Module 12

병행성 제어

NHN Academy

Overview

- 트랜잭션
- 일관성
- Lock
- SQL의 트랜잭션 지원
- 잠금 없는 병행 제어

트랜잭션

- 트랜잭션 개요
- 트랜잭션과 일관성
- 트랜잭션과 스케줄

트랜잭션 개요

- 데이터베이스 객체들에 대한 일련의 판독(Read) 또는 기록(Write)
  - 판독시에는 객체의 내용이 디스크에서 버퍼풀의 프레임으로 페이징되고 그 값이 프록그램 변수로 복사됨
  - 기록시에는 프레임의 객체 복사본이 수정된 후 디스크에 기록
- 데이터베이스 객체는 프로그램이 정보를 기록하거나 판독하는 단위
  - 페이지, 레코드등의 될 수 있으며 DBMS마다 다르게 구현
- ACID
  - **원자성**(Atomicity) 트랜잭션과 관련된 작업들이 부분적으로 실행되다가 중단되지 않음을 보장
  - **일관성**(Consistency) 실행을 완료하면 언제나 일관성 있는 데이터베이스 상태로 유지됨을 보장
  - **격리성**(Isolation) 트랜잭션 수행 시 다른 트랜잭션의 연산작업이 끼어들지 못하도록 보장
  - **지속성**(Durability) 성공적으로 수행된 트랜잭션은 영구적으로 데이터베이스에 반영됨을 보장

트랜잭션과 일관성

- 트랜잭션의 인터리빙과 관계없이 모든 트랜잭션은 순서대로 실행한 결과를 보장해야 함
  - 두 트랜잭션 t1과 t2가 동시에 실행되더라도 실제 효과는 t1이 종료된 후 t2가 실행된 결과와 같아야 함
  - 격리성(Isolation) 만족
  - 무결성 제역조건을 집행할 메커니즘 제공
- DBMS는 서로 다른 트랜잭션 단위 작업을 인터리빙
  - DBMS는 인터리빙해서 실행한 실제 효과가 **트랜잭션들을 선형으로 실행한 것과 동등**하도록 보장함
- 손상 복구
  - 트랜잭션이 정상적으로 완료되지 못하는 경우
    - 트랜잭션을 철회(abort)하는 경우
    - 시스템 손상
    - 예상치 못한 상황
  - 일관성 보장을 위해 미완료 트랜잭션의 단위 작업을 무효화 함

트랜잭션과 스케줄

- DBMS 관점에서 트랜잭션 하나는 단위 작업(action – read와 write)들의 나열
  - 트랜잭션은 부분적으로 순서화 된(partially ordered) 집합으로 정의할 수 있음
  - 단위작업의 리스트로 취급할 수 있음
- 트랜잭션 T에서 객체 O를 읽는 작업에서,
  - 판독 단위 작업 $\mathbf{R_T(O)}$, 기록 단위 작업 $\mathbf{W_T(O)}$, 철회 작업 $\mathbf{A_T}$, 완결 작업 $\mathbf{C_T}$
- 트랜잭션 스케줄
  - 어떤 트랜잭션에 있는 단위작업의 리스트
  - DBMS가 보는 트랜잭션의 단위 작업들을 기술
  - 각 트랜잭션에 Abort와 Commit 연산을 넣은 스케줄을 완전한 스케줄(Complete Schedule)로 부름
  - 다른 단위 작업이 인터리빙 되지 않는 경우 직렬 스케줄(Serial Schedule)로 부름

T1	T2
R(A) W(A)	
	R(B) W(B)
R(C) R(C)	

Lab 12-1 트랜잭션 제어

일관성

- 직렬 가능성
- 교차수행에 의한 이상
- 기록-판독 충돌(WR Conflict)
- 판독-기록 충돌(RW Conflict)
- 기록-기록 충돌(WW Conflict)

직렬가능성

- 직렬 가능 스케줄(Serializable Schedule)
  - 일관적인 데이터베이스 인스턴스에 대한 효과가 완결된 트랜잭션 집합 S에 대대한 완전한 직렬 스케줄의 효과와 동등함이 보장되는 스케줄
  - 주어진 스케줄을 수행해서 나온 데이터베이스 인스턴스는 해당 트랜잭션들을 직렬 순서대로 수행해서 나온 데이터베이스 인스턴스와 동등
- 트랜잭션들을 똑같이 직렬로 수행하더라도 순서를 바꾸면 결과도 달라질 수 있음 – 이 결과는 수용 가능함
- Abort된 트랜잭션이 있는 스케줄까지 포함하지는 않음

교차수행에 의한 일부 이상

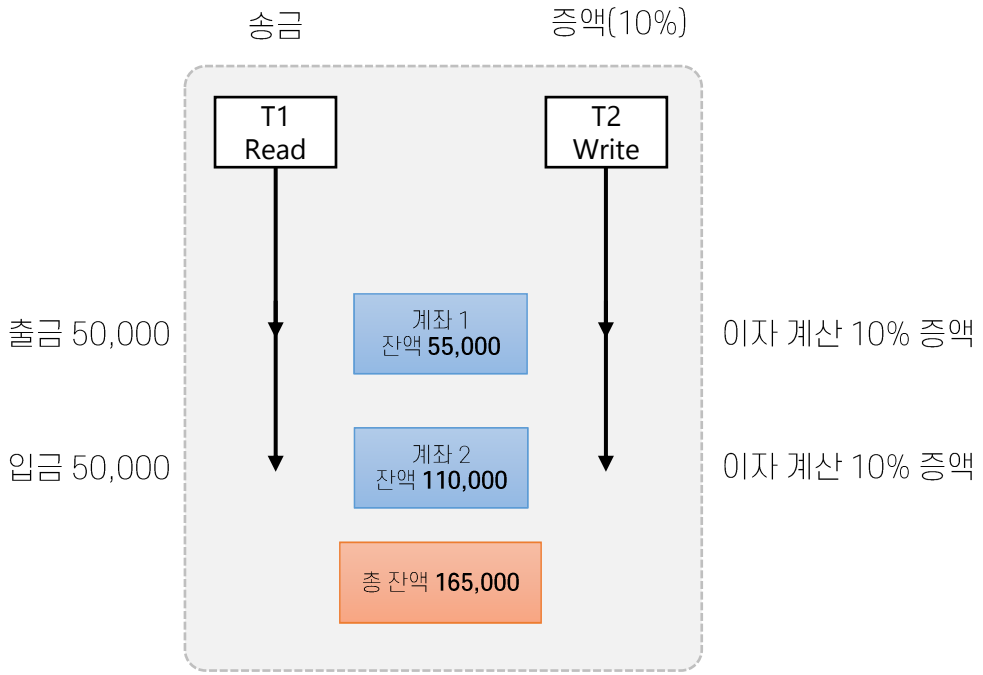
- 트랜잭션 T1과 T2의 단위 작업이 충돌(conflict)하는 경우
  - 동일한 개체에 대해 두 개 이상의 단위 작업이 수행될 때 하나 이상의 트랜잭션이 기록 연산일 경우
- 기록-판독 충돌(WR Conflict)
- 판독-기록 충돌(RW Conflict)
- 기록-기록 충돌(WW Conflict)

기록-판독 충돌(WR 충돌)

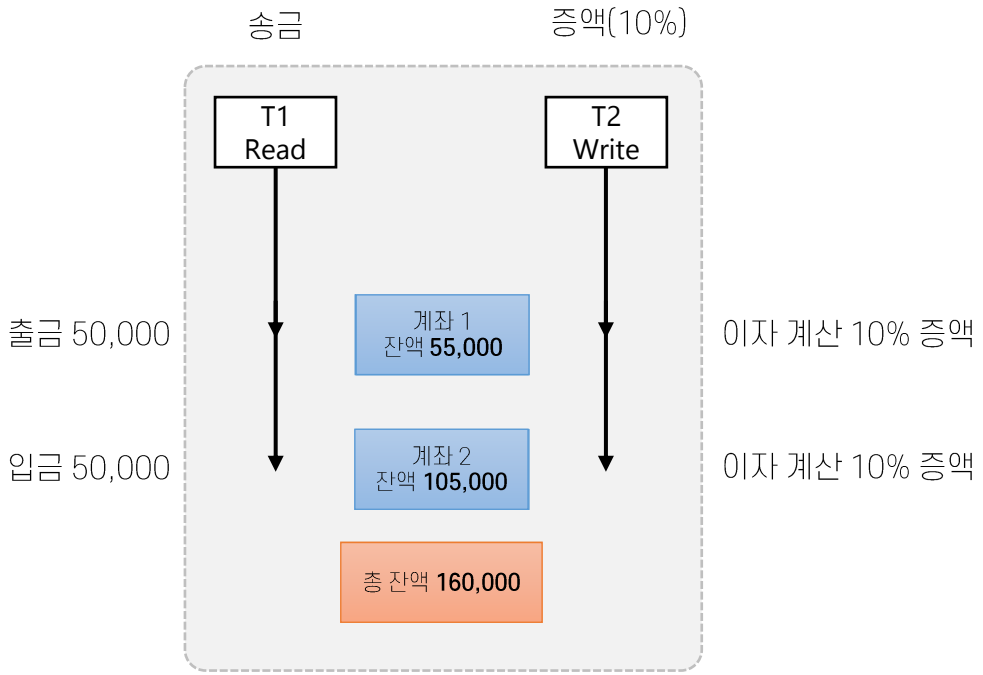
- 미 완결된 데이터를 읽는 경우
- 기록-판독 충돌(WR 충돌)
  - 트랜잭션 T1이 수정한 데이터베이스 객체 A를,
    T1이 Commit되기 전에 트랜잭션 T2가 판독(Read)하는 경우
  - 이러한 판독을 **오손 판독(Dirty Read)**라고 부름

T1	T2
R(A) W(A)	
	R(A) W(A) R(B) R(B) Commit
R(B) W(B) Commit	

순차적 트랜잭션


송금
증액(10%)
T1
Read
T2
Write
출금 50,000
입금 50,000
계좌 1
잔액 55,000
계좌 2
잔액 110,000
총 잔액 165,000
이자 계산 10% 증액
이자 계산 10% 증액

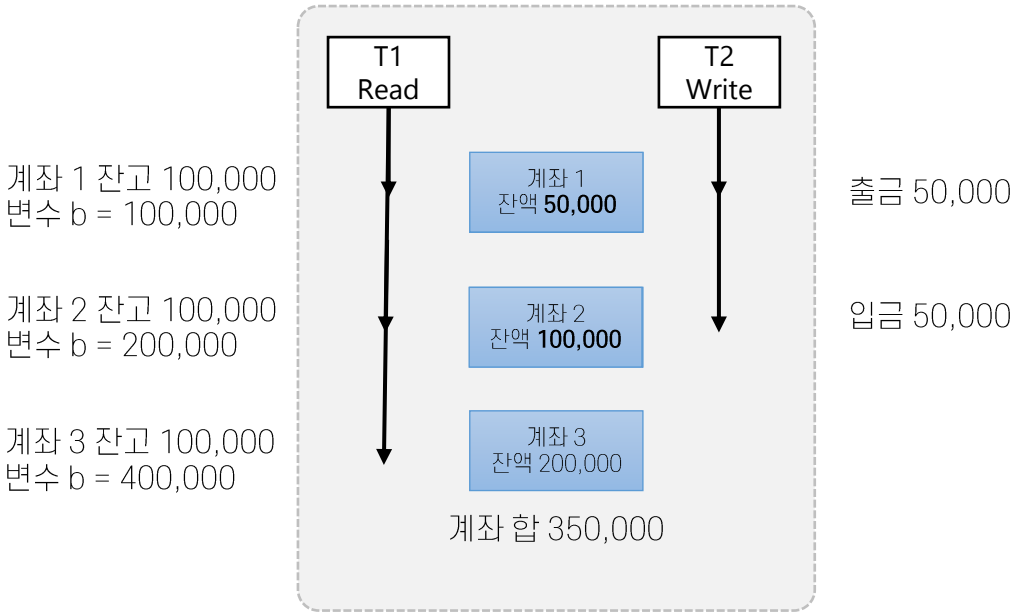
기록-판독 충돌(WR 충돌)


송금
증액(10%)
T1
Read
T2
Write
출금 50,000
계좌 1
잔액 55,000
이자 계산 10% 증액
입금 50,000
계좌 2
잔액 105,000
이자 계산 10% 증액
총 잔액 160,000

판독-기록 충돌(RW 충돌)

- 반복 불가능한 읽기를 수행하는 경우
- 판독-기록 충돌(RW 충돌)
  - T1이 객체 A의 값을 판독중에 T2가 객체 A의 값을 변경하는 경우
  - 이런 판독을 반복 불가능한 판독(Unrepeatable Read)라고 부름

판독-기록 충돌


T1
Read
T2
Write
계좌 1 잔고 100,000
변수 b = 100,000
계좌 2 잔고 100,000
변수 b = 200,000
계좌 3 잔고 100,000
변수 b = 400,000
계좌 1
잔액 50,000
계좌 2
잔액 100,000
계좌 3
잔액 200,000
계좌 합 350,000
출금 50,000
입금 50,000

기록-기록 충돌(WW 충돌)

- 미 완결된 데이터를 덮어 쓰는 경우
- 기록-기록 충돌(WW 충돌)
  - T10이 어떤 객체 A의 값을 수정하고, 진행 중인 상태에서 R2가 값을 덮어 쓰는 경우
  - 이런 기록을 맹목 기록(Blind Write)라고 부름

Lock

- Lock Mode
- Shared Lock Mode
- Exclusive Lock Mode
- Strict 2 Phase Lock

Lock Mode

- 직렬성을 보장하기 위한 방법으로, 데이터 객체들이 상호 배타적으로 액세스 되도록 함
- 한 트랜잭션이 한 데이터 객체에 액세스 했을 때 다른 트랜잭션이 해당 객체를 수정하지 못하도록 함
- 다중 트랜잭션 환경에서 데이터베이스의 일관성과 무결성을 유지하기 위한 순차적 진행을 보장
- 잠금 모드
  - Shared Lock Mode(Read Lock)
    - 트랜잭션 T_i가 데이터 객체 A에 Shared Lock을 가지고 있다면 T_i는 A를 읽을 수 있지만 갱신할 수 없음
  - Exclusive Lock Mode(Write Lock – **X로 표시**)
    - 트랜잭션 T_i가 데이터 객체 A에 Exclusive Lock을 가지고 있다면 T_i는 A를 읽을 수도, 갱신할 수도 있음

Shared Lock Mode

- Shared Lock 또는 Read Lock - **S로 표시함**
- 다른 트랜잭션에서 데이터를 읽을 수 있는 Lock Mode
  - Shared Lock은 다른 트랜잭션에 대한 읽기 시도를 허용
- 동일 데이터베이스 객체에 여러 Shared Lock이 적용될 수 있음
  - 트랜잭션 T1이 객체 A에 Shared Lock을 가지고 있으면, 트랜잭션 Ti에 대해 Shared Lock을 허용
  - 트랜잭션 T1이 객체 A에 Shared Lock을 가지고 있으면, 트랜잭션 Ti에 대해 Exclusive Lock을 허용하지 않음
- SELECT 쿼리에 적용됨

Exclusive Lock Mode

- Write Lock – X로 표시함
- 다른 트랜잭션에서 데이터를 읽을 수도, 쓸 수도 없는 잠금 모드
  - Exclusive Lock은 다른 트랜잭션의 접근을 허용하지 않음
- 트랜잭션 T1이 객체 A에 Exclusive Lock을 가지고 있을 경우, Ti는 A에서 Shared Lock을 가질 수 없음
- 트랜잭션 T1이 객체 A에 Exclusive Lock을 가지고 있을 경우, Ti는 A에서 Exclusive Lock을 가질 수 없음
- UPDATE, DELETE, INSERT 등의 갱신 쿼리에 적용됨

Strict 2 Phase Lock

- 가장 널리 사용되는 잠금 규약으로, 두 가지 규칙을 사용
  1. 트랜잭션 T가 어떤 객체를 판독(수정)하려면, 그 객체에 대해 공유(배타적) 잠금을 요청
  2. 트랜잭션이 종료될 때 가지고 있던 모든 잠금을 풀어줌
- 잠금 규약은 안전한 인터리빙만을 어용함
  - 두 트랜잭션이 동일한 객체에 접근하며, 한 트랜잭션이 객체를 수정하려 할 때, 단위 작업이 직렬 순서로 수행한 효과를 얻도록 함

잠금 기반 병행 제어

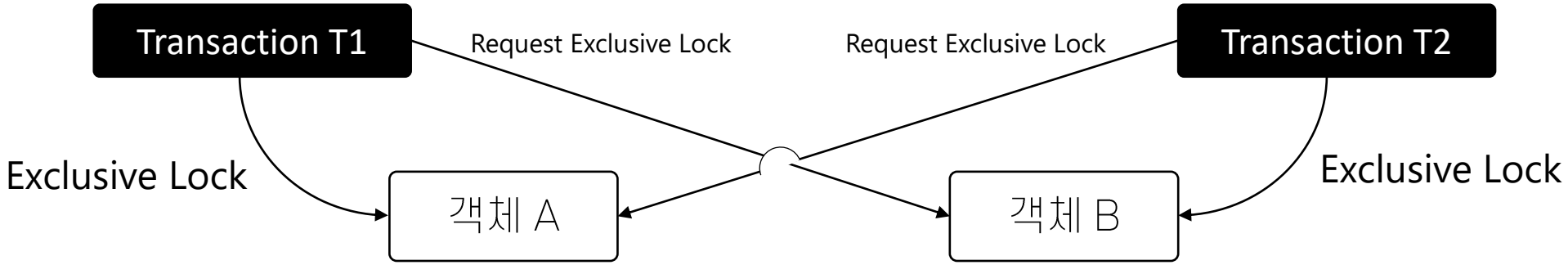
- 잠금 관리
- 교착 상태
- 교착 상태 예방
- 교착 상태 탐지

잠금 관리

- DBMS는 잠금 관리자(Lock Manager)를 제공
  - 잠금 관리자는 잠금 테이블(Lock Table)과 트랜잭션 테이블(Transaction Table)을 유지
  - 잠금 테이블 엔트리(Lock Table Entry)로 객체에 대한 잠금 정보 관리
- 잠금 요청과 잠금 해제 요청 구현
  - Shared Lock을 요청하고, 요청 큐는 비어 있으며(객체에 대한 Lock을 가진 트랜잭션이 없음), 해당 객체가 현재 Exclusive로 Lock되어 있지 않은 경우에는, Lock Manager는 이 Lock을 허가하고 해당 객체에 대한 Lock Entry를 갱신
  - Exclusive Lock을 요청하고, 현재 그 객체에 대한 Lock을 가진 트랜잭션이 없는 경우에는 Lock Manager는 이 Lock을 허가하고 Lock Table Entry를 갱신
  - 이 밖의 경우에는 요청된 Lock이 바로 허가될 수 없으며, 해당 객체에 대한 Lock Request Queue에 요청을 추가

교착상태(Deadlock)

- 두 트랜잭션이 잠금 해제를 기다리는 관계에 사이클이 생기는 경우



- 논리적으로 교착상태를 막을 수 있는 방법은 없음
  - 예방으로 프로세스의 시작을 막음
  - 탐지로 교착상태를 검출하여 프로세스를 Kill

교착상태 예방

- 트랜잭션마다 우선순위를 부여해주고 우선 순위가 낮은 트랜잭션은 우선순위가 높은 트랜잭션을 기다릴 수 없도록 하여 예방
- 우선 순위 지정에는 타임 스탬프를 사용
  - 가장 오래된 트랜잭션이 가장 높은 우선순위를 가짐
- 잠금 관리자는 두 가지 전략 중 하나를 사용
  - 트랜잭션 Ti가 잠금을 요청하고 트랜잭션 Tj가
  - Wait-Die
    Ti 입장에서 자신의 우선순위가 높으면 기다리고 그렇지 않으면 철회
  - Wound-Wait
    Ti 입장에서 자신의 우선순위가 높으면 Tj를 철회, 그렇지 않으면 자신이 기다림

교착상태 탐지

- Deadlock은 매우 드물게 발생되며, 몇 안되는 트랜잭션만 관련됨
- 발생 후 탐지해서 해결하는 것이 좋을 수 있음
  - DBMS는 주기적으로 Deadlock인지를 점검
- 잠금 관리자는 Wait-for 그래프라는 자료구조를 유지하면서 교착상태 사이클을 탐지
  - Tj의 잠금 해제를 Ti가 기다린다면 Ti에서 Tj로 가는 간선이 존재함
  - 잠금 관리자는 잠금 요청이 큐에 들어올때 마다 그래프에 간선을 추가, 허가 후 삭제

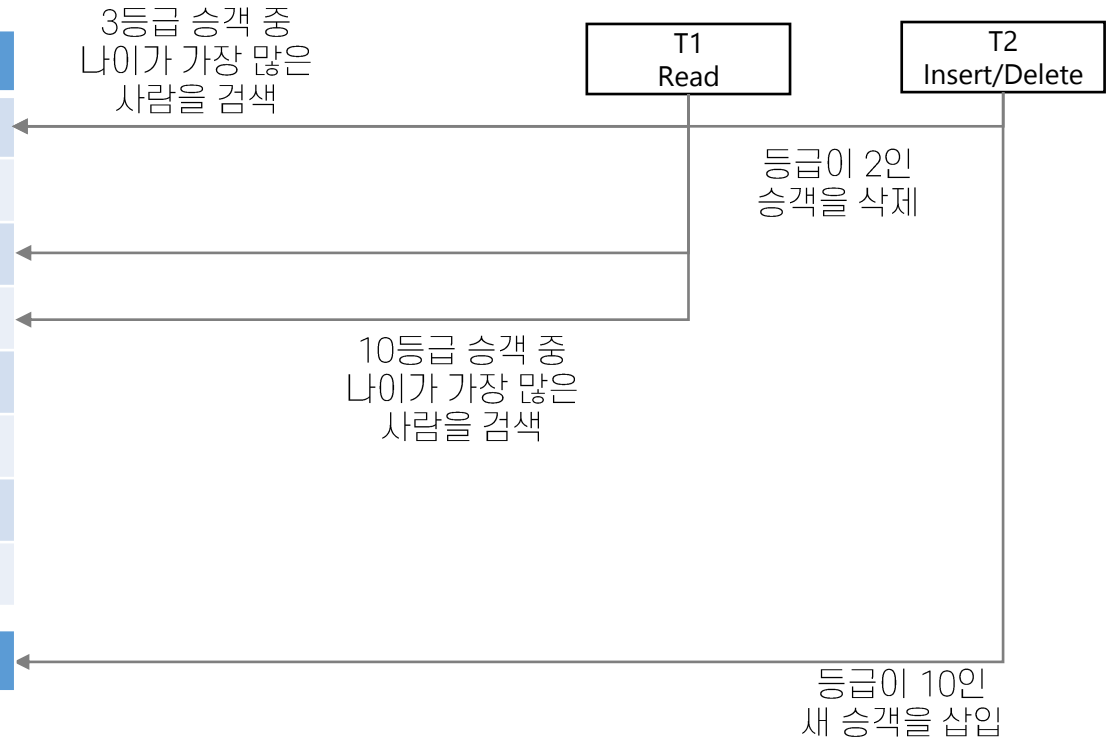
Lab 12-2 Deadlock

특수한 잠금 기법

- 동적 데이터베이스와 Phantom 문제
- 그외 잠금 기법

동적 데이터베이스와 팬텀 문제

No	Name	Grade	Age
1	홍길동	2	44
2	이순신	11	44
3	안중근	10	30
4	김영랑	10	54
5	김소월	9	45
6	윤동주	11	26
7	천상병	9	55
8	조지훈	9	43
10	박목월	10	62



- 트랜잭션 T1의 동작에 따라,
  - 인덱스가 없어서 파일에 있는 모든 페이지를 모두 살펴보아야 한다면, T1은 기존 페이지들을 모두 잠금하고 파일에 새 페이지가 추가될 수 없도록 만들어야 함
  - 인덱스가 있다면 등급이 1인 데이터 엔트리에 잠금을 할 수 있음. T2는 T1의 잠금을 해제를 기다려야 함. 이를 인덱스 잠금(Index Locking)이라고 함

그외 잠금 기법

- 트리 구조 인덱스의 경우, 인덱스 구조를 무시하고 각 페이지를 하나의 데이터 객체로 간주해서 2PL 방식을 사용
- 다단위 잠금
  - 의도 공용 잠금 (Intention Shared Lock: IS)
    - 트랜잭션이 테이블의 개별 Row의 Shared Lock을 얻음
  - 의도 전용 잠금(Intention Exclusive Lock: IX)
    - 트랜잭션이 테이블의 개별 Row의 Exclusive Lock을 얻음

SQL의 트랜잭션 지원

- 트랜잭션 특성
- Isolation Level

트랜잭션 특성

- 트랜잭션은 세 가지 특성을 가짐
  - Access Mode: UNSPECIFIED, READ ONLY, READ/WRITE
  - Diagnostics-size: 기록할 수 있는 오류 조건의 수
  - Isolation Level: 동시에 실행되고 있는 다른 트랜잭션의 작업들에 대해 주어진 트랜잭션이 노출되는 정도를 제어

Isolation Level

- SERIALIZABLE (Level 3)
  - 트랜잭션 T는 완결된 트랜잭션들이 만들어 놓은 변경만 판독
  - T가 판독하거나 기록한 값은 T가 종료된 후에 변경할 수 있음
- REPEATABLE READ (Level 2)
  - 트랜잭션 T는 완결된 트랜잭션들이 만들어 놓은 변경만 판독
  - T가 판독하거나 기록한 값은 T가 종료된 후에 변경할 수 있으나, Index Locking은 하지 않음
- READ COMMITTED (Level 1)
  - 트랜잭션 T는 완결된 트랜잭션들이 만들어 놓은 변경만 판독
  - T가 판독하거나 기록한 값은 T가 종료된 후에 변경할 수 있으나, T가 판독한 Row는 다른 트랜잭션이 접근 가능
- READ UNCOMITTED (Level 0)
  - 트랜잭션 T는 진행중인 트랜잭션이 만들어 놓은 변경도 판독할 수 있음

ISOLATION LEVEL

- SERIALIZABLE (Level 3)
  - 다른 트랜잭션이 읽고 있는 데이터는 다른 트랜잭션에서 수정/삭제/삽입 불가능
  - 모든 이상현상 방지
- REPEATABLE READ (Level 2)
  - 다른 트랙잭션에서 읽고 있는 데이터는 다른 트랜잭션에서 수정/삭제 불가능
  - Phantom Read 발생
- READ COMMITTED (Level 1)
  - Commit 된 데이터만 다른 트랜잭션에서 읽을 수 있음
  - Non-Repeatable Read, Phantom Read 발생
- READ UNCOMITTED (Level 0)
  - Commit 되지 않은 데이터를 다른 트랜잭션에서 읽을 수 있음
  - Dirty Read, Non-Repeatable Read, Phantom Read 발생

Isolation Level

Level	Dirty Read	Non-Repeatable Read	Phantom
READ UNCOMMITTED	가능성 있음	가능성 있음	가능성 있음
READ COMMITTED	불가	가능성 있음	가능성 있음
REPEATABLE READ	불가	불가	가능성 있음
SERIALIZABLE	불가	불가	불가

Lab 12-3 트랜잭션 격리 수준과 잠금