23. Transaction

▼ Transaction concept

▼ Trasaction

- **transaction** : 다양한 data item에 접근 혹은 update하는 program 실행 단위
- example
  - 계좌 A에서 계좌 B로 $50 송금하는 transaction 생각해보자!

$$
\begin{aligned}
&\textbf{read}(A)\\
&A := A - 50\\
&\textbf{write}(A)\\
&\textbf{read}(B)\\
&B := B + 50\\
&\textbf{write}(B)
\end{aligned}
$$

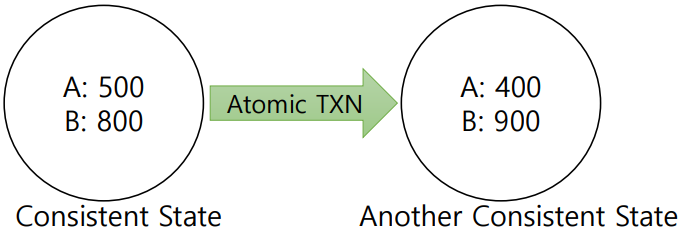
    - 성능적인 이슈 발생(h/w crash 혹은 system crash)
    - 여러 transaction의 concurrent execution 발생
  - atomic requirement : transaction 수행이 모두 되거나 아예 되지 않거나

    ⇒ 중간에 끊기면 update 문제가 발생함

    → 원자성이 필요함
  - Durability requirement : 한 번 commit된 transaction의 내용은 persistent

    → s/w 혹은 h/w가 실패하더라도 유지되어야 함
  - Consistency requirement : transaction의 실행에 따라 바뀌지 말아야 함

    → 일관성을 지녀야 함



- explicit integrity : primary keys and foreign keys
- implicit integrity

◦ Isolation requirement :

T1	T2
1.**read**(*A*) 2.*A* := *A* – 50 3.**write**(*A*)	read(A), read(B), print(A+B)
4.**read**(*B*) 5.*B* := *B* + 50 6.**write**(*B*)	

⇒ T1 중간에 T2 실행 : 부분적으로 update 됨

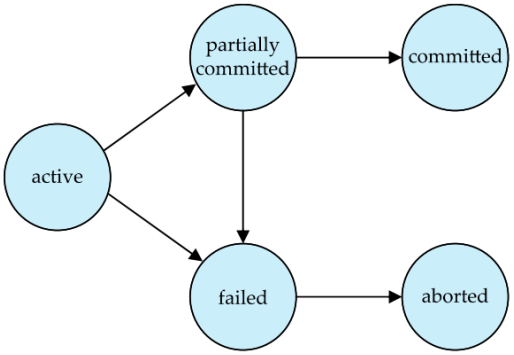
→ 실행 중인 transaction이 serial하게 실행되도록(독립적으로!)

▼ ACID principles

database system의 data들이 보장해줘야 하는 것

1. Atmoicity : all or nothing
2. Consistency : database의 일관성
3. Isolation : 동시에 실행되고 있는 다른 transaction 인식 x
4. Durability : transaction 완료 이후에는 update 내용이 영구적으로 지속

▼ Transaction State



- Active : 초기 상태, 즉 실행 중
- Partially committed : 마지막 statement가 실행되고 난 후
- Failed : proceed가 다 되지 않은 상태를 발견했을 때
- Aborted : transaction roll back 혹은 transaction 시작 상태로 restore → 중단!

    - abort되었을 때?
        1. restart
        2. kill
- Committed : 성공적으로 완전히 commit 완료되었을 때

▼ Concurrent Executions

- 한 번에 여러 개의 transaction이 system 내에서 concurrent하게 실행됨
- 장점
    1. processor와 disk의 utilization 향상
    2. 평균 response time 감소
- Concurrency control schemes : isolation을 성공하기 위한 mechanism

▼ Schedules

concurrent한 transaction의 instruction이 실행될 순서

- 성공적으로 실행이 완료된 transaction은 마지막 statement에서 commit
- 실행 완료에 실패한 transaction은 마지막 statement에서 instruction 중단

- serial schedule 1

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

- serial schedule 2

T_1	T_2
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	

not

- serial schedule 3

but, equivalent

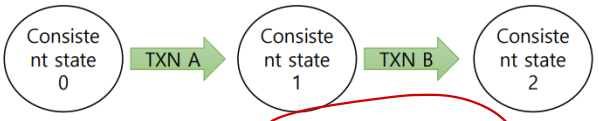
T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

- wrong schedule → not preserve

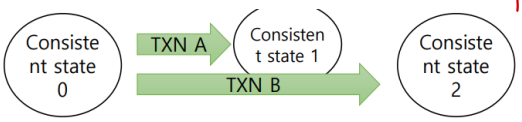
A=100

T_1	T_2
read (A)	
$A := A - 50$ A=50	
But, write ✗	read (A) A=100
	$temp := A * 0.1$ temp=
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

- **Serializability**
  - basic assumption : 각 transaction이 consistency를 보장한다고 가정
    - transaction 집합의 serial한 실행은 database consistency 보장함



- serial schedule과 동등한 schedule → **serializable이라고 함!**



- Serializability 종류

⇒ operation이 read/write만 있는 simple한 ver만 확인해보자

- Conflict serializability
  - conflict : 두 개의 operation이 충돌하는 것
    - operation이 서로 다른 transaction에 속하는 경우
    - operation이 같은 데이터에 작업 하는 경우
    - 둘 중 하나라도 write 작업을 하는 경우

1. I_i = **read**(Q), I_j = **read**(Q). I_i and I_j don't conflict.
2. I_i = **read**(Q), I_j = **write**(Q). They conflict.
3. I_i = **write**(Q), I_j = **read**(Q). They conflict
4. I_i = **write**(Q), I_j = **write**(Q). They conflict

  - conflict는 순서 바꿨을 때도 발생하는지 check
    - 문제 발생 안 하면 순서 바꾸더라도 결과 똑같음
  - conflict equivalent
    - 같은 transaction들의 operation들로 구성된 schedule
    - 양쪽 transaction 내의 conflicting operation의 실행 순서가 동일

⇒ 이때의 S는 conflict serializable이라고 함!

↳ consistency 보장? — ..

  - example

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Schedule 3

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Schedule 6

순서바뀌면 conflict.

- example → not Conflict serializability

T_3	T_4
read (Q)	
	write (Q)
write (Q)	

- View serializability
  - S와 S′ : 같은 transaction이지만 다르게 scheduled
  - view equivalent(조건?)
    1. S의 Ti이 Q의 초기값을 read했다면 → S′의 Ti도 Q의 initial value를 읽을 수 있어야 함
    2. S의 Ti가 Tj에 의해 생성된 Q를 read했다면

       → S′도 같은 Q 값을 읽어야 함
    3. S에서의 마지막 write(Q)를 실행

       → S′에서도 마지막 write(Q)를 실행해야 함

  ⇒ view equivalance : read/write에 기반
  - Schedule S : serial schedule에 view equivalent한다면

T1	T2	T3
R(Q)		
	W(Q)	
W(Q)		
	R(Q)	
		R(Q)
		W(Q)