트랜잭션

▼ 트랙잭션

- 데이터베이스 내에서 수행되는 **작업의 최소 단위**
- 데이터베이스의 **무결성을 유지**하며 **DB의 상태를 변화**시키는 기능을 수행
- `COMMIT` : 모든 작업이 성공해서 데이터베이스에 정상 반영하는 것
- `ROLLBACK` : 작업 중 하나라도 실패해서 트랜잭션 이전으로 되돌리는 것
- `SAVEPOINT` **:** 현 시점에서 SAVEPOINT까지 트랜잭션의 일부만 롤백

ACID (트랜잭션의 특징)

- `Atomicity(원자성)` : 트랜잭션은 **모두 수행**되거나 **모두 수행되지 않는다**
  - 트랜잭션안의 단 하나의 작업이라도 실패시 전체를 롤백하여 원자성을 보장
- `Consistency(일관성)` : 하나의 **트랜잭션 이후** 데이터베이스의 **상태는 이전과 같이** 유효해야 한다 (모든 트랜잭션은 일관성 있는 데이터베이스 상태를 유지해야 함)
  - 트랜잭션이 일어난 이후에도 데이터베이스의 제약이나 규칙을 만족
- `Isolation(격리성)` : 트랜잭션 수행 중 다른 트랜잭션의 작업이 **서로에게 영향을 미치지 않는다**
  - 트랜잭션 간에 서로 독립적으로 수행
  - Isolation level(격리 수준) : 트랜잭션에서 일관성 없는 데이터를 허용하도록 하는 수준
- `Durability(지속성)` : 트랜잭션이 **성공**시 적용된 결과는 **영구적**으로 **지속**된다.
  - 시스템이 다운되거나, 기타 문제가 발생해도 성공한 트랜잭션 내용을 복구

Isolation level(격리 수준) : 내려갈수록 느려짐

- **READ UNCOMMITED** (커밋되지 않은 읽기)
  - 트랜잭션1 진행 중에 A+1한 데이터를 트랜잭션2가 읽어왔는데 트랜잭션1이 롤백되면 트랜잭션2는 존재하지 않는 데이터를 읽어온 것이 될 수 있는 문제 발생
- **READ COMMITED**(커밋된 읽기) : **일반적**으로 많이 사용
  - 트랜잭션2가 읽은 A를 트랜잭션3이 수정을 할 수 있어 트랜잭션2가 다시 읽으면 값이 다른 문제 발생
- **REPEATABLE READ** (반복 가능한 읽기)
  - 하나의 트랜잭션이 읽은 로우를 다른 트랜잭션이 수정할 수 없게 함
  - 트랜잭션1이 샐리라는 로우를 얻었을 때 트랜잭션2가 바다라는 새로운 로우를 추가하면 다시 조회했을때 발견되지 않은 새로운 로우가 발견될 수 있다는 문제 발생
- **SERIALIZABLE**(직렬화 가능)
  - 동시에 같은 테이블에 접근할 수 없음

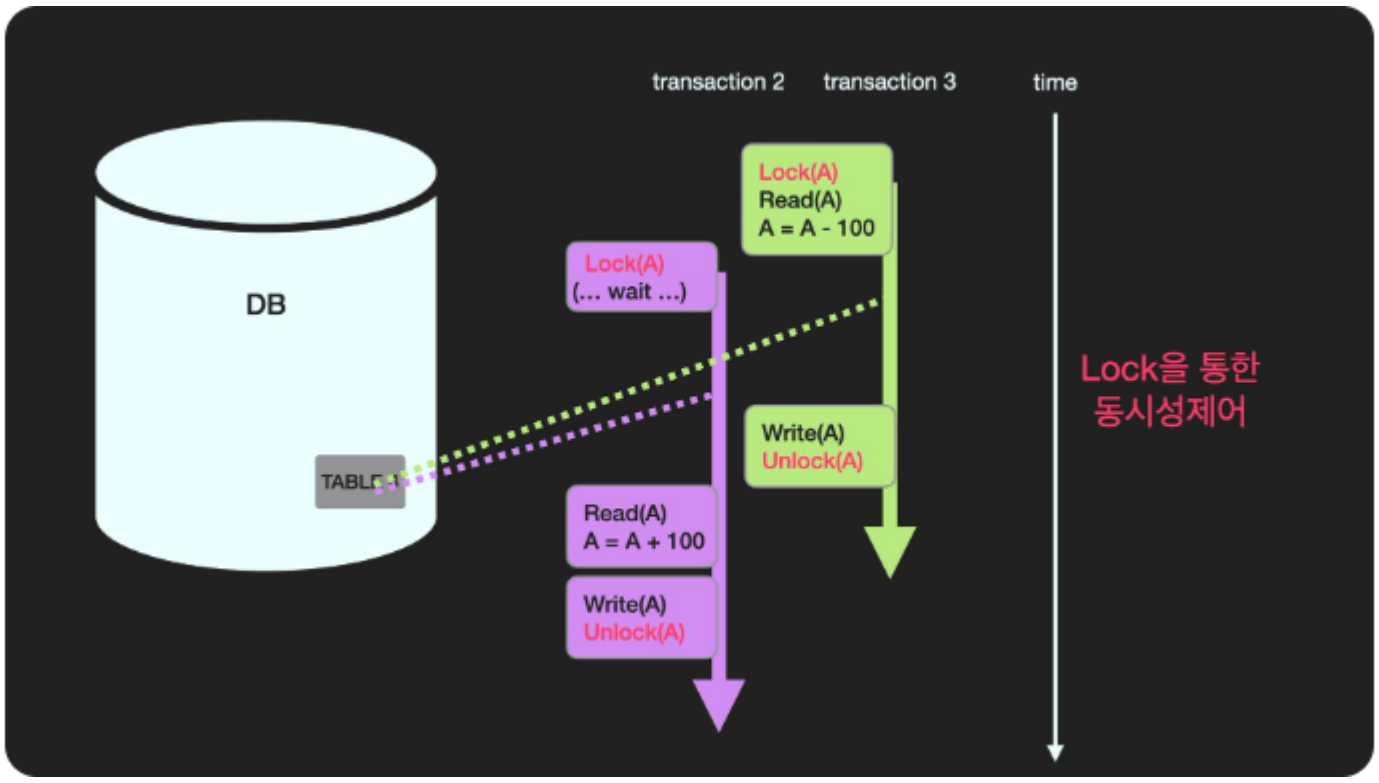
동시성 제어

- 다중 사용자 환경에서 둘 이상의 **트랜잭션이 동시에** 수행될 때, 일관성을 해치지 않도록 트랜잭션의 **데이터 접근 제어**
- 다중 사용자 환경을 지원하는 DBMS의 경우, 반드시 지원해야 하는 기능

	트랜잭션1	트랜잭션2	발생 문제	동시접근
[상황 1]	읽기	읽기	읽음(읽기만 하면 아무 문제가 없음)	허용
[상황 2]	읽기	쓰기	오손 읽기, 반복불가능 읽기, 유령 데이터 읽기	허용 혹은 불가 선택
[상황 3]	쓰기	쓰기	갱신손실, 모순성, 연쇄 복구	허용불가(LOCK을 이용)

Loking(로킹)기법 : 트랜잭션들이 동일한 데이터 항목에 대해 임의적인 **병행 접근을 하지 못하도록 제어**하는 것

- 방법
  - 트랜잭션 3이 데이터 A에 대해 Read(A) or Write(A)연산을 수행하려면 반드시 lock(A) 연산을 먼저 해주어야 함
  - 트랜잭션 3이 실행한 lock(A)에 대해서는 해당 트랜잭션이 종료되기 전에 반드시 unlock(A)연산을 해주어야 함
  - 트랜잭션 2는 다른 트랜잭션에 의해 이미 lock이 걸려 있는 A에 대해 다시 lock(A)를 수행시키지 못하므로 대기하고, 락을 획득한 후 부터 연산 수행 가능



Lock의 종류

1. **Shared Lock (공유 lock, Read Lock)**: 보통 **데이터를 읽을 때** 사용하는 lock
   - Shared lock의 경우, 같은 데이터에 여러 개의 shared lock을 설정 가능
   - Shared lock이 설정된 데이터에 대해 exclusive lock을 설정 불가
2. **Exclusive Lock (배타 lock, Write Lock)**: 보통 **데이터를 변경할 때** 사용하는 lock
   - Exclusive lock이 해제될 때까지 다른 트랜잭션은 해당 데이터에 접근 불가
   - 다른 트랜잭션이 수행되고 있는 데이터에 대해 exclusive lock 설정 불가

▼ DeadLock(교착상태)

- 여러 트랜잭션들이 각각 자신의 데이터에 대하여 **lock을 획득한 상태에서 상대방 데이터에 대하여 접근하고자** 대기를 할 때 교차 대기를 하게 되면서 **서로 영원히 기다리는 상태**

해결방법

[예방 기법]

- 각 트랜잭션이 실행되기 전에 필요한 **데이터를 모두 잠금(Locking)**하는 것
- 하지만 데이터가 많은 경우 모든 데이터를 잠금해야 되기 때문에 트랜잭션의 병행성을 보장하지 못함

[방지 기법]

1. MySQL의 경우, isolation level를 READ-COMMITED로 변경해 교착상태 발생 횟수를 줄일 수 있음

- READ-COMMITED로 변경할 경우, 쿼리문 단위로 exclusive lock을 걸어 트랜잭션 진행 중이더라도 exclusive lock이 걸린 데이터가 아니라면 다른 트랜잭션이 접근해 작업 가능

2. LOCK_TIMEOUT 시간을 설정해 lock이 지속할 수 있는 **최대 시간 설정**
    - 해당 시간 지나면 트랜잭션 및 쿼리 취소

[회피 기법]

- 자원 할당 시, 트랜잭션의 **시간 스탬프(Time Stamp)를 활용**해 교착 상태를 회피하는 방법

1. **Wait-Die 방식**: 다른 트랜잭션이 lock을 건 데이터에 대해서 접근하려고 할 때,
    - 현재 트랜잭션이 먼저 실행된 트랜잭션이라면 대기
    - 현재 트랜잭션이 나중에 실행된 트랜잭션이라면 포기, 나중에 재요청
2. **Wound-Wait 방식:** 다른 트랜잭션이 lock을 건 데이터에 대해서 접근하려고 할 때,
    - 현재 트랜잭션이 먼저 실행된 트랜잭션이라면 선점(빼앗기)
    - 현재 트랜잭션이 나중에 실행된 트랜잭션이라면 대기

[낙관적 병행 제어 기법]

- 트랜잭션이 실행되는 동안에는 **검사 X**
- 트랜잭션이 다 실행된 이후에 검사해서 **문제가 있다면 되돌리고**, 문제가 없어야 커밋 가능