CHƯƠNG IV

GIAO DỊCH
(Transaction)

MỤC ĐÍCH

Giới thiệu khái niệm giao dịch, các tính chất một giao dịch cần phải có để sự hoạt động của nó trong môi trường có "biến động" vẫn đảm bảo cho CSDL luôn ở trạng thái nhất quán.

Giới thiệu các khái niệm khả tuần tự, khả tuần tự xung đột, khả tuần tự view, khả phục hồi và cascadeless, các thuật toán kiểm thử tính khả tuần tự xung đột và khả tuần tự view

YÊU CẦU

Hiểu khái niệm giao dịch, các tính chất cần phải có của giao dịch và "ai" là người đảm bảo các tính chất đó

Hiểu các khái niệm về khả tuần tự, khả phục hồi và mối tương quan giữa chúng

Hiểu và vận dụng các thuật toán kiểm thử

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 câu 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 T_i, T_j , hoặc T_j kết thúc thực hiện trước khi T_i khởi động hoặc T_j bắt đầu sự thực hiện sau khi T_i 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.

T_i là một giao dịch chuyển 50 từ tài khoản A sang tài khoản B. Giao dịch này có thể được xác định như sau:

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

figure IV- 1

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 T_i, giá trị của các tài khoản A và B tương ứng là 1000 và 2000. Giả sử rằng trong khi thực hiện giao dịch T_i, một sự cố xảy ra cản trở T_i 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 T_i , 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 cố 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).

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 trạng thái sau:

- **Hoạt động (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:

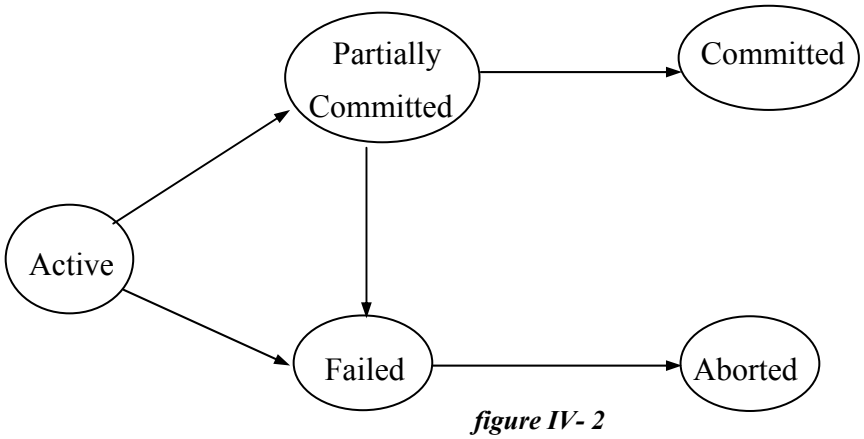


figure IV- 2

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 điểm 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 hoạt động 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.

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 hoạt động 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*.

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 T_1 và T_2 là hai giao dịch chuyển khoản từ một tài khoản sang một tài khoản khác. Giao dịch T_1 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);

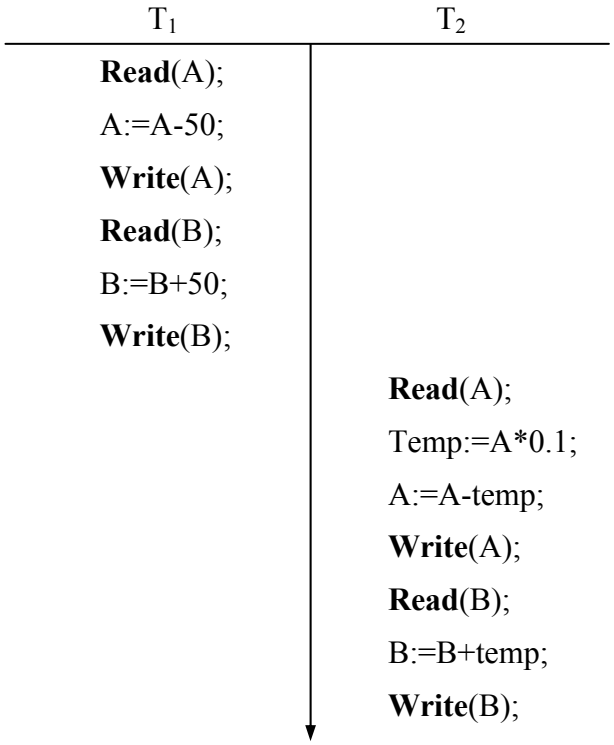
figure IV- 3

Giao dịch T_2 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);

figure IV- 4

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 T_1 rồi tới T_2. 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 T_1 nằm ở cột trái còn các chỉ thị của T_2 nằm ở cột phải:

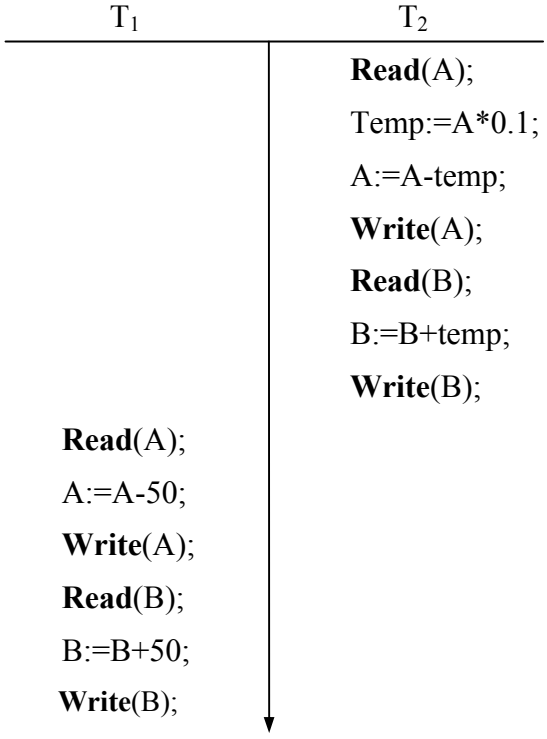


Schedule-1

figure IV- 5

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 T_2 rồi đến T_1 , khi đó dãy thực hiện sẽ là:



Schedule-2

figure IV- 6

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 T_1 , 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.

T_1	T_2
Read(A);	
A:=A-50;	
Write(A);	
	Read(A);
	Temp:=A*0.1;
	A:=A-temp;
	Write(A);
Read(B);	
B:=B+50;	
Write(B);	
	Read(B);
	B:=B+temp;
	Write(B);

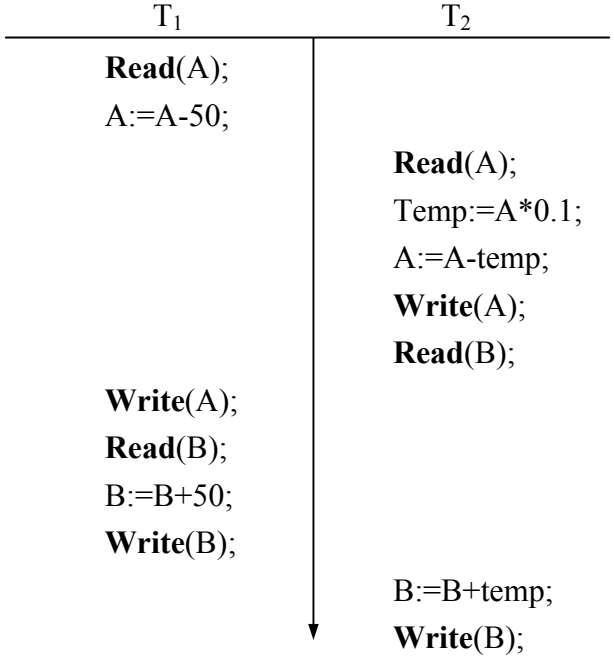
Schedule-3 --- một lịch trình cạnh tranh tương đương schedule-1

figure IV- 7

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.

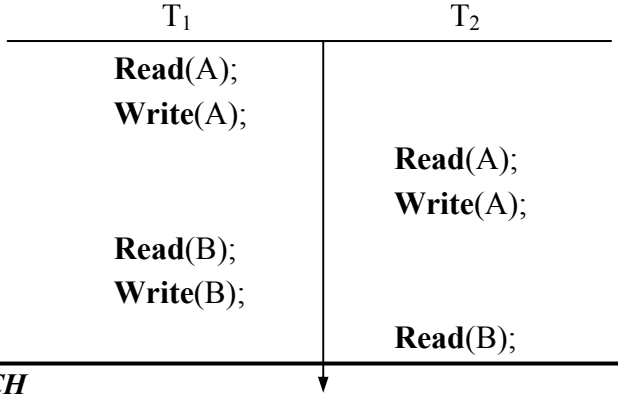


Schedule-4 --- một lịch trình cạnh tranh

figure IV- 8

TÍNH KHẢ TUẦN TỰ (Serializability)

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:



Write(B);

Schedule-3 (viết dưới dạng thoả thuận)

figure IV- 9

TUẦN TỰ XUNG ĐỘT (Conflict Serializability)

Xét lịch trình S trong đó có hai chỉ thị liên tiếp I_i và I_j của các giao dịch T_i , T_j tương ứng ($i \neq j$). Nếu I_i và I_j tham khảo đến các hạng mục dữ liệu khác nhau, ta có thể đổi chỗ I_i và I_j mà không làm ảnh hưởng đến kết quả của bất kỳ chỉ thị nào trong lịch trình. Tuy nhiên, nếu I_i và I_j 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. I_i = **Read**(Q); I_j = **Read**(Q): Thứ tự của I_i và I_j không gây ra vấn đề nào, do T_i và T_j đọc cùng một giá trị Q bất kể đến thứ tự giữa I_i và I_j.
2. I_i = **Read**(Q); I_j = **Write**(Q): Nếu I_i thực hiện trước I_j, Khi đó T_i không đọc giá trị được viết bởi T_j bởi chỉ thị I_j . Nếu I_j thực hiện trước I_i , T_i sẽ đọc giá trị của Q được viết bởi I_j , như vậy thứ tự của I_i và I_j là quan trọng.
3. I_i = **Write**(Q); I_j = **Read**(Q): Thứ tự của I_i và I_j là quan trọng do cùng lý do trong trường hợp trước.
4. I_i = **Write**(Q); I_j = **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 T_i và T_j . 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 I_i và I_j trong S, thứ tự của I_i vf I_j 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 I_i và I_j là các chỉ thị **Read**, thứ tự thực hiện của hai chỉ thị này (trong S) là không gây ra vấn đề.

Ta nói I_i và I_j 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.

T_1	T_2
Read(A);	
Write(A);	
	Read(A);
	Write(A);
Read(B);	
Write(B);	
	Read(B);
	Write(B);

figure IV- 10

Chỉ thị **Write(A)** trong T_1 xung đột với **Read(A)** trong T_2. Tuy nhiên, chỉ thị **Write(A)** trong T_2 không xung đột với chỉ thị **Read(B)** trong T_1 do các chỉ thị này truy xuất các hạng mục dữ liệu khác nhau.

I_i và I_j là hai chỉ thị liên tiếp trong lịch trình S. Nếu I_i và I_j 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 T_2 không xung đột với chỉ thị **Read(B)** của T_1, ta có thể đổi chỗ các chỉ thị này để được một lịch trình tương đương – schedule-5 dưới đây

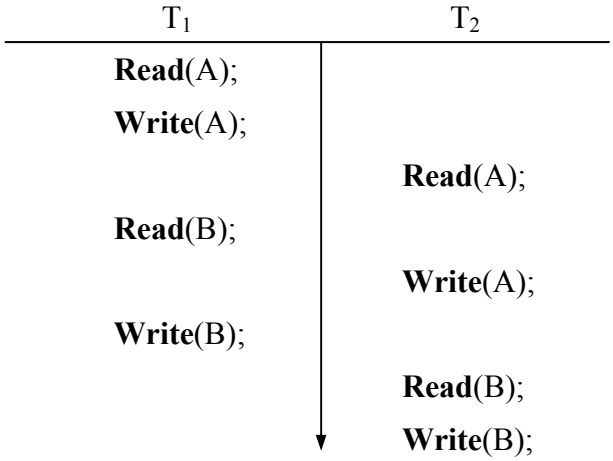
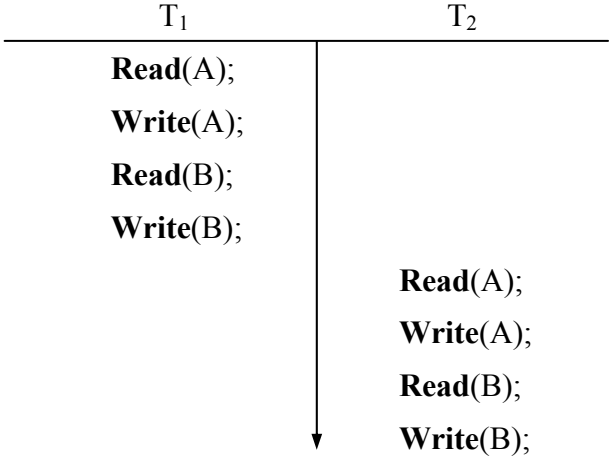


figure IV- 11

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

- Đổi chỗ chỉ thị **Read(B)** của T_1 với chỉ thị **Read(A)** của T_2
- Đổi chỗ chỉ thị **Write(B)** của T_1 với chỉ thị **Write(A)** của T_2
- Đổi chỗ chỉ thị **Write(B)** của T_1 với chỉ thị **Read(A)** của T_2

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):



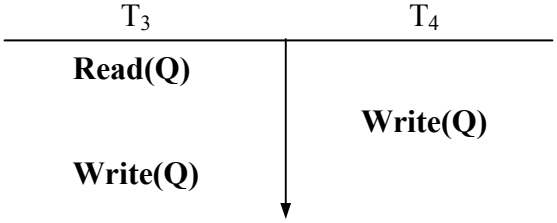
Schedule-6

figure IV- 12

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.

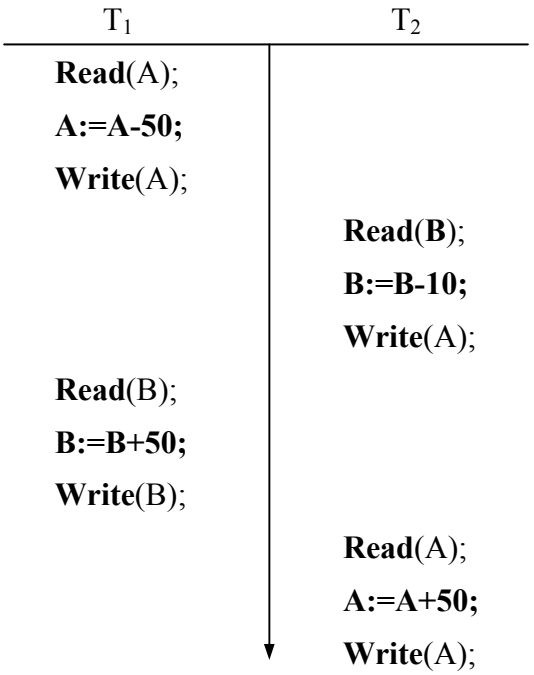
Khái niệm tương đương xung đột dẫn đến khái niệm tuần tự xung đột. Ta nói một lịch trình S là khả tuần tự xung đột (*conflict serializable*) nếu nó tương đương xung đột với một lịch trình tuần tự. Như vậy, schedule-3 là khả tuần tự xung đột. Như một ví dụ, lịch trình schedule-7 dưới đây không tương đương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



Schedule-7

figure IV- 13

Có thể có hai lịch trình sinh ra cùng kết quả, nhưng không tương đương xung đột. Ví dụ, giao dịch T_5 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 $< T_1, T_5 >$ do trong lịch trình schedule-8 chỉ thị **Write(B)** của T_5 xung đột với chỉ thị **Read(B)** của T_1 như vậy ta không thể di chuyển tất cả các chỉ thị của T_1 về trước các chỉ thị của T_5 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 $<T_1, T_5 >$ 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.** Tuy nhiên sự phân tích như vậy sẽ nặng nề và phải trả một giá tính toán cao hơn.



Schedule-8

figure IV- 14

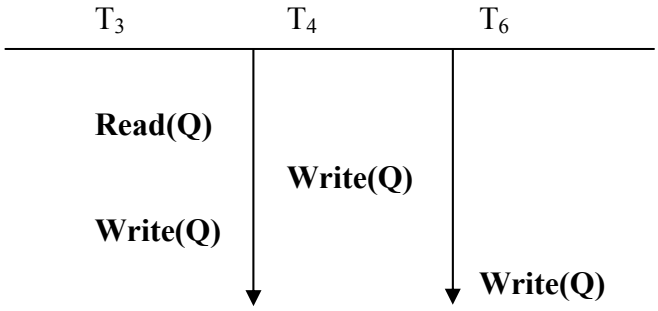
TUẦN TỰ VIEW (View Serializability)

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 T_i đọc giá trị khởi đầu của Q trong lịch trình S, thì giao dịch T_i 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 T_i thực hiện **Read(Q)** trong lịch trình S và giá trị đó được sản sinh ra bởi giao dịch T_j thì T_i cũng phải đọc giá trị của Q được sinh ra bởi giao dịch T_j 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 T_2 được sinh ra bởi T_1 , 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 T_2 được sinh ra bởi T_1 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

figure IV- 15

Nó tương đương view với lịch trình tuần tự < T_3, T_4, T_6 > do chỉ thị **Read(Q)** đọc giá trị khởi đầu của Q trong cả hai lịch trình và T_6 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 T_4 và T_6 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.

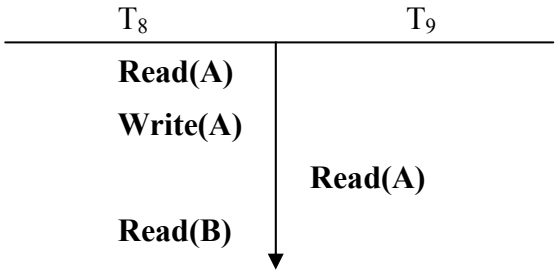
TÍNH KHẢ PHỤC HỒI (Recoverability)

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 T_i 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 T_i 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.

LỊCH TRÌNH KHẢ PHỤC HỒI (Recoverable Schedule)

Xét lịch trình schedule-10 trong đó T_9 là một giao dịch chỉ thực hiện một chỉ thị **Read(A)**. Giả sử hệ thống cho phép T_9 bàn giao (commit) ngay sau khi thực hiện chỉ thị **Read(A).** Như vậy T_9 bàn giao trước T_8 . Giả sử T_8 thất bại trước khi bàn giao, vì T_9 vì T_9 đã đọc giá trị của hạng mục giữ liệu A được viết bởi T_8 , ta phải bỏ dở T_9 để đảm bảo tính nguyên tử giao dịch. Song T_9 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 T_8 .



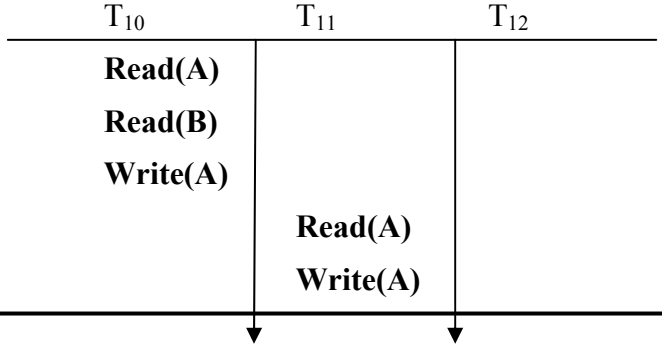
Schedule-10

figure IV- 16

Lịch trình schedule-10 là một ví dụ về lịch trình không phục hồi được và không dược phép. Hầu hết các hệ CSDL đòi 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 T_i , T_j , nếu T_j đọc hạng mục dữ liệu được viết bởi T_i thì hoạt động bàn giao của T_j phải xảy ra sau hoạt động bàn giao của T_i .*

LỊCH TRÌNH CASCADELESS (Cascadeless Schedule)

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



Read(A)

Schedule-11

figure IV- 17

Giao dịch T_{10} viết một giá trị được đọc bởi T_{11} . Giao dịch T_{12} đọc một giá trị được viết bởi T_{11} . Giả sử rằng tại điểm này T_{10} thất bại. T_{10} phải cuộn lại, do T_{11} phụ thuộc vào T_{10} nên T_{11} cũng phải cuộn lại và cũng như vậy với T_{12}. 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 T_i , T_j nếu T_j đọc một hạng mục dữ liệu được viết trước đó bởi T_i, hoạt động bàn giao của T_i phải xuất hiện trước hoạt động đọc của T_J* . Một lịch trình cascadeless là khả phục hồi.

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.

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 dụ, 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.

KIỂM THỬ TÍNH KHẢ TUẦN TỰ

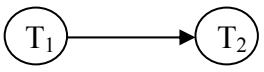
Khi thiết kế các sơ đồ điều khiển cạnh tranh, ta phải chứng tỏ rằng các lịch trình được sinh ra bởi sơ đồ là khả tuần tự. Để làm điều đó, trước tiên ta phải biết làm thế nào để xác định, với một lịch trình cụ thể đã cho, có là khả tuần tự hay không.

KIỂM THỬ TÍNH KHẢ TUẦN TỰ XUNG ĐỘT

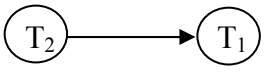
Giả sử S là một lịch trình. Ta xây dựng một đồ thị định hướng, được gọi là đồ thị trình tự (precedence graph), từ S. Đồ thị gồm một cặp (V, E) trong đó V là tập các đỉnh và E là tập các cung. Tập các đỉnh bao gồm tất cả các giao dịch tham gia vào lịch trình. Tập các cung bao gồm tất cả các cung dạng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

1. T_i thực hiện **Write(Q)** trước T_j thực hiện **Read(Q).**
2. T_i thực hiện **Read(Q)** trước khi T_j thực hiện **Write(Q).**
3. T_i thực hiện **Write(Q)** trước khi T_j thực hiện **Write(Q).**

Nếu một cung $T_i \rightarrow T_j$ tồn tại trong đồ thị trình tự, thì trong bất kỳ lịch trình tuần tự S' nào tương đương với S, T_i phải xuất hiện trước T_j .

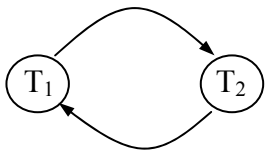
Đồ thị trình tự đối với schedule-1 là: 

vì tất cả các chỉ thị của T_1 được thực hiện trước chỉ thị đầu tiên của T_2

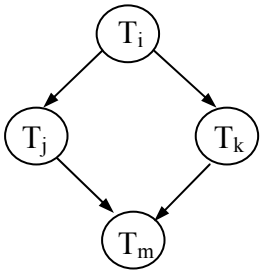
Đồ thị trình tự đối với schedule-2 là: 

vì tất cả các chỉ thị của T_2 được thực hiện trước chỉ thị đầu tiên của T_1

Đồ thị trình tự đối với schedule-4 chứa các cung $T_1 \rightarrow T_2$ vì T_1 thực hiện **Read(A)** trước T_2 thực hiện **Write(A).** Nó cũng chứa cung $T_2 \rightarrow T_1$ vì T_2 thực hiện **Read(B)** trước khi T_1 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 xung đột. 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:

figure IV- 18

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.

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:

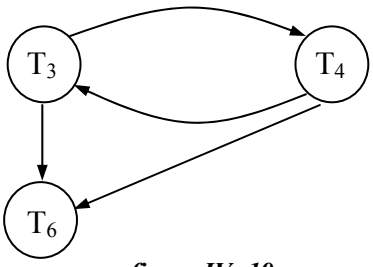


figure IV- 19

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 < T_3, T_4 , T_6 >).

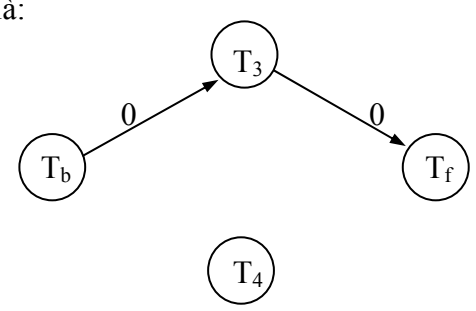
Cung $T_3 \rightarrow T_4$ không được xen vào đồ thị vì các giá trị của hạng mục Q được sản sinh bởi T_3 và T_4 không được dùng bởi bất kỳ giao dịch nào khác và T_6 sản sinh ra giá trị cuối mới của Q. Các chỉ thị **Write(Q)** của T_3 và T_4 được gọi là các **Write vô dụng (Useless Write)**. Điều trên chỉ ra rằng không thể sử dụng đơn thuần sơ đồ đồ thị trình tự để 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 T_j đọc hạng mục dữ liệu Q được viết bởi T_i . Rõ ràng là nếu S là khả tuần tự view, khi đó, trong bất kỳ lịch trình tuần tự S' tương đương với S, T_i phải đi trước T_j . Bây giờ giả sử rằng, trong lịch trình S, giao dịch T_k thực hiện một **Write(Q)**, khi đó, trong lịch trình S', T_k phải hoặc đi trước T_i hoặc đi sau T_j . Nó không thể xuất hiện giữa T_i và T_j vì như vậy T_j không đọc giá trị của Q được viết bởi T_i 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 $T_k \rightarrow T_i$ và $T_j \rightarrow T_k$ 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). Cũng 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:

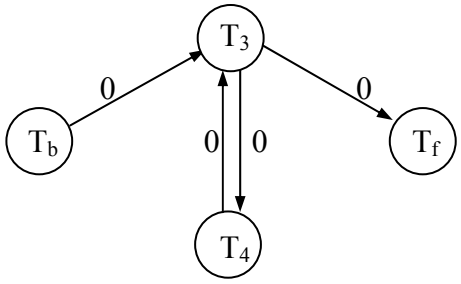
Giả sử S là lịch trình gồm các giao dịch { T_1 , T_2 , ... , T_n }. T_b và T_f là hai giao dịch giả: T_b phát ra **Write(Q)** đối với mỗi Q được truy xuất trong S, T_f phát ra **Read(Q)** đối với mỗi Q được truy xuất trong S. Ta xây dựng lich trình mới S' từ S bằng cách xen T_b ở bắt đầu của S và T_f ở cuối của S. Đồ thị trình tự gán nhãn đố với S' được xây dung dựa trên các quy tắc:

1. Thêm cung $T_i \rightarrow^0 T_j$, nếu T_j 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 T_i được gọi là vô dụng nếu không có con đường nào trong đồ thị trình tự dẫn từ T_i đến T_f .
3. Đối với mỗi hạng mục dữ liệu Q sao cho T_j đọc giá trị của Q được viết bởi T_i và T_k thực hiện **Write(Q),** $T_k \neq T_b$ tiến hành các bước sau
   a. Nếu $T_i = T_b$ và $T_j \neq T_f$, khi đó xen cung $T_j \rightarrow^0 T_k$ vào đồ thị trình tự gán nhãn
   b. Nếu $T_i \neq T_b$ và $T_j = T_f$ khi đó xen cung $T_k \rightarrow^0 T_i$ vào đồ thị trình tự gán nhãn
   c. Nếu $T_i \neq T_b$ và $T_j \neq T_f$ khi đó xen cả hai cung $T_k \rightarrow^p T_i$ và $T_j \rightarrow^p T_k$ 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 T_i viết hạng mục dữ liệu được đọc bởi T_j thì một giao dịch T_k viết cùng hạng mục dữ liệu này phải hoặc đi trước T_i hoặc đi sau T_j . Quy tắc 3a và 3b là trường hợp đặc biệt là kết quả của sự kiện T_b và T_f 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 $T_3 \rightarrow T_4$ là kết quả của 3a, cung $T_4 \rightarrow T_3$ là kết quả của 3b) :



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

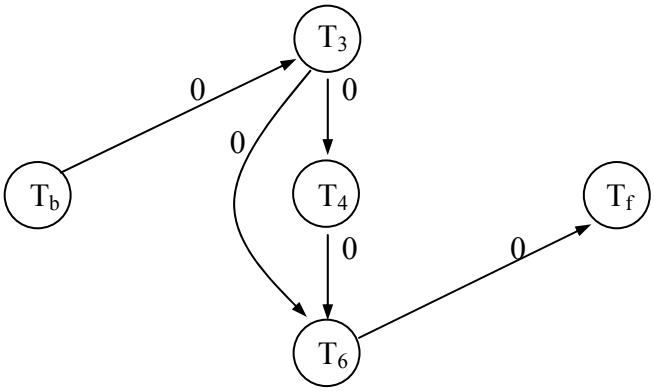
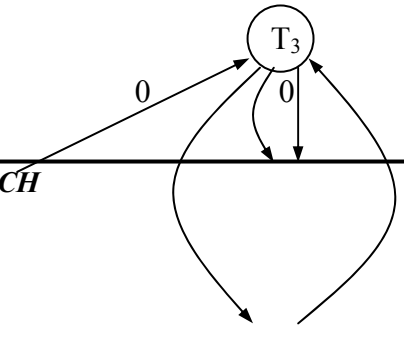


figure IV- 20

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:



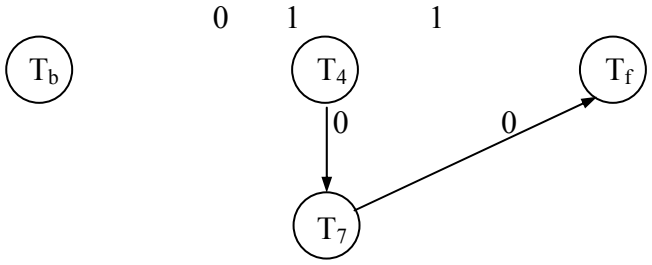


figure IV- 21

Đồ thị trình tự gán nhãn của schedule-7 chứa chu trình tối tiểu : $T_3 \rightarrow T_4 \rightarrow T_3$

Đồ thị trình tự gán nhãn của schedule-10 chứa chu trình tối tiểu: $T_3 \rightarrow T_1 \rightarrow T_3$

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

Nếu đồ thị trình tự gán nhận 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 đồ 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 dựng đồ thị trình tự.* Khi đó có 2^n 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:

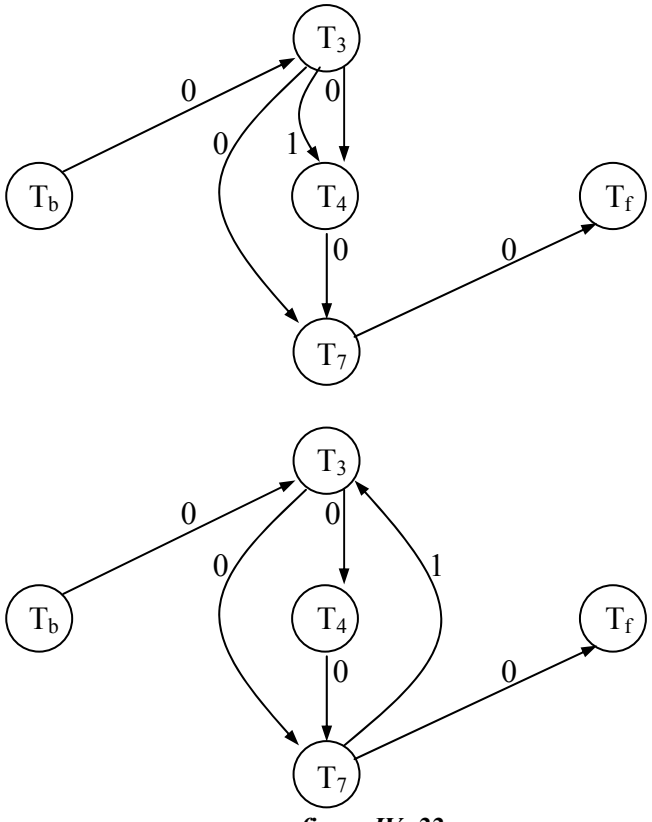


figure IV- 22

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

BÀI TẬP CHƯƠNG IV

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

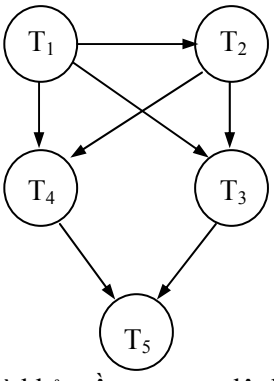
a. Chứng tỏ rằng mỗi sự thự hiện tuần tự bao gồm hai giao dịch này bảo tồn tính nhất quán của CSDL.

b. Nêu một sự thực hiện cạnh tranh của T_1 và T_2 sinh ra một lịch trình không khả tuần tự.

c. Có một sự thực hiện cạnh tranh của T_1 và T_2 sinh ra một lịch trình khả tuần tự không ?

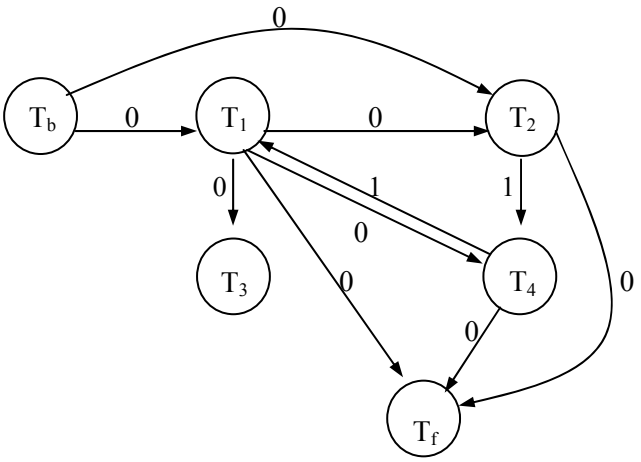
IV.5 Do một lịch trình khả tuần tự xung đột là một lịch trình khả tuần tự view. Tại sao ta lại nhấn mạnh tính khả tuần tự xung đột hơn tính khả tuần tự view?

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?