**Chapter 15: Concurrency Control**

**Concurrency Control**

스케쥴은 serializable하고 recoverable 해야 함. (데이터베이스 consistency를 위해서)

그리고 cascadeless면 좋음.

동시에 하나의 트랜잭션만 실행하는 것은 serial schedule을 만듦. 하지만 concurrency는 구림.

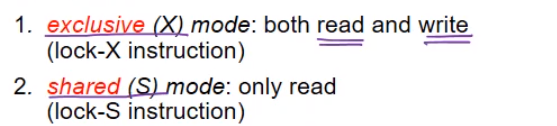
Concurrency control은 concurrency의 정도랑 overhead의 정도랑 tradeoff가 있음.

Isolation과 관련 깊음.

**Lock-Based Protocols**

Lock은 concurrent access를 제어하기 위한 메커니즘.

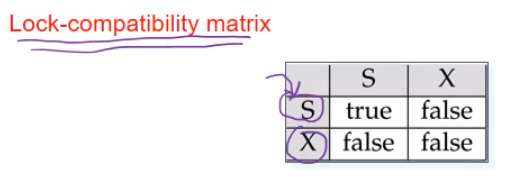
두 가지 모드가 있음.



Lock request는 concurrency-control manager에 의해 만들어짐.

트랜잭션은 request가 승인된 후에만 진행될 수 있음.

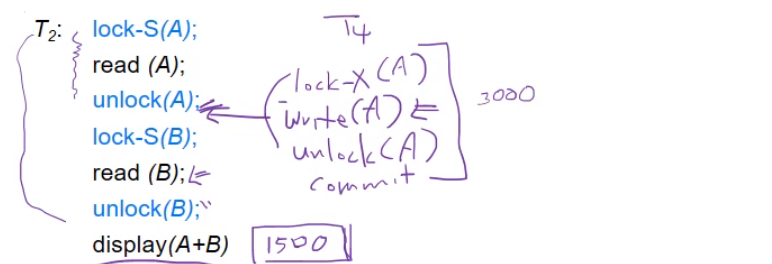
**Granting of Locks**



T1이 shared일 때 T2가 shared를 요청할 때만 가능.

트랜잭션은 만약 요청한 lock이 다른 트랜잭션에 의해 잡혀 있는 lock과 compatible하면 lock의 권한을 얻을 수 있음.

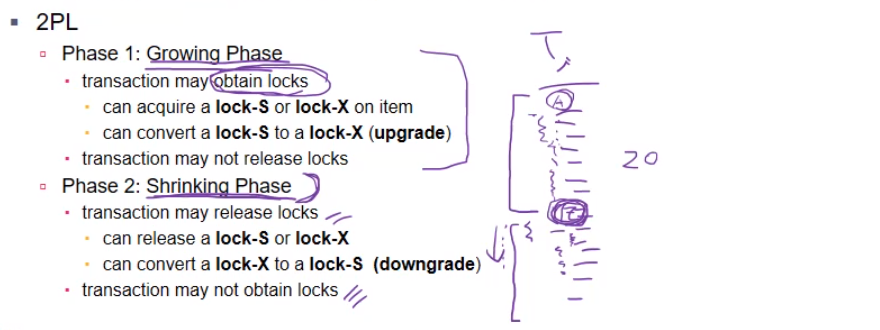
**Example**



T2와 T4는 non-serializable임. Write가 커밋되기 전에 read해서.

**Two-Phase Locking Protocol (2PL)**

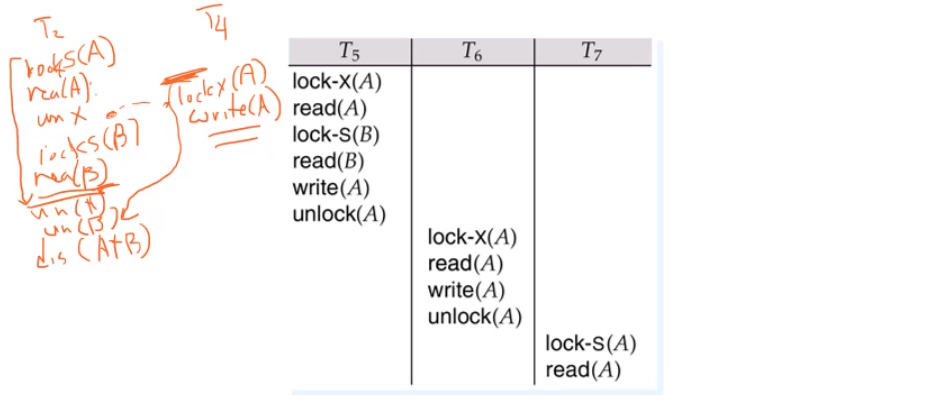
아래의 규칙들이 모든 트랜잭션에 대해 lock을 요청하고 풀어줄 때 사용됨.



Phase 1에서 lock-S를 lock-X로 바꾸는 건 upgrade로 보고, lock을 얻는 것과 같은 개념으로 봄.

반대로 Phase 2에서 lock-X를 lock-S로 바꾸는 건 downgrade, lock을 release하는 것과 같음.

**Example**



앞의 예시의 T2, T4의 경우 왼쪽 필기 같은 식으로 순서가 바뀜.

2PL 때문에 serializability를 강제하게 되어 있음.

**Features of 2PL**

Protocol이 (conflict) serializability를 보장함.

트랜잭션들은 lock point의 순서에 의해 serialize될 수 있음.

2PL이 사용될 경우 허용되지 않는 conflict serializable 스케쥴도 있음.

하지만 중요한 것은 2PL을 사용하면 모두 serializable하다는 것.

Deadlocks:

2PL을 사용할 경우 deadlock, starvation이 일어날 수 있음.

Cascading rollback도 가능함.

**Strict / Rigorous 2PL**

Strict 2PL

Exclusive lock은 commit/abort할 때까지 release하지 않고 가지고 있음.

Cascading rollback을 피할 수 있음. (dirty read를 방지하기 때문)

Rigorous 2PL (많이 사용)

모든 lock을 commit/abort까지 release하지 않고 hold함.

트랜잭션들은 커밋된 순서로 serialize될 수 있음.