**Chapter 14: Transactions**

**Transactions**

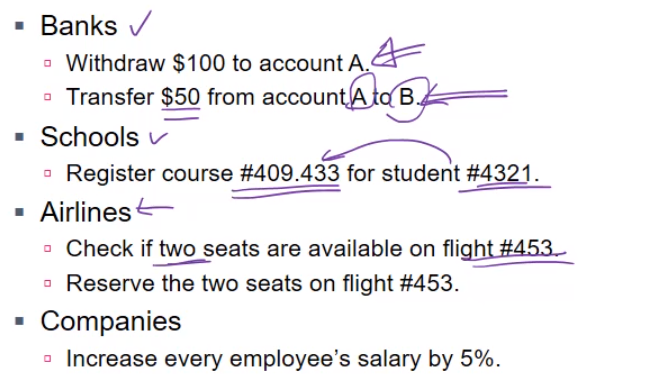
transactions이란? 거래, 일을 의미함.

작업의 논리적 단위. (유저 입장)

여러 가지 데이터 아이템을 접근하고 업데이트하는 프로그램 단위 (시스템 입장)

DBMS는 트랜잭션으로 선언된 작업 단위들을 보장하기 위해 특징들 (ACID)을 보장해야 함.

**Example of Transactions**



**Danger for Transactions**

여러 리스크들: system crash, disk failure, system error

Delayed disk write

Disk access는 page (block) 단위로 수행됨

Write operation이 특정한 data의 양이 수집된 이후 수행됨.

Buffer manager가 page를 pin할 수 있다.

다양한 트랜잭션의 동시적 수행

**Properties of a Transaction**

ACID properties

Atomicity: all or nothing

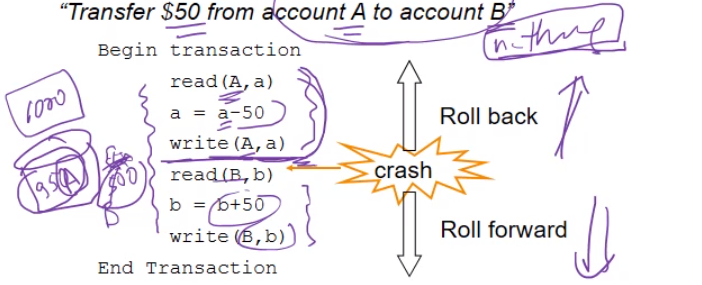
Consistency (Correctness): consistent 상태에서 다른 consistent 상태로 전이되는 것.

Isolation: 각 트랜잭션은 interfered 되면 안 됨. Concurrency. 방해받지 않음.

Durability: 트랜잭션이 끝난다면 그 효과는 durable & public 해야 함.

**Atomicity**

All or nothing.

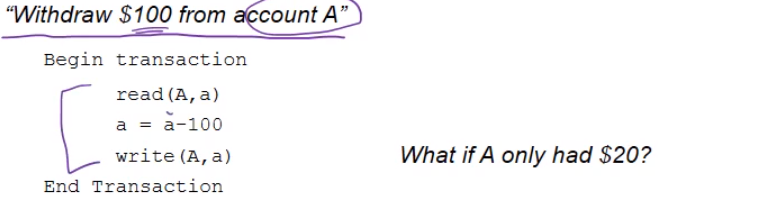


트랜잭션 수행 도중 crash 발생 시 roll back (nothing)또는 roll forward (all)수행.

**Consistency**

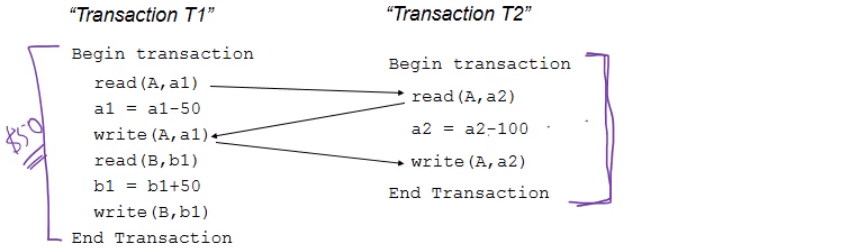
Move from a consistent state to another consistent state. 시간이 지나도state가 correct했으면 좋겠음.

이건 프로그래머의 책임.



**Isolation**

Should not be interfered by other transactions (concurrency)



이걸 동시에 수행하면 a에 대해 문제가 생김. 그런 문제를 방지해야 함.

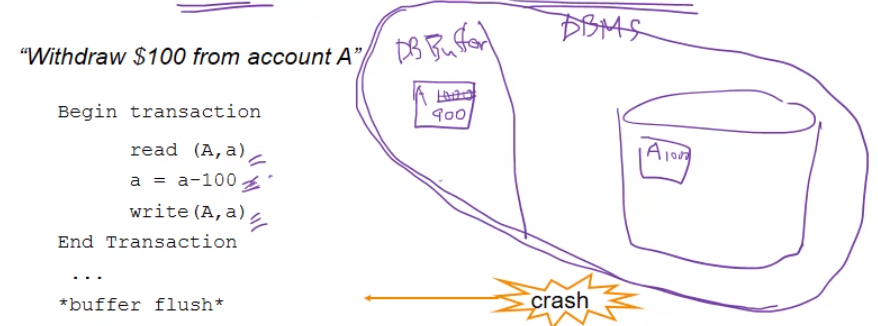
Serial Execution vs Concurrent Execution.

Transaction 각각이 consistent해야 하는데, 동시에 수행하면 아래와 같이 inconsistent한 상태가 있을 수 있음. 그 상황을 처리해야 함.



**Durability**

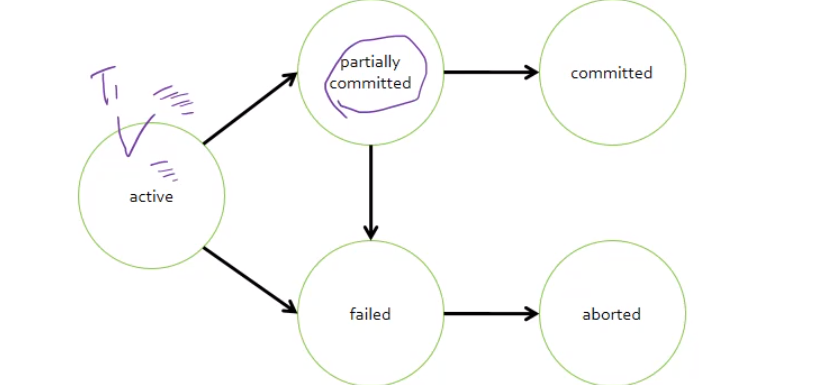
The effect of a completed transaction should be durable & public.



Buffer와 디스크까지가 DBMS가 관리하는 영역. 디스크에 쓰기 전 crash가 나면 트랜잭션 결과가 디스크에 반영이 안 됨. 이 경우를 처리해야 durable.

public이라는 건 트랜잭션이 끝나면 업데이트가 되지 않더라도 다른 연산들이 해당 결과를 바탕으로 수행되어야 함.

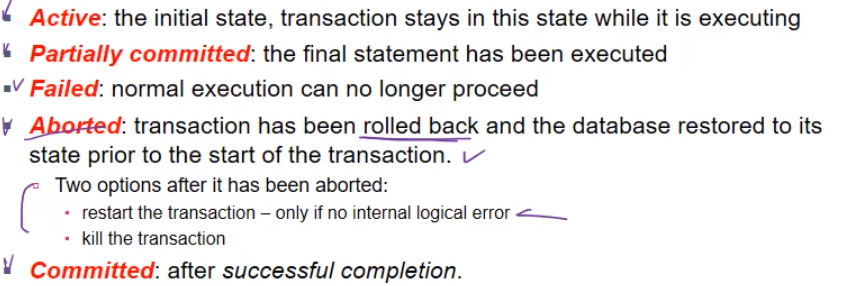
**Transactions States**



Partially committed는 모든 작업이 끝났지만 DBMS가 ACID 도장을 찍지 않은 상태.

committed되면 DBMS가 무조건 ACID를 보장해야 함.

Failed 되면 또 DBMS가 일련의 장치를 한 뒤 aborted state로 넘어감. 그 때도 ACID를 보장.



**Concurrent Executions**

시스템에서 여러 개의 트랜잭션이 동시에 수행되는 것이 허용됨.

- increased processor and disk utilization

- reduced average response time

Concurrency control schemes – isolation을 수행하기 위한 메커니즘들

데이터베이스의 consistency가 무너지는 것을 막기 위해 여러 transaction들 사이의 interaction을 control.

**Schedules**

트랜잭션의 simplified view

트랜잭션이 local buffer에서 read와 write를 하며 데이터에 대한 임의의 연산을 수행한다고 가정.

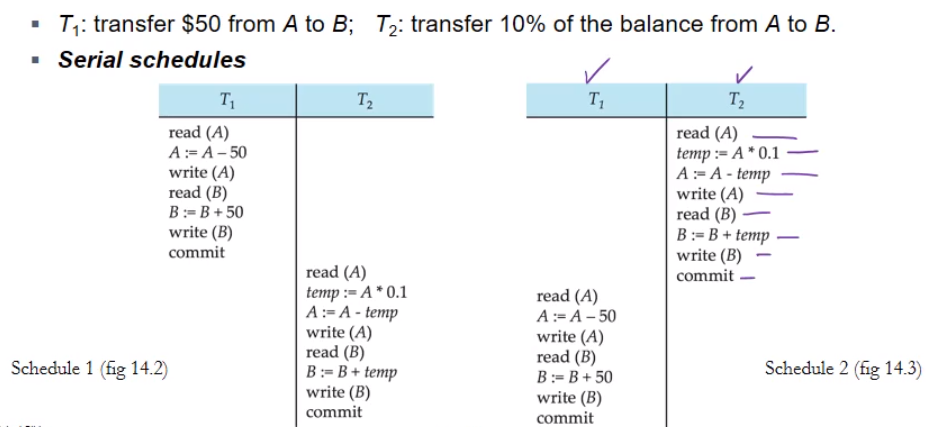
Read와 write를 제외한 다른 연산들은 무시. – 오직 read, write 연산으로만 simplified schedule이 이루어짐.

트랜잭션의 타임라인이라고 생각할 수 있음. Sequences of instructions that indicate the chronological order of execution

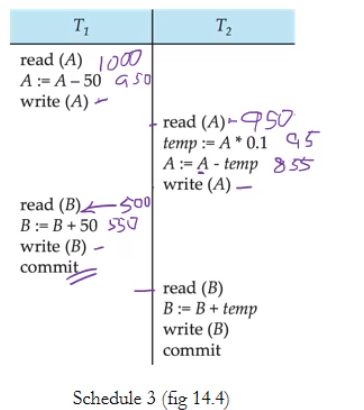
Schedule for a set of transactions

이 트랜잭션들의 모든 instruction으로 이루어져야 함.

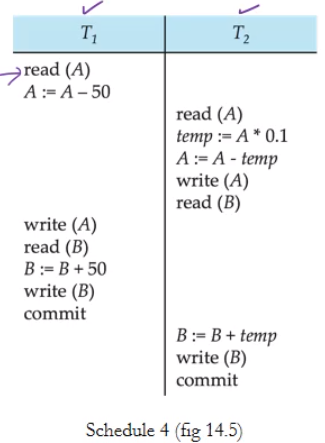
각각의 트랜잭션에서 나타난 instruction들의 순서가 반드시 지켜져야 함.

**Example Schedules** ****

이런 schedule들은 serial schedule이라고 함. 하나의 트랜잭션이 다 끝날 때까지 다른 트랜잭션이 수행되지 않음.

얘는 serial schedule이 아님. 하지만 schedule 1과 equivalent함. (효과가 같은 schedule을 서로 equivalent하다고 한다.)

스케쥴 1과 3에서 A+B의 합은 같음. 스케쥴 2는 T2를 먼저 했기 때문에 나머지와 inequivalent.

스케쥴 4는 A+B의 합이 보존되지 않음.

따라서 스케쥴 4는 스케쥴 1과 equivalent 하지 않다.

**Serializability**

기본 가정 – 각각의 트랜잭션은 데이터베이스의 consistency를 유지해야 함.

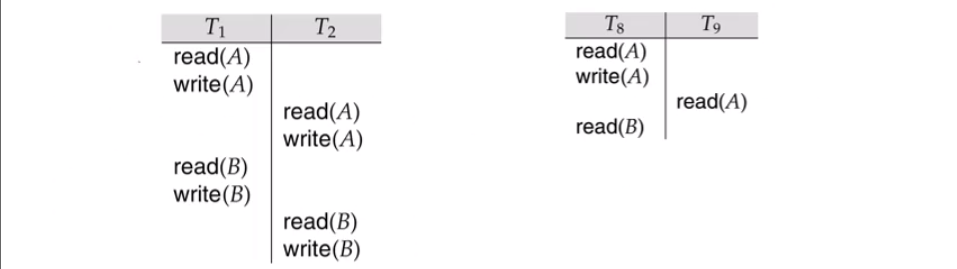
즉, 트랜잭션들의 serial execution은 데이터베이스 consistency를 유지함.

여러 개가 동시에 실행되어도 이렇게 serial의 효과를 보장해야 한다는 뜻.

Concurrent schedule은 serial과 같으면, serializable하다.

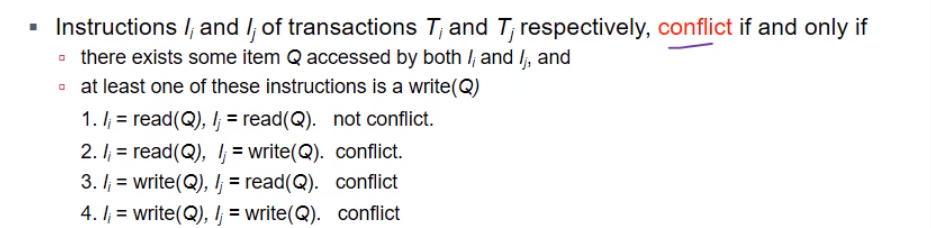
S1은 모든 DB의 가능한 인스턴스에서 S2의 실행과 같은 DB 상태 결과가 되면 동치이다.

**Serializable Schedules**



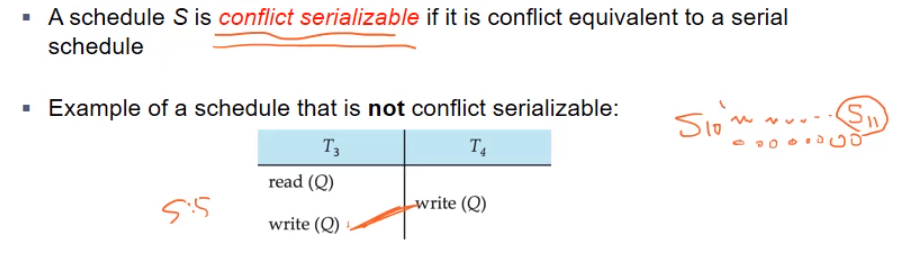
두 경우 모두 Serializable. Serial은 아님.

**Conflict Serializability**



같은 아이템 Q에 대해서 트랜잭션 두개가 모두 접근하고, 둘 중 하나가 write(Q)를 가지고 있으면 conflict. 둘 다 read일 경우는 conflict 아님.

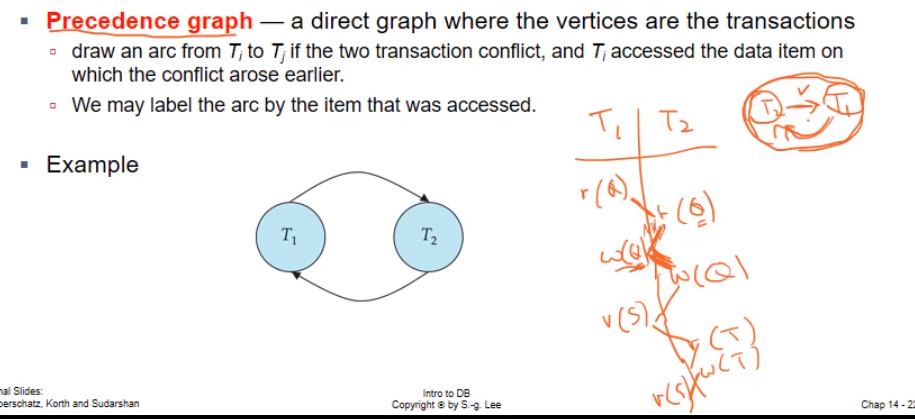
직관적으로, conflict는 둘 사이의 temporal order를 강요함.



Conflict serializable이란 non-conflict한 스케쥴에서 순서를 바꿔서 같은 스케쥴이 된다면 conflict serializable이라고 함.

이 때 한 트랜잭션의 순서를 바꾸는 것은 아님. 두 트랜잭션 사이의 순서를 바꾸는 것.

**Testing for Serializability**



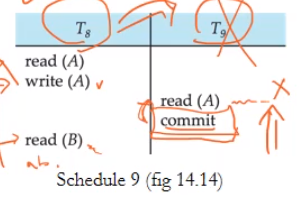
**Test for Conflict Serializability**

Conflict serializable하려면 해당 스케쥴에 cycle이 없어야 함(acyclic).

Cycle 탐지는 n^2. (혹은 n+e 가능)

만약 precedence graph가 사이클이 없으면, topological sorting을 통해서 serial한 스케쥴을 얻을 수 있음.

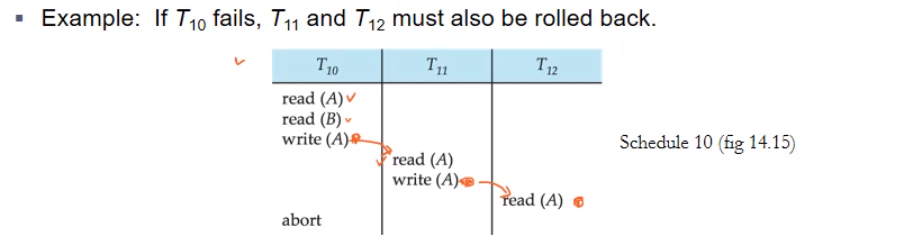
**Recoverability**

****

스케쥴 9는 recoverable하지 않음. T8에서 문제가 생기면 T9도 롤백해야하는데 이미 커밋이 되어있기 때문.

스케쥴이 Recoverable하려면 T\_j가 아이템을 읽었는데 T\_i가 그 전에 해당 아이템을 write한 경우, T\_i의 커밋이 T\_j보다 앞에 있어야 함.

**Cascading Rollback**

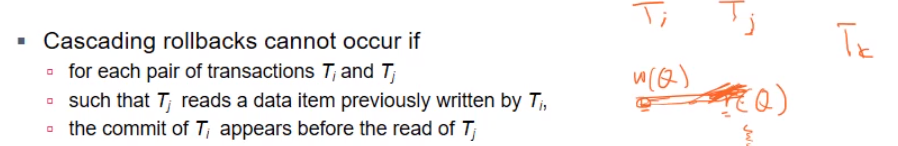


하나의 트랜잭션의 failure는 여러 개의 트랜잭션의 롤백을 일으킬 수 있음.

하지만 엄청난 양의 undoing을 해야할 수도 있음.

**Cascadeless Schedules**

Cascading rollback은 다음 경우 일어날 수 없음.



T\_j는 T\_i가 커밋할 때까지 기다리는 것.

Dirty read를 하지 않는 것 (transaction에서 commit되지 않은 write는 읽지 않는다.

모든 cascadeless 스케쥴은 또한 복구 가능하다.

그래서, 스케쥴들을 cascadeless한 것들로 제한하는 게 바람직함.