Chapter 15 : 동시성 제어

Database System Concepts, 6th Ed.

Chapter 15-2:동시성 제어

- 로크 기반 규약
- 교착상태 처리
- 타임스탬프 기반 규약

교착 상태 처리

- 다음 두 트랜잭션을 고려해 보자 :

  T_1: write (A)　　T_2: write(B)

  write(B)　　write(A)

- 교착상태가 있는 스케쥴

T_1	T_2
lock-X on A	
write (A)	
	lock-X on B
	write (B)
	wait for **lock-X** on A
wait for **lock-X** on B	

교착 상태 처리(계속)

- 집합내의 각 트랜잭션이 집합내의 다른 트랜잭션을 기다리고 있는 트랜잭션의 집합이 있으면, 시스템은 교착상태에 있다.

- 교착상태 방지 규약(***Deadlock prevention*** protocol)은 시스템이 결코 교착상태에 돌입하지 않도록 보장한다. 몇 가지 기법은 다음과 같다 :
  - 각 트랜잭션이 실행을 시작하기 전에 모든 데이터 항목에 로크를 건다. (사전 선언)
  - 모든 데이터 항목의 부분 순서를 부과하고 트랜잭션은 부분 순서로 지정된 순서로만 데이터 항목에 로크를 걸 수 있도록 한다. (그래프 기반 규약)

보다 나은 교착상태 방지 전략

- 다음 기법들은 교착 상태 방지만을 위해 트랜잭션 타임스탬프를 사용한다.

- **대기 복귀** 기법(**wait-die** scheme) - 비선점
  - 오래된 트랜잭션이 최근의 트랜잭션에서 데이터 항목을 해제하기를 기다린다.
    최근의 트랜잭션은 오래된 트랜잭션을 기다리지 않고, 대신 복귀된다.
  - 트랜잭션은 필요한 데이터 항목을 얻기 전에 여러 번 복귀될 수 있다.

- **복귀 대기** 기법 (**wound-wait** scheme) - 선점
  - 오래된 트랜잭션이 최근의 트랜잭션을 기다리는 대신 강제로 복귀시킨다.
    최근의 트랜잭션은 오래된 트랜잭션을 기다린다.
  - 대기 복귀 기법보다 복귀가 적다.

- 대기 복귀와 복귀 대기 기법 모두에서 복귀된 트랜잭션은 원래의 타임스탬프를 가지고 재시작한다. 따라서, 오래된 트랜잭션은 새로운 트랜잭션에 대해 우선하며 기아를 피하게 된다.

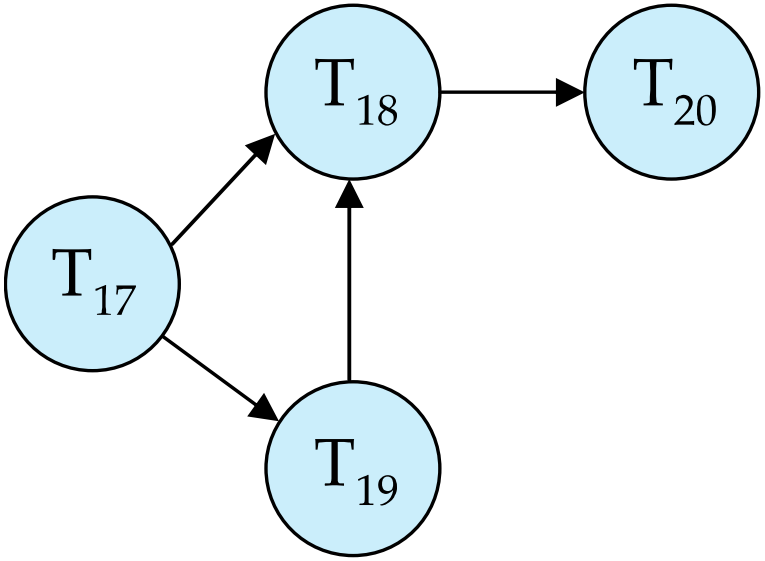
교착 상태 방지(계속)

- 타임아웃 기반 기법 :
  - 트랜잭션은 정해진 시간만큼만 로크를 기다린다. 그 후에는 타임아웃에 걸려 복귀된다.
  - 따라서 교착상태는 불가능하다.
  - 구현하기가 쉽지만, 기아의 가능성이 있다. 또한 타임아웃의 간격값을 정하기가 어렵다.

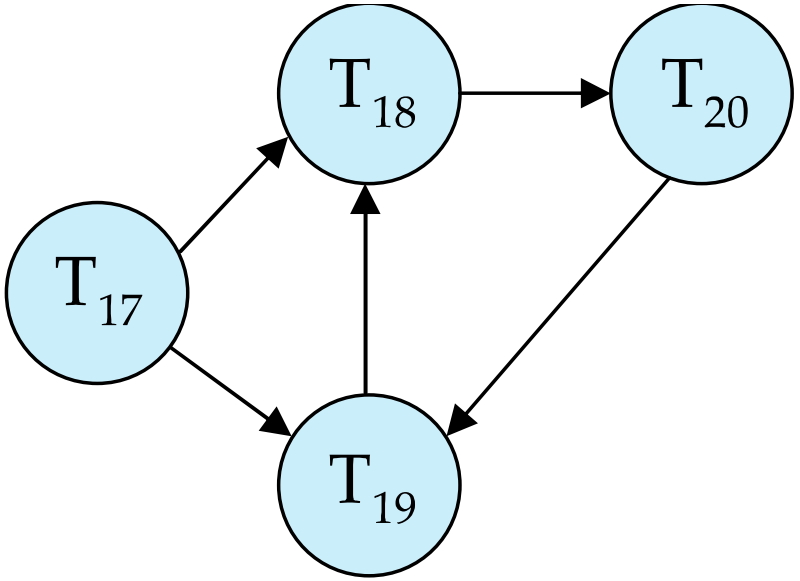
교착 상태 탐지

- 교착 상태는 G = (V, E)로 구성된 대기 그래프(wait-for graph)로 설명될 수 있다.
  - V 는 정점의 집합이다(시스템내의 모든 트랜잭션).
  - E 는 간선의 집합으로 각 요소는 $T_i \rightarrow T_j$ 형태의 순서 쌍이다.
- E내의 $T_i \rightarrow T_j$ 는 T_j 가 데이터 항목을 해제하기를 T_i가 기다리고 있음을 의미한다.
- T_j 가 현재 가지고 있는 데이터 항목을 T_i가 요구하면, 대기 그래프내에 간선 $T_i \rightarrow T_j$ 가 삽입된다. 이 간선은 T_i가 필요로 하는 데이터 항목을 T_j 가 더 이상 가지고 있지 않을 때만 제거된다.
- 시스템이 교착상태에 있는 필요 충분 조건은 대기 그래프에 순환을 가질 때이다. 순환이 존재하는지를 탐지하기 위해 교착상태 탐지 알고리즘을 주기적으로 호출해야 한다.

교착 상태 탐지(계속)



Wait-for graph without a cycle



Wait-for graph with a cycle

교착상태의 회복

- 교착상태를 탐지하면 :

  – 교착상태를 깨기 위해 몇 개의 트랜잭션을 복귀시켜야 한다.
  최소의 비용을 초래할 트랜잭션을 희생자로 선택한다.

  – 복귀 - 트랜잭션의 어디까지 복귀시킬지를 결정한다.
  * 전체 복귀 : 트랜잭션을 중단시키고 재시작한다.
  * 교착상태를 깨는데 필요한 부분만을 복귀시키는 것이 보다 효율적이다.

  – 동일한 트랜잭션이 항상 희생자로 선택되면 기아가 발생한다.
  기아를 피하기 위해 비용 요인 내에 복귀 횟수를 포함시킨다.

타임스탬프 기반 규약

- 각 트랜잭션이 시스템에 들어올 때 타임스탬프를 부여한다. 이전 트랜잭션 T_i가 타임스탬프 TS(T_i)를 가지고, 새로운 트랜잭션 T_j에 타임스탬프 TS(T_j) 가 할당되면, TS(T_i) < TS(T_j)가 된다.
- 이 규약은 타임스탬프가 정렬한 직렬성 순서대로 동시 실행을 관리한다.
- 그러한 행위를 보장하기 위해, 규약은 각 테이터 항목 Q에 대해 두 개의 타임스탬프 값을 유지한다 :
  - **W-타임스탬프**(Q)는 **write**(Q)를 성공적으로 실행한 트랜잭션들 중 가장 큰 타임스탬프이다.
  - **R-타임스탬프**(Q)는 **read**(Q)를 성공적으로 실행한 트랜잭션들 중 가장 큰 타임스탬프이다.

타임스탬프 기반 규약(계속)

- 타임스탬프 순서 규약은 어떤 충돌하는 **read**와 **write** 연산이 타임스탬프 순서로 실행되도록 보장한다.
- 트랜잭션 T_i가 **read**(Q)를 낸다고 가정하자.

  1. TS(T_i) < W-TS(Q)이면, T_i가 이미 덧씌워진 Q의 값을 읽으려 하는 것이므로, **read** 연산은 거절되고 T_i는 복귀된다.
  2. TS(T_i) ≥W-TS(Q)이면, **read** 연산은 실행되고 R-TS(Q)는 R-TS(Q) 와 TS(T_i)중 큰 값으로 설정된다.

타임스탬프 기반 규약(계속)

- 트랜잭션 T_i가 **write**(Q)를 낸다고 가정하자.

1. TS(T_i) < R-TS(Q)이면, T_i가 생성하려는 Q의 값은 이전에 다른 트랜잭션에서 처리를 위해 읽어 간 것으로 시스템에서는 결코 생성되지 않으리라고 가정한 것이다. 따라서, **write** 연산은 거절되고 T_i는 복귀된다.

2. TS(T_i) < W-TS(Q)이면, T_i가 쓸데없는 Q의 출력을 시도하는 것이어서 **write** 연산은 거절되고 T_i는 복귀된다.

3. 그렇지 않으면, **write** 연산은 실행되고 W-TS(Q)는 W-TS(Q)와 TS(T_i) 중 큰 값으로 설정된다.

규약 사용의 예

타임스탬프 1,2,3,4,5를 가진 트랜잭션들의 여러 데이터 항목에 대한 부분 스케쥴

T_1	T_2	T_3	T_4	T_5
				read (X)
	read (Y)			
read (Y)				
		write (Y)		
		write (Z)		
				read (Z)
	read (Z)			
	abort			
read (X)				
			read (W)	
		write (W)		
		abort		
				write (Y)
				write (Z)

$TS(T_1)=1$, $TS(T_2)=2$, $TS(T_3)=3$, $TS(T_4)=4$, $TS(T_5)=5$

R-timestamp(X):

W-timestamp(X):

R-timestamp(Y):

W-timestamp(Y):

R-timestamp(Z):

W-timestamp(Z):

R-timestamp(W):

W-timestamp(W):

타임 스템프 규약 하에서 생성될 수 있는 스케쥴

타임스탬프 1,2를 가진 트랜잭션들의 2개 데이터 항목에 대한 스케쥴

T25:

read *(B)*;
read *(A)*;
display*(A+B)*.

T26:

read(B);
B:=B-50;
write(B);
read(A);
A:=A+50;
write(A);
display(A+B)

스케쥴 3

T_{25}	T_{26}
read (B)	
	read (B)
	$B := B - 50$
	write (B)
read (A)	
	read (A)
display $(A + B)$	
	$A := A + 50$
	write (A)
	display $(A + B)$

TS(T_{25})=1
TS(T_{26})=2

R-timestamp(B):

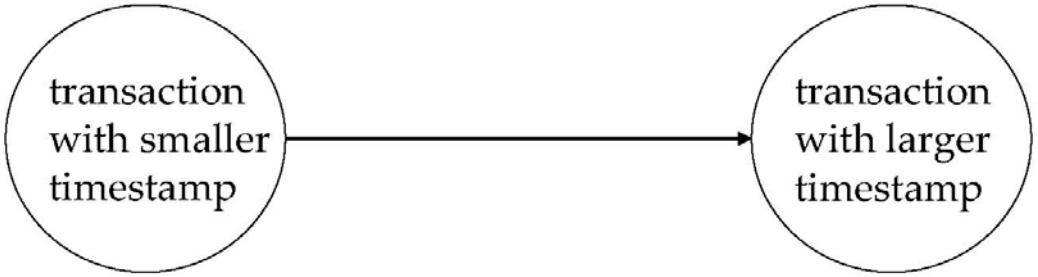
W-timestamp(B):

R-timestamp(A):

W-timestamp(A):

타임스탬프 순서 규약의 정확성

- 타임스탬프 순서 규약은 선행 그래프내의 모든 호가 다음과 같은 형태이다. 따라서, 선행 그래프 내에 순환이 없을 것이다.
- 선행그래프: 충돌 직렬성 검사 방법
- 타임스템프 순서규약은 충돌 직렬성을 보장한다 :

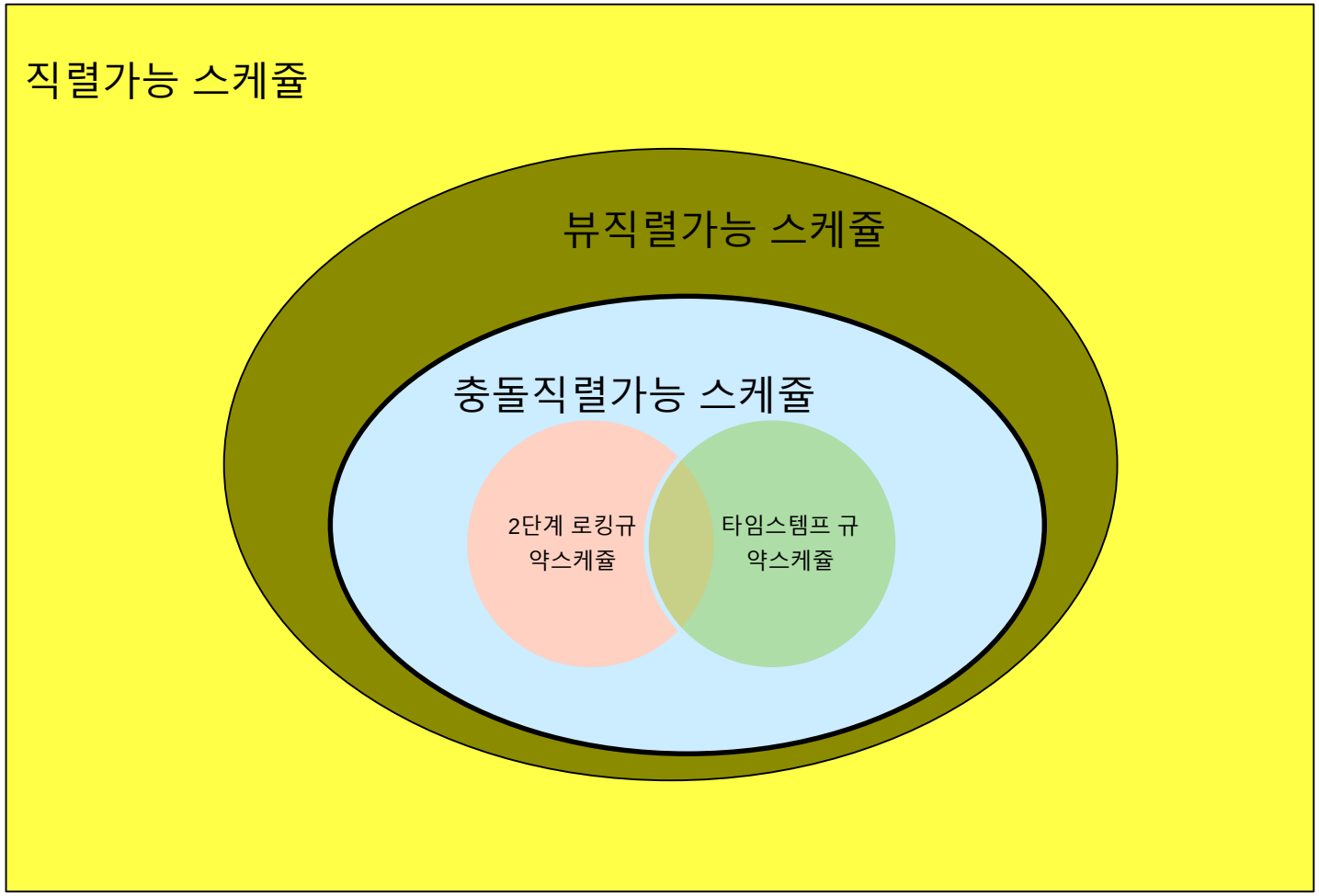


- 타임스탬프 규약은 트랜잭션이 결코 기다리지 않으므로 교착상태가 발생하지 않는다.
- 그러나, 스케쥴이 연쇄 복귀 행위가 발생할 수도 있으며 회복 가능하지 않을 수도 있다.

회복성과 연쇄 복귀의 회피

- 타임스탬프 순서 규약의 문제점 :
  - T_i는 중단되었지만 T_j는 T_i가 이미 쓴 데이터 항목을 읽었다고 가정하자.
  - 그러면 T_j는 중단되어야 한다; T_j가 보다 먼저 완료하면, 스케쥴은 회복 불가능이다.
  - 또한, T_j가 쓴 데이터 항목을 읽은 트랜잭션은 중단되어야 한다.
  - 이것은 연쇄 복귀를 야기할 수 있다.

- 해결책 :
  - write는 처리의 마지막에서 모두 수행되도록 트랜잭션을 구축한다.
  - 트랜잭션의 모든 쓰기는 원자적 행위를 이루도록 한다; 트랜잭션이 쓰기를 진행하는 중에는 어떤 트랜잭션도 실행할 수 없다.
  - 중단된 트랜잭션은 새로운 타임스탬프를 가지고 재시작한다.



End of Chapter