Tj thực hiện Write(Q).

Nếu một cung Ti (Tj tồn tại trong đồ thị trình tự, thì trong bất kỳ lịch trình tuần tự S' nào tương đương với S, Ti phải xuất hiện trước Tj .

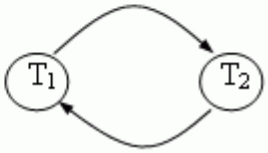
Đồ thị trình tự đối với schedule-1 là: 

vì tất cả các chỉ thị của T1 được thực hiện trước chỉ thị đầu tiên của T2

Đồ thị trình tự đối với schedule-2 là: 

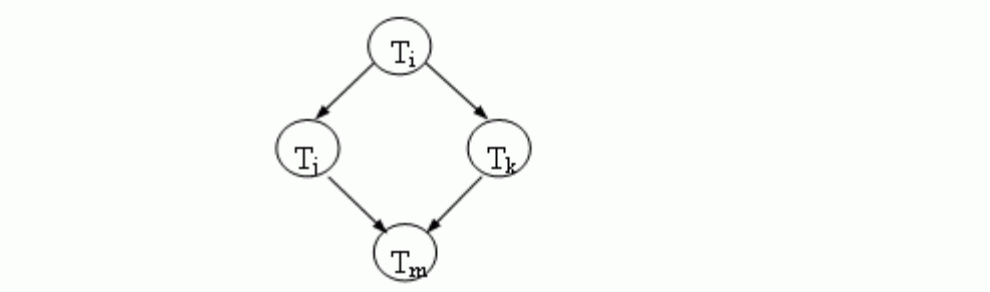
vì tất cả các chỉ thị của T2 được thực hiện trước chỉ thị đầu tiên của T1

Đồ thị trình tự đối với schedule-4 chứa các cung T1 (T2 vì T1 thực hiện Read(A) trước T2 thực hiện Write(A). Nó cũng chứa cung T2 (T1 vì T2 thực hiện Read(B) trước khi T1 thực hiện Write(B):

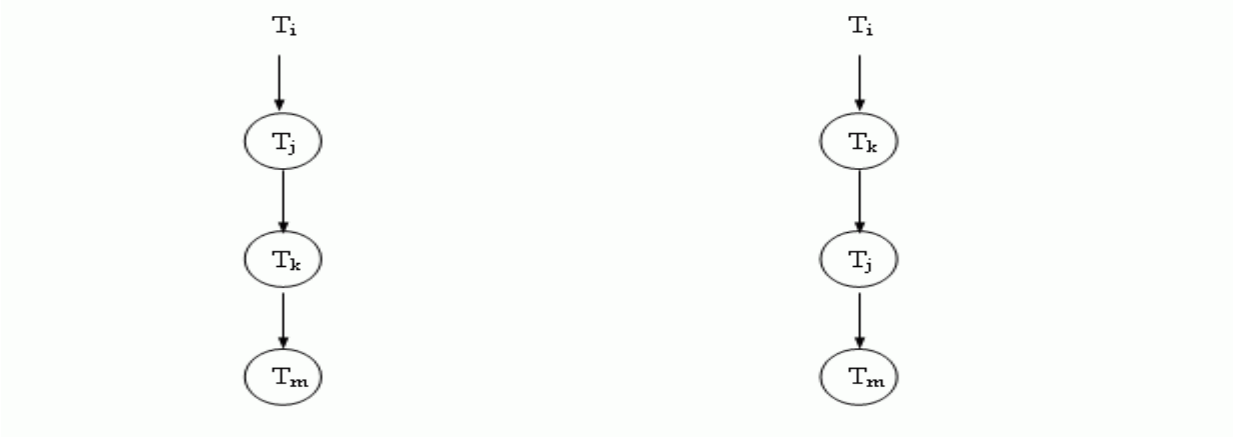


Nếu đồ thị trình tự đối với S có chu trình, khi đó lịch trình S không là khả tuần tự xungđột. Nếu đồ thị không chứa chu trình, khi đó lịch trình S là khả tuần tự xung đột. Thứ tự khả tuần tự có thể nhận được thông qua sắp xếp topo (topological sorting), nó

xác định một thứ tự tuyến tính nhất quán với thứ tự bộ phận của đồ thị trình tự. Nói chung, có một vài thứ tự tuyến tính có thể nhận được qua sắp xếp topo. Ví dụ, đồ thị sau:



Có hai thứ tự tuyến tính chấp nhận được là:

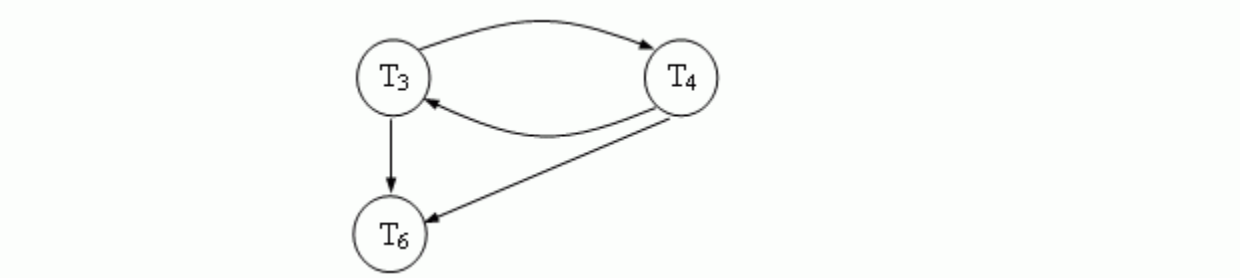


Như vậy, để kiểm thử tính khả tuần tự xung đột, ta cần xây dựng đồ thị trình tự và gọi thuật toán phát hiện chu trình. Ta nhận được một sơ đồ thực nghiệm để xác định tính khả tuần tự xung đột. Như ví dụ, schedule-1 và schedule-2, đồ thị trình tự của chúng không có chu trình, do vậy chúng là các chu trình khả tuần tự xung đột, trong khi đồ thị trình tự của schedule-4 chứa chu trình do vậy nó không là khả tuần tự xung đột.

IV.9.2 Kiểm thử tính khả tuần tự view

Ta có thể sửa đổi phép kiểm thử đồ thị trình tự đối với tính khả tuần tự xung đột dể kiểm thử tính khả tuần tự view. Tuy nhiên, phép kiểm thử này phải trả giá cao về thời gian chạy.

Xét lịch trình schedule-9, nếu ta tuân theo quy tắc trong phép kiểm thử tính khả tuần tự xung đột để tạo đồ thị trình tự, ta nhận được đồ thị sau:



Đồ thị này có chu trình, do vậy schedule-9 không là khả tuần tự xung đột. Tuy nhiên, đã đã thấy nó là khả tuần tự view (do nó tương đương với lịch trình tuần tự < T3, T4 , T6 >). Cung T3 (T4 không được xen vào đồ thị vì các giá trị của hạng mục Q được sản sinh bởi T3 và T4 không được dùng bởi bất kỳ giao dịch nào khác và T6 sản sinh ra giá trị cuối mới của Q. Các chỉ thị Write(Q) của T3 và T4 được gọi là các Write vô **dụng (Useless Write). Điều trên chỉ ra rằng không thể sử dụng đơn thuần sơ đồ đồ thị** trình tự dể kiểm thử tính khả tuần tự view. Cần thiết phát triển một sơ đồ cho việc quyết định cung nào là cần phải xen vào đồ thị trình tự.

Xét một lịch trình S. Giả sử giao dịch Tj đọc hạng mục dữ liệu Q được viết bởi Ti . Rõ ràng là nếu S là khả tuần tự view, khi đó, trong bất kỳ lịch trình tuần tự S' tương đương với S, Ti phải đi trước Tj . Bây giờ giả sử rằng, trong lịch trình S, giao dịch Tk thực hiện một Write(Q), khi đó, trong lịch trình S', Tk phải hoặc đi trước Ti hoặc đi sau Tj . Nó không thể xuất hiện giữa Ti và Tj vì như vậy Tj không đọc giá trị của Q được viết bởi Ti và như vậy S không tương đương view với S'. Các ràng buộc này không thể biểu diễn được trong thuật ngữ của mô hình đồ thị trình tự đơn giản được nêu lên trước đây. Như trong ví dụ trước, khó khăn nảy sinh ở chỗ ta biết một trong hai cung Tk (Ti và Tj (Tk phải được xen vào đồ thị nhưng ta chưa tạo được quy tắc để xác định sự lựa chọn thích hợp. Để tạo ra quy tắc này, ta cần mở rộng đồ thị định hướng để bao hàm các cung gán nhãn, ta gọi đồ thị như vậy là đồ thị trình tự gán nhãn (Label precedence graph). Cnũng như trước đây, các nút của đồ thị là tất cả các giao dịch tham gia vào lịch trình. Các quy tắc xen cung gán nhãn được diễn giải như sau:

Giả sử S là lịch trình gồm các giao dịch { T1 , T2 , ... , Tn }. Tb và Tf là hai giao dịch giả: Tb phát ra Write(Q) đối với mỗi Q được truy xuất trong S, Tf phát ra Read(Q) đối với mỗi Q được truy xuất trong S. Ta xây dựng lich trình mới S' từ S bằng cách xen Tb ở bắt đầu của S và Tf ở cuối của S. Đồ thị trình tự gán nhãn đố với S' được xây dung dựa trên các quy tắc:

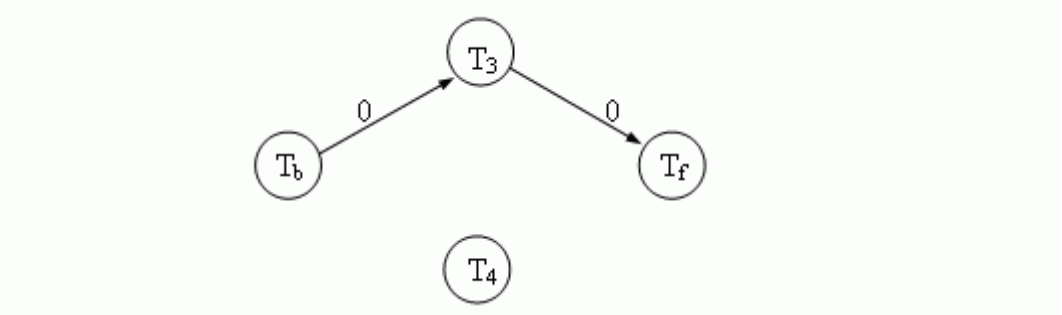
1. Thêm cung Ti (0 Tj , nếu Tj đọc giá trị của hạng mục dữ liệu Q được viết bởi T_i
2. Xoá tất cả các cung liên quan tới các giao dịch vô dụng. Một giao dịch Ti được gọi là vô dụng nếu không có con đường nào trong đồ thị trình tự dẫn từ Ti đến Tf .
3. Đối với mỗi hạng mục dữ liệu Q sao cho Tj đọc giá trị của Q được viết bởi Ti và Tk thực hiện Write(Q), Tk (Tb tiến hành các bước sau

a. Nếu Ti = Tb và Tj (Tf, khi đó xen cung Tj (0 Tk vào đồ thị trình tự gán nhãn

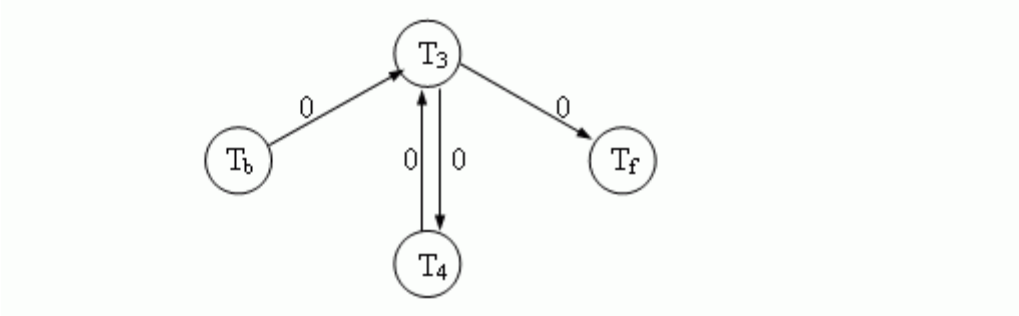
b. Nếu Ti (Tb và Tj = Tf khi đó xen cung Tk (0 Ti vào đồ thị trình tự gán nhãn

c. Nếu Ti (Tb và Tj (Tf khi đó xen cả hai cung Tk (p Ti và Tj (p Tk vào đồ thị trình tự gán nhãn, trong đó p là một số nguyên duy nhất lớn hơn 0 mà chưa được sử dụng trước đó để gán nhãn cung.

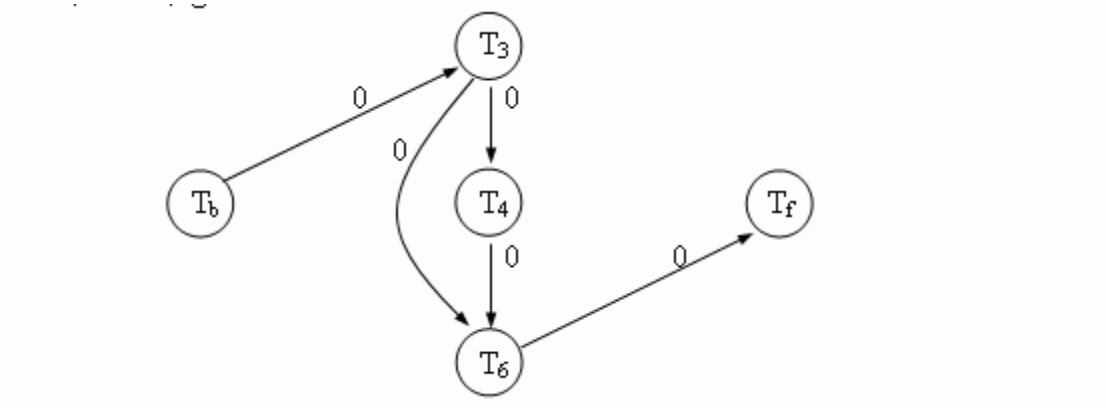
Quy tắc 3c phản ánh rằng nếu Ti viết hạng mục dữ liệu được đọc bởi Tj thì một giao dịch Tk viết cùng hạng mục dữ liệu này phải hoặc đi trước Ti hoặc đi sau Tj . Quy tắc 3a và 3b là trường hợp đặc biệt là kết quả của sự kiện Tb và Tf cần thiết là các giao dịch đầu tiên và cuối cùng tương ứng. Như một ví dụ, ta xét schedule-7. Đồ thị trình tự gán nhãn của nó được xây dựng qua các bước 1 và 2 là:



Đồ thị sau cùng của nó là (cung T3 (T4 là kết quả của 3a, cung T4 (T3 là kết quả của 3b) :



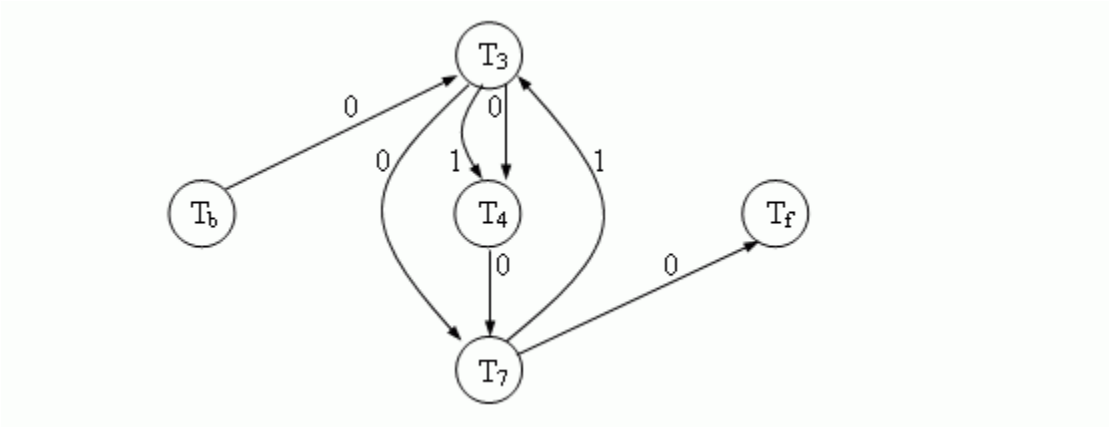
Đồ thị trình tự gán nhãn của schedule-9 là:



Cuối cùng, ta xét lịch trình schedule-10:

T_3	T_3	T_7
Read(Q)		
	Write(Q)	
		Read(Q)
Write(Q)		
		Write(Q)

Đồ thị trình tự gán nhãn của schedule-10 là:



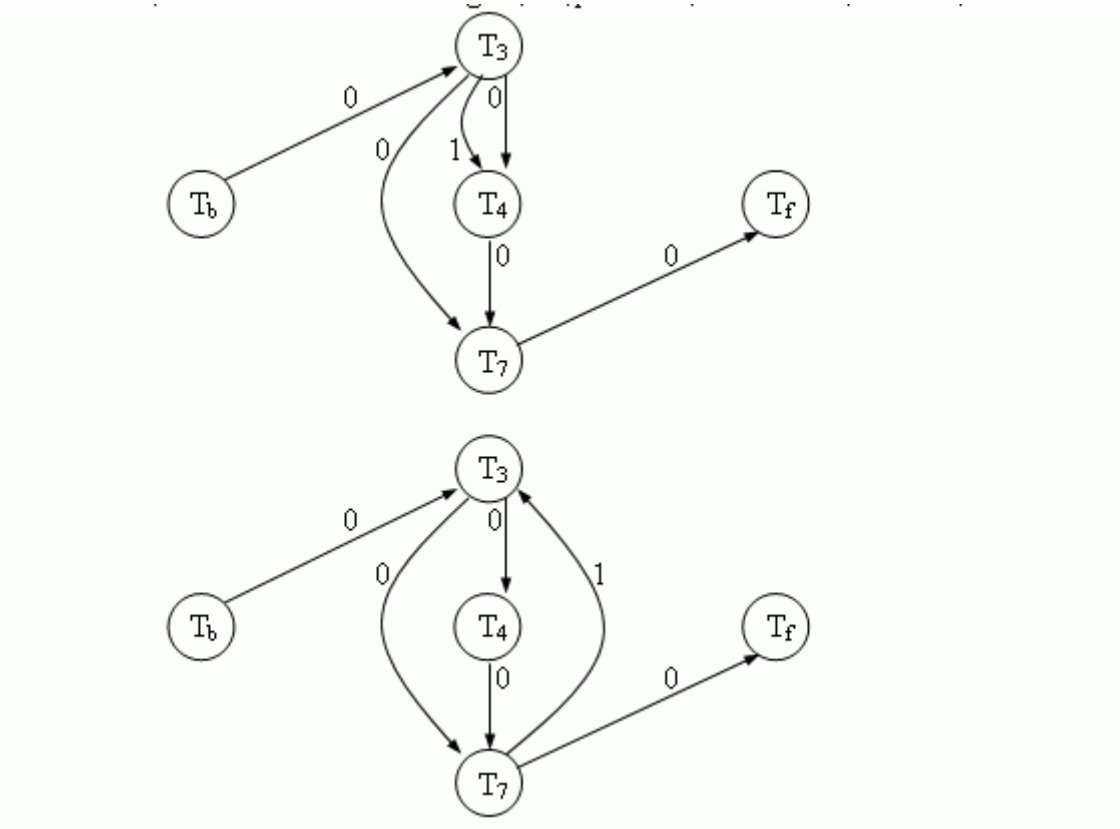
Đồ thị trình tự gán nhãn của schedule-7 chứa chu trình tối tiểu : T3 (T4 (T3

Đồ thị trình tự gán nhãn của schedule-10 chứa chu trình tối tiểu: T3 (T1 (T3

Đồ thị trình tự gán nhãn của schedule-9 không chứa chứa chu trình nào.

Nếu đồ thị trình tự gán nhãn không chứa chu trình, lịch trình tương ứng là khả tuàn tự view, như vậy schedule-9 là khả tuần tự view. Tuy nhiên, nếu dồ thị chứa chu trình, điều kiện này không kéo theo lịch trình tương ứng không là khả tuần tự view. Đồ thị trình tự gán nhãn của schedule-7 chứa chu trình và lịch trình này không là khả tuần tự view. Bên cạnh đó, lịch trình schedule-10 là khả tuần tự view, nhưng đồ thị trình tự gán nhãn của nó có chứa chu trình. Bây giờ ta giả sử rằng có n cặp cung tách biệt, đó là do *ta đã áp dụng n lần quy tắc 3c trong sự xây dung đồ thị trình tự. Khi đó có 2n đồ thị khác* nhau, mỗi một chứa đúng một cung trong mỗi cặp. Nếu một đồ thị nào đó trong các đồ thị này là phi chu trình, khi đó lịch trình tương ứng là khả tuần tự view. Thuật toán này đòi hỏi một phép kiểm thử vét cạn các đồ thị riêng biệt, và như vậy thuộc về lớp vấn đề NP-complet !!

Ta xét đồ thị schedule-10. nó có đúng một cặp tách biệt. Hai đồ thị triên biệt là:



Đồ thị thứ nhất không chứa chu trình, do vậy lịch trình là khả tuần tự view.

BÀI TẬP

***

IV.1 Liệt kê các tính chất ACID. Giải thích sự hữu ích của mỗi một trong chúng.

IV.2 Trong khi thực hiện, một giao dịch trải qua một vài trạng thái đến tận khi nó bàn giao hoặc bỏ dở. Liệt kê tất cả các dãy trạng thái có thể giao dịch có thể trải qua. Giải thích tại sao mỗi bắc cầu trạng thái có thể xảy ra.

IV.3 Giải thích sự khác biệt giữa lịch trình tuần tự (Serial schedule) và lịch trình khả tuần tự (Serializable schedule).

IV.4 Xét hai giao dịch sau:

T_1 : **Read(A);**
Read(B);
If A=0 then B:=B+1;
Write(B).

T_2 : **Read(B);**
Read(A);

If B=0 then A:=A+1;

Write(A).

Giả thiết yêu cầu nhất quán là A=0 V B=0 với A=B=0 là các giá trị khởi đầu

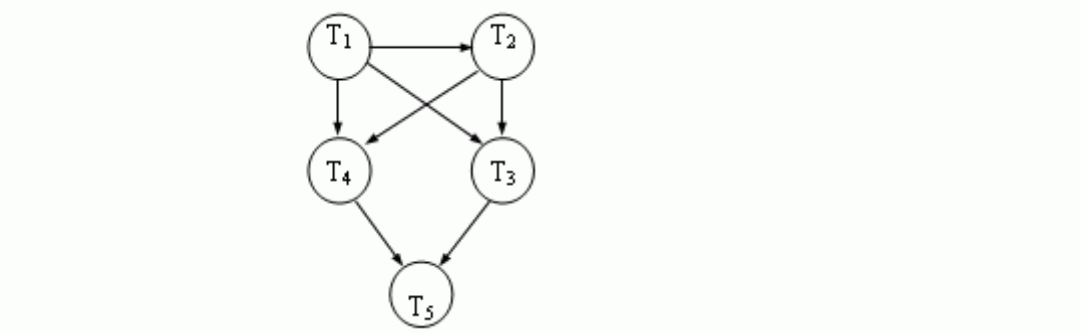
a. Chứng tỏ rằng mỗi sự thự hiện tuần tự bao gồm hai giao dịch này bảo tồn tính nhất quán của CSDL.

b. Nêu một sự thực hiện cạnh tranh của T1 và T2 sinh ra một lịch trình không khả tuần tự.

c. Có một sự thực hiện cạnh tranh của T1 và T2 sinh ra một lịch trình khả tuần tự không ?

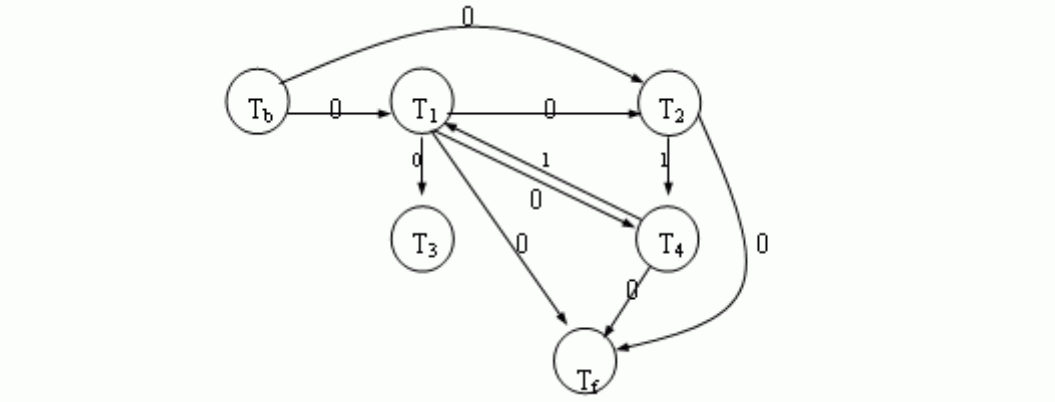
IV.5 Do một lịch trình khả tuần tự xung đột là một lịch trình khả tuần tự view. Tại sao ta **lại nhấn mạnh tính khả tuần tự xung đột hơn tính khả tuần tự view?**

IV.6 Xét đồ thị trình tự sau:



Lịch trình tương ứng là khả tuần tự xung đột không? Giải thích

IV.7 Xét đồ thị trình tự gán nhãn sau:



Lịch trình tương ứng là khả tuần tự view không? Giải thích.

IV.8 Lịch trình khả phục hồi là gì? Tại sao cần thiết tính khả phục hồi của một lịch trình?

IV.9 Lịch trình cascadeless là gì? Tại sao cần thiết tính cascadeless của lịch trình?

ĐIỀU KHIỂN CẠNH TRANH

***

* Mục tiêu

* Kiến thức cần có để học chương này

* Tài liệu tham khảo liên quan đến chương

* Nội dung

* Vấn đề nghiên cứu của chương kế tiếp

***

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.

V.1 GIAO THỨC DỰA TRÊN CHỐT:

V.1.1 Chốt (Lock)

V.1.2 Cấp chốt

V.1.3 Giao thức chốt hai kỳ

V.1.4 Giao thức dựa trên đồ thị

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 Ti nhận được một chốt ở phương thức shared (ký** hiệu là S) trên hạng mục Q, khi đó Ti có thể đọc, nhưng không được viết Q.

2. Exclusive. Nếu một giao dịch Ti nhận được một chốt ở phương thức Exclusive (ký hiệu là X), khi đó Ti 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 Ti đòi hỏi một chốt phương thức A trên hạng mục Q mà trên nó giao dich Tj (Tj (Ti) hiện đang giữ một chốt phương thức B. Nếu giao dịch Ti 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.

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 Ti 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 Ti phải chờ đến tận khi tất cả các chốt không tương thích bị giữ bởi các giao dịch khác được giải phóng.

Giao dịch Ti 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 T1 và T2 . Giao dịch T1 chuyển 50$ từ tài khoản B sang tài khoản A và đươch xác định như sau:

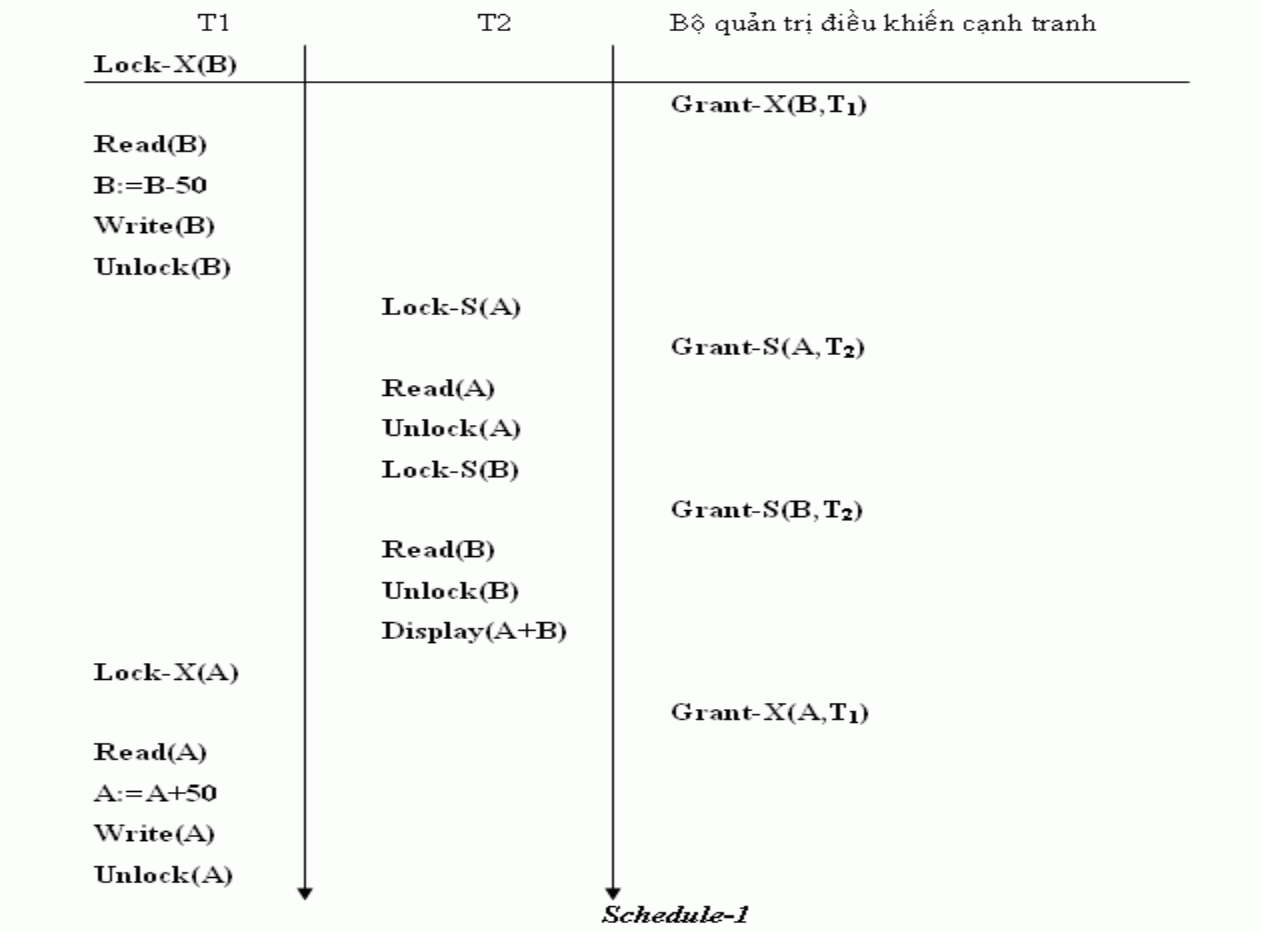
T_1 : **Lock-X(B);**
Read(B);
B:=B-50;
Write(B);
Unlock(B);
Lock-X(A);
Read(A);
A:=A+50;
Write(A);
Unlock(A);

Giao dịch T2 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);

Giả sử giá trị của tài khoản A và B tương ứng là 100$ và 200$. Nếu hai giao dịch này thực hiện tuần tự, hoặc theo thứ tự T1, T2 hoặc theo thứ tự T2 , T1 , và khi dó T2 sẽ hiển thị giá trị 300$. 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 T2 sẽ hiển thị giá trị 250$ --- một

kết quả không đúng. Lý do của sai lầm này là do giao dịch T1 đã tháo chốt hạng mục B quá sớm và T2 đã tham khảo một trạng thái không nhất quán !!!



Lịch trình 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

Bây giờ giả sử rằng tháo chốt bị làm trễ đến cuối giao dịch. Giao dịch T3 tương ứng với T1 với tháo chốt bị làm trễ được định nghĩa như sau:

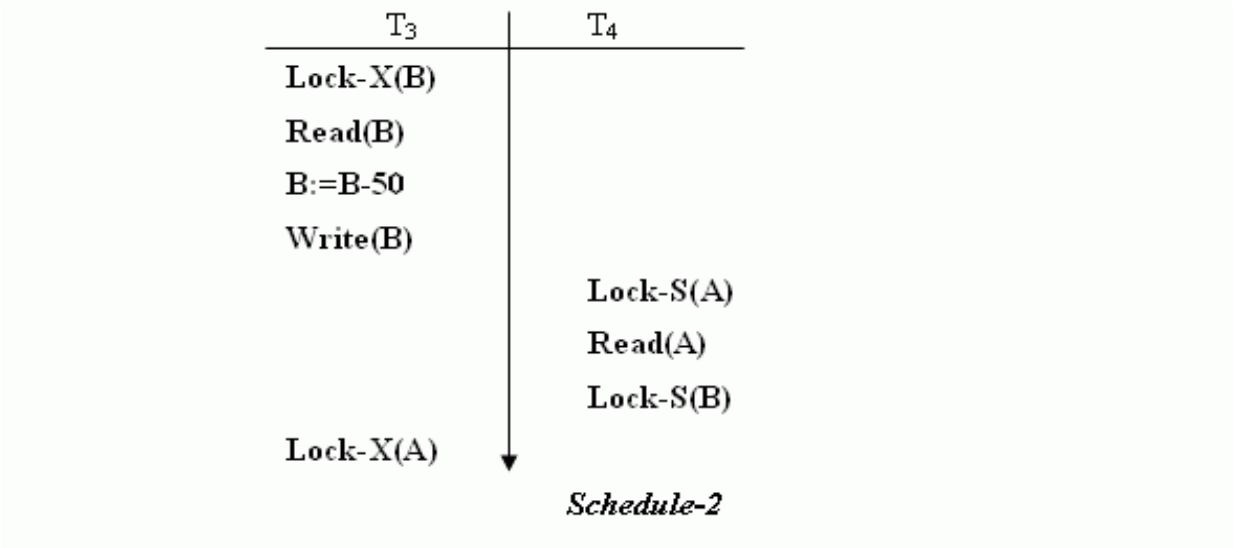
T_3 : **Lock-X(B);**
Read(B);
B:=B-50;
Write(B);
Lock-X(A);
Read(A);
A:=A+50;

Write(A);
Unlock(B);
Unlock(A);

Giao dịch T4 tương ứng với T2 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);

Các lịch trình có thể trên T3 và T4 không để cho T4 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 T3 và T4 sau:



Schedule-2

Do T3 giữ một chốt phương thức Exclusive trên B, nên yêu cầu một chốt phương thức shared của T4 trên B phải chờ đến khi T3 tháo chốt. Cũng vậy, T3 yêu cầu một chốt Exclusive trên A trong khi T4 đang giữ một chốt shared trên nó và như vậy phải chờ. Ta gặp phải tình huống trong đó T3 chờ đợi T4 đồng thời T4 chờ đợi T3 -- 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 { T0 , T1 , ..., Tn } một tập các giao dịch tham gia vào lịch trình S. Ta nói Ti đi trước Tj trong S, và được viết là Ti (Tj , nếu tồn tại một hạng mục dữ liệu Q sao cho Ti giữ chốt phương thức A trên Q , Tj giữ chốt phương thức B trên Q muộn hơn và comp(A,B) = false. Nếu Ti (Tj , thì Ti sẽ xuất hiện trước Tj 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 (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 T2 giữ một chốt phương thức shared trên một hạng mục dữ liệu, một giao dịch khác T1 yêu cầu một chốt phương thức exclusive cũng trên hạng mục này, rõ ràng T1 phải chờ T2 tháo chốt. Trong khi đó một giao dịch khác T3 yêu cầu một chốt phương thức shared, do yêu cầu chốt này tương thích với phương thức chốt được giữ bởi T1 nên nó được cấp cho T3. Tại thời điểm T2 tháo chốt, T1 vẫn phải chờ sự tháo chốt của T3, nhưng bây giờ lại có một giao dịch T4 yêu cầu một chốt phương thức shared và nó lại được cấp do tính tương thích và cứ như vậy, có thể T1 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 Ti 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 Ti

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 phình to (Growing phase). Một giao dịch có thể nhận được các chốt, nhưng có không thể tháo bất kỳ chốt nào
2. Kỳ thu nhỏ (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 phình to. Giao dịch tậu được nhiều chốt như cần thiết. Mỗi khi giao dịch tháo một chốt, nó đi vào kỳ thu nhỏ và nó không thể phát ra các yêu cầu chốt nữa. Các giao dich T3 và T4 là hai kỳ. Các giao dịch T1 và T2 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 phình to, còn hạ cấp chỉ được diễn ra trong kỳ thu nhỏ. 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.

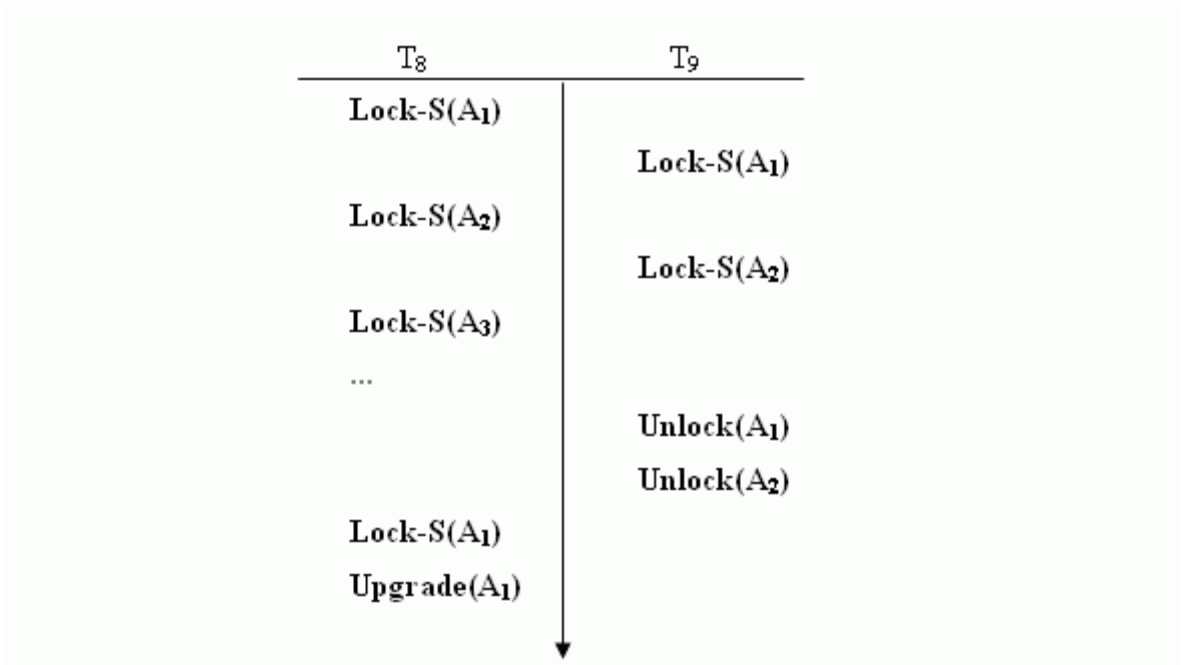
Ta xét một ví dụ: Các giao dịch T8 và T9 được nêu trong ví dụ chỉ được trình bày bởi các hoạt động ý nghĩa là Read và Write.

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

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

Nếu ta sử dụng giao thức chốt hai kỳ, khi đó T8 phải chốt A1 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 T8 cần một chốt exclusive trên A1 chỉ ở cuối sự thực hiện của nó, khi nó write(A1). Như vậy, T8 có thể khởi động chốt A1 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 T8 và T9 có thể truy xuất đến A1 và A2 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 phình to và downgrade chỉ được phép xảy ra trong kỳ thu nhỏ. 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.

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.

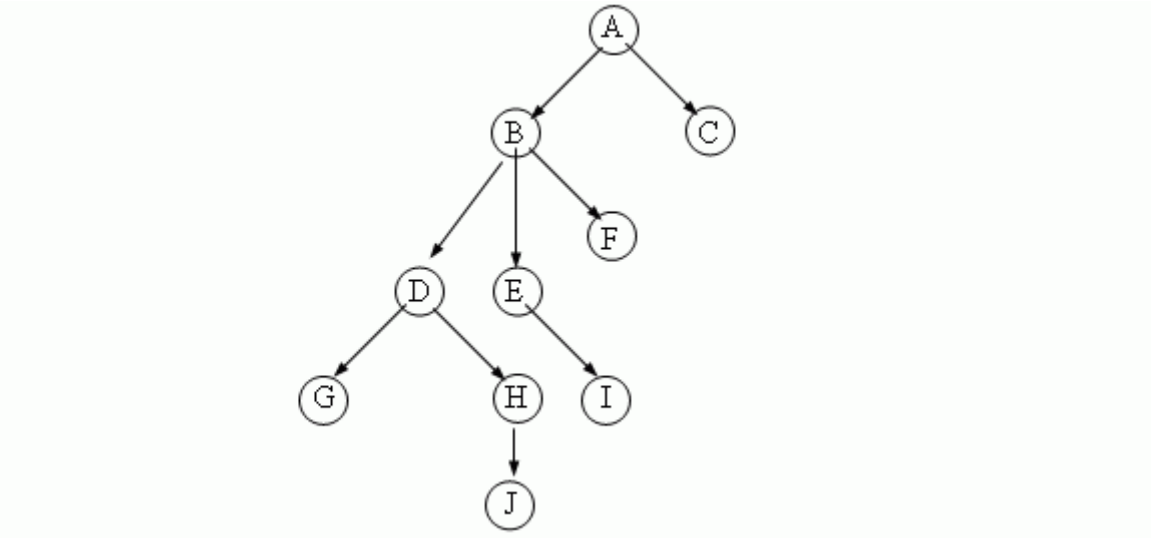
Để có được hiểu biết trước như vậy, ta áp đặt một thứ tự bộ phận, ký hiệu (, trên tập tất cả các hạng mục dữ liệu D ={ d1 , d2 , ..., dn }. Nếu di (dj , bất kỳ giao dịch nào truy xuất cả di và dj phải truy xuất di trước khi truy xuất dj . Thứ tự bộ phận này cho phép xem D như một đồ thị định hướng phi 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 tuân theo giao thức cây là khả tuần tự xung đột.

Ví dụ: Cây CSDL là:



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).

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

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

V.2.1 Tem thời gian

V.2.2 Giao thức thứ tự tem thời gian

V.2.3 Quy tắc viết Thomas

V.2.1 Tem thời gian (Timestamp)

Ta kết hợp với mỗi giao dịch Ti trong hệ thống một tem thời gian cố định duy nhất, được biểu thị bởi TS(Ti). Tem thời gian này được gán bởi hệ CSDL trước khi giao dịch Ti bắt đầu thực hiện. Nếu một giao dịch Ti đã được gán tem thời gian TS(Ti) và một giao dịch mới Tj đi vào hệ thống, khi đó TS(Ti) < TS(Tj). 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(Ti) < TS(Tj), hệ thống phải đảm bảo rằng lịch trình được sinh ra là tương dương với một lịch trình tuần tự trong đó Ti xuất hiện trước Tj .

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 Ti phát ra Read(Q).

a. Nếu TS(Ti) < W-Timestamp(Q), Ti cần đọc một giá trị của Q đã được viết rồi. Do đó, hoạt động Read bị vứt bỏ và Ti bị cuộn lại.

b. Nếu TS(Ti) (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(Ti).

2. Giả sử giao dịch Ti phát ra Write(Q).

a. Nếu TS(Ti) < R-Timestamp(Q), Giá trị của Q mà Ti 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 nũa. Do vậy, hoạt động **Write này bị vứt bỏ và Ti bị cuộn lại**

b. Nếu TS(Ti) < W-Timestamp(Q), Ti đang thử viết một giá trị đã quá hạn của Q, Từ đó, hoạt động Write bị vứt bổ và Ti bị cuộn lại

c. Ngoài ra, hoạt động Write được thực hiện và W-Timestamp(Q) được đặt là TS(Ti)

Một giao dịch Ti 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.

Ta xét các giao dịch T14 và T15 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).

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(T14) < TS(T15), 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

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

Nếu áp dụng giao thức thứ tự tem thời gian, ta có TS(T16) < TS(T17). Hoạt động Read(Q) của T16 và Write(Q) của T17 thành công, khi T16 toan thực hiện hoạt động Write(Q) của nó, vì TS(T16) < TS(T17) = W-timestamp(Q), nên Write(Q) của T16 bị vứt bỏ và giao dịch T16 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 dổi, các quy tắc đối với Write được thay đổi chút ít như sau:

Giả sử giao dịch Ti phát ra Write(Q).

1. Nếu TS(Ti) < R-timestamp(Q), giá trị của Q mà Ti đang sinh ra trước đây là cần thiết và được giả thiết là không bao giờ được sinh ra. Do vậy, hoạt động **Write này bị vứt bỏ và Ti bị cuộn lại.**

2. Nếu TS(Ti) < W-timestamp(Q), Ti 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, nhung Ti không bị cuộn lại).

3. Ngoài ra, hoạt động Write được thực hiện và W-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.3 G IAO 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 Ti , 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 Ti . 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 Ti 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 Ti thành công trong kỳ hợp lệ, các cập nhật hiện hành được áp dụng vào CSDL, nếu không Ti 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 Ti xảy ra. Do vậy, ta sẽ kết hợp ba tem thời gian với giao dịch Ti :

1. Start(Ti). Thời gian khi Ti bắt đầu sự thực hiện.
2. Validation(Ti). Thời gian khi Ti kết thúc kỳ đọc và khởi động kỳ hợp lệ.
3. Finish(Ti). Thời gian khi Ti 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(Ti). Như vậy, giá trị TS(Ti) = Validation(Ti) và nếu TS(Tj) < TS(Tk) thì bất kỳ lịch trình nào được sinh ra phải tương đương với một lịch trình tuần tự trong đó giao dịch Ti xuất hiện trước giao dịch Tk . Lý dota chọn Validation(Ti) như tem thời gian của Ti , mà không chọn Start(Ti), là vì ta hy vọng thời gian trả lời sẽ nhanh hơn.

Phép kiểm thử hợp lệ đối với Tj đòi hỏi rằng, đối với tất cả các giao dịch Ti với TS(Ti) < TS(Tj), một trong các điều kiện sau phải được thoả mãn:

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

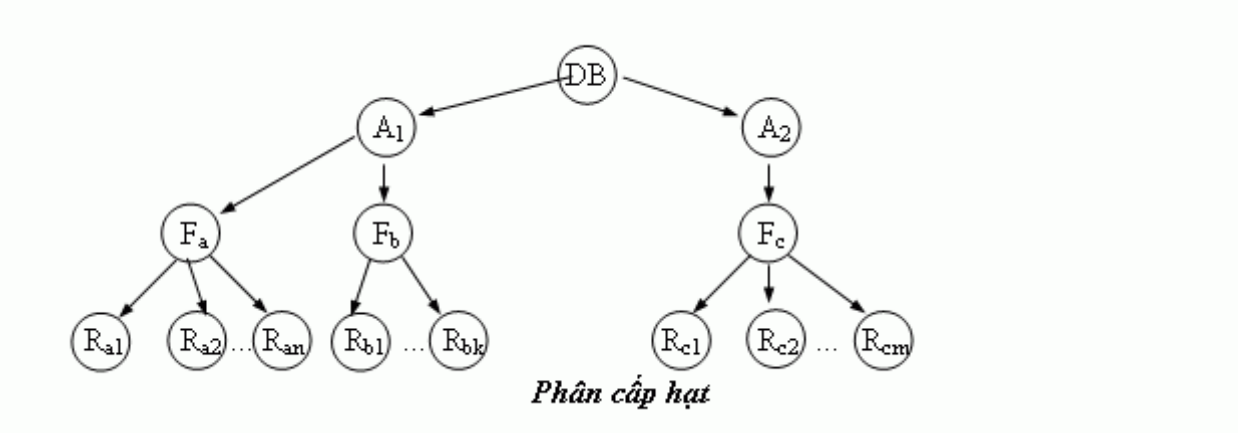
T_{14}	T_{15}
Read(B)	
	Read(B)
	B:=B-50
	Read(A)
	A:=A+50
Read(A)	
Xác nhận tính hợp lệ	
Display(A+B)	
	Xác nhận tính hợp lệ
	Write(B)
	Write(A)

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.4 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 Ti phải truy xuất toàn bộ CSDL và giao thức chốt được sử dụng, khi đó Ti 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 Ti chỉ cần một yêu cầu chốt để chốt toàn bộ CSDL. Mặt khác, nếu Ti 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

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 Ti chốt tường minh file Fb ở phương thức exclusive, nó đã chốt ẩn tất cả các mẩu tin của Fb cũng trong phương thức exclusive.

Giả sử giao dịch Tj muốn chốt mẩu tin Rb6 của file Fb. vì giao địch Ti đã chốt tường minh file Fb, mẩu tin Rb6 cũng bị chốt ẩn. Song làm thế nào để hệ thống biết được Tj có thể chốt Rb6 hay không: Tj phải duyệt cây từ gốc đến mẩu tin Rb6 , nếu có một nút bất kỳ tren đường dẫn bị chốt ở phương thức không tương thích, Tj phải chờ. Bây giờ, nếu Tk muốn chốt toàn bộ CSDL, nó phải chốt nút gốc. Tuy nhiên, do Ti hiện đang giữ một chốt trên Fb, một bộ phận của cây, nên Tk 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

Giao thức chốt đa hạt dưới đây đảm bảo tính khả tuần tự. Mỗi giao địch 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 T18 đọc mẩu tin Ra2 của file Fa. Khi đó T18 cần phải chốt CSDL, vùng A1 và Fa ở phương thức IS và Ra2 ở phương thức S.
- Giả sử giao dịch T19 sửa đổi mẩu tin Ra9 trong file Fa , khi đó T19 cần phải chốt CSDL, vùng A1 và file Fa ở phương thức IX và Ra9 ở phương thức X
- Giả sử giao dịch T20 đọc tất cả các mẩu tin của file Fa , khi đó T20 cần phải chốt CSDL, và vùng A1 ở phương thức IS và chốt Fa ở phương thức S.
- Giả sử giao dịch T21 đọc toàn bộ CSDL, nó có thể lầm điều đó sau khi chốt CSDL ở phương thức S.

Chú ý rằng T18, T20 và T21 có thể truy xuất đồng thời CSDL, giao dịch T19 có thể thực hiện cạnh tranh với T18 nhưng không với T20 hoặc T21

V.5 C ÁC 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.5.1 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(Ti). 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 <Q1, Q2, ... , Qm> mỗi phiên bản Qk chứa ba trường dữ liệu:

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

Một giao dịch, gọi là Ti , tạo ra phiên bản Qk 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 Ti . W-timestamp và R-timestamp được khởi động là TS(Ti). Giá trị R-timestamp được cập nhật mỗi khi một giao dịch Tj đọc nội dung của Qk và R-timestamp(Qk) < TS(Tj).

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 Ti phát ra một hoạt động Read(Q) hoặc Write(Q). Qk 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 Tj phát ra một Read(Q), khi đó giá trị trả lại là nội dung của phiên bản Qk

Nếu Tj phát ra một Write(Q) và nếu TS(Tj) < R-timestamp(Q) khi đó giao dịch Tj bị cuộn lại.

Nếu không, nếu TS(Tj) = W-timestamp(Q) nội dung của Qk 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 Qk và Qj 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 Qj và Qk 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 têu 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 giảo quyết bằng cuộn lại.

V.5.2 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 viết 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 Ti 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(Ti).

Khi một giao dịch cập nhật đọc một hạng mục, nó nhận một chốt shared trên hạng mục, và đọc phiên bản mới nhất của hạng mục. Khi một giao dịch cập nhật muốn viết một hạng mục, đầu tiên nó nhận 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 (

Khi một giao dịch cập nhật Ti hoàn tất các hoạt động của nó, nó thực hiện xử lý bàn giao như sau: Đầu tiên, Ti đặt tem thời gian trên mỗi phiên bản nó đã tạo là TS-counter+1; sau đó Ti 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.6 QUẢN LÝ DEADLOCK:

V.6.1 Phòng ngừa deadlock

V.6.2 Sơ đồ dựa trên Timeout

V.6.3 Phát hiện deadlock và khôi phục

V.6.3.1 Phát hiện deadlock

V.6.3.2 Khôi phục từ 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 { T0 , T2 , ..., Tn } sao cho T0 đang chờ một hạng mục dữ liệu được giữ bởi T1, T1 đang chờ một hạng mục dữ liệu đang bị chiếm bởi T2 , ..., Tn-1 đang chờ một hạng mục dữ liệu được giữ bởi Tn và Tn đang chờ một hạng mục T0 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.6.1 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 T2 yêu cầu một chốt bị giữ bởi giao dịch T1 , chốt đã cấp cho T1 có thể bị lấy lại và cấp chgo T2 , T1 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 Ti yêu cầu một hạng mục dữ liệu bị chiếm bởi Tj , Ti được phép chờ chỉ nếu nó có tem thời gian nhỏ hơn của Tj nếu không Ti bị cuộn lại (die).
2. Sơ đồ Wound-Wait dựa trên kỹ thuật ưu tiên. Khi giao dịch Ti yêu cầu một hạng mục dữ liệu hiện đang bị giữ bởi Tj , Ti được phép chờ chỉ nếu nó có tem thời gian lớn hơn của Tj , nếu không Tj 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 Ti chết và bị cuộn lại vì nó đòi hỏi một hạng mục dữ liệu bị giữ bởi giao dịch Tj , khi đó Ti 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 Tj , Ti bị chết lần nữa. Như vậy, Ti có thể bị chết vài lần trước khi tậu được hạng mục dữ liệu cần thiết. Trong sơ đồ Wound-Wait, Giao dịch Ti bị thương và bị cuộn lại do Tj yêu cầu hạng mục dữ liệu nó chiếm giữ. Khi Ti khởi động lại, và yêu cầu hạng mục dữ liệu, bây giờ, đang bị Tj giữ, Ti 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.6.2 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 đươc cấp trong khoảng thời gian này, giao dịch đươc 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 thục 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.6.3 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:

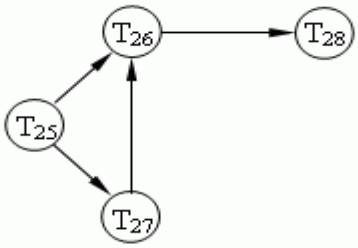
- Duy trì thông tin về sự cấp phát hiện hành các hạng mục dữ liệu cho các giao dịch cũng như các yêu cầu hạng mục dữ liệu chưa được chưa được giải quyết.
- Cung cấp một thuật toán sử dụng các thông tin này để xác định hệ thống đã đi vào trạng thái deadlock chưa.
- Phục hồi từ deadlock khi phát hiện được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

V.6.3.1 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 Ti (Tj , nó chỉ ra rằng Ti chờ Tj giải phóng một hạng mục dữ liệu nó cần.

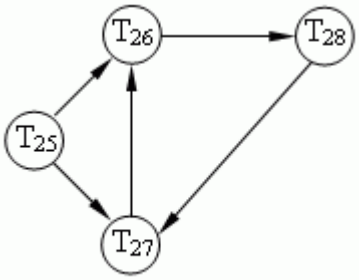
Khi giao dịch Ti yêu cầu một hạng mục dữ liệu đang bị giữ bởi giao dịch Tj khi đó cung Ti (Tj được xen vào đồ thị. Cạnh này bị xoá đi chỉ khi giao dịch Tj không còn giữ hạng mục dữ liệu nào mà Ti cần.

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 deadloch, 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)

Do đồ thị không có chu trình nên hệ thông không trong trạng thái deadlock. Bây giờ, giả sử T28 yêu cầu một hạng mục dữ liệu được giữ bởi T27 , cung T28 (T27 được xen vào đồ thị, điều này dẫn đến tồn tại một chu trình T26 (T27 (T28 (T26 có nghĩa là hệ thống rơi vào tình trạng deadlock và T26 , T27 , T28 bị deadlock.



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 có để 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.6.3.2 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.