Transaction

Tính chất của giao tác: ACID

- **Atomic – Tính nguyên tố**
  Không thể chia nhỏ.
- **Consistency – Tính nhất quán**
  Chuyển CSDL từ trạng thái nhất quán này sang trạng thái nhất quán khác.
- **Isolation – Tính cô lập**
  Các giao tác xử lý đồng thời phải độc lập với những thay đổi của giao tác khác.
- **Durability – Tính lâu dài, bền vững**
  Khi giao tác hoàn tất, tất cả thay đổi phải được ghi nhận chắc chắn lên CSDL.

- **Atomic:** Nguyên tố → Thực thi tất cả hoặc không gì cả
- **Consistent**: Nhất quán → Trước và sau khi thực thi transaction, CSDL luôn ở 1 trạng thái nhất quán
- **Isolate:** Độc lập → Tuân thủ nguyên tắc xử lý đồng thời, các giao tác (transaction) được xử lý cùng lúc mà không cần chờ đợi, tuy nhiên vẫn đảm bảo tính đúng đắn
  - Trong hệ thống xử lý tuần tự từng giao tác, tính độc lập được đảm bảo → Tính đúng đắn được đảm bảo. Tuy nhiên tính hiệu quả không cao
  - Trong hệ thống xử lý đồng thời, các giao tác được phép bắt đầu ngay cả khi giao tác khác chưa hoàn thành, tính hiệu quả cao nhưng cần có cơ chế đảm bảo tính độc lập và tính đúng đắn. Ví dụ như giao tác A đang cập nhật 1 giá trị x nhưng chưa hoàn thành giao tác. Tính đồng thời cho phép giao tác B được quyền xem xét giá trị của x cùng lúc đó, nếu như A rollback transaction (dự tính cập nhật nhưng không hoàn thành được nên trả về trạng thái trước đó) → B có thể đọc dữ liệu x sai
- **Durable**: Tính bền vững
  - Cần có nguyên tắc rollback những transaction chưa hoàn thành vì 1 nguyên nhân nào đó, cho dù các thao tác của transaction đó có được ghi nhận vật lý hay chưa (ghi nhận xuống ổ cứng trong quá trình thao tác)
  - Cần có 1 cơ chế để ghi nhận kết quả thực thi của transaction xuống ổ đĩa vật lý sau khi thực thi thành công, dù cho sau đó vì một lý do gì đó mà hệ thống ngưng kết nối (mất điện)

CÁC VẤN ĐỀ CỦA XỬ LÝ ĐỒNG THỜI:

- **Mất dữ liệu cập nhật (Lost update):**

Cả 2 giao tác T1 và T2 xử lý đồng thời trên 1 dữ liệu A, T1 thực hiện cập nhật giá trị A nhưng chưa thực hiện Commit tran (cập nhật giá trị vào A) thì T2 đã gọi đến A và cập nhật đè lên A (mất đi giá trị ở A mà T1 vừa cập nhật) → Sai lệch về dữ liệu cập nhật trên A

- **Đọc dữ liệu chưa Commit (Uncommit data, Dirty read):**

T1 và T2 cùng truy cập vào A, T1 cập nhật A nhưng chưa Commit tran, T2 gọi đến A và sử dụng kết quả cập nhật của T1, tuy nhiên T1 sau đó Rollback tran → Sai lệch về dữ liệu đọc từ T2

- **Thao tác đọc không thể lặp lại (Unrepeatable data):**

T1 và T2 cùng truy cập vào giá trị A, T1 đang đọc dữ liệu A, tuy nhiên trong lúc này T2 cập nhật giá trị của A, sau đó T1 lại đọc dữ liệu A lần 2 → 2 lần đọc của T1 mặc dù không thay đổi nhưng vẫn mang giá trị khác nhau, vi phạm tính isolate

- **Dữ liệu bóng ma (Phantom):**

T1 và T2 truy cập vào dữ liệu A, T1 thực hiện select trên A và sử lý các select đó. Tuy nhiên, lúc này T2 thực hiện insert hoặc delete trên A → T1 không cập nhật được các giá trị insert hoặc delete này nên vẫn xử lý trên dữ liệu A select được ban đầu → Sai lệch về dữ liệu select từ A

- **Các giải pháp giải quyết vấn đề xử lý đồng thời**

Có 3 cơ chế để giải quyết vấn đề này: cơ chế khóa (lock-base), nhãn thời gian (timestamp-base), xác nhận hợp lệ (validation)

Lịch biểu

1. Hoạt động của các giao dịch đồng thời được coi là đúng đắn nếu và chỉ nếu tác dụng của nó giống như tác dụng có được khi cho thực hiện chúng một cách tuần tự.
2. Lịch biểu (schedule)

   Đn1: Lịch biểu của một tập các giao tác là thứ tự trong đó các thao tác trong giao tác được thực hiện. Thứ tự của các thao tác trong lịch biểu phải tuân theo đúng thứ tự của chúng trong giao tác cho trước.

   Đn2: Một lịch biểu là một chuỗi sắp theo thời gian các hành động được thực hiện bởi một hoặc nhiều giao tác.

- Lịch biểu được gọi là tuần tự (serial) nếu thứ tự thực hiện các thao tác trong lịch biểu là tất cả các thao tác của giao tác này rồi đến tất cả các thao tác của giao tác khác và cứ như vậy.
- Mọi lịch biểu tuần tự đều đảm bảo tính nhất quán cho cơ sở dữ liệu.
- Lịch biểu được gọi là khả tuần tự (serializable) nếu tác dụng của nó giống như tác dụng của một lịch biểu tuần tự nào đó. Tức là chúng sinh ra cùng một giá trị cho mỗi đơn vị dữ liệu.

- Lịch biểu có thể là khả tuần tự do hành vi cụ thể của các giao tác trong lịch biểu.
- Bộ lập lịch không điều khiển đồng thời dựa vào hành vi cụ thể của từng thao tác trong giao tác.
- Bộ lập lịch dựa trên một nguyên tắc chung để điều khiển đồng thời.

- Lịch biểu có thể phục hồi được (Recoverable Schedule)

Mọi HQT CSDL yêu cầu lịch biểu phải có thể phục hồi được.

Định nghĩa: Một lịch biểu có thể phục hồi được là lịch biểu mà mọi cặp Ti, Tj, khi Tj đọc đơn vị dữ liệu vừa được ghi bởi Ti, thao tác commit của Ti xuất hiện trước thao tác commit của Tj.

- Lịch biểu không rollback dây chuyền (Uncascading rollback Schedule)

T1	T2	T3
Read A		
Read B		
Write A		
	Read A	
	Write A	
		Read A

Khi T1 fail, phải rollback T2 và kéo theo T3 phải bị rollback.

Hiện tượng một transaction rollback, dẫn đến một loạt các transaction khác phải rollback gọi là rollback dây chuyền (Cascading rollback).

- Định nghĩa: Lịch biểu không rollback dây chuyền là lịch biểu mà trong đó mọi cặp giao tác Ti, Tj, nếu Tj đọc đơn vị dữ liệu được viết trước đó bởi Ti, thao tác commit của Ti phải xuất hiện trước thao tác đọc của Tj.

- Một lịch biểu không rollback dây chuyền là lịch biểu có khả năng phục hồi được.

Lịch biểu

❑Tính tương thích của 2 thao tác

Định nghĩa: Hai thao tác O_i, O_j là tương thích nếu kết quả của việc thực hiện đồng thời O_i, O_j, giống như kết quả của việc thực hiện tuần tự O_i, O_j hoặc O_j, O_i
Hai thao tác tương thích thì không xung đột nhau.

T1	T2
Read (A)	
	Read(A)

T1	T2
Read (A)	
	Read(B)

T1	T2
Read (A)	
	Write(B)

Hai thao tác không tương thích nhau thì xung đột nhau.

T1	T2
Read (A)	
	Write(A)

T1	T2
	Write(A)
Read(A)	

T1	T2
Write(A)	
	Write(A)

Tính khả hóan vị của 2 thao tác

❑ Định nghĩa: Hai thao tác Oi, Oj là khả hoán vị nếu kết quả của việc thực hiện Oi, Oj hay Oj, Oi là như nhau.

❑ Hai thao tác tương thích thì khả hóan vị.

❑ Hai thao tác truy xuất trên cùng đơn vị dữ liệu, ma trận tương thích

	Read A	Write A
Read A	1	0
Write A	0	0

• Các thao tác truy xuất trên các đơn vị dữ liệu khác nhau thì tương thích và khả hóan vị.

❑ Với một lịch biểu S1 cho trước, ta có thể lặp lại việc hoán đổi vị trí của hai thao tác không xung đột liền kề, cho đến khi đạt được 1 lịch biểu tuần tự S2, nếu có thể. Khi đó, lịch biểu S1 ban đầu là lịch biểu khả tuần tự.

❑ Mọi tác động đến CSDL hoàn toàn không thay đổi khi ta thực hiện hóan đổi vị trí các thao tác không xung đột nhau.

❑ Hai lịch biểu S1 và S2 là ***tương đương xung đột*** nếu ta có thể chuyển lịch biểu S1 thành lịch biểu S2 bằng một/một số các thao tác hóan đổi các thao tác không xung đột kề nhau.

❑ Một lịch biểu là ***khả tuần tự xung đột*** nếu nó tương đương xung đột với một lịch biểu tuần tự.

❑ Khả tuần tự xung đột là điều kiện đủ để đảm bảo tính khả tuần tự. Nghĩa là, một lịch biểu khả tuần tự xung đột là một lịch biểu khả tuần tự.

❑ Không đòi hỏi một lịch biểu phải khả tuần tự xung đột để là lịch biểu khả tuần tự, nhưng khả tuần tự xung đột là điều kiện để những bộ lập lịch trong các hệ thống thương mại bảo đảm tính khả tuần tự.

Lịch biểu

Xét tính khả tuần tự của S

S

T1	T2	T3	T4
	1.Read A		
		2.Read A	
	3.Write B		
		4.Write A	
5.Read B			
			6.Read B
7.Read A			
8.Write C			
			9.Write A

S'

T1	T2	T3	T4
	1.Read A		
	3.Write B		
		2.Read A	
		4.Write A	
5.Read B			
7.Read A			
8.Write C			
			6.Read B
			9.Write A

Lịch biểu S' tương đương xung đột với S nên S là khả tuần tự xung đột → S khả tuần tự

- **Khả tuần tự xung đột không là điều kiện cần cho tính khả tuần tự.**

  $S1: w_1(Y); w_1(X); w_2(Y); w_2(X); w_3(X)$

  $S2: w_1(Y); w_2(Y); w_2(X); w_1(X); w_3(X)$
- **S1: T3 ghi X, T2 ghi Y**
- **S2: T1, T2 ghi trên X nhưng T3 ghi đè, T2 ghi Y.**
- **Hai lịch biểu trên cho giá trị trên X và Y giống nhau.**
- **S1 tuần tự**

⇒ **S2 khả tuần tự**

⇒ **nhưng không khả tuần tự xung đột do không thể chuyển S2 về 1 lịch biểu tuần tự bằng cách hóan đổi.**

Khả tuần tự xung đột ⇒ ⇏ **Khả tuần tự**

Kiểm tra tính khả tuần tự

Với lịch biểu có Rlock và Wlock

Input: LB S gồm T1, T2, …, Tk

Output: S có khả tuần tự hay không

1. **Nếu Ti Slock trên đvdl X, tiếp theo Tj yêu cầu Xlock trên X, thì có cung Ti → Tj.**
2. **Ti khóa Xlock trên X, Tj yêu cầu Xlock trên X, có cung từ Ti→Tj.**
3. **Tm là GT giữ Slock trên X sau khi Ti nhả khóa độc quyền trên X, nhưng trước khi Tj khóa Xlock trên X, (nếu không có Tj thì Tm là giao tác yêu cầu Slock trên X sau khi Ti nhả khóa trên X), thì có cung từ Ti đến Tm.**
4. **Nếu đồ thị có chu trình thì S không khả tuần tự. Ngược lại thì S khả tuần tự và có thể tìm được lịch tuần tự tương đương.**

T1	T2
RL(A)	
Read(A)	
UL(A)	
	RL(B)
	Read(B)
	UL(B)
	WL(A)
	Read(A)
	A:=A+B
	Write(A)
	UL(A)
WL(B)	
Read(B)	
B:=B+A	
Write(B)	
UL(B)	

B1 

B2 

G

B3 G có chu trình => S không khả tuần tự xung đợt

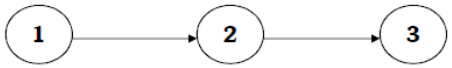
Lịch biểu

- Đồ thị ưu tiên (đồ thị đụng độ) để kiểm tra một lịch biểu là khả tuần tự xung đột hay không.
  - Input: Lịch biểu S.
  - Output: S khả tuần tự?
- Cho *S* là lịch biểu gồm các thao tác của 2 giao tác Ti và Tj. Ti ưu tiên hơn Tj (Ti <Tj) nếu tồn tại cặp thao tác Oin ∈Ti, Ojm ∈Tj không khả hoán vị và Oin thực hiện trước Ojm.
- Đồ thị ưu tiên có nút là các giao tác, cung có hướng đi từ nút i đến nút j nếu Ti < Tj.
- Lịch biểu là khả tuần tự xung đột nếu đồ thị ưu tiên không có chu trình.
- Nếu đồ thị ưu tiên có chu trình, lịch biểu là không khả tuần tự xung đột.

Ví dụ

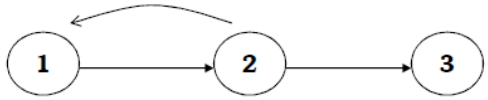
- S1: $r_2(A); r_1(B); w_2(A); r_3(A); w_1(B); w_3(A); r_2(B); w_2(B)$

T1	T2	T3
	r(A)	
r(B)		
	w(A)	
		r(A)
w(B)		
		w(A)
	r(B)	
	w(B)	



- Lịch biểu đã cho khả tuần tự, tương đương với lịch tuần tự T1< T2 < T3

- S: $r_2(A); r_1(B); w_2(A); r_2(B); r_3(A); w_1(B); w_3(A); w_2(B)$



T1	T2	T3
	r(A)	
r(B)		
	w(A)	
	r(B)	
		r(A)
w(B)		
		w(A)
	w(B)	

- Lịch biểu đã cho không khả tuần tự xung đột.

5. ĐIỀU KHIỂN ĐỒNG THỜI DÙNG KỸ THUẬT KHÓA

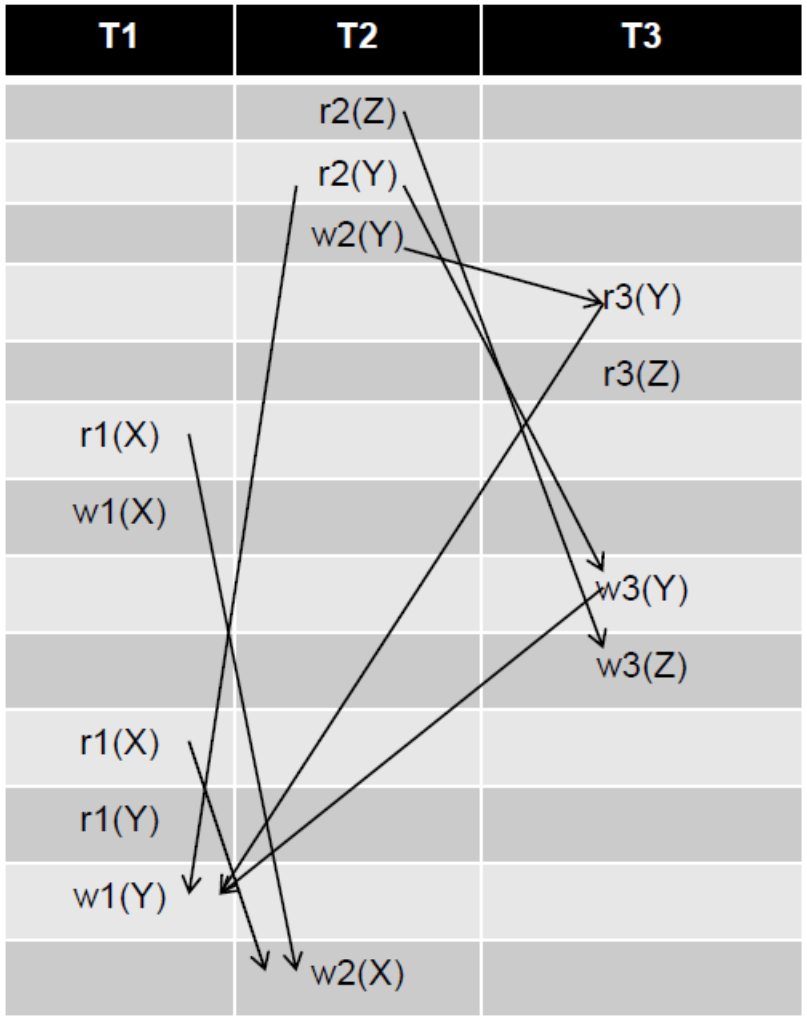
- Kỹ thuật khóa đơn giản: Lock (A), Unlock (A).
- Nghi thức khóa 2 giai đoạn (2PL).
- Kỹ thuật khóa Đọc/ Ghi: Rlock(A), Wlock(A), Unlock (A).
  - Tăng cấp (upgrading).
  - Khóa cập nhật (update lock).
  - Khóa tăng/ giảm (increment lock)
  - Khóa trên đvdl có kích thước khác nhau.
- Nghi thức khóa trên đvdl tổ chức phân cấp.

Lịch biểu

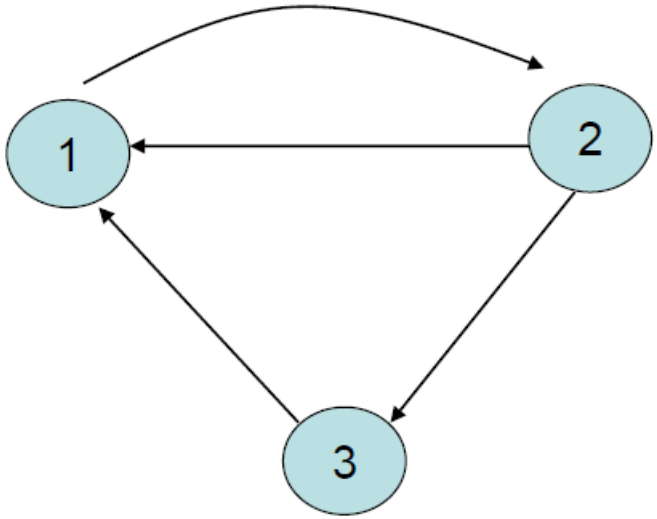
Dạng bài tập 1: Input: Lịch biểu S

Output: S có khả tuần tự? Dùng đồ thị ưu tiên.

r2(Z), r2(Y), w2(Y), r3(Y), r3(Z), r1(X), w1(X), w3(Y), w3(Z), r1(X), r1(Y), w1(Y), w2(X)



T2 < T3



Đồ thị có chu trình. Lịch biểu đã cho là không khả tuần tự xung đột

Lịch biểu

Kỹ thuật khóa đơn giản

Bộ lập lịch dùng kỹ thuật khóa yêu cầu:

- Phải khóa và nhả khóa ngoài việc Đọc/Ghi dữ liệu.
- Việc dùng khóa phải tuân theo 2 điều kiện sau:

1. Tính nhất quán của giao tác:
   a) Giao tác chỉ có thể đọc hoặc ghi trên đơn vị dữ liệu nếu trước đó có yêu cầu lock trên đơn vị dữ liệu và chưa nhả lock.
   b) Nếu giao tác đã lock trên đvdl thì sau đó phải unlock.
2. Tính hợp lệ của lịch biểu:
   - Không thể có 2 giao tác đồng thời khóa trên 1 đvdl.

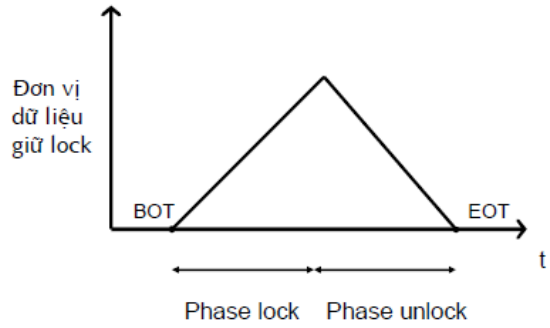
$l_i(X)$:	T_i yêu cầu lock trên đvdl X
$u_i(X)$:	Ti unlock trên trên đvdl X

Hạn chế của PP ĐKĐT dùng khóa

- Livelock: tình trạng 1 giao tác chờ hoài để được làm việc trên 1 đvdl.
- Deadlock: tình trạng hai giao tác cứ chờ nhau mãi để được làm việc trên 1 đơn vị dữ liệu mà không có giao tác nào có thể thực hiện trước.

Nghi thức khóa 2 giai đoạn (Two-phase lock Protocol - 2PL)

Phát biểu 2PL: Một giao tác thực hiện cơ chế khóa 2 giai đoạn khi giao tác không thực hiện lock nào nữa sau khi đã unlock.



T1	T2
L(A)	L(B)
Read(A)	Read(B)
L(B)	L(A)
Read(B)	Read(A)
B:=B+A	UL(B)
Write(B)	A:=A+B
UL(A)	Write(A)
UL(B)	UL(A)

T1, T2 thỏa nghi thức khóa 2 giai đoạn

Lịch biểu

Nghi thức khóa nghiêm ngặt (Strict locking)

- Khóa nghiêm ngặt: Tất cả các khóa độc quyền của 1 giao tác bất kỳ phải giữ cho đến khi giao tác commit hoặc abort, và lệnh commit/ abort phải được ghi nhật ký trên đĩa.
- Lịch biểu trong đó các giao tác tuân thủ cách khóa nghiêm ngặt là lịch biểu nghiêm ngặt (strict schedule).
  - Mọi lịch biểu nghiêm ngặt là lịch biểu không rollback dây chuyền.
  - Mọi lịch biểu nghiêm ngặt là khả tuần tự.
    - Vì lịch biểu nghiêm ngặt tương đương với lịch biểu tuần tự theo thứ tự mà GT commit.

Nguyên tắc khóa

1. Tính nhất quán của giao tác
   - Thao tác đọc $r_i(X)$ phải đi sau $sl_i(X)$ hoặc $xl_i(X)$ mà không có thao tác $u_i(X)$ xen vào giữa.
   - Thao tác ghi $w_i(X)$ phải đi sau $xl_i(X)$ mà không có thao tác $u_i(X)$ xen vào giữa.
   - Tất cả các lock phải được unlock trên cùng đvdl.
2. Giao tác thỏa 2PL
   - Không có thao tác $u_i(Y)$ nào đi trước $sl_i(X)$ hoặc $xl_i(X)$
3. Lịch biểu hợp lệ
   - Nếu có thao tác $xl_i(X)$ trong lịch biểu thì không thể có $xl_j(X)$ hoặc $sl_j(X)$ theo sau, $j \neq i$ mà không có lệnh $u_i(X)$ nào ở giữa.
   - Nếu có thao tác $sl_i(X)$ trong lịch biểu thì không thể có $xl_j(X)$ theo sau, $j \neq i$ mà không có lệnh $u_i(X)$ ở giữa.

Các phương thức khóa

- Shared lock (S)
  - Shared Lock ⇔ Read Lock
  - Khi đọc 1 đơn vị dữ liệu, SQL Server tự động thiết lập Shared Lock trên đơn vị dữ liệu đó
  - Có thể được thiết lập trên 1 bảng, 1 trang, 1 khóa hay trên 1 dòng dữ liệu.
  - Nhiều giao tác có thể đồng thời giữ Shared Lock trên cùng 1 đơn vị dữ liệu.
  - Không thể thiết lập Exclusive Lock trên đơn vị dữ liệu đang có Shared Lock.
  - Shared Lock thường được giải phóng ngay sau khi sử dụng xong dữ liệu được đọc, trừ khi có yêu cầu giữ shared lock cho đến hết giao tác.
- Exclusive Locks (X)
  - Exclusive Lock ⇔ Write Lock
  - Khi thực hiện thao tác ghi (insert, update, delete) trên 1 đơn vị dữ liệu, SQL Server tự động thiết lập Exclusive Lock trên đơn vị dữ liệu đó.
  - Exclusive Lock luôn được giữ đến hết giao tác.
  - Tại 1 thời điểm, chỉ có tối đa 1 giao tác được quyền giữ Exclusive Lock trên 1 đơn vị dữ liệu.

Ma trận tương thích

	S	X
S	yes	no
X	no	no

Khoá tăng cấp

❑ Khóa tăng cấp (Upgrading lock)

- Shared lock: thân thiện
- T muốn đọc X trước, sau đó ghi X thì trước tiên khóa đọc trên X, sau đó khi muốn ghi thì nâng cấp (upgrade) khóa đọc thành khóa ghi.
  - Giao tác hoàn toàn có thể yêu cầu lock trên cùng một đvdl với nhiều chế độ lock khác nhau.
- Cách này cho phép tăng tính đồng thời.

T1	T2
$sl_1(A); r_1(A);$	
	$sl_2(A); r_2(A);$
	$sl_2(B); r_2(B);$
$sl_1(B); r_1(B); \ldots$	
$xl_1(B)$ denied	
	$u_2(A); u_2(B);$
$xl_1(B); w_1(B);$	
$u_1(A); u_1(B);$	

Nếu T1 khóa độc quyền trên B đầu tiên thì T1 và T2 không thể thực hiện được nhiều việc đồng thời.

Upgrading lock giúp tăng tính đồng thời

❑ Update Lock $ul_i(X)$

- Update Lock sử dụng khi đọc dữ liệu với dự định ghi trở lại trên đơn vị dữ liệu này.
- $Ul_i(X)$ chỉ cho phép giao tác T_i quyền đọc X, không cho phép T_i quyền ghi trên X. Tuy nhiên, chỉ có update lock mới có thể được nâng cấp lên thành khóa độc quyền sau đó, khóa đọc thì không thể nâng cấp.
- Update Lock là chế độ khóa trung gian giữa Shared Lock và Exclusive Lock.
- Update lock giúp tránh deadlock mà khóa tăng cấp gặp phải.
- Khi thực hiện thao tác ghi lên 1 đơn vị dữ liệu thì bắt buộc Update Lock phải được nâng cấp thành Exclusive Lock.
- Tại 1 thời điểm chỉ cho phép 1 giao tác giữ Update lock trên 1 đvdl

	S	X	U
S	Yes	No	Yes
X	No	No	No
U	No	No	No

Có thể cấp lock U trên A khi đã có S lock trên A. Nhưng khi đã có lock U trên A thì không thể cấp bất cứ loại lock nào trên A, vì nếu cho phép thì lock U không bao giờ có cơ hội nâng cấp thành khóa độc quyền.

Lịch biểu

Input: Lịch biểu S, các phương thức khóa HQT CSDL dùng để thực hiện S, gồm:

a. Khóa đơn giản, dùng {L, UN}

b. Khóa đọc - ghi, dùng {SL, XL, UN}

c. Khóa đọc - ghi, cho phép tăng cấp, dùng {SL, XL, UN, upgrading)

d. Khóa đọc - ghi, update lock, dùng {SL, XL, UL, UN}

e. Khóa đọc - ghi và khóa tăng/giảm, dùng {SL, XL, IL, UN}

Cho biết quá trình thực hiện của S

S: r1(A), r2(B), r3(C), r1(B), r2(C), r3(D), w1(A), w2(B), w3(C)

a. Dùng {lock, unlock}

T1	T2	T3
L(A)		
R(A)		
	L(B)	
	R(B)	
		L(C)
		R(C)
L(B) Chờ		
	L(C) Chờ	
		L(D)
		R(D)
		W(C)
		UN(C)

T1	T2	T3
	L(C)	
	R(C)	
		UN(D)
	W(B)	
	UN(B)	
L(B)		
R(B)		
	UN(C)	
W(A)		
UN(A)		
UN(B)		

S: r1(A), r2(B), r3(C), r1(B), r2(C), r3(D), w1(A), w2(B), w3(C)

a. Dùng {SL, XL}

T1	T2	T3
XL(A)		
R1(A)		
	XL(B)	
	R2(B)	
		XL(C)
		R3(C)
SL(B) Chờ		
	SL(C) Chờ	
		SL(D)
		R3(D)
		W3(C)
		Un(C)

	SL(C)	
	R2(C)	
		Un(D)
	W2(B)	
	Un(B)	
SL(B)		
R1(B)		
	Un(C)	
W1(A)		
Un(A)		
Un(B)		

Lịch tuần tự tương đương: T3 < T2 < T1

Lịch biểu

S: r1(A), r2(B), r3(C), r1(B), r2(C), r3(D), w1(A), w2(B), w3(C)

b. Dùng {SL, XL và khóa tăng cấp} upgrading

T1	T2	T3
SL(A)		
R1(A)		
	SL(B)	
	R2(B)	
		SL(C)
		R3(C)
SL(B)		
R1(B)		
	SL(C)	
	R2(C)	
		SL(D)
		R3(D)
XL(A)		
W1(A)		
	XL(B) Chờ	

		XL(C) Chờ
Un(A)		
Un(B)		
	XL(B)	
	W2(B)	
	Un(B)	
	Un(C)	
		XL(C)
		W3(C)
		Un(C)
		Un(D)

Lịch tuần tự tương đương: T1 < T2 < T3

S: r1(A), r2(B), r3(C), r1(B), r2(C), r3(D), w1(A), w2(B), w3(C)

c. Dùng {SL, XL và UL}

T1	T2	T3
UL(A)		
R1(A)		
	UL(B)	
	R2(B)	
		UL(C)
		R3(C)
SL(B) Chờ		
	SL(C) Chờ	
		SL(D)
		R3(D)
		XL(C)
		W3(C)
		Un(C)

	SL(C)	
	R2(C)	
		Un(D)
	XL(B)	
	W2(B)	
	Un(B)	
SL(B)		
R1(B)		
		Un(C)
XL(A)		
W1(A)		
Un(A)		
Un(B)		

Lịch tuần tự tương đương: T3 < T2 < T1

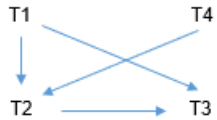
Lịch biểu

xl1(A); r1(A); xl2(A); w2(A); rl1(B); r1(B); xl2(B); r2(B); rl3(A); r3(A); rl4(B); r4(B); wl3(B); w3(B); w1(A); un1(A); un1(B); w2(B); un2(A); un2(B); un3(A); un3(B); un4(B)

Cho biết bộ lập lịch (BLL) được trang bị phương thức khóa gì? Hãy cho biết quá trình thực hiện của lịch biểu. Lịch biểu đã cho có khả tuần tự không? Nếu có, lịch tuần tự tương đương là gì?
=> BLL được trang bị phương thức khóa đọc, khóa ghi, nhả khóa ở cuối giao tác. Quá trình thực hiện lịch biểu: 1, 2, 1, 2, 3, 4, 3, 1, 2

T1	T2	T3	T4
XL(A)			
R(A)			
	XL(A) chờ		
RL(B)			
R(B)			
	XL(A) chờ		
		RL(A) chờ	
			RL(B)
			R(B)
W(A)			
UN(A)			
	XL(A)		
	W(A)		
UN(B)			
			UN(B)
	XL(B)		
	R(B)		
	W(B)		
	UN(A)		
		RL(A)	
		R(A)	
	UN(B)		
		WL(B)	
		W(B)	
		UN(A)	
		UN(B)	

Đồ thị ưu tiên:



Đồ thị trên không có chu trình nên lịch biểu đã cho là khả tuần tự xung đột

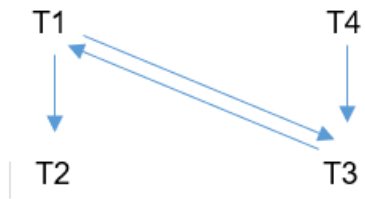
=> Lịch biểu đã cho là khả tuần tự. Đồng thời, khả tuần tự vì các giao tác nhất quán , thoả 2PL, và lịch biểu hợp lệ.

Lịch tuần tự tương đương là: T1 < T4 < T2 < T3

Với lịch biểu S đã cho, giả sử BLL được trang bị phương thức khóa đọc, khóa ghi, có hỗ trợ tăng cấp, nhả khóa ở cuối giao tác thì quá trình thực hiện của lịch biểu ra sao? Lịch biểu đã cho có khả tuần tự không? Nếu có, lịch tuần tự tương đương là gì?

T1	T2	T3	T4
RL(A)			
R(A)			
	WL(A) chờ		
RL(B)			
R(B)			
		RL(A)	
		R(A)	
			RL(B)
			R(B)
		WL(B) chờ	
WL(A) chờ			
			UN(B)

Đồ thị ưu tiên:

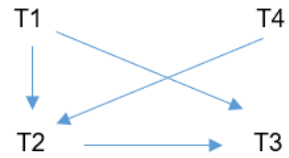


Đồ thị trên có chu trình nên lịch biểu đã cho không phải là khả tuần tự.

Với lịch biểu S đã cho, nhưng giả sử BLL được trang bị phương thức khóa đọc, khóa ghi, khóa update, nhả khóa ở cuối giao tác thì quá trình thực hiện của lịch biểu ra sao? Lịch biểu đã cho có khả tuần tự không? Nếu có, lịch tuần tự tương đương là gì?

T1	T2	T3	T4
UL(A)			
R(A)			
	WL(A) CHỜ		
RL(B)			
R(B)			
		RL(A) CHỜ	
			RL(B)
			R(B)
WL(A)			
W(A)			
UN(A)			
	WL(A)		
	W(A)		
	UL(B)		
	R(B)		
UN(B)			
			UN(B)
	WL(B)		
	W(B)		
	UN(A)		
		RL(A)	
		R(A)	
	UN(B)		
		WL(B)	
		W(B)	
		UN(A)	
		UN(B)	

Đồ thị ưu tiên:



....
Lịch tuần tự tương đương là: T1 < T4 < T2 < T3

Lịch biểu

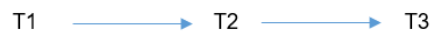
r1(B); r2(A); r3(C); r1(A); r2(C); r3(D); w1(B); w2(A); w3(C);

Giả sử bộ lập lịch hỗ trợ khóa chia sẻ (shared lock/ read lock), khóa độc quyền (exclusive lock/ write lock), cho phép tăng cấp (upgrading), nhả khóa (unlock) cuối giao tác. Cho biết quá trình thực hiện của lịch biểu? Lịch biểu có khả tuần tự không? Vì sao? Nếu có thì cho biết lịch biểu tuần tự tương đương.

=> Quá trình thực hiện lịch biểu: 1, 2, 3, 1, 2, 3, 1, 2, 3

T1	T2	T3
RL(B)		
R(B)		
	RL(A)	
	R(A)	
		RL(C)
		R(C)
RL(A)		
R(A)		
	RL(C)	
	R(C)	
		RL(D)
		R(D)
WL(B)		
W(B)		
	WL(A) CHỜ	
		WL(C) CHỜ
UN(A)		
UN(B)		
	WL(A)	
	W(A)	
	UN(A)	
	UN(C)	
		WL(C)
		W(C)
		UN(C)
		UN(D)

Đồ thị ưu tiên:



Đồ thị trên không có chu trình nên lịch biểu đã cho là khả tuần tự xung đột

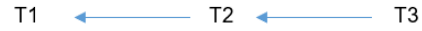
=> Lịch biểu đã cho là khả tuần tự. Đồng thời, khả tuần tự vì các giao tác nhất quán , thoả 2PL, và lịch biểu hợp lệ.

Lịch tuần tự tương đương là:
T1 < T2 < T3

Giả sử bộ lập lịch hỗ trợ khóa chia sẻ (shared lock/ read lock), khóa độc quyền (exclusive lock/ write lock), khóa cập nhật (update lock), nhả khóa (unlock) cuối giao tác. Cho biết quá trình thực hiện của lịch biểu? Lịch biểu có khả tuần tự không? Vì sao? Nếu có thì cho biết lịch biểu tuần tự tương đương.

T1	T2	T3
UL(B)		
R(B)		
	UL(A)	
	R(A)	
		UL(C)
		R(C)
RL(A) CHỜ		
	RL(C) CHỜ	
		RL(D)
		R(D)
		WL(C)
		W(C)
		UN(C)
	RL(C)	
	R(C)	
		UN(D)
	WL(A)	
	W(A)	
	UN(A)	
RL(A)		
R(A)		
	UN(C)	
WL(B)		
W(B)		
UN(A)		
UN(B)		

Đồ thị ưu tiên:



Đồ thị trên không có chu trình nên lịch biểu đã cho là khả tuần tự xung đột

=> Lịch biểu đã cho là khả tuần tự. Đồng thời, khả tuần tự vì các giao tác nhất quán , thoả 2PL, và lịch biểu hợp lệ.

Lịch tuần tự tương đương là:
T1 < T2 < T3

Lịch biểu

Lịch biểu hợp lệ gồm các giao tác nhất quán và thỏa nghi thức khóa 2 giai đoạn thì khả tuần tự xung đột và khả tuần tự.
Input: Ti
Output: S gồm Ti và S khả tuần tự.
Nhất quán, thỏa 2PL: áp dụng trên giao tác
Hợp lệ: lịch biểu
Đúng + Nhanh

	Phương thức khóa	Ưu điểm	Khuyết điểm	
Khóa đơn giản	Lock(), unlock()	Giải quyết được các vđ của truy xuất đồng thời	Không nhanh	
Khóa đọc, ghi	RL(), WL(), UN() (SL(), XL(), UN())	Giải quyết được các vđ của truy xuất đồng thời, NHANH	Làm cho các GT đọc trước ghi sau ngăn cản những thao tác đọc đồng thời của GT khác	
Khóa đọc, ghi, tăng cấp	RL(), WL(), tăng cấp, UN() (GT T đọc trước, ghi sau thì trong thời gian đọc chỉ giữ khóa đọc, khi thực sự có nhu cầu ghi thì hãy giữ khóa ghi)	Đúng, nhanh	Deadlock Wait-for graph - Đồ thị chờ 1 giao tác: 1 node Tj chờ Ti thì có cung đi từ j tới i	
Khóa đọc, ghi, update lock	RL(), WL(), UL(), UN()			

Nếu các GT không có đặc điểm đọc trước ghi sau thì chèn khóa đối với 3 nghi thức khóa sau hoàn toàn giống nhau: đọc ghi, đọc ghi tăng cấp, đọc ghi update lock

BLL được trang bị khóa đơn giản L(X), Un(X)

L(X), R(X), …UN(X)
Chỉ 1 GT được lock trên ĐVDL tại 1 thời điểm bất kỳ
Để đọc trên A: L(A): mang ý nghĩa là khóa độc quyền
L(X) cả khi đọc và ghi => T1 giữ lock thì T2 chờ

BLL được trang bị khóa đọc, ghi RL(), WL(), UN()

RL(X); R(X), … UN(X)
T1: …;R(A), …; W(A)
T1: …WL(A) ;R(A), …; W(A); …; UN(A)

Đọc trước ghi sau: WL(A); R(A), …; W(A)
RL = SL = SLOCK WL = XL = XLOCK

Để đọc trên A:
RL(A) để đọc trên A => T1 giữ lock thì T2 vẫn RL được
WL(A) để ghi trên A => T1 giữ lock thì T2 chờ
Dùng WL(A) luôn để đọc trên A đối với GT đọc trước ghi sau

BLL được trang bị khóa đọc, ghi tăng cấp RL(), WL(), UN()

T1: …;R(A), …; W(A)
T1: …RL(A) ;R(A), …WL(A) ; W(A)
Để đọc trên A:
RL(A) để đọc trên A => T1 giữ lock thì T2 vẫn RL được
WL(A): để ghi trên A => T1 giữ lock thì T2 chờ

BLL được trang bị khóa đọc, ghi, update lock UL(), RL(), WL(), UN()

T1: …;R(A), …; W(A)
T1: …UL(A);R(A), …WL(A); W(A), UN(A)
Để đọc trên A:
RL(A) để đọc trên A => T1 giữ lock thì T2 chờ
UL(A) để ghi trên A => T1 giữ lock thì T2 chờ
UL(A) để đọc trên A đối với GT đọc trước ghi sau

Lịch biểu

T1	T2	T3
r1(A)	r2(B)	r3(C)
r1(B)	r2(C)	r3(D)
w1(A)	w2(B)	w3(C)

BLL được trang bị khóa đơn giản

T1	T2	T3
L(A)	L(B)	L(C)
R(A)	R(B)	R(C)
L(B)	L(C)	L(D)
R(B)	R(C)	R(D)
W(A)	W(B)	W(C)
UN(A)	UN(B)	UN(C)
UN(B)	UN(C)	UN(D)

BLL được trang bị khóa đọc ghi

T1	T2	T3
WL(A)	WL(B)	WL(C)
R(A)	R(B)	R(C)
RL(B)	RL(C)	RL(D)
R(B)	R(C)	R(D)
W(A)	W(B)	W(C)
UN(A)	UN(B)	UN(C)
UN(B)	UN(C)	UN(D)

BLL được trang bị khóa đọc ghi tăng cấp

T1	T2	T3
RL(A)	RL(B)	RL(C)
R(A)	R(B)	R(C)
RL(B)	RL(C)	RL(D)
R(B)	R(C)	R(D)
WL(A)	WL(B)	WL(C)
W(A)	W(B)	W(C)
UN(A)	UN(B)	UN(C)
UN(B)	UN(C)	UN(D)

BLL được trang bị khóa đọc ghi update lock

T1	T2	T3
UL(A)	UL(B)	UL(C)
R(A)	R(B)	R(C)
RL(B)	RL(C)	RL(D)
R(B)	R(C)	R(D)
WL(A)	WL(B)	WL(C)
W(A)	W(B)	W(C)
UN(A)	UN(B)	UN(C)
UN(B)	UN(C)	UN(D)

Lịch biểu

BLL được trang bị khóa đơn giản

T1	T2	T3
r1(A)	r2(B)	r3(C)
r1(B)	r2(C)	r3(D)
w1(A)	w2(B)	w3(C)
l1(A)		
r1(A)		
	l(B)	
	r2(B)	
		l(C)
		r3(C)
l(B) Chờ		
	l(C) Chờ	
		l(D)
		r3(D)
		w3(C)
		un(C)
	l(C)	
	r2(C)	
		un(D)
	w2(B)	
	un(B)	
l(B)		
r1(B)		
	un(C)	
w1(A)		
un(A)		
un(B)		

BLL được trang bị khóa đọc ghi

T1	T2	T3
r1(A)	r2(B)	r3(C)
r1(B)	r2(C)	r3(D)
w1(A)	w2(B)	w3(C)
wl(A)		
r1(A)		
	wl(B)	
	r2(B)	
		wl(C)
		r3(C)
rl(B) Chờ		
	rl(C) Chờ	
		rl(D)
		r3(D)
		w3(C)
		un(C)
	rl(C)	
	r2(C)	
		un(D)
	w2(B)	
	un(B)	
rl(B)		
r1(B)		
	un(C)	
w1(A)		
un(A)		
un(B)		

BLL được trang bị khóa đọc ghi tăng cấp

T1	T2	T3
r1(A)	r2(B)	r3(C)
r1(B)	r2(C)	r3(D)
w1(A)	w2(B)	w3(C)
rl(A)		
r1(A)		
	rl(B)	
	r2(B)	
		rl(C)
		r3(C)
rl(B)		
r1(B)		
	rl(C)	
	r2(C)	
		rl(D)
		r(D)
wl(A)		
w1(A)		
	wl(B) Chờ	
		wl(C) Chờ
un(A)		
un(B)		
	wl(B)	
	w2(B)	
	un(B)	
	un(C)	
		wl(C)
		w2(C)
		un(C)
		un(D)

BLL được trang bị khóa đọc ghi update lock

T1	T2	T3
r1(A)	r2(B)	r3(C)
r1(B)	r2(C)	r3(D)
w1(A)	w2(B)	w3(C)
UL(A)		
r1(A)		
	UL(B)	
	r2(B)	
		UL(C)
		r3(C)
rl(B) Chờ		
	rl(C) Chờ	
		rl(D)
		r(D)
		WL(C)
		w2(C)
		un(C)
	RL(C)	
	R(C)	
		un(D)
	WL(B)	
	W(B)	
	UN(B)	
RL(B)		
R(B)		
	UN(C)	
WL(A)		
W(A)		
UN(A)		
UN(B)		

Khả tuần tự vì các GT nhất quán, thỏa 2PL và lịch biểu hợp lệ. T3 < T2 < T1

Mức cô lập

Các mức độ cô lập (Isolation levels)

- Mức độ cô lập của 1 giao tác quy định mức độ nhạy cảm của 1 giao tác đối với những sự thay đổi trên CSDL do các giao tác khác tạo ra.
- Mức độ cô lập của giao tác quy định cách thức lock và thời gian giữ lock trên đơn vị dữ liệu mà giao tác có truy cập.
- Các mức độ cô lập được SQL Server hỗ trợ:
  - Read Uncommitted.
  - Read Committed (default).
  - Repeatable Read.
  - Serializable.

- *SET TRANSACTION ISOLATION LEVEL <READ UNCOMMITTED|READ COMMITTED|REPEATABLE READ|SERIALIZABLE>*

Read Uncommitted

Không thiết lập Shared Lock trên những đơn vị dữ liệu cần đọc, thiết lập Exclusive lock khi ghi.

Không bị ảnh hưởng bởi những lock của các giao tác khác trên những đơn vị dữ liệu cần đọc.

Không phải chờ khi đọc dữ liệu (kể cả khi dữ liệu đang bị lock bởi giao tác khác).

Ưu điểm:

Tốc độ xử lý rất nhanh.

Không cản trở những giao tác khác thực hiện ddoc dữ liệu.

Khuyết điểm:

Các vấn đề gặp phải khi xử lý đồng thời: Dirty Reads, Unrepeatable Read, Phantoms.

Nhận xét:

Chỉ nên dùng để đọc dữ liệu trong trường hợp cần dữ liệu tổng quan về CSDL, ví dụ như tạo những báo cáo về tình hình chung.

Không dùng khi cần đọc những số liệu chính xác.

Read Committed

Là MCL mặc định của SQL Server

Tạo Shared Lock trên đvdl được đọc, Shared Lock được giải phóng ngay sau khi đọc xong dữ liệu => Giải quyết vấn đề Dirty Reads

Tạo Exclusive Lock trên đvdl được ghi, và giữ cho đến hết giao tác

Ưu điểm:

Giải quyết vấn đề Lost update, Dirty Reads

Shared Lock được giải phóng ngay, không cần phải giữ cho đến hết giao tác nên không ngăn cản thao tác cập nhật của các giao tác khác.

Khuyết điểm:

Chưa giải quyết được vấn đề Unrepeatable Reads, Phantoms

Phải chờ khi chưa thể được đáp ứng yêu cầu lock trên đơn vị dữ liệu đang bị giữ lock bởi giao tác khác.

Mức cô lập

Read Uncommitted (MCL1)	Read committed (MCL2)	Repeatable read (MCL3)	Serializable (MCL4)
1. Khóa ghi? 2. Khóa đọc? 3. Còn xảy ra tình trạng gì của ĐKĐT?	4. Khóa ghi? 5. Khóa đọc? Thời gian khóa đọc? 6. Còn xảy ra tình trạng gì của XLĐT?	7. Khóa ghi? 8. Khóa đọc? Thời gian khóa đọc? 9. Còn xảy ra tình trạng gì của XLĐT?	10. Khóa ghi? 11. Khóa đọc? Thời gian khóa đọc? 12. Còn xảy ra tình trạng gì của XLĐT?
3.1 Lost Update (TH1)	6.1 Lost Update (TH1)	9.1 Lost Update (TH1)	12.1 Lost Update (TH1)
3.2 Dirty read	6.2 Dirty read	9.2 Dirty read	12.2 Dirty read
3.3 Unrepeatable read	6.3 Unrepeatable read	9.3 Unrepeatable read	12.3 Unrepeatable read
3.4 Phantom	6.4 Phantom	9.4 Phantom	12.4 Phantom

Repeatable Read

Repeatable Read = Read Committed+ Giải quyết Unrepeatable Reads

Tạo Shared Lock trên đvdl được đọc, Shared Lock được giữ cho đến hết giao tác => Không cho phép các giao tác khác cập nhật trên đvdl này.

Tạo Exclusive Lock trên đvdl được ghi, Exclusive Lock được giữ cho đến hết giao tác.

Ưu điểm:

Giải quyết vấn đề lost update, dirty Read và Unrepeatable Read

Khuyết điểm:

Chưa giải quyết được vấn đề Phantom.

Phải chờ khi chưa thể được đáp ứng yêu cầu lock trên đơn vị dữ liệu đang bị giữ lock bởi giao tác khác.

Các giao tác khác không được phép cập nhật trên những đơn vị dữ liệu đang bị giữ Shared Lock.

Vẫn cho phép Insert những dòng dữ liệu thỏa mãn điều kiện thiết lập những Shared Lock => Phantoms

Serializable

Serializable = Repeatable Read + Giải quyết Phantom

Tạo Shared Lock trên đvdl được đọc, Shared Lock được giữ cho đến hết giao tác => Không cho phép các giao tác khác cập nhậtrên đvdl này.

Không cho phép Insert những dòng dữ liệu thỏa mãn điều kiện thiết lập những Shared Lock.

Tạo Exclusive Lock trên đvdl được ghi, Exclusive Lock được giữ cho đến hết giao tác.

Ưu điểm:

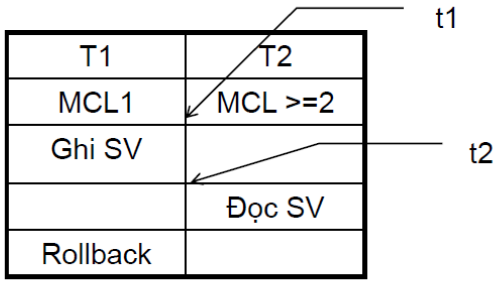
Giải quyết được cả 4 vấn đề của TXĐT.

Khuyết điểm:

Phải chờ khi chưa thể được đáp ứng yêu cầu lock trên đvdl đang bị giữ lock bởi giao tác khác.

Mức cô lập

TH1: MCL1 có khóa ghi khi ghi?



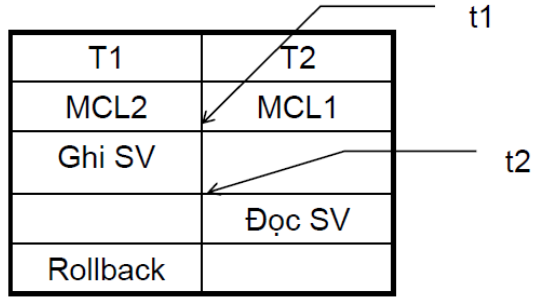
Giả sử ta biết rằng T2 có mức cô lập >=2 thì sẽ khóa đọc khi đọc.

Nếu T2 đọc ra dữ liệu tại thời điểm t2 có nghĩa là T1 không khóa ghi trên SV nên T2 khóa đọc và đọc dữ liệu được ngay.

Nếu T2 đọc ra dữ liệu tại thời điểm t1 có nghĩa là T1 có khóa ghi trên SV và T2 chờ cho T1 hoàn tất, nhả khóa ghi xong rồi mới đọc, nghĩa là T1 có khóa trên bảng SV khi ghi.

→ MCL1 khắc phục được Lost Update TH1.

TH2: MCL1 có khóa đọc khi đọc?



Giả sử ta biết rằng T2 có mức cô lập =2 thì sẽ khóa ghi khi ghi.

Nếu T2 đọc ra dữ liệu tại thời điểm t2 có nghĩa là T2 không khóa đọc trên SV nên đọc dữ liệu được ngay và đọc ra dữ liệu sai.

Nếu T2 đọc ra dữ liệu tại thời điểm t1 có nghĩa là T2 có khóa đọc trên SV và T2 chờ cho T1 hoàn tất, nhả khóa ghi xong rồi mới đọc.

→ MCL1 vẫn còn gặp phải tình trạng Dirty read.

Phục hồi dữ liệu

- Bắt đầu giao tác: Start T1
- Wrtie T1, A, Image before, Image after
- Commit T1
- Checkpoint: đã ghi nhận vật lý

⇒ **Đọc log file từ dưới lên**

- Giao tác **chưa được commit**, giá trị **đã được ghi vật lý** trước checkpoint thì thực hiện **Undo vật lý** ⇧
- Giao tác **chưa được commit**, thao tác **thực hiện sau checkpoint** thì thực hiện **Undo buffer** [⇧]
- Giao tác **đã được commit trước sự cố** thì phải ghi nhận hoàn thành bằng **Redo** ✕

<START T1>		E = 19
<T1,E,6,5>		A = 3
<T1,A,15,12>		D
<CHECKPOINT>		M = 3
<START T2>		Q = 7
<T1,E,5,19>	x	
<COMMIT T1>		
<T2,A,12,3>	x	
<START T3>		
<T3,D,7,6>	[↑]	
<T3,M,3,7>	[↑]	
<T2,D,6,5>	x	
<COMMIT T2>		
<T3,Q,7,4>	[↑]	
<T3,E,19,8>	[↑]	

<start T1>		M = 11
Write <T1, M, 4, 5>		N = 15
<start T3>		O = 6
Write <T3, O, 6, 7>	[↑]	P = 15
<Checkpoint>		Q = 19
Write <T1, N, 9, 10>	x	
<commit T1>		
<start T2>		
Write <T2, M, 5, 11>	x	
Write <T2, N, 10, 15>	x	
Write <T3, O, 7, 20>	[↑]	
Write <T2, P, 14, 15>	x	
<commit T2>		
Write <T3, Q, 19, 20>	[↑]	

<Check point>		M = 11
[Begin tran, T1]		N = 15
[Read, T1, A]		O = 6
[Write, T1, A, 30, 40]	[↑]	P = 15
[Read, T1, B]		Q = 19
[Write, T1, B, 20, 10]	x	
[Begin tran, T2]		
[Read, T2, C]		
[Write, T2, C, 10, 15]	x	
[Begin tran, T3]	x	
[Read, T3, D]	[↑]	
[Write, T3, D, 10, 20]	x	
[Commit, T3]		
[Read, T2, C]	[↑]	
[Write, T2, C, 15, 40]		
[Read, T2, D]		
[Write, T2, D, 20, 40]		

Phục hồi dữ liệu

[Begin-Transaction, T1]		A = 5
[Write,T1,A,5,10]	↑	B = 20 / 50
[check_point]		D = 20
[Begin-Transaction, T2]		
[Write,T2,B,20,40]	[↑]	
[Begin_transaction,T3]		
[Read,T3,B]	x	
[Write,T3,B,40,50]	x	
[Begin_Transaction,T4]		
[Write,T4,D,10,20]	x	
[Commit,T3]		
[Read,T1,A]	[↑]	
[Commit,T4]		
<hết tập nhật ký>		

<start T2>		M = 6
<start T1>		N = 10
<T2, M, 6, 11>		P = 15
<T1, M, 11, 6>		Q = 19
<commit T1>		O = 6
<start T3>		
<T2, N, 9, 10>		
<T3, O, 6, 7>	↑	
<check point>		
<T2, P, 14, 15>	x	
<T3, Q, 19, 20>	[↑]	
<commit T2>		
<hết tập nhật ký>		

<start T1>		X = 28
<T1,X,14,28>		Y = 5
<T1,Y,15,5>		Z = 10
<Start T2>		W = 7
<T2,Z,20,10>		
<Commit T1>		
<Start ckpt (T2)>		
<T2,W,4,7>	x	
<Start T3>		
<end ckpt>		
<T3,X,28,17>	[↑]	
<commit T2>		

Điều khiển đồng thời dựa trên kỹ thuật xác nhận hợp lệ

- Bộ lập lịch cứ cho giao tác thực hiện.
- Tại 1 thời điểm sẽ xát xem có tranh chấp gì không.
- 1 giao tác có 3 đường tạo thành hình tam giác

I - Bắt đầu giao tác (Read)

X - Tại đường cao là thời điểm kiểm tra hợp lệ (V)

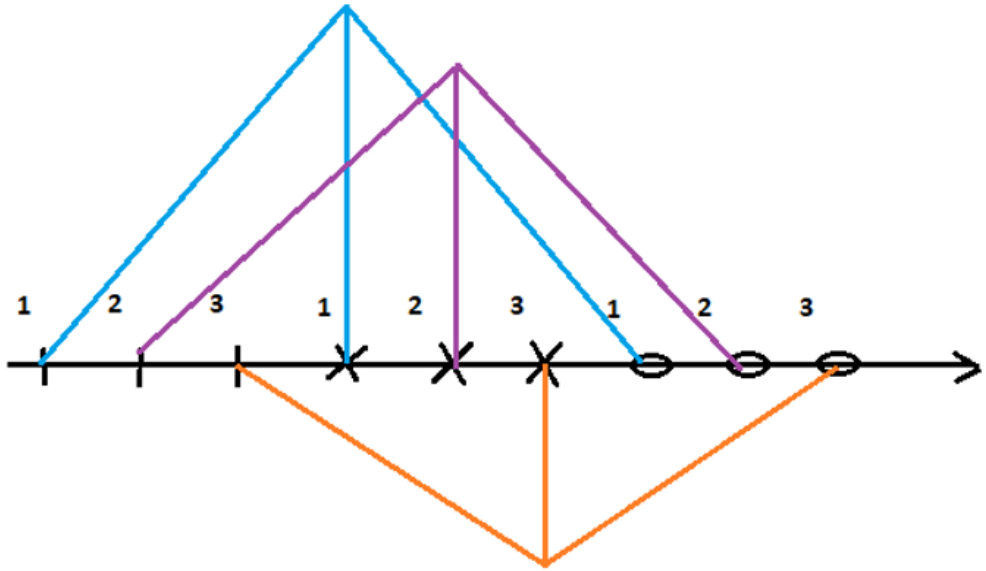
O - Kết thúc giao tác (Write)

Từ I tới X là giai đoạn đọc

Từ X tới O là giai đoạn ghi

- **Thứ tự kiểm tra** giao tác là **thứ tự xuất hiện V (dấu X)** trong lịch biểu từ trái sang phải.
- Giao tác sau **kiểm tra dựa trên** các **giao tác đã hợp lệ** trước đó.
- **Giao tác đầu tiên kiểm tra thì hiển nhiên hợp lệ.**
- **Đọc + Đọc => không cần kiểm tra**
- **Đọc + Ghi, Ghi + Ghi => cần kiểm tra tranh chấp**

R1(A, B); R2(B, C); R3(C); V1; V2; V3; W1(A); W2(C); W3(B)



*T1 hợp lệ vì trước đó chưa có giao tác hợp lệ

*Kiểm tra T2:

$RS(T2) \cap WS(T1) = \{B,C\} \cap \{A\} = \Phi$

$WS(T2) \cap WS(T1) = \{C\} \cap \{A\} = \Phi$

=> T2 hợp lệ

*Kiểm tra T3:

$RS(T3) \cap WS(T1) = \{C\} \cap \{A\} = \Phi$

$RS(T3) \cap WS(T2) = \{C\} \cap \{C\} = \{C\}$

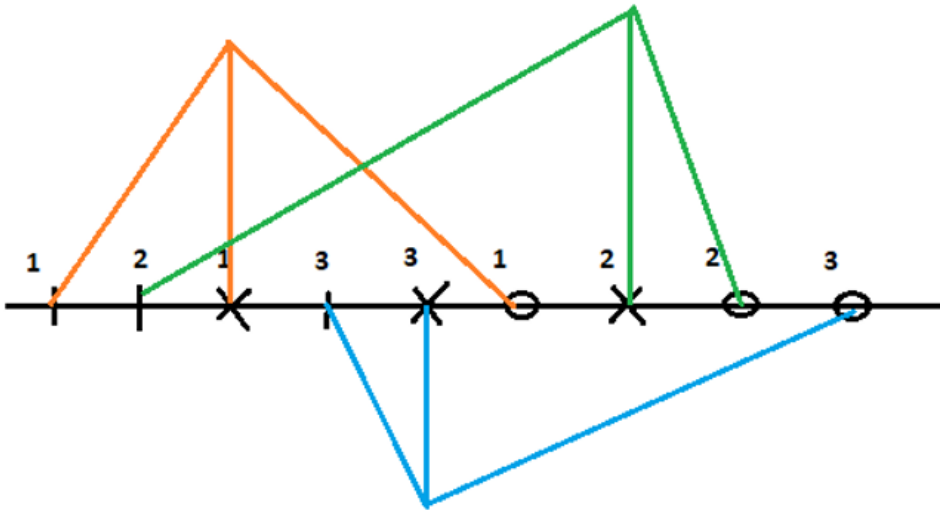
$WS(T3) \cap WS(T1) = \{B\} \cap \{A\} = \Phi$

$WS(T3) \cap WS(T2) = \{B\} \cap \{C\} = \Phi$

=> T3 không hợp lệ, rollback

Điều khiển đồng thời dựa trên kỹ thuật xác nhận hợp lệ

R1(A,B); R2(B,C); V1; R3(C); V3; W1(C); V2;W2(B); W3(A);



*T1 hợp lệ vì trước đó chưa có giao tác hợp lệ

*Kiểm tra T3:

RS(T3) ∩ WS(T1)= {C} ∩ {C} = {C}

WS(T3) ∩ WS(T1)={A} ∩ {C} = Φ

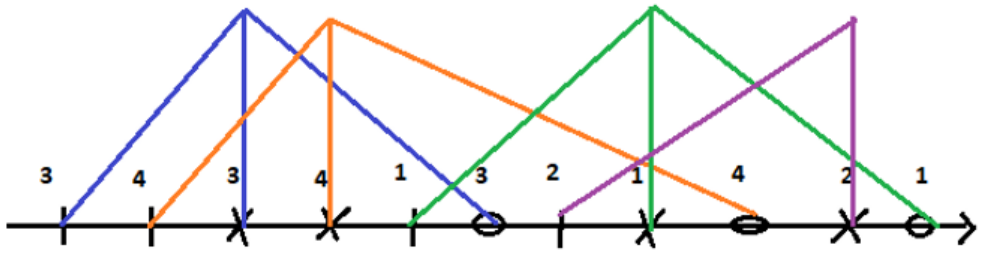
=> T3 không hợp lệ, rollback

*Kiểm tra T2:

RS(T2) ∩ WS(T1) = {B,C} ∩ {C} = C

=> T2 không hợp lệ, rollback

R3(B); R4(A,B); V3; V4; R1(B); W3(D); R2(A, D); V1; W4(A, C); V2; W1(D, E);



*T3 hợp lệ vì trước đó chưa có giao tác hợp lệ

*Kiểm tra T4:

RS(T4) ∩ WS(T3) = {A, B} ∩ {D} = Φ

WS(T4) ∩ WS(T3) = {A, C} ∩ {D} = Φ

=> T4 hợp lệ

*Kiểm tra T1:

RS(T1) ∩ WS(T3)= {B} ∩ {D} = Φ

RS(T1) ∩ WS(T4)= {B} ∩ {A, C} = Φ

WS(T1) ∩ WS(T4)= {D, E} ∩ {A, C} = Φ

⇒ T1 hợp lệ

*Kiểm tra T2:

RS(T2) ∩ WS(T1) = {A, D} ∩ {D, E} = {D}

RS(T2) ∩ WS(T4)= {A, D} ∩ {A, C} = {A}

=> T2 không hợp lệ, rollback

Điều khiển đồng thời dựa trên nhãn thời gian

- TS(X): nhãn thời gian của giao tác – thời gian bắt đầu giao tác
- RT(X): nhãn thời gian đọc của giao tác – là nhãn thời gian lớn nhất của giao tác đã đọc trên X
- WT(X): nhãn thời gian ghi của giao tác – là nhãn thời gian lớn nhất của giao tác đã ghi trên X
- **T muốn đọc X thì: TS(T) >= WT(X)**
- **T muốn ghi X thì: TS(T) >= WT(X) và TS(T) >= RT(X)**

*** Thuật toán điều khiển từng phần (riêng phần / 1 phiên bản):** giao tác có mặt trước làm trước, có mặt trước mà làm sau thì rollback

- **Khi T yêu cầu Read(X):**

  - Nếu TS(T) >= WT(X) thì cho đọc và cập nhật RT(X) = TS(T), WT(X) giữ nguyên

  - Nếu TS(T) < WT(X) thì T rollback và không khởi động lại

- **Khi T yêu cầu Write(X):**

  - Nếu TS(T) >= WT(X) và TS(T) >= RT(X) thì cho đọc và cập nhật W(X) = TS(T), RT(X) giữ nguyên

  - Nếu TS(T) < WT(X) hoặc/và TS(T) < RT(X) thì T rollback và không khởi động lại

*** Thuật toán điều khiển nhãn thời gian đa phiên bản:** thao tác ghi có thể tạo ra nhiều phiên bản khác nhau của đơn vị dữ liệu

- **Khi T yêu cầu Read(X):**

  - Tìm tất cả các phiên bản của X có TS(T) >= WT(X) => sau đó lấy phiên bản có WT(X) lớn nhất

  - Cập nhật RT(X) cho phiên bản đó: So TS(T) với RT(X) của phiên bản đó, cái nào lớn hơn thì lấy gán cho RT(X)

  - Giữ nguyên WT(X) cho phiên bản đó

- **Khi T yêu cầu Write(X):**

  - Tìm tất cả các phiên bản của X có TS(T) >= WT(X) => sau đó lấy phiên bản có WT(X) lớn nhất

  - Nếu RT(X) của phiên bản đó > TS(T) thì cho T rollback và không khởi động lại

  - Nếu RT(X) của phiên bản đó <= TS(T) thì:

    + Nếu TS(T) = WT(X): ghi đè lên phiên bản đó

    + Nếu TS(T) > WT(X): tạo phiên bản mới với RT(X) = WT(X) = TS(T)

Điều khiển đồng thời dựa trên nhãn thời gian

Lịch biểu: r1(A); w1(A); r2(A); w2(A);r3(A); r4(A)

Trong đó thời điểm giao tác T1, T2, T3, T4 bắt đầu là 150, 200,175 225.

Nhãn thời gian từng phần (riêng phần)

T1	T2	T3	T4	A
150	200	175	225	RT=WT=0
R1(A)				RT = 150
W1(A)				WT = 150
	R2(A)			RT = 200
	W2(A)			WT = 200
		R3(A)		T3 roll back
			R4(A)	RT = 225

Nhãn thời gian đa phiên bản

T1	T2	T3	T4	A0	A1	A2
150	200	175	225	RT=WT=0		
R1(A)				RT = 150 WT =0		
W1(A)					RT = WT =150	
	R2(A)				RT = 200 WT = 150	
	W2(A)					RT = WT =200
		R3(A)			RT = 200 WT = 150	
			R4(A)			RT = 225 WT =200

Nhận xét về tính năng đặc trưng của hai kỹ thuật nhãn thời gian từng phần và nhãn thời gian đa phiên bản

- Phương pháp nhãn thời gian từng phần: sử dụng ít tài nguyên do chỉ lưu phiên bản có nhãn đọc / ghi lớn nhất của đơn vị dữ liệu, xử lý nhanh hơn. Tuy nhiên phương pháp này dễ khiến các giao tác rollback.
- Phương pháp nhãn thời gian đa phiên bản: ngoài phiên bản mới nhất của đơn vị dữ liệu được lưu lại, còn có các phiên bản trước đó, giúp cho thao tác đọc không bao giờ làm giao tác rollback, giao tác T thay vì abort vì lí do trên sẽ tiếp tục đọc phiên bản (của đơn vị dữ liệu cần đọc) phù hợp với nhãn thời gian của T. Tuy nhiên phương pháp này sẽ tốn kém chi phí do phải lưu trữ nhiều phiên bản của đơn vị dữ liệu

Điều khiển đồng thời dựa trên nhãn thời gian

xl1(A); r1(A); xl2(A); w2(A); rl1(B); r1(B); xl2(B); r2(B); rl3(A); r3(A); rl4(B); r4(B); wl3(B); w3(B); w1(A); un1(A); un1(B); w2(B); un2(A); un2(B); un3(A); un3(B); un4(B)

Nhãn thời gian như sau: ts(T1) =100; ts(T2) =200; ts(T3) =300; ts(T4) =400

Nhãn thời gian từng phần

T1	T2	T3	T4	A	B
100	200	300	400	RT=WT =0	RT=WT =0
R(A)				RT=100 WT=0	
	W(A)			RT=100 WT=200	
R(B)					RT=100 WT=0
	R(B)				RT=200 WT=0
		R(A)		RT=300 WT=200	
			R(B)		RT=400 WT=0
		W(B)			T3 Rollback
W(A)				T1 Rollback	
	W(B)				T2 Rollback

Nhãn thời gian đa phiên bản

T1	T2	T3	T4	A0	A1	A2	B0
100	200	300	400	RT=WT=0			RT=WT=0
R(A)				RT=100 WT=0			
	W(A)				RT=200 WT=200		
R(B)							RT=100 WT=0
	R(B)						RT=200 WT=0
		R(A)			RT=300 WT=200		
			R(B)				RT=400 WT=0
		W(B)					T3 Rollback
W(A)						RT=100 WT=100	
	W(B)						T2 Rollback

Điều khiển đồng thời dựa trên nhãn thời gian

w1(A); w3(A); r4(A); r2(A);w4(B); r1(B)

Trong đó thời điểm giao tác T1, T2, T3, T4 bắt đầu là 100, 200, 300, 400.

Nhãn thời gian từng phần (riêng phần)

T1	T2	T3	T4	A	B
100	200	300	400	RT=WT=0	RT=WT=0
W1(A)				RT=0 WT=100	
		W3(A)		RT=0 WT=300	
			R4(A)	RT=400 WT=300	
	R2(A)			T2 Rollback	
			W4(B)		RT=0 WT=400
R1(B)					T1 Rollback

Nhãn thời gian đa phiên bản

T1	T2	T3	T4	A0	A1	A2	B0	B1
100	200	300	400	RT=WT=0			RT=WT=0	
W1(A)					RT=100 WT=100			
		W3(A)				RT=300 WT=300		
			R4(A)			RT=400 WT=300		
	R2(A)				RT=200 WT=100			
			W4(B)					RT=400 WT=400
R1(B)							RT=100 WT=0	

Điều khiển đồng thời dựa trên nhãn thời gian

r4(A); r1(A); w4(B); w1(A); r2(B); r3(B); r2(A); w2(C); r3(A)

Trong đó thời điểm giao tác T1, T2, T3, T4 bắt đầu là 420, 400, 425, 415.

Nhãn thời gian từng phần (riêng phần)

T1	T2	T3	T4	A	B	C
420	400	425	415	RT = WT= 0	RT = WT= 0	RT = WT= 0
			Read(A)	RT =415 WT = 0		
Read(A)				RT = 420 WT = 0		
			Write(B)		RT = 0 WT = 415	
Write(A)				RT = 420 WT = 420		
	Read(B)				T2 ROLLBACK	
		Read(B)			RT = 425 WT = 415	
	Read(A)					
	Write(C)					
		Write(A)		RT = 420 WT =425		

Nhãn thời gian đa phiên bản

T1	T2	T3	T4	A0	A1	A2	B0	B1	C0	C1
420	400	425	415	RT = WT= 0			RT = WT= 0		RT = WT= 0	
			Read(A)	RT= 415 WT= 0						
Read(A)				RT = 420 WT= 0						
			Write(B)					RT=WT= 415		
Write(A)					RT=WT = 420					
	Read(B)						RT= 400 WT= 0			
		Read(B)						RT=425 WT= 415		
	Read(A)			RT = 420 WT= 0						
	Write(C)									RT = WT = 400
		Write(A)				RT=WT = 425				

Điều khiển đồng thời dựa trên nhãn thời gian

r1(B), r2(A), r3(C), w1(B), w1(A), w2(C), w3(A)

TS(T1) = 200, TS(T2) = 150, TS(T3) = 175

Nhãn thời gian từng phần (riêng phần)

T1	T2	T3	A	B	C
200	150	175	RT = WT= 0	RT = WT= 0	RT = WT= 0
Read(B)				RT = 200 WT= 0	
	Read(A)		RT = 150 WT= 0		
		Read(C)			RT = 175 WT= 0
Write(B)				RT = 200 WT= 200	
Write(A)			RT = 150 WT= 200		
	Write(C)				T2 Roll back
		Write(A)	T3 Roll back		

Nhãn thời gian đa phiên bản

T1	T2	T3	A0	A1	A2	B0	B1	C0	C1
200	150	175	RT = WT= 0			RT = WT= 0		RT = WT= 0	
Read(B)						RT = 200 WT= 0			
	Read(A)		RT = 150 WT= 0						
		Read(C)						RT = 175 WT= 0	
Write(B)							RT = 200 WT= 200		
Write(A)				RT = 200 WT= 200					
	Write(C)							T2 Roll back	
		Write(A)			RT = 175 WT= 175				

Deadlock

- Deadlock xảy ra khi các giao tác có liên quan đến nhau không thể thực hiện tiếp các thao tác của nó mà phải chờ lẫn nhau mãi.
- **Xác định lịch biểu có gây ra deadlock không dựa vào Đồ thị chờ:**
  - Đồ thị có đỉnh là các giao tác
  - Xét 2 giao tác cùng thực thi trên 1 đơn vị dữ liệu: Nếu Tj diễn ra trước, Ti phải chờ Tj thì ta có cung Tj ->Ti
  - Sau khi vẽ xong, nếu đồ thị chờ có chu trình => xảy ra deadlock
  - Có thể giải quyết deadlock (khi đã diễn ra) bằng cách huỷ đỉnh ứng với giao ác có nhiều cung vào ra nhất (cho rollback)
- Để ngăn ngừa deadlock có thể sử dụng thuật toán Wait-die hoặc Wound-wait

PP dùng nhãn thời gian		ĐT chờ
Wait - Die	Wound-Wait	
GT rollback ở thời điểm xảy yêu cầu lock, là giai đoạn sớm, nên có thể có nhiều GT bị rollback hơn, và GT rollback thường thực hiện ít công việc hơn.	Nếu GT yêu cầu lock gần thời điểm GT bắt đầu, ít khi xảy ra tình trạng GT già không yêu cầu được lock giữ bởi GT trẻ hơn, vì vậy rollback ít xảy ra. GT bị rollback đã thực hiện nhiều việc trước khi bị rollback.	Đồ thị có thể rất lớn, phân tích và tìm chu trình sẽ tốn nhiều thời gian.
Có khi không có deadlock xảy ra vẫn yêu cầu GT rollback.		Chỉ yêu cầu 1 GT rollback khi thực sự GT gây ra deadlock
Ưu tiên GT "già", GT "già" hơn sẽ kill giao tác "trẻ" hơn. Đảm bảo mọi GT đều hoàn tất, không có tình trạng starvation.		
Dễ cài đặt hơn đồ thị chờ		Cài đặt khó, đặc biệt đối với hệ thống phân tán.

Ngăn ngừa Deadlock

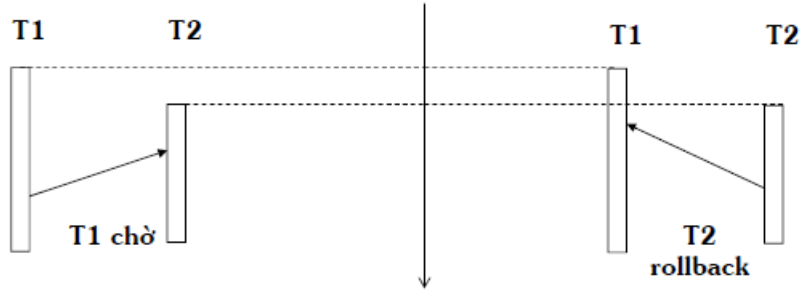
*** Thuật toán Wait-die:** ưu tiên các giao tác già hơn (bắt đầu trước, timestamp nhỏ hơn)

- Giao tác thực hiện trước tranh chấp lock với giao tác được thực hiện sau => chờ
- Giao tác thực hiện sau tranh chấp lock với giao tác được thực hiện trước => bị rollback và thực hiện lại

T1 có timestamp là tT1, T2 có timestamp là tT2

T1 yêu cầu lock trên 1 đơn vị dữ liệu đang bị giữ lock bởi T2 (tranh chấp):

- Nếu nhãn thời gian của T1 < T2 (tT1 < tT2, T1 già hơn) thì T1 chờ
- Nếu nhãn thời gian của T1 > T2 thì T1 rollback



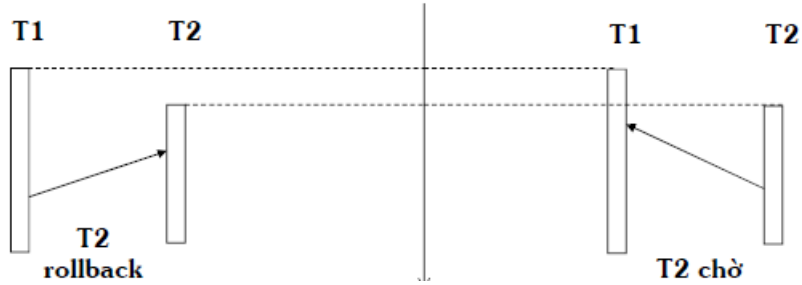
*** Thuật toán Wound-wait:** ưu tiên các giao tác già hơn (bắt đầu trước, timestamp nhỏ hơn)

- Giao tác thực hiện trước không bao giờ bị rollback hay chờ khi tranh chấp lock với một giao tác thực hiện sau đang giữ lock.
- Giao tác thực hiện sau tranh chấp lock với giao tác được thực hiện trước => phải chờ, hoặc bị rollback và thực hiện lại

T1 có timestamp là tT1, T2 có timestamp là tT2

T1 yêu cầu lock trên 1 đơn vị dữ liệu đang bị giữ lock bởi T2 (tranh chấp):

- Nếu nhãn thời gian của T1 < T2 (tT1 < tT2, T1 già hơn) thì T2 rollback
- Nếu nhãn thời gian của T1 > T2 thì T1 chờ



B1: Cho lịch biểu: r1(A); r2(B); r3(C); w1(B); w2(C); w3(A). Biết rằng BLL được trang bị khóa đọc, khóa ghi, nhả khóa cuối giao tác.

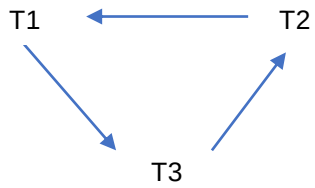
Dùng đồ thị chờ cho biết lịch biểu này có xảy ra deadlock không? Nếu có hãy đưa ra giải pháp để giải quyết deadlock. Hãy chứng tỏ rằng deadlock đã được giải quyết, nếu có thể.

* Thứ tự thực hiện lịch biểu: 1, 2, 3, 1, 2, 3

T1	T2	T3
RL(A)		
R(A)		
	RL(B)	
	R(B)	
		RL(C)
		R(C)
WL(B) chờ		
	WL(C) chờ	
		WL(A) chờ

* Quy ước: Nếu Tj chờ Ti thì có cung từ Ti -> Tj

* Đồ thị chờ: có chu trình nên xảy ra deadlock



* Để giải quyết deadlock, ta hủy đỉnh (ứng với giao tác có nhiều cung ra vào nhất). Ở đây các đỉnh đều như nhau nên chọn đỉnh bất kỳ, ta chọn đỉnh T3 để rollback.

T1	T2	T3
RL(A)		
R(A)		
	RL(B)	
	R(B)	
		RL(C)
		R(C)
ROLLBACK T3		
WL(B) CHỜ		
	WL(C)	
	W(C)	
WL(B) CHỜ		
	UN(B)	
	UN(C)	
WL(B)		
W(B)		
UN(A)		
UN(B)		
		RL(C)
		R(C)
		WL(A)
		W(A)
		UN(C)
		UN(A)

Hãy đưa ra 03 (ba) phương pháp cụ thể để tránh deadlock. Giả sử nhãn thời gian của các giao tác: ts(T1) < ts(T2) < ts(T3)

* Phương pháp 1: Sắp xếp các đơn vị dữ liệu theo 1 thứ tự cố định (thứ tự alphabet) và các giao tác yêu cầu lock trên chúng theo thứ tự này: r1(A); r2(B); r3(C); w1(B); w2(C); w3(A) => r1(A); r2(B); w3(A); w1(B); w2(C); r3(C)

T1	T2	T3
RL(A)		
R(A)		
	RL(B)	
	R(B)	
		WL(A) CHỜ
WL(B) CHỜ		
	WL(C)	
	W(C)	
		WL(A) CHỜ
WL(B) CHỜ		
	UN(B)	
	UN(C)	
		WL(A) CHỜ
WL(B)		
W(B)		
		WL(A) CHỜ
UN(A)		
		WL(A)
		W(A)
UN(B)		
		RL(C)
		R(C)
		UN(A)
		UN(C)

*** Phương pháp 2: Dùng nhãn thời gian Wait-Die**

T1	T2	T3
RL(A)		
R(A)		
	RL(B)	
	R(B)	
		RL(C)
		R(C)
WL(B) CHỜ (do nhãn tg T1 < T2)		
	WL(C) CHỜ (do nhãn tg T2 < T3)	
		WL(A) Rollback (do nhãn tg T3 > T1)
WL(B) CHỜ		
	WL(C)	
	W(C)	
WL(B) CHỜ		
	UN(B)	
	UN(C)	
WL(B)		
W(B)		
UN(A)		
UN(B)		
		RL(C)
		R(C)
		WL(A)
		W(A)
		UN(A)
		UN(C)

*** Phương pháp 3: Dùng nhãn thời gian Wound-wait**

T1	T2	T3
RL(A)		
R(A)		
	RL(B)	
	R(B)	
		RL(C)
		R(C)
WL(B)	Rollback (do nhãn tg T1 < T2)	
W(B)		
		WL(A) CHỜ (do nhãn tg T3 > T1)
UN(A)		
UN(B)		
		WL(A)
		W(A)
		UN(A)
		UN(C)
	RL(B)	
	R(B)	
	WL(C)	
	W(C)	
	UN(B)	
	UN(C)	

* Các giao tác cùng rollback => cho bắt đầu lại theo thứ tự nhãn thời gian, **Giả sử các giao tác rollback được điều phối theo thứ tự thực thi ban đầu.**

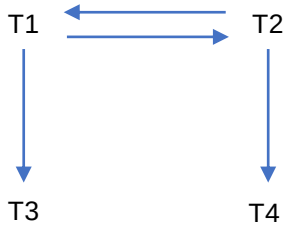
B2: Cho lịch biểu: r1(A); r2(B); w1(C); w2(D); r3(C); w1(B); w4(D); w2(A). Biết rằng BLL được trang bị khóa đọc, khóa ghi, nhả khóa cuối giao tác. ts(T1) < ts(T2) < ts(T3). Giao tác khởi động lại theo thứ tự mà giao tác đã roll back, và giao tác không bắt đầu lại trước khi (các) giao tác khác kết thúc.

* Thứ tự thực hiện: 1, 2, 1, 2, 3, 1, 4, 2

T1	T2	T3	T4
RL(A)			
R(A)			
	RL(B)		
	R(B)		
WL(C)			
W(C)			
	WL(D)		
	W(D)		
		RL(C) chờ	
WL(B) chờ			
			WL(D) chờ
	WL(A) chờ		

* Quy ước: Nếu Tj chờ Ti thì có cung từ Ti -> Tj

* Đồ thị chờ: có chu trình nên xảy ra deadlock



*** Ngăn ngừa deadlock bằng thuật toán Wait-Die**

T1	T2	T3	T4
RL(A)			
R(A)			
	RL(B)		
	R(B)		
WL(C)			
W(C)			
	WL(D)		
	W(D)		
		RL(C) RB	
WL(B) chờ			
			WL(D) RB
	WL(A) RB		
WL(B)			
W(B)			
UN(A)			
UN(C)			
UN(B)			
	RL(B)		
	R(B)		
	WL(D)		
	W(D)		
		RL(C)	
		R(C)	
			WL(D) RB
	WL(A)		
	W(A)		
	UN(B)		
	UN(D)		
	UN(A)		
		UN(C)	
			WL(D)
			W(D)
			UN(D)

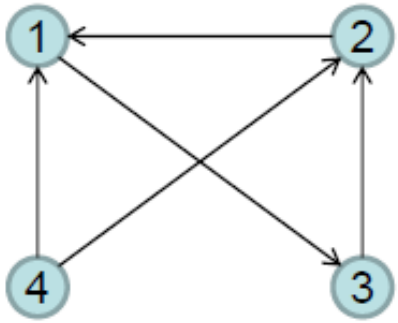
*** Ngăn ngừa deadlock bằng thuật toán Wound-wait**

T1	T2	T3	T4
RL(A)			
R(A)			
	RL(B)		
	R(B)		
WL(C)			
W(C)			
	WL(D)		
	W(D)		
		RL(C) chờ	
WL(B)	Rollback		
W(B)			
			WL(D)
			W(D)
UN(A)			
UN(C)			
UN(B)			
		RL(C)	
		R(C)	
			UN(D)
		UN(C)	
	RL(B)		
	R(B)		
	WL(D)		
	W(D)		
	WL(A)		
	W(A)		
	UN(B)		
	UN(D)		
	UN(A)		

* 3 giao tác cùng rollback => cho bắt đầu lại theo thứ tự nhãn thời gian, **Giả sử các giao tác rollback được điều phối theo thứ tự thực thi ban đầu.**

Dạng bài tập 4: Input: Lịch biểu S gồm các Ti cho trước
Output: Có deadlock không? Giải quyết?

T1: l1(A); r1(A); l1(B); w1(B); u1(A); u1(B)
T2: l2(C); r2(C); l2(A); w2(A); u2(C); u2(A)
T3: l3(B); r3(B); l3(C); w3(C); u3(B); u3(C)
T4: l4(D); r4(D); l4(A); w4(A); u4(D); u4(A)



Giao tác transaction, thao tác operation
(Insert, update, delete: Write|
select: Read)

T1	T2	T3	T4
l1(A); r1(A);			
	l2(C); r2(C);		
		l3(B); r3(B);	
			l4(D); r4(D);
	l2(A);		
		l3(C);	
			l4(A);
l1(B);			

Đồ thị có chu trình → có deadlock. Giải quyết deadlock bằng cách hủy giao tác ứng với node có nhiều cung vào ra nhất.

Dạng bài tập 5: Input: Lịch biểu S gồm các GT Ti (có xảy ra deadlock)
Output: Hãy ngăn ngừa deadlock dùng pp sắp xếp và lock ĐVDL theo thứ tự alphabet.

T1: l1(A); r1(A); l1(B); w1(B); u1(A); u1(B) → lock trên đvdl theo thứ tự alphabet
T2: l2(C); r2(C); l2(A); w2(A); u2(C); u2(A) → T2: l2(A); l2(C); r2(C); w2(A);u2(C); u2(A)
T3: l3(B); r3(B); l3(C); w3(C); u3(B); u3(C) → lock trên đvdl theo thứ tự alphabet
T4: l4(D); r4(D); l4(A); w4(A); u4(D); u4(A) → T4: l4(A); l4(D); r4(D); w4(A); u4(D); u4(A)

T1	T2	T3	T4
L1(A)			
R1(A)			
	L2(A) Chờ		
		L3(B)	
		R3(B)	
			L4(A) Chờ
		L3(C)	
		W3(C)	
L1(B) Chờ			
		U3(B)	
		U3(C)	

L1(B)			
W1(B)			
U1(A)			
	L2(A)		
	L2(C)		
U1(B)			
	R2(C)		
	W2(A)		
	U2(C)		
	U2(A)		
			L4(A)
			L4(D)
			R4(D)
			W4(A)
			U4(D)
			U4(A)

Dạng bài tập 6a: Input: Lịch biểu S gồm các Ti cho trước
Output: Hãy ngăn ngừa deadlock dùng thuật toán Wait-Die

st1< st2<st3<st4

T1: l1(A); r1(A); l1(B); w1(B); u1(A); u1(B)
T2: l2(C); r2(C); l2(A); w2(A); u2(C); u2(A)
T3: l3(B); r3(B); l3(C); w3(C); u3(B); u3(C)
T4: l4(D); r4(D); l4(A); w4(A); u4(D); u4(A)

T1	T2	T3	T4
l1(A), r1(A)			
	l2(C), r2(C)		
		l3(B), r3(B)	
			l4(D) r4(D)
	l2(A) Rollback		
		l3(C); W(C)	
			l4(A) Rollback
L1(B) Chờ			

		W3(C)	
		U3(B)	
L1(B)			
W1(B)			
		U3(C)	
U1(A)			
U1(B)			
	L2(C)		
	R2(C)		
			L4(D)
			R4(D)
	L2(A)		
	w2(A)		
			L4(A) Rollback
	U2(A)		
	U2(C)		
			T4 bắt đầu lại

Dạng bài tập 6b: Input: Lịch biểu S gồm các GT Ti (có xảy ra deadlock)
Output: Hãy ngăn ngừa deadlock dùng thuật toán Wound-Wait

st1< st2<st3<st4

T1: l1(A); r1(A); l1(B); w1(B); u1(A); u1(B)
T2: l2(C); r2(C); l2(A); w2(A); u2(C); u2(A)
T3: l3(B); r3(B); l3(C); w3(C); u3(B); u3(C)
T4: l4(D); r4(D); l4(A); w4(A); u4(D); u4(A)

T1	T2	T3	T4
l1(A), r1(A)			
	l2(C), r2(C)		
		l3(B) r3(B)	
			l4(D), r4(D)
	l2(A) Chờ		
		l3(C) Chờ	
			l4(A) Chờ
L1(B) Cho T3 rollback			

W1(B)			
U1(A)			
	L2(A)		
	w2(A)		
U1(B)			
	U2(A)		
	U2(C)		
			L4(A)
			w4(A)
			U4(A), u4(D)
		L3(B), r3(B)	
		L3(C), w3(C)	
		U3(B)	
		U3(C)	