**Chương 1: Tổ chức lưu trữ**

* **Các công thức tính trên đĩa lưu trữ:**
  + Dung lượng track toàn phần (1 track) = (***B*** + ***G***) \* số block mỗi track
  + Dung lượng track hiệu dụng (1 track) = ***B*** \* số block mỗi track
  + Dung lượng cylinder toàn phần (1 cylinder) = dung lượng track toàn phần \* số đĩa \* số mặt đĩa
  + Dung lượng cylinder hiệu dụng (1 cylinder) = dung lượng track hiệu dụng \* số đĩa \* số mặt đĩa
  + Dung lượng đĩa toàn phần (của toàn bộ đĩa) = dung lượng cylinder toàn phần \* số cylinder
  + Dung lượng đĩa hiệu dụng (của toàn bộ đĩa) = dung lượng cylinder hiệu dụng \* số cylinder
  + Thời gian trễ quay (Rotaional delay) **rd** = 60000 / (2 \* ***p***) (ms)
  + Tốc độ truyền (Transfer rate) **tr** = (kích thước toàn phần 1 track ) / (60000/***p***) (Bytes/ms)
  + Thời gian truyền khối (Block transfer time) **btt** = ***B***/***tr*** (ms)
  + Thời gian tìm và truyền duy nhất một khối = ***s*** + ***rd*** + ***btt*** (ms)
  + Thời gian tìm và truyền **k** khối liên tục trên 1 track = ***s*** + ***rd*** + (k \* ***btt***) (ms)
  + Thời gian tìm và truyền **k** khối không liên tục trên 1 track = ***s*** + (k \* (***rd*** + ***btt***)) (ms)
  + Tốc độ truyền gộp (Bulk transfer rate) **btr = (*B*** / (***B*** + ***G***)) \* ***tr*** (Bytes/ms)
  + Thời gian truyền **k** khối liên tục khi đó = ***s*** + ***rd*** + (k \* (***B*** / ***btr***)) (ms)

**Chương 2: Cấu trúc chỉ mục**

* **Chỉ mục đơn mức:**
  + Trên tập tin dữ liệu:
    - Hệ số phân khối bfr = ⌊B/R⌋ (records/block)
    - Số khối trên tập tin b = ⌈r/bfr⌉ (blocks)
  + Trên tập tin chỉ mục:
    - Kích thước 1 mục chỉ mục Ri = V + PR (bytes)
    - Hệ số phân khối chỉ mục bfri = ⌊B/Ri⌋ (entries/block)
    - Số khối trên tập tin chỉ mục bi = ⌈r/bfri⌉ (blocks)
  + Chi phí truy đạt không dùng tập tin chỉ mục (khi không có thứ tự. Dùng phương pháp tìm kiếm tuần tự) = ⌈b/2⌉ (blocks access)
  + Chi phí truy đạt dùng tập tin chỉ mục (khi có thứ tự. Dùng phương pháp tìm kiếm nhị phân) = ⌈log2b⌉ (blocks access)
* **Các loại chỉ mục đơn mức**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Loại chỉ mục** | **Đặc điểm** | **Các cách hiện thực** | **Số mục chỉ mục** | **Chỉ mục dày hoặc thưa** | **Neo khối** |
| Sơ cấp | - Chỉ mục đặt trên mục tin khóa chính, và tập tin đang có thứ tự theo mục tin này  - 1 mục chỉ mục có dạng <K(i), PB(i)> | Dùng bản ghi ở đầu mỗi khối hoặc cuối mỗi khối để làm neo khối | Bằng số khối được tính ở tập tin dữ liệu | Thưa (vì mỗi khối ở tập tin dữ liệu chỉ được đánh dấu bằng 1 mục chỉ mục) | Bản ghi đầu tiên của khối còn gọi là neo khối |
| Cụm | - Chỉ mục đặt trên mục tin không phải khóa, và tập tin đang có thứ tự theo mục tin này  - 1 mục chỉ mục có dạng <K(i), PB(i)>  - Con trỏ khối ở mỗi mục chỉ mục sẽ trỏ đến khối đầu tiên xuất hiện bản ghi có giá trị ở mục tin đang được chỉ mục = K(i) | - C1: Ở một khối có thể chứa nhiều giá trị phân biệt khác nhau  - C2: Dùng trọn 1 khối hoặc cụm khối liên tục nhau của tập tin dữ liệu để chứa duy nhất 1 giá trị phân biệt | Bằng số giá trị phân biệt ở tập tin dữ liệu | Thưa (vì một giá trị phân biệt mới có một mục chị mục) | - Có neo khối khi hiện thực theo cách 2  - Không có neo khối khi hiện thực theo cách 1 |
| Thứ cấp trên mục tin khóa | - Chỉ mục đặt trên mục tin là khóa (khóa dự tuyển), và tập tin đang không có thứ tự theo mục tin này  - 1 mục chỉ mục có dạng <K(i), PB(i)>  - Con trỏ khối ở mỗi mục chỉ mục sẽ trỏ đến khối xuất hiện bản ghi có giá trị ở mục tin đang được chỉ mục = K(i) |  | Bằng số bản ghi trên tập tin dữ liệu | Dày (vì mỗi bản ghi trên tập tin dữ liệu cần có 1 mục chỉ mục) | Không có |
| Thứ cấp trên mục tin không khóa | - Chỉ mục đặt trên mục tin không phải khóa, và tập tin đang không có thứ tự theo mục tin này | - C1: Nhiều chỉ mục cùng giá trị, có dạng <K(i), PR(i)>  - C2: Mục chỉ mục có chiều dài thay đổi, có dạng <K(i), PR1(i), PR2(i), …>  - C3: Dùng 1 hoặc vài khối trung gian chưa các con trỏ bản ghi và chỉ mục sẻ trỏ đến các khối này, có dạng <K(i), PB(i)> | - Ở C1: bằng số bản ghi  - Ở C2 và C3: bằng số giá trị phân biệt | - Ở C1: dày  - Ở C2 và C3: thưa | Không có |

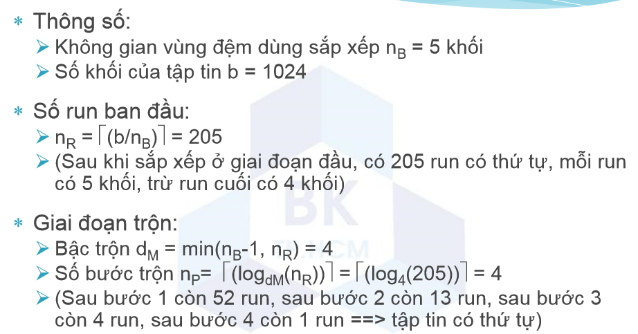
**\*Lưu ý:** vì 1 tập tin chỉ có tối đa 1 thứ tự trên 1 một field nên chỉ có thể dùng chỉ mục sơ cấp hoặc chỉ mục cụm, nhưng không được đồng thời. Một tập tin có thể có nhiều chỉ mục thứ cấp

* **Chỉ mục đa mức:**
  + Hệ số phân khối chỉ mục trong chỉ mục đa mức còn gọi là độ phân nhánh (fan-out). Một khối ở mức cao hơn 1 có thể chứa được bao nhiêu mục chỉ mục thì ta suy ra được có bấy nhiêu khối ở mức thấp hơn
  + Số mức tối đa của 1 mục chỉ mục đa mức: Giả sử mức 1 có ri mục chỉ mục và độ phân nhánh là fo => Số mức tối đa t = ⌈(logfo(ri))⌉
  + Một chỉ mục đa mức có thể được tạo cho bất kỳ kiểu chỉ mục mức 1 nào (sơ cấp, cụm, thứ cấp) miễn là mức đầu tiên có nhiều hơn 1 khối trên đĩa
  + Công thức tổng quát tính số khối ở mức **i** = ⌈bi-1 / fo⌉ (blocks)
* **Các loại chỉ mục đa mức động**

|  |  |  |
| --- | --- | --- |
| **Loại chỉ mục đa mức động** | **Cách tính bậc** | **Tính sức chứa của chỉ mục** |
| B - Tree | -Mỗi nút với bậc p ta có: (p-1) phần tử chỉ mục có dạng <K(i), Pr(i)> và p con trỏ khối  - Với kích thước của mục tin chỉ mục là V, con trỏ khối là P, con trỏ bản ghi là Pr, kích thước khối là B. Mỗi nút phải nằm gọn trong một khối, nên: (p \* P) + ((p - 1) \* (Pr + V)) <= B (Giải biểu thức này để tìm giá trị p lớn nhất có thể có) | - Tại mỗi mức, số mục chỉ mục của mức = số khối \* số phần tử của mỗi khối  - Số khối tại mỗi mức = số con trỏ khối (fan-out) của mức trên  => Tổng số mục tin chỉ mục = tổng số mục tin chỉ mục tại mỗi mức |
| B+ - Tree | - Mỗi nút nội bậc p ta có: (p-1) phần tử chỉ mục có dạng <K(i)> và p con trỏ khối  - Mỗi nút lá bậc pleaf ta có: pleaf phần tử chỉ mục có dạng <K(i), Pr(i)> và 1 con trỏ khối trỏ đến nút kế tiếp  - Mỗi nút phải nằm gọn trong một khối, nên:  (p \* P) + ((p - 1) \* V)) <= B (ở nút nội)  (pleaf \* (V + Pr)) + P <= B (ở nút lá) | - Tại mỗi mức không phải lá, số phần tử trung gian = số khối \* số phần tử của mỗi khối  - Số khối tại mỗi mức = số con trỏ khối (fan-out) của mức trên  - Số mục chỉ mục thực sự chỉ nằm ở mức lá  => Tổng số mục chỉ mục là số phần tử tính được ở mức lá |

**\*Lưu ý**: Mỗi nút trên cây B/B+ là một khối đĩa. Thường một cây B+ cùng cấu hình với một cây B sẽ diễn đạt nhiều mục tin chỉ mục hơn so với cây B

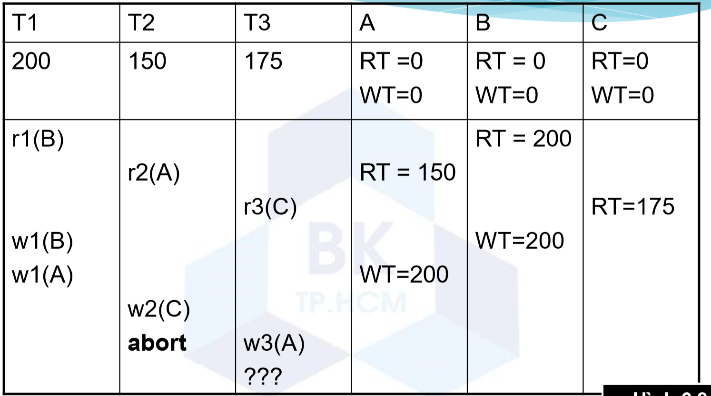
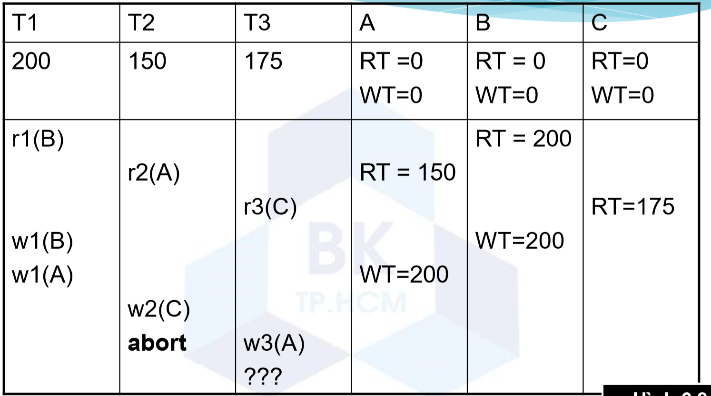
**Chương 3: Tối ưu hóa truy vấn**

* **Có 2 phương pháp chủ yếu về tối ưu:** dùng các quy luật kinh nghiệm (heuristic rule) và Ước lượng chi phí (cost estimate)
* **Giải thuật sort-merge:**
  + **nR**: số run ban đầ
  + **b**: số khối tập tin dữ liệu
  + **nB:** số khối tập tin dữ liệu
  + **dM:** bậc trộn = số run có thể trộn cùng lúc ở mỗi bước trộn
  + **nP:** số bước trộn
  + Ở giai đoạn sắp xếp **nR** = ⌈b/ nB⌉
* **Tóm tắt phương pháp tối ưu hóa dùng heuristic:**
  + Heuristic chính là áp dụng trước các tác vụ thu gọn kích thƣớc của kết quả trung gian
  + Thi hành các tác vụ chọn sớm đến mức có thể để thu gọn số dòng dữ liệu và thi hành các tác vụ chiếu sớm đến mức có thể để thu gọn số thuộc tính của dữ liệu (trung gian)
  + Các tác vụ chọn và kết cho ra ít kết quả cần phải được thi hành trƣớc các tác vụ tương tự
* **Các giải thuật cho phép kết**
  + Kết lặp-lồng (Nested-loop join): Ứng với mỗi bản ghi t trên R truy xuất tất cả bản ghi s trên S và kiểm tra nếu hai bản ghi thoả điều kiện kết (join condition) t[A] = s[B]
  + Kết lặp-đơn (Single-loop join): Nếu một trong hai thuộc tính kết có chỉ mục (hoặc tập tin băm trên thuộc tính đó), giả sử B trên S, ứng với mỗi bản ghi t trên R, truy xuất các bản ghi tương ứng s trên S so cho s[B] = t[A]
  + Kết xếp-và-trộn (Sort-merge join): Khi các bản ghi trên R và S đã được sắp thứ tự theo các thuộc tính kết (A và B tương ứng). Quét R và S đồng thời để truy xuất các bản ghi tương ứng kết được với nhau
  + Kết băm (Hash-join) (\*): Các bản ghi của R và S sẽ được chứa vào cùng một tập tin băm và dùng cùng một hàm băm trên thuộc tính A và B tương ứng. Bước 1: băm tất cả các bản ghi của tập tin ít bản ghi hơn, giả sử R, vào các thùng trong tập tin băm. Bước 2: băm tất cả các bản ghi của S vào các thùng tương ứng và kết bản ghi từ S với các bản ghi tương ứng của R trong cùng thùng

**Chương 4: Xử lý giao tác**

* **Các vần đề khi có tương tranh:**
  + Vấn đề mất cập nhật (Lost Update Problem) : Xảy ra khi hai giao tác truy đạt vào cùng các mục liệu theo cách đan xen và làm cho giá trị của một số mục liệu không đúng. VD: một giao tác sau ghi đè lên một mục liệu mà không biết là mục liệu này đã được cập nhật từ giao tác trước
  + Vấn đề cập nhật tạm (hoặc đọc phải dữ liệu tạm) (The Temporary Update or Dirty Read Problem): Xảy ra khi một giao tác cập nhật một mục liệu và sau đó bị hỏng, trong khi đó, một giao tác khác truy đạt vào mục liệu này trước khi nó được khôi phục giá trị gốc
  + Vấn đề tóm tắt sai (hoặc không đọc lặp lại được) (The Incorrect Summary Problem): Khi một giao tác đang tính toán một tác vụ gộp (như SUM, AVG, …) trên một số bản ghi trong khi các giao tác khác đang cập nhật các bản ghi này, tác vụ gộp có thể sử dụng một số giá trị trước khi chúng được cập nhật và sử dụng một số giá trị sau khi chúng được cập nhật
* **4 tính chất của giao tác:**
  + Atomicity – Tính đơn thể: Một giao tác là một đơn vị đơn thể (atomic unit) các tác vụ: được thực thi toàn bộ (các tác vụ) hoặc không thực thi tác vụ nào cả
  + Consistency preservation – Tính bảo toàn tính nhất quán: Một sự thực thi đúng của một giao tác phải chuyển CSDL từ trạng thái nhất quán này sang trạng thái nhất quán khác
  + Isolation – Tính đơn lập: Một giao tác cần được thực thi một cách đơn lập so với các giao tác khác: Sự thực thi của nó không thể bị can thiệp bởi bất kỳ các giao tác đang được thực thi cùng lúc
  + Durability hoặc permanency – Tính bền vững: Một khi một giao tác thay đổi CSDL đã được commit, các thay đổi này cần phải đảm bảo để không thể mất mát do các hỏng hóc xảy ra sau đó
* **Lịch biểu giao tác**
  + Khi các giao tác được thực thi tương tranh theo cách đan xen, thứ tự thực thi của các tác vụ từ các giao tác tạo thành lịch biểu giao tác (hoặc lịch sử giao tác)
  + Hai tác vụ trong một lịch biểu là có xung đột (conflict) nếu: (1) chúng thuộc về hai giao tác khác nhau. (2) chúng truy đạt vào cùng một mục liệu. (3) ít nhất một trong chúng là lệnh ghi (write\_item)
* **Recoverable schedule – Lịch biểu khả khôi phục:** (1) Không có giao tác nào cần phải quay ngược. (2) Lịch biểu S là khả khôi phục (recoverable) nếu không có giao tác T trong S được commit cho đến khi tất cả các giao tác T’ ghi lên mục liệu mà T đã đọc (sau đó) commit xong. (3) Lịch biểu này có thể yêu cầu các quay ngược dắt dây (cascading rollback): buộc quay ngược một giao tác chưa commit đọc một mục liệu đã ghi bởi một giao tác hỏng
* **Cascadeless schedule – Lịch biểu không dắt dây:** Khi các giao tác chỉ đọc các mục liệu đã ghi bởi các giao tác đã commit
* **Strict Schedules – Lịch biểu nghiêm cách:** Một giao tác không được đọc hoặc ghi một mục liệu X cho đến khi giao tác cuối có ghi lên X đã được commit
* **Lịch biểu tuần tự (Serial schedule):** với mỗi giao tác T tham gia vào lịch biểu, tất các các tác vụ của T được thực thi liên tiếp nhau trong S (Lưu ý: Các giao tác phải được thực hiện 1 cách tuần tự, không có sự đan xen giữa các tác vụ của các giao tác khác nhau)
* **Lịch biểu khả tuần tự hoá (Serializable schedule):** Lịch biểu S là khả tuần tự hoá (serializable) nếu nó tương đương với một lịch biểu tuần tự với cùng các giao tác trong lịch biểu này (Lưu ý: khi thực hiện lịch biểu thì có sự đan xen giữa các tác vụ của các giao tác khác nhau)
* **Kiểm tra tính khả tuần tự hóa:** Với 1 lịch biểu S được lập từ N giao tác (T1,T2,T3,..,Tn) thì ta sẻ kiểm tra bằng đồ thị có hướng như sau:
  + Với 1 giao tác Ti ta vẽ ra 1 đỉnh trên đồ thị
  + Nếu một tác vụ của Ti xuất hiện trước một tác vụ có xung đột (với nó) là Tj trong lịch biểu thì ta sẽ vẽ một cạnh Ti -> Tj
  + Cách nhận biết 1 cặp tác vụ xung đột: Read – Write, Write – Write, Write – Read
  + Khi duyệt hết lịch biểu S mà ta được 1 đồ thì không có chu trình thì kết luận rằng lịch biểu S này khả tuần tự hóa, và ngược lại

**Chương 5: Điều khiển tương tranh**

* **Kỹ thuật dùng khóa:**
  + Giao tác T phải dùng tác vụ lock\_item(X) trước bất kỳ lệnh read\_item(X) hoặc write\_item(X) nào trong T
  + Giao tác T phải dùng tác vụ unlock\_item(X) sau tất cả các lệnh read\_item(X) và write\_item(X) đã hoàn thành trong T
  + Có 2 chế độ khóa:
    - Khóa chia sẻ (shared lock – read lock): read\_lock (X) - Các giao tác có thể dùng chung khóa cùng lúc để đọc cùng 1 mục liệu(X), không thể dùng write lock (để ghi dữ liệu) trên X tại cùng thời điểm
    - Khóa loại trừ (exclusive lock – write lock): write\_lock (X) - Chỉ có tối đa một write lock trên mục liệu X tại bất kỳ thời điểm nào, không thể có thêm share lock nào khác trên X tại cùng thời điểm
* **Kỹ thuật khóa 2 pha:**
  + Trong một giao tác, hai pha này cần phải loại trừ lẫn nhau: Trong pha khoá, không có tác vụ mở khoá nào. Trong pha mở khoá, không có tác vụ khoá nào
  + **Lưu ý:** Nếu tất cả các giao tác trong lịch biểu đều dùng kỹ thuật khoá hai pha, lịch biểu được đảm bảo sẽ là khả tuần tự hoá
  + Giải thuật khóa 2 pha bảo thủ: Ngăn chặn khoá chết (deadlock) bằng cách khoá tất cả các mục liệu cần thiết trước khi bắt đầu thực hiện giao tác
  + Giải thuật khóa 2 pha nghiêm cách: Đòi hỏi giao tác không mở khoá ghi (write or exclusive lock) của nó cho đến khi kết thúc (bằng commit hoặc abort)
  + Giải thuật khóa 2 pha khắt khe: Đòi hỏi các giao tác chỉ mở khoá (đọc hoặc ghi) khi kết thúc (bằng commit hoặc abort)
* **Phát hiện khỏa chết (dead lock) bằng đồ thị đợi:**
  + Mỗi giao tác xuất hiện trong lịch biểu là một đỉnh cùng tên
  + Khi giao tác Ti đợi để khoá mục liệu X đang khoá bởi giao tác Tj, thêm cạnh Ti  Tj vào đồ thị
  + Nếu đồ thị có chu trình => lịch biểu có khóa chết. Khi này 1 giao tác nào đó trong chu trình sẽ được chọn là vật hi sinh để quay ngược (rollback)
* **Kỹ thuật dùng tem thời gian đơn giản:**
  + Ý tưởng:mỗi 1 mục liệu sẽ được gán hai tem thời gian là Read\_TS(X) và Write\_TS(X). Read\_TS(X) sẽ mang giá trị lớn nhất của tất cả các tem thời gian của các giao tác đã đọc mục liệu X, tương tự cho Write\_TS(X)
  + Giao tác được xếp trước (older) sẽ có nhãn thời gian nhỏ hơn giao tác xếp sau (younger)
  + Khoá chết không xảy ra nếu dùng giải thuật này
  + Khi 1 giao tác T cần thực hiện tác vụ write\_item(X): Nếu read\_TS(X) > TS(T) hoặc write\_TS(X) > TS(T), huỷ và quay ngược T và từ chối tác vụ ghi này (*nghĩa là có một giao tác trẻ hơn (vào sau) đã đọc hoặc ghi mục liệu X, tức vi phạm thứ tự thực thi về tem thời gian*). Ngược lại, thi hành tác vụ write\_item(X) của T và gán write\_TS(X) bằng TS(T)
  + Khi 1 giao tác T cần thực hiện tác vụ read\_item(X): Nếu write\_TS(X) > TS(T), huỷ và quay ngược T và từ chối tác vụ đọc này (*nghĩa là có một giáo tác trẻ hơn đã ghi mục liệu X, tức vi phạm thứ tự thực thi về tem thời gian*). Ngược lại, thi hành tác vụ read\_item(X) của T và gán read\_TS(X) là giá trị lớn hơn giữa TS(T) và giá trị read\_TS(X) hiện tại
* **Kỹ thuật dùng tem thời gian đa phiên bản:**
  + Ý tưởng:duy trì một số phiên bản (X1, X2,…,Xn) của mỗi mục liệu và mỗi phiên bản có kèm theo 2 tem đọc – ghi là read\_TS(Xi) - write\_TS(Xi), tìm và cấp đúng phiên bản cho tác vụ đọc của một giao tác, như vậy, tác vụ đọc sẽ không bao giờ bị từ chối
  + read\_TS(Xi): Tem thời gian lớn nhất của các giao tác đã đọc phiên bản Xi
  + write \_TS(Xi): Tem thời gian lớn nhất của các giao tác đã ghi phiên bản Xi
  + **Lưu ý:** Một phiên bản mới của mục liệu X chỉ được tạo ra bởi một tác vụ ghi X nào đó
  + Luật để có được tính khả tuần tự hoá:
    - Nếu giao tác T cần thực hiện tác vụ write\_item(X) và phiên bản Xi của X có giá trị write\_TS(Xi) là cao nhất trong tất cả các phiên bản k của X mà có write\_TS(Xk) <= TS(T) và read \_TS(Xi) > TS(T), thì huỷ bỏ và quay ngược T. Ngược lại, tạo ra phiên bản mới Xj và gán read\_TS(Xj) = write\_TS(Xj) = TS(T)
    - Nếu giao tác T cần thực hiện tác vụ read\_item(X), tìm phiên bản i của X có giá trị write\_TS(Xi) là cao nhất trong tất cả các phiên bản k của X mà có write\_TS(Xk) <= TS(T), trả về giá trị Xi cho T và gán read \_TS(Xi) bằng giá trị lớn hơn trong hai giá trị TS(T) và giá trị hiện tại của read\_TS(Xi). (*Luật này đảm bảo là không có lệnh đọc nào bị từ chối)*