트렌젝션

트렌젝션의 특징

* 데이터베이스 응용의 오작동
  + 예 : 전체 사원의 급여를 6%인상

**UPDATE**  EMPLOYEE

**SET** SALARY = SALARY \* 1.06;

업데이트 도중에 서버가 다운된다면?

320번째 사원까지 급여를 6% 인상

* 컴퓨터에서 이런 작동을 하는 도중에 다운되면 프로세스작업이기 때문에 그냥 중단된다.

(컴퓨터가 이 작업의 책임을 져주지 않는다.)

하지만 DB는 이러한 문제를 사람에게 탓하지 않고 시스템을 탓하게 한다.

* + 예 : 계좌 이체

UPDATE CUSTOMER

SET BALANCE = BALANCE – 100000

WHERE CUST\_NAME = ‘정미림’;

프로세스의 개념을 발전시켜서 나온 개념이 ‘트렌젝션’이다.

트렌젝션의 ACID 특성 (시험에 나옴)

원자성(Atomicity)

일관성(Consistency)

고립성(Isolation)

지속성(Durability)

* 원자성

트렌젝션을 쪼갤 수 없다!

모두 수행되거나 혹은 전혀 수행되지 않거나…!

데이터 베이스의 상태(스키마 상태)

수행 상태와 수행되지 않는 상태가 존재를 하지만 애매한 상태는 나오면 안된다!

* 일관성

어떤 트렉젝션이 수행되기 전에 데이터베이스가 일관된 상태를 가졌다면 트렌젝션이 수행된 후에 데이터베이스는 또 다른 일관된 상태를 가진다.

일관적 상태의 데이터베이스 =====트랜잭션=====🡺 새로운 일관적 상태의 데이터베이스

데이터베이스의 값(상태)는 변화 하였지만 무결성은 지켜져야 한다.

중요한 이유? (숨겨진 의미)

1. 트랜젝션이 시작되기 전에 일관된 상태였다면 종료된 후에도 일관된 상태여야 하는데.. 만약에 시작되기 전에 무결성이 아니라면? => 책임질 필요가 없어! 보장할 수 없다.
2. 트랜젝션이 시작되기 전에 일관된 상태였는데 트랜젝션 도중에 무결성이 지켜지지 않는다면? => 책임지지 않는다. 모른다. 보장할 수 없다.

일시적으로 일관되지 않은 상태일 수 있다.

하지만 정상적으로 트랜젝션이 된다면 일관성을 유지시킬 수 있다.

정상적으로 수행되도록 하기 위해서는 원자성이 필요하다. 다 끝날 때까지 기다린다. 순서대로 수행시킨다.

원자성으로 인해 무조건 모두 트랜젝션 된 상태이거나 그렇지 않은 상태로 간다.

멀티태스킹 (타임 쉐어링) (Context Switching) 뭔가 동시에 실행시켜야 느리지 않겠다!

* 고립성

하나씩 실행하면 최상이지만 이게 문제가 있으니 동시에 수행하도록 한다. 하지만 동시에 수행을 하더라도 혼자서 수행된 것 이랑 같게 보장되어야 한다.

고립화 되어 있는 것처럼 수행되어야 한다.

여러 트랜젝션이 동시에 수행되더라도 마치 혼자 수행한 것과 같아야 함

직렬 스케쥴 : 트랜젝션들을 하나씩 순서대로 수행하는 스케쥴

T1 -> T2 T2 -> T1

비직렬 스케쥴 : 트랜젝션들이 순서대로 수행되지 않는 스케쥴

T11 -> T21 -> T12 -> T22 …..

T11 -> T21 -> T31 -> T22 ….

무수히 많다!

고립성 => 비직렬 스케쥴로 수행하더라도 마치 직렬 스케쥴로 수행하는 것과 같아야 함

직렬 스케쥴 < 비직렬 스케쥴

직렬 가능성이 있는 비직렬 스케쥴이라면 트랜젝션을 비직렬로 하더라도 무결성이 지켜진다.

그럼 이 스케쥴을 어떻게 찾아야 하는가? => 동시성 제어

* 지속성

트랜젝션이 완료되면 \*(무슨일이 있어도)\* 그 결과를 유지해야 한다.

원자성 => DBMS 회복 기능

일관성 => DBMS 동시성 제어

고립성 => DBMS 동시성 제어

지속성 => 무결성 제약 조건

* Commit과 Rollback(Abort)

Commit => 완료 (성공적인 종료)

Rollback => 철회(비성공적인 종료)

* 진행하다가 무결성이 지켜지지 않을 조짐이 보이거나 이미 있으면 다시 되돌린다.
* 동시성 제어
* 직렬 스케쥴

여러 트랜젝션들의 집합을 한 번에 트랜젝션씩 차례대로 수행함

* 비직렬 스케쥴

여러 트랜젝션들을 동시에 수행함

* 직렬 가능

비직렬 스케쥴의 결과가 어떤 직렬 스케쥴의 수행 결과와 동등함

* 동시성 제어 오류

갱신 손실(lost update) : 업데이트 한 것을 잃어 버렸다. 트랜젝션이 오버라이트 해서

오손 데이터 읽기(dirty read) : 완료되지 않은 트렌젝션이 갱신한 데이터를 읽는것

반복할 수 없는 읽기(unrepeatable read) : 한 트랜젝션이 동일한 데이터를 두 번 읽을 때 서로 다른 값을 읽는 것

* + 갱신 손실 : 비직렬 스케쥴링시 발생하는 원치않는 오버라이트로 인한 무결성 오류
  + 오손데이터 읽기 : 변화를 철회 하려고 하는데 다른 곳에서 변화된 결과를 읽을 때
  + 반복할 수 없는 읽기 : 정보를 읽는 것을 두 번 이상 할 때 중간에서 변화가 된 경우
* 고립화 수준(Isolation Level)

한 트랜젝션이 다른 트랜젝션과 고립되는 정도

* 고립 수준이 낮으면 동시성은 높아지지만 데이터의 정확성은 떨어짐
* 고립 수준이 높으면 데이터가 정확해지지만 동시성이 저하됨

(정의한 방식대로 해라!)

* 고립화 수준의 종류