

데이터베이스

- 순서 : Ch9(트랜잭션)
- 학기 : 2018학년도 2학기
- 학과 : 가천대학교 컴퓨터공학과 2학년
- 교수 : 박양재

데이터베이스

- 목차

9.1 트랜잭션 개요

9.2 동시성 제어

9.3 회복

9.4 PL/SQL의 트랜잭션

9장. 트랜잭션

□ 트랜잭션(transaction)

- ✓ 데이터베이스는 기업의 귀중한 공유자원
- ✓ DBMS는 동시에 여러 사용자의 요청을 처리
- ✓ 항공기 예약, 은행, 신용 카드 처리, 대형 할인점 등에서는 대규모 데이터베이스를 수백, 수천 명 이상의 사용자들이 동시에 접근함
- ✓ 많은 사용자들이 동시에 데이터베이스의 서로 다른 부분 또는 동일한 부분을 접근하면서 데이터베이스를 사용함(동일 릴레이션 동일 튜플까지)
- ✓ DBMS는 일반적인 프로그램들에 비하여 중앙처리장치보다 입출력장치를 빈번히 접근
- ✓ 데이터베이스를 접근하려는 사용자 수가 다수 있을 때 한 사람씩 접근하도록 한다면 입출력장치보다 중앙처리장치의 속도가 매우 빠르므로 중앙처리장치가 쉬는 경우가 많이 생긴다. 따라서 운영체제가 다수의 프로그램을 동시에 수행하듯이 여러 트랜잭션을 동시에 수행해야 한다.

9장. 트랜잭션(계속)

□ 트랜잭션(transaction)(계속)

✓ 동시성 제어(concurrency control)

- 동시에 수행되는 트랜잭션들이 데이터베이스에 미치는 영향은 이들을 순차적으로 수행하였을 때 데이터베이스에 미치는 영향과 같도록 보장
- 다수 사용자가 데이터베이스를 동시에 접근하도록 허용하면서 데이터베이스의 일관성을 유지함
- DBMS의 동시성 제어 모듈 기능 : 동시에 수행되는 트랜잭션들 간의 간섭(상호작용)을 제어하여 데이터베이스의 일관성을 보장

✓ 회복(recovery)

- 데이터베이스를 갱신하는 도중에 시스템이 고장 나도 데이터베이스의 일관성을 유지함
- 동시성 제어 모듈 + 회복모듈 = 트랜잭션 관리 모듈.
- 오라클이나 MS-SQL Server와 같은 다수 사용자용 DBMS에는 트랜잭션 관리기능 필수이나 ACCESS처럼 PC용 단일사용자 DBMS에는 제공되지 않는다.

9.1 트랜잭션 개요

❑ 데이터베이스 시스템 환경에서 흔히 볼 수 있는 몇 가지 응용의 예

예 1 : 전체 사원의 급여를 6% 인상

사원이 500명 재직하고 있는 회사에서 모든 사원의 급여를 6% 인상하는 연산을 데이터베이스의 EMPLOYEE 릴레이션에서 수행한다.

```
UPDATE    EMPLOYEE
SET        SALARY = SALARY * 1.06;
```



[그림 9.1] 데이터베이스를 갱신하는 중에 컴퓨터 시스템의 다운

- ✓ 이때 500명 **전원의 급여가 수정되거나 한 명의 급여도 갱신되지 않도록 DBMS가 보장해야 함**
- ✓ 그림 9.1과 같이 320번째 사원까지 수정한 상태에서 컴퓨터 시스템이 다운된 후에 재기동되었을 때 DBMS는 어떻게 대응해야 하는가?
- ✓ DBMS가 추가로 정보를 유지하지 않는다면 DBMS가 재기동된 후에 어느 직원의 투플까지 수정되었는가를 알 수 없음 → **로그(log)** 유지

9.1 트랜잭션 개요(계속)

❑ 데이터베이스 시스템 환경에서 흔히 볼 수 있는 몇 가지 응용의 예(계속)

예 2 : 계좌 이체

은행 고객은 자신의 계좌에서 다른 계좌로 송금할 수 있다. 정미림은 자신의 계좌에서 100,000원을 인출하여 안명석의 계좌로 이체하려고 한다. 고객들의 계좌 정보가 CUSTOMER 릴레이션에 들어 있다.

```
UPDATE    CUSTOMER
SET        BALANCE = BALANCE - 100000
WHERE     CUST_NAME = '정미림';
```

```
UPDATE    CUSTOMER
SET        BALANCE = BALANCE + 100000
WHERE     CUST_NAME = '안명석';
```

9.1 트랜잭션 개요(계속)

- ✓ 두 개의 UPDATE문을 사용하여, 하나의 UPDATE문에서는 정미림의 잔액을 100,000원 감소시키고, 또 다른 UPDATE문에서는 안명석의 잔액을 100,000원 증가시킴
- ✓ 첫 번째 UPDATE문을 수행한 후에 두 번째 UPDATE문을 수행하기 전에 컴퓨터 시스템이 다운되면 재기동한 후에 DBMS가 어떻게 대응해야 하는가?
 - 정미림=-200,000, 안명석=+100,000
 - 재시동 후 DBMS가 아무 일도 하지 않았을 경우 : 정미림=-100000, 안명석 = 그대로
- ✓ 위의 두 개의 UPDATE문은 둘 다 완전하게 수행되거나 한 UPDATE 문도 수행되어서는 안되도록, 즉 하나의 트랜잭션(단위)처럼 DBMS가 보장해야 함
- ✓ 기본적으로 각각의 SQL문이 하나의 트랜잭션으로 취급됨
- ✓ 두 개 이상의 SQL문들을 하나의 트랜잭션으로 취급하려면 사용자가 이를 명시적으로 표시해야 함
- ✓ 성공적인 계좌이체 트랜잭션의 결과는 데이터베이스에 지속적으로 반영되어야 한다.

9.1 트랜잭션 개요(계속)

□ 트랜잭션의 특성(ACID 특성)

✓ 원자성(Atomicity)

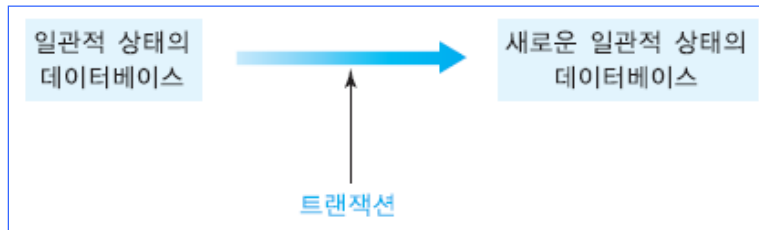
- 한 트랜잭션 내의 모든 연산들이 완전히 수행되거나 전혀 수행되지 않음(all or nothing)을 의미
- DBMS의 회복 모듈은 시스템이 다운되는 경우에, 부분적으로 데이터베이스를 갱신한 트랜잭션의 영향을 취소함으로써 트랜잭션의 원자성을 보장함
- 완료된 트랜잭션이 갱신한 사항도 트랜잭션이 취소 되면 수행 전의 상태로 되돌릴 수 있는 원자성을 보장함.
- 완료되지 않은 트랜잭션은 수행 전 상태로 되돌린다.(예: 계좌이체와 급여 인상)

9.1 트랜잭션 개요(계속)

❑ 트랜잭션의 특성(ACID 특성)(계속)

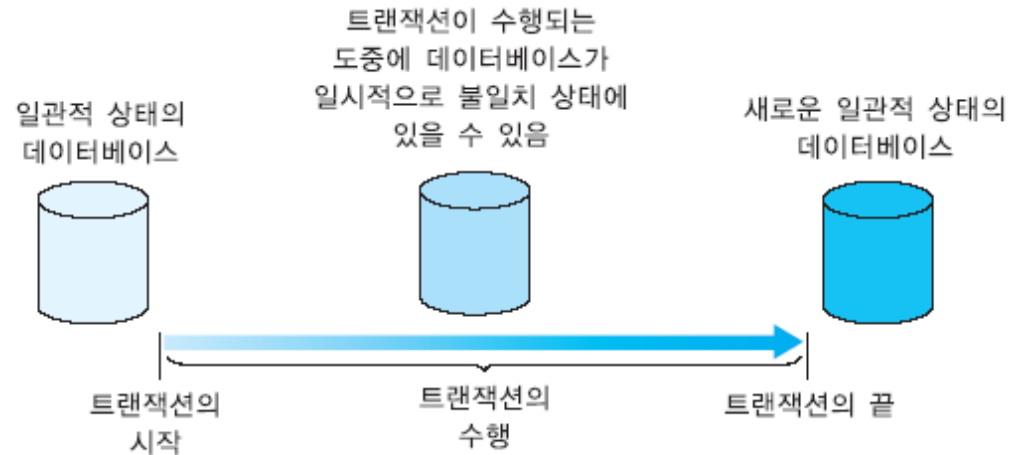
✓ 일관성(Consistency)

- 어떤 트랜잭션이 수행되기 전에 데이터베이스가 일관된 상태를 가졌다면 트랜잭션이 수행된 후에 데이터베이스는 또 다른 일관된 상태를 가짐



- 트랜잭션이 수행되는 도중에는 데이터베이스가 일시적으로 일관된 상태를 갖지 않을 수 있음
- 일관성 유지 : 트랜잭션을 작성하는 사용자(응용프로그래머)와 무결성 제약조건을 유지하는 DBMS
- DBMS는 CREATE TABLE문에 정의된 무결성 제약조건을 유지하나 데이터의 의미는 알지 못한다(DBMS는 은행계좌의 잔액의 이자가 어떻게 계산되는지 알지 못한다)

9.1 트랜잭션 개요(계속)



[그림 9.2] 데이터베이스의 일시적인 불일치 상태

9.1 트랜잭션 개요(계속)

□ 트랜잭션의 특성(ACID 특성)(계속)

✓ 고립성(Isolation)

- 한 트랜잭션이 데이터를 갱신하는 동안 이 트랜잭션이 완료되기 전에는 갱신 중인 데이터를 다른 트랜잭션들이 접근하지 못하도록 해야 함
- 다수의 트랜잭션들이 동시에 수행되더라도 그 결과는 어떤 순서에 따라 트랜잭션들을 하나씩 차례대로 수행한 결과와 같아야 함
- DBMS의 동시성 제어 모듈이 트랜잭션의 고립성을 보장함
- DBMS는 응용들의 요구사항에 따라 다양한 고립 수준(isolation level)을 제공함
- 예: 은행계좌에서 정미림 계좌에서 100,000원 빼고, 안명석 계좌에 100,000원을 증가 시키는 트랜잭션을 수행하는 동안 다른 트랜잭션이 정미림과 안명석 계좌에 접근하지 못하도록 해야 한다. 만일 접근하면 데이터 불일치 발생

9.1 트랜잭션 개요(계속)

□ 트랜잭션의 특성(ACID 특성)(계속)

✓ 지속성(Durability)

- 일단 한 트랜잭션이 완료되면 이 트랜잭션이 갱신한 것은 그 후에 시스템에 고장이 발생하더라도 손실되지 않음
- 완료된 트랜잭션의 효과는 시스템이 고장난 경우에도 데이터베이스에 반영됨
- DBMS의 회복 모듈은 시스템이 다운되는 경우에도 트랜잭션의 지속성을 보장함



[그림 9.3] 트랜잭션의 네 가지 특성과 DBMS의 기능과의 관계

9.1 트랜잭션 개요(계속)

□ 트랜잭션의 사용

- ✓ 대부분의 DBMS들은 데이터를 갱신하는 명령들을 트랜잭션으로 관리
- ✓ 데이터베이스 연산들은 사용자(프로그래머)가 정의하는 트랜잭션으로 묶는 것이 편리
- ✓ 하나의 응용 프로그램은 단일 트랜잭션이나 트랜잭션들의 그룹으로 구성
- ✓ 데이터베이스를 수정하는 임의의 SQL문으로 시작

□ 트랜잭션의 완료(commit)

- ✓ 트랜잭션에서 변경하려는 내용이 데이터베이스에 완전하게 반영됨
- ✓ SQL 구문상으로 COMMIT WORK

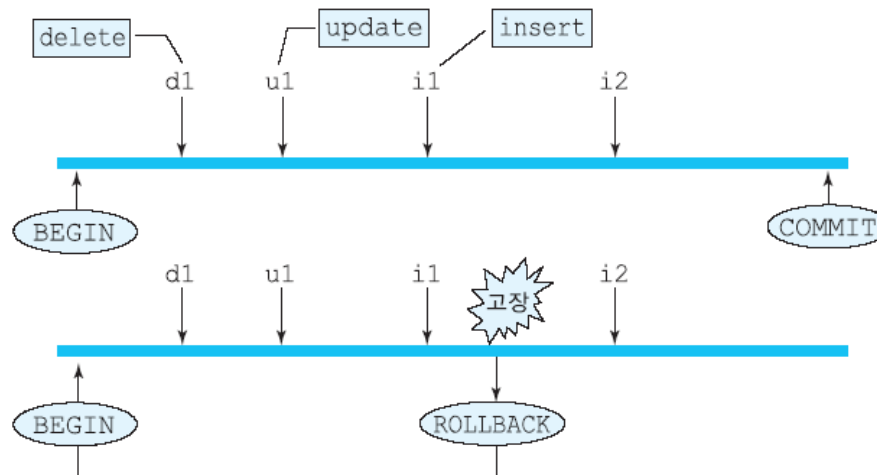
□ 트랜잭션의 철회(abort)

- ✓ 트랜잭션에서 변경하려는 내용이 데이터베이스에 일부만 반영된 경우에는 원자성을 보장하기 위해서, 트랜잭션이 갱신한 사항을 트랜잭션이 수행되기 전의 상태로 되돌림
- ✓ SQL 구문상으로 ROLLBACK WORK

9.1 트랜잭션 개요(계속)

〈표 9.1〉 COMMIT과 ROLLBACK의 비교

연산	COMMIT	ROLLBACK
의미	완료(성공적인 종료)	철회(비성공적인 종료)
DBMS의 트랜잭션 관리 모듈에 게 알리는 사항	<ul style="list-style-type: none"> 트랜잭션이 성공적으로 끝났음 데이터베이스는 새로운 일관된 상태를 가짐 트랜잭션이 수행한 갱신을 데이터베이스에 반영해야 함 	<ul style="list-style-type: none"> 트랜잭션의 일부를 성공적으로 끝내지 못했음 데이터베이스가 불일치 상태를 가질 수 있음 트랜잭션이 수행한 갱신이 데이터베이스에 일부 반영되었다면 취소해야 함



[그림 9.4] COMMIT과 ROLLBACK

9.1 트랜잭션 개요(계속)

□ 트랜잭션이 성공하지 못하는 원인

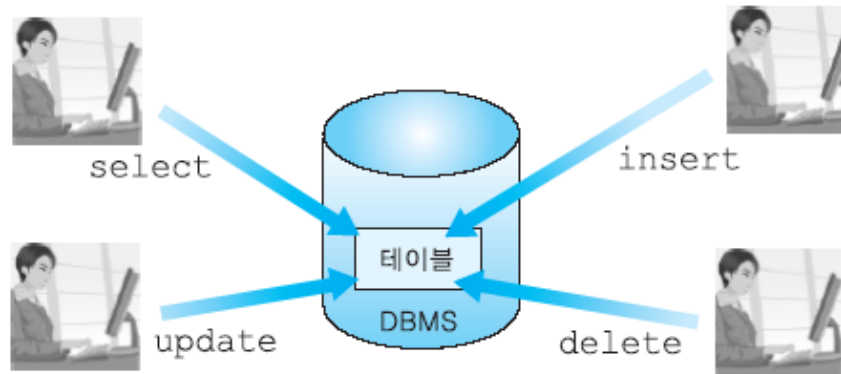
- ✓ 시스템(사이트) 고장
 - 중앙 처리 장치, 주기억 장치, 전원 공급 장치 등이 고장남
 - 주기억장치 내용은 없어지지만 보조기억장치의 내용은 안전
- ✓ 트랜잭션 고장
 - 트랜잭션 고장은 트랜잭션이 수행되는 도중에 철회됨
 - 트랜잭션 내에서 어떤 값을 0으로 나누거나 오버플로우가 발생해도 철회됨
- ✓ 매체 고장
 - 디스크 헤드, 디스크 컨트롤러 등이 고장 나서 보조 기억 장치의 전부 또는 일부 내용이 지워짐(대책 :백업)
- ✓ 통신 고장
- ✓ 자연적 재해
- ✓ 부주의 또는 고의적인 고장(해킹)

9.2 동시성 제어

□ 동시성 제어

- ✓ 대부분의 DBMS들은 다수 사용자용
- ✓ 여러 사용자들이 동시에 동일한 테이블을 접근하기도 함
- ✓ DBMS의 성능을 높이기 위해 여러 사용자의 질의나 프로그램들을 동시에 수행하는 것이 필수적
- ✓ 동시에 수행되는 다수의 트랜잭션들이 존재할 때 고립성을 유지하는 것
- ✓ DBMS가 아무런 조치없이 트랜잭션들을 동시에 수행 시키면 여러가지 문제가 발생할 수 있다.
- ✓ 동시성 제어 기법은 여러 사용자들이 다수의 트랜잭션들을 동시에 수행하는 환경에서 부정확한 결과를 생성할 수 있는, 트랜잭션들 간의 간섭이 생기지 않도록 함
- ✓ 트랜잭션을 수행하는 각 사용자가 혼자서 데이터베이스를 접근하는 것처럼 생각할 수 있도록 해야 한다.

9.2 동시성 제어



[그림 9.5] 다수 사용자의 동시 접근

❑ 직렬 스케줄(serial schedule)

- ✓ 여러 트랜잭션들의 집합을 한 번에 한 트랜잭션씩 차례대로 수행함
- ✓ 데이터베이스 일관성 유지

❑ 비직렬 스케줄(non-serial schedule)

- ✓ 여러 트랜잭션들을 동시에 수행함

❑ 직렬가능(serializable)

- ✓ 비직렬 스케줄의 결과가 어떤 직렬 스케줄의 수행 결과와 동등함

9.2 동시성 제어(계속)

□ 데이터베이스 연산

- ✓ 응용 프로그램에서 접근하려는 데이터베이스 항목을 포함하고 있는 디스크 블록을 주기억 장치와 디스크 간에 이동하는 연산은 **Input(X)**와 **output(X)**이다.
- ✓ 주기억 장치의 버퍼와 응용 간에 데이터베이스 이동하는 연산 **read_item(X)**와 **write_item(X)**
- ✓ **Input(X)** 연산 : 데이터베이스 항목 X를 포함하고 있는 블록을 주기억 장치의 버퍼로 읽어들이
- ✓ **Output(X)** 연산 : 데이터베이스 항목 X를 포함하고 있는 블록을 디스크에 기록함
- ✓ **read_item(X)** 연산 : 주기억 장치 버퍼에서 데이터베이스 항목 X의 값을 프로그램 변수 X로 복사함
- ✓ **write_item(X)** 연산 : 프로그램 변수 X의 값을 주기억 장치 내의 데이터베이스 항목 X에 기록함

9.2 동시성 제어(계속)

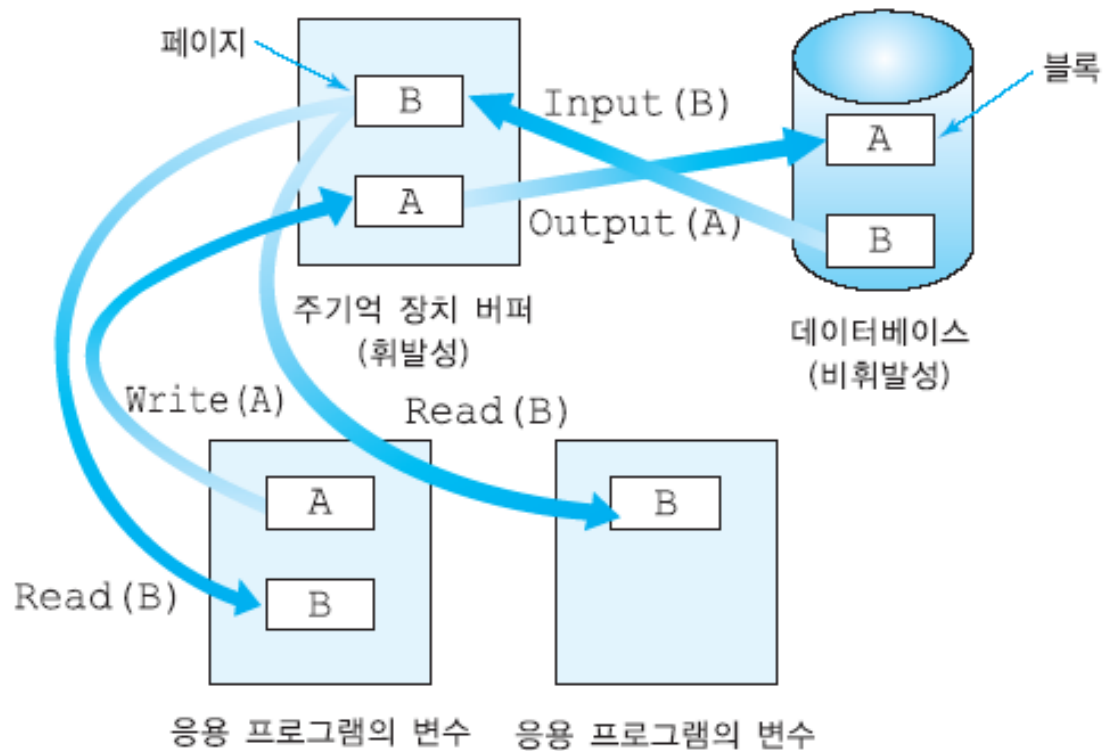


그림 [9.6] 디스크, 주기억장치 버퍼, 응용 프로그램변수간의 값 이동

9.2 동시성 제어(계속)

- 동시성 제어를 하지 않고 다수의 트랜잭션을 동시에 수행할 때 생길 수 있는 문제
 - ✓ 갱신 손실(lost update) : 수행 중인 트랜잭션이 갱신한 내용을 다른 트랜잭션이 덮어 씌우므로써 갱신이 무효가 되는 것
 - ✓ 오손 데이터 읽기(dirty read): 완료되지 않은 트랜잭션이 갱신한 데이터를 읽는 것
 - ✓ 반복할 수 없는 읽기(unrepeatable read): 한 트랜잭션이 동일한 데이터를 두 번 읽을 때 서로 다른 값을 읽는 것

9.2 동시성 제어(계속)

□ 로킹(locking) 기법

- ✓ 트랜잭션들이 동일한 데이터에 동시에 접근하지 못하도록 lock과 unlock연산으로 제어한다.
- ✓ 로킹단위가 커질수록 병행성이 낮아지지만 제어가 쉽고, 로킹단위가 작아질수록 제어가 어렵지만 병행성은 높아진다.
 - lock 연산 : 트랜잭션이 데이터에 대한 독점권을 요청하는 연산
 - unlock연산 : 트랜잭션이 데이터에 대한 독점권을 반환하는 연산

□ 2단계 로킹 규약

- 트랜잭션이 lock과 unlock연산을 확장 단계와 축소단계로 나누어 수행한다. **모든 트랜잭션이 2단계 로킹 규약을 준수하면 해당 스케줄은 직렬가능함을 보장한다.**
- 확장단계: 트랜잭션이 lock연산만 실행할 수 있고, unlock연산을 실행할 수 없는 단계
- 축소단계: 트랜잭션이 unlock연산만 실행할 수 있고, lock연산은 실행할 수 없는 단계

9.2 동시성 제어(계속)

□ 로킹(locking)

- ✓ 데이터 항목을 로킹하는 개념은 동시에 수행되는 트랜잭션들의 동시성을 제어하기 위해서 가장 널리 사용되는 기법
- ✓ **로크(lock)**는 동일한 데이터 항목에 대한 여러 트랜잭션들의 동시 접근을 조정하기 위해서 사용
- ✓ 데이터베이스 내의 각 데이터 항목과 연관된 하나의 변수(모든 데이터항목마다 존재)
- ✓ 각 트랜잭션이 수행을 시작하여 데이터 항목을 접근할 때마다 요청한 로크에 관한 정보는 **로크 테이블(lock table)** 등에 유지됨
- ✓ 트랜잭션에서 **갱신을 목적으로** 데이터 항목을 접근할 때는 **독점 로크(X-lock, exclusive lock)**를 요청함
- ✓ 트랜잭션에서 **읽을 목적으로** 데이터 항목을 접근할 때는 **공유 로크(S-lock, Shared lock)**를 요청함
- ✓ 트랜잭션이 데이터 항목에 대한 접근을 끝낸 후에 로크를 **해제(unlock)**함

9.2 동시성 제어(계속)

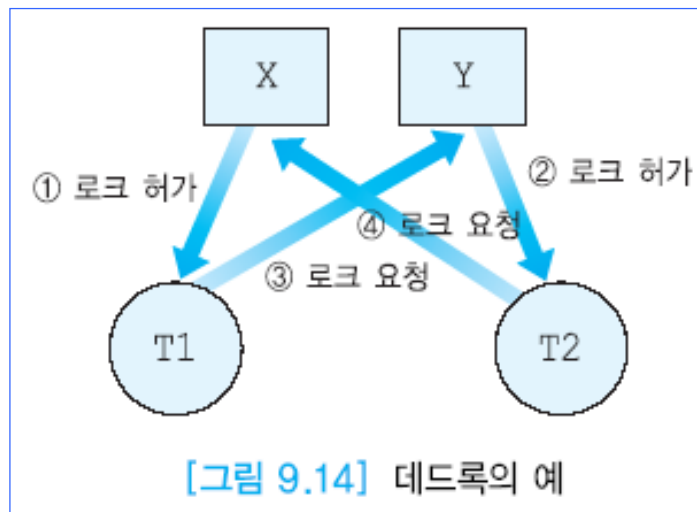
❑ 데드록(deadlock)

- ✓ 2단계 로킹 프로토콜에서 동시성 제어를하면 데이터베이스의 일관성을 보장 할 수 있다.(직렬가능)
- ✓ 2단계 로킹 프로토콜에서는 데드록이 발생할 수 있음
- ✓ 데드록은 두 개 이상의 트랜잭션들이 서로 상대방이 보유하고 있는 로크를 요청하면서 기다리고 있는 상태를 말함
- ✓ 데드록을 해결하기 위해서는 데드록을 방지하는 기법이나, 데드록을 탐지하고 희생자를 선정하여 데드록을 푸는 기법 등을 사용함

9.2 동시성 제어(계속)

□ 데드록(계속)

- ① T1이 X에 대해 **독점 로크**를 요청하여 허가받음
- ② T2이 Y에 대해 **독점 로크**를 요청하여 허가받음
- ③ T1이 Y에 대해 **공유 로크나 독점 로크**를 요청하면 로크가 해제될 때까지 기다리게 됨
- ④ T2가 X에 대해 **공유 로크나 독점 로크**를 요청하면 로크가 해제될 때까지 기다리게 됨



9.2 동시성 제어(계속)

□ 다중 로크 단위(multiple granularity)

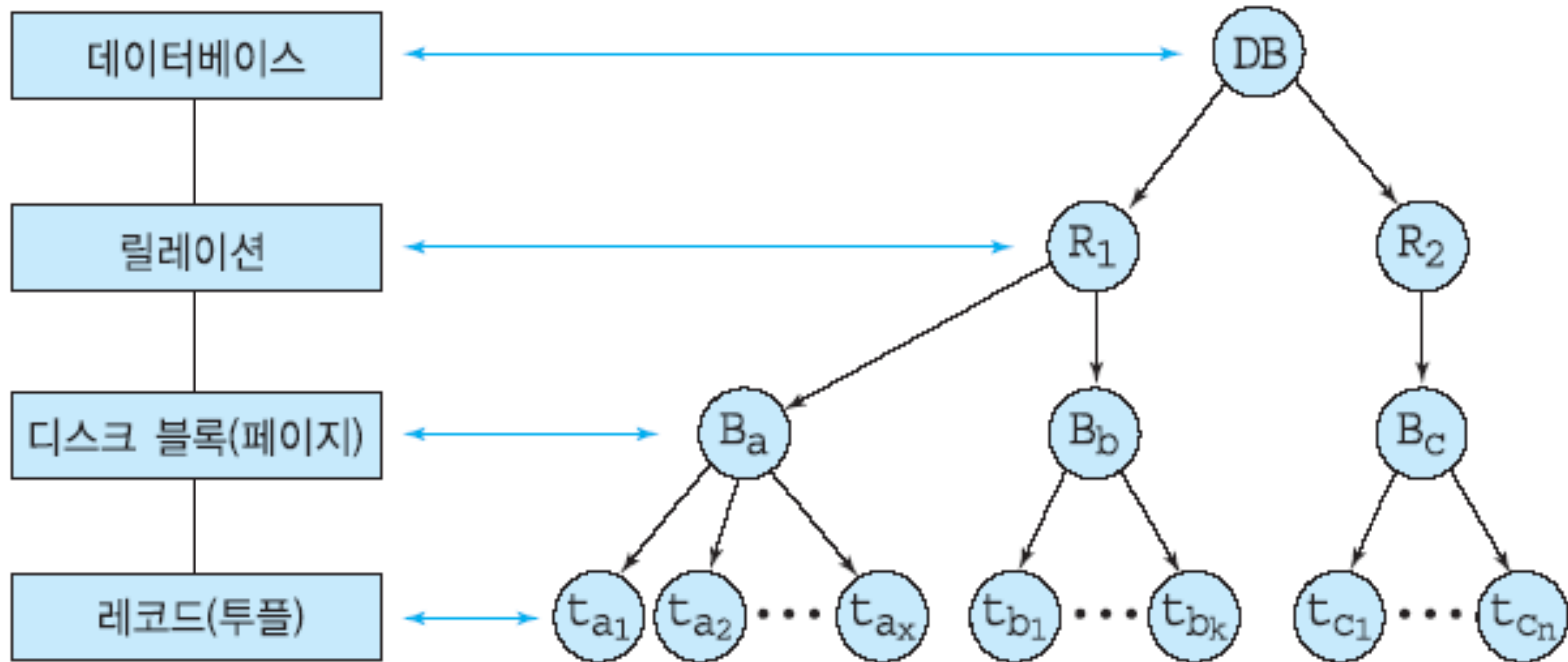
- ✓ 대부분의 트랜잭션들이 소수의 튜플들을 접근하는 데이터베이스 응용에서는 튜플 단위로 로크를 해도 로크 테이블을 다루는 시간이 오래 걸리지 않음
- ✓ 트랜잭션들이 많은 튜플을 접근하는 데이터베이스 응용에서 튜플 단위로만 로크를 한다면 로크 테이블에서 로크 충돌을 검사하고, 로크 정보를 기록하는 시간이 오래 걸림
- ✓ 트랜잭션이 접근하는 튜플의 수에 따라 로크를 하는 데이터 항목의 단위를 구분하는 것이 필요함
- ✓ 한 트랜잭션에서 로크 할 수 있는 데이터 항목이 두 가지 이상 있으면 다중 로크 단위라고 말함
- ✓ 데이터베이스에서 로크 할 수 있는 단위로는 데이터베이스, 릴레이션, 디스크 블록, 튜플 등

9.2 동시성 제어(계속)

□ 다중 락 단위(계속)

- ✓ 최적의 락 단위는 트랜잭션에서 접근하는 튜플 수에 의존
- ✓ 적은 수의 튜플에 접근하는 트랜잭션에서는 튜플을 락 단위로하고, 한 릴레이션에 속하는 대부분의 튜플들을 접근하는 트랜잭션에 대해서는 릴레이션 단위로 락
- ✓ 일반적으로 DBMS는 각 트랜잭션에서 접근하는 튜플 수에 따라 자동적으로 락 단위를 조정함
- ✓ 락 단위가 작을수록 로킹에 따른 오버헤드가 증가함
 - 트랜잭션이 각 튜플에 대하여 락할 때마다 락 테이블에 이에 대한 정보를 입력하고, 트랜잭션이 끝날 때는 락 테이블에서 이 트랜잭션에 관한 모든 락정보를 삭제해야 한다.
- ✓ 락 단위가 작을수록 동시성의 정도는 증가함

9.2 동시성 제어(계속)



[그림 9.15] 다중 로크 단위

✓로크 단위가 작을수록 동시성의 정도는 증가함

9.2 동시성 제어(계속)

❑ 팬텀 문제(phantom problem :유령 문제)

- ✓ 두 개의 트랜잭션 T1과 T2가 그림 4.8의 EMPLOYEE 릴레이션에 대해서 아래와 같은 순서대로 수행된다고 가정
- ✓ 그림 9.16에서 트랜잭션 T1은 EMPLOYEE 릴레이션에서 1번 부서에 근무하는 직원들의 이름을 검색하는 동일한 SELECT문을 두 개 포함하고, 트랜잭션 T2는 1번 부서에 근무하는 직원 투표를 한 개 삽입하는 INSERT문을 포함

시간	트랜잭션 T1	트랜잭션 T2
1	SELECT ENAME FROM EMPLOYEE WHERE DNO= 1;	
2		INSERT INTO EMPLOYEE VALUES (3474, '정희연', '사원', 2106, 1500000, 1); COMMIT;
3	SELECT ENAME FROM EMPLOYEE WHERE DNO= 1; COMMIT;	

[그림 9.16] 팬텀 문제가 발생하는 수행 순서

9.2 동시성 제어(계속)

❑ 팬텀 문제(phantom problem :유령 문제)(계속)

- ✓ 시간 1에 트랜잭션 T1의 SELECT문이 수행되면 1번 부서에 근무하는 직원들의 이름인 박영권, 김상원이 검색됨
- ✓ 시간 2에 트랜잭션 T2의 INSERT문이 수행되면 EMPLOYEE 릴레이션에 1번 부서에 근무하는 정희연 투플이 삽입됨
- ✓ 시간 3에 트랜잭션 T1의 두 번째 SELECT문이 수행되면 박영권, 김상원, 정희연이 검색됨
- ✓ 즉 한 트랜잭션 T1에 속한 첫 번째 SELECT문과 두 번째 SELECT문의 수행 결과가 다르게 나타남 → 이런 현상을 **팬텀 문제**라고 부름

EMPLOYEE	EMPNO	EMPNAME	TITLE	MANAGER	SALARY	DNO
	2106	김창섭	대리	1003	2500000	2
	3426	박영권	과장	4377	3000000	1
	3011	이수민	부장	4377	4000000	3
	1003	조민희	과장	4377	3000000	2
	3427	최종철	사원	3011	1500000	3
	1365	김상원	사원	3426	1500000	1
	4377	이성래	사장	^	5000000	2
✓ 추가	3474	정희연	사원	2106	1500000	1

9.3 회복

□ 회복의 필요성

- 어떤 이유로 DBMS가 다운되든지 **DBMS회복 모듈은 트랜잭션의 원자성과 지속성을 보장해야 한다.**
- ✓ 어떤 트랜잭션 T를 수행하는 도중에 시스템이 다운되었을 때, T의 수행 효과가 디스크의 데이터베이스에 일부 반영되었을 수 있음
어떻게 T의 수행을 취소하여 원자성을 보장할 것인가?
- ✓ 또한 트랜잭션 T가 완료된 직후에 시스템이 다운되면 T의 모든 갱신 효과가 주기억 장치로부터 디스크에 기록되지 않았을 수 있음
어떻게 T의 수행 결과가 데이터베이스에 완전하게 반영되도록 하여 지속성을 보장할 것인가?
- ✓ 디스크의 헤드 등이 고장 나서 디스크의 데이터베이스를 접근할 수 없다면 어떻게 할 것인가?

9.3 회복(계속)

□ 회복의 개요

- ✓ 어떤 응용1에서 데이터베이스 항목 X가 필요해서 디스크로부터 주기억장치 버퍼로 읽어들이면(input(X), 응용2에서 동일한 항목 X가 필요할 때 디스크로부터 읽을 필요없이 주기억장치 버퍼의 항목 X를 사용한다.
- ✓ 응용2에서도 항목 X를 갱신할 수 있으므로, 응용1에서 X를 갱신 후에 디스크에 즉시 기록할 필요는 없다.
- ✓ 각 응용에서 어떤 데이터베이스 항목을 갱신할 때마다 즉시 디스크에 기록한다면 디스크 접근횟수가 크게 증가하여 성능이 저하

9.3 회복(계속)

□ 회복의 개요

- ✓ 여러 응용이 주기억 장치 버퍼 내의 동일한 데이터베이스 항목을 갱신한 후에 디스크에 기록함으로써 성능을 향상시키는 것이 중요함
- ✓ 버퍼의 내용을 디스크에 기록하는 것을 가능하면 최대한 줄이는 것이 일반적
 - 예: 버퍼가 꽉 찼을 때 또는 트랜잭션이 완료될 때 버퍼의 내용이 디스크에 기록될 수 있음
- ✓ 트랜잭션이 버퍼에는 갱신 사항을 반영했지만 버퍼의 내용이 디스크에 기록되기 전에 고장이 발생할 수 있음

9.3 회복(계속)

□ 회복의 개요(계속)

- ✓ 고장이 발생하기 전에 트랜잭션이 완료 명령을 수행했다면 회복 모듈은 이 트랜잭션의 갱신 사항을 **재수행(REDO)**하여 트랜잭션의 갱신이 지속성을 갖도록 해야 함
- ✓ 고장이 발생하기 전에 트랜잭션이 완료 명령을 수행하지 못했다면 원자성을 보장하기 위해서 이 트랜잭션이 데이터베이스에 반영했을 가능성이 있는 갱신 사항을 **취소(UNDO)**해야 함
- ✓ DBMS는 데이터베이스 회복을 위하여 백업, 로깅, 체크 포인트 등을 수행한다.
 - **백업(backup)** : 데이터베이스를 주기적으로 자기 테이프 등에 복사하는 것
 - **로깅(logging)** : 현재 수행중인 트랜잭션의 상태와 데이터베이스의 갱신 사항을 기록
 - **체크 포인트(check point)** : 시스템이 붕괴된 후 재가동 되었을 때 재수행하거나 취소해야하는 트랜잭션들의 수를 줄여준다.

9.3 회복(계속)

□ 저장 장치의 유형

- ✓ 주기억 장치와 같은 휘발성 저장 장치에 들어 있는 내용은 시스템이 다운된 후에 모두 사라짐
- ✓ 디스크와 같은 비휘발성 저장 장치에 들어 있는 내용은 디스크 헤드 등이 손상을 입지 않는 한 시스템이 다운된 후에도 유지됨
- ✓ **안전 저장 장치**(stable storage)는 모든 유형의 고장을 견딜 수 있는 저장 장치를 의미
- ✓ 두 개 이상의 비휘발성 저장 장치가 동시에 고장날 가능성이 매우 낮으므로 비휘발성 저장 장치에 두 개 이상의 사본을 중복해서 저장함으로써 **안전 저장 장치를 구현함**

9.3 회복(계속)

□ 재해적 고장과 비재해적 고장

✓ 재해적 고장

- 디스크가 손상을 입어서 데이터베이스를 읽을 수 없는 고장
- 재해적 고장으로부터의 회복은 데이터베이스를 백업해 놓은 자기 테이프를 기반으로 함

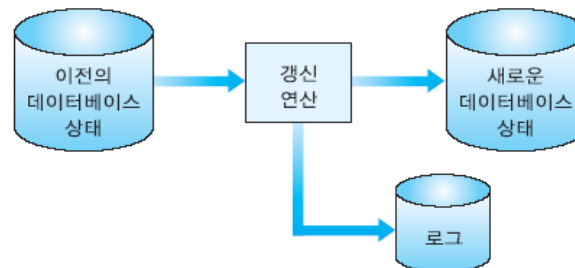
✓ 비재해적 고장

- 그 이외의 고장
- 대부분의 회복 알고리즘들은 비재해적 고장에 적용됨
- **로그를 기반으로 한 즉시 갱신**, **로그를 기반으로 한 지연 갱신**, 그림자 페이징(shadow paging) 등 여러 알고리즘
- **대부분의 상용 DBMS에서 로그를 기반으로 한 즉시 갱신 방식을 사용**

9.3 회복(계속)

□ 로그를 사용한 즉시 갱신

- ✓ 즉시 갱신에서는 트랜잭션이 데이터베이스를 갱신한 사항이 주기억 장치의 버퍼에 유지되다가 트랜잭션이 완료되기 전이라도 디스크의 데이터베이스에 기록될 수 있음
- ✓ 데이터베이스에는 완료된 트랜잭션의 수행 결과뿐만 아니라 철회된 트랜잭션의 수행 결과도 반영될 수 있음
- ✓ 트랜잭션의 원자성과 지속성을 보장하기 위해 DBMS는 로그라고 부르는 특별한 파일을 유지함
- ✓ 데이터베이스의 항목에 영향을 미치는 모든 트랜잭션의 연산들에 대해서 로그 레코드를 기록함
- ✓ 각 로그 레코드는 로그 순서 번호(LSN: Log Sequence Number)로 식별됨



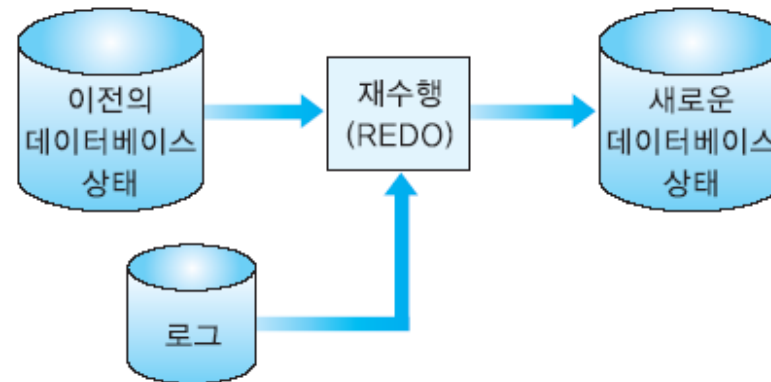
[그림 9.16] 로그 생성

9.3 회복(계속)

□ 로그를 사용한 즉시 갱신(계속)

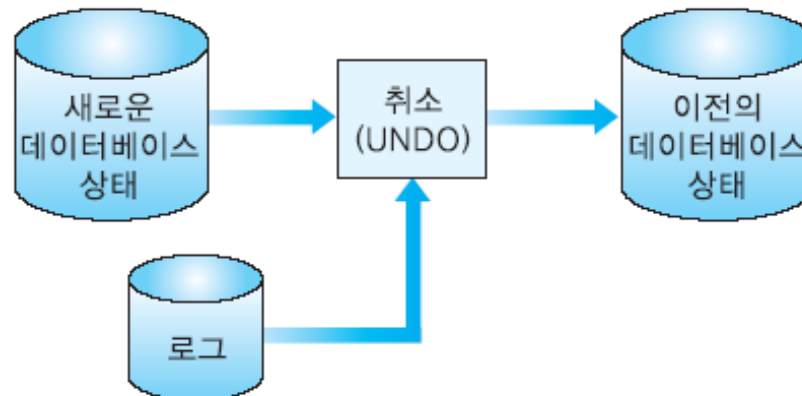
- ✓ 주기억 장치 내의 로그 버퍼에 로그 레코드들을 기록하고 로그 버퍼가 꽉 찰 때 디스크에 기록함
- ✓ 로그는 데이터베이스 회복에 필수적이므로 일반적으로 안전 저장 장치에 저장됨
- ✓ 이중 로그(dual logging): 로그를 두 개의 디스크에 중복해서 저장하는 것
- ✓ 각 로그 레코드가 어떤 트랜잭션에 속한 것인가를 식별하기 위해서 각 로그 레코드마다 트랜잭션 ID를 포함시킴
- ✓ 동일한 트랜잭션에 속하는 로그 레코드들은 연결 리스트로 유지됨

9.3 회복(계속)



[그림 9.17] 트랜잭션의 재수행

이전의 데이터베이스 상태에 **로그를 적용하여 트랜잭션을 재수행하면** 새로운 데이터베이스 상태가 생성



[그림 9.18] 트랜잭션의 취소

새로운 터베이스 상태에 **로그를 적용하여 트랜잭션을 취소하면** 이전의 데이터베이스 상태가 생성

9.3 회복(계속)

❑ 체크포인트(checkpoint)

- DBMS는 시스템이 다운된 후 재기동 되었을 때
 - [Trans-ID, start]로그 레코드와 [Trans-ID, commit]로그 레코드가 모두 디스크의 로그에 존재하면 이 트랜잭션을 재수행(REDO)하고,
 - [Trans-ID, commit]로그 레코드가 존재하지 않으면 트랜잭션을 취소(UNDO)
-
- ✓ DBMS는 회복시 재수행할 트랜잭션의 수를 줄이기 위해서 주기적으로 체크포인트를 수행함
 - ✓ 체크포인트 시점에는 주기억 장치의 버퍼 내용이 디스크에 강제로 기록되므로, 체크포인트를 수행하면 디스크 상에서 로그와 데이터베이스의 내용이 일치하게 됨
 - ✓ 체크포인트 작업이 끝나면 로그에 [checkpoint] 로그 레코드가 기록됨
 - ✓ 일반적으로 체크포인트를 10~20분마다 한 번씩 수행함

9.3 회복(계속)

❑ 데이터베이스 백업과 재해적 고장으로부터의 회복

- ✓ 아주 드물지만, 데이터베이스가 저장되어 있는 디스크의 헤드 등이 고장나서 데이터베이스를 읽을 수 없는 경우가 발생함
- ✓ 데이터베이스를 **회복**하는 한 가지 방법은 **주기적으로 자기 테이프에 전체 데이터베이스와 로그를 백업하고, 자기 테이프를 별도의 공간에 안전하게 보관하는 것(시간 많이 소요)**
- ✓ 사용자들에게 데이터베이스 사용을 계속 허용하면서, 지난 번 백업 이후에 갱신된 내용만 백업을 하는 **점진적인 백업(incremental backup)**이 바람직

9.4 PL/SQL의 트랜잭션

□ 트랜잭션의 시작과 끝

- ✓ 오라클에서 한 트랜잭션은 암시적으로 끝나거나 명시적으로 끝날 수 있음
- ✓ 한 트랜잭션은 실행 가능한 첫 번째 SQL문이 실행될 때 시작되어 데이터 정의를 만나거나, 데이터 제어어를 만나거나, COMMIT이나 ROLLBACK 없이 Oracle SQL Developer를 정상적으로 종료했을 때는 수행 중이던 트랜잭션이 **암시적으로 완료(commit)**됨
- ✓ Oracle SQL Developer를 비정상적으로 종료하거나 시스템에 장애가 발생하면 수행 중이던 **트랜잭션을 암시적으로 철회한다.**

9.4 PL/SQL의 트랜잭션(계속)

□ 트랜잭션의 시작과 끝(계속)

- ✓ COMMIT, ROLLBACK, SAVEPOINT문을 사용하여 트랜잭션의 논리를 명시적으로 제어할 수 있음
- **COMMIT문** : 현재의 트랜잭션에서 수행한 한 개 이상의 데이터 조작어의 결과를 데이터베이스에 모두 반영하고 **현재의 트랜잭션을 완료**
- **ROLLBACK문** : 현재의 트랜잭션에서 수행한 한 개 이상의 데이터 조작어의 결과를 데이터베이스에서 **모두 되돌리고 현재의 트랜잭션을 철회**
- **SAVEPOINT문** : 현재의 트랜잭션 내에 저장점을 표시하여 트랜잭션을 더 작은 부분으로 나눔
- **ROLLBACK TO SAVEPOINT문**을 사용하면 현재의 트랜잭션에서 **지정된 저장점 이후에 갱신된 내용만 되돌림**

9.4 PL/SQL의 트랜잭션(계속)

□ 트랜잭션의 시작과 끝(계속)

- ✓ SQL*Plus에서는 묵시적으로 한 트랜잭션에서 데이터 정의어나 데이터 제어어 이전까지 입력한 여러 개의 데이터 조작어로 이루어짐
- ✓ 사용자가 **각 데이터 조작어 마다 COMMIT문을 입력해야 하므로 번거롭다**. 이럴 경우
- ✓ **set 명령**을 사용하여 각 데이터 조작어를 한 트랜잭션으로 처리할 수 있음

```
set auto on;
```

또는

```
set autocommit on;
```

```
SQL> DELETE FROM EMPLOYEE ;
```

```
7 rows deleted.
```

```
SQL>ROLLBACK
```

```
Rollback complete.
```

```
SQL>DELETE FROM EMPLOYEE WHERE DNO=3 ;
```

```
2 rows deleted.
```

9.4 PL/SQL의 트랜잭션(계속)

□ 트랜잭션의 속성

- ✓ 만일 트랜잭션이 데이터베이스를 읽기만 한다면 트랜잭션이 읽기 전용임을 명시하여 DBMS가 동시성의 정도를 높일 수 있음

```
SET TRANSACTION READ ONLY;
```

```
SELECT  AVG(SALARY)  
FROM    EMPLOYEE  
WHERE   DEPT='개발부';
```

- ✓ 만일 어떤 트랜잭션이 읽기 전용이라고 명시했으면 **그 트랜잭션은 어떠한 갱신 작업도 수행할 수 없다.** 예를 들어 아래와 같은 SQL문은 허용되지 않음

```
SET TRANSACTION READ ONLY;
```

```
UPDATE  EMPLOYEE  
SET      SALARY = SALARY * 1.06;
```

9.4 PL/SQL의 트랜잭션(계속)

□ 트랜잭션의 속성(계속)

- ✓ 트랜잭션에 대해 SET TRANSACTION READ WRITE를 명시하면 **SELECT, INSERT, DELETE, UPDATE**문을 모두 수행할 수 있음

```
SET TRANSACTION READ WRITE;
```

```
UPDATE    EMPLOYEE
```

```
SET      SALARY = SALARY * 1.06;
```

9.4 PL/SQL의 트랜잭션(계속)

□ 고립 수준

✓ 고립성(Isolation)-p11참조

- 한 트랜잭션이 데이터를 갱신하는 동안 이 트랜잭션이 완료되기 전에는 갱신 중인 데이터를 다른 트랜잭션들이 접근하지 못하도록 해야 함
- 다수의 트랜잭션들이 동시에 수행되더라도 그 결과는 어떤 순서에 따라 트랜잭션들을 하나씩 차례대로 수행한 결과와 같아야 함
- DBMS의 동시성 제어 모듈이 트랜잭션의 고립성을 보장함
- DBMS는 응용들의 요구사항에 따라 다양한 고립 수준(isolation level)을 제공함

9.4 PL/SQL의 트랜잭션(계속)

□ 고립 수준

- ✓ SQL2에서 사용자가 동시성의 정도를 몇 가지로 구분하여 명시할 수 있음
- ✓ 고립 수준은 한 트랜잭션이 다른 트랜잭션과 고립되어야 하는 정도를 나타냄
- ✓ 고립 수준이 낮으면 동시성은 높아지지만 데이터의 정확성은 떨어짐
- ✓ 고립 수준이 높으면 데이터가 정확해지지만 동시성이 저하됨
- ✓ 응용의 성격에 따라 허용 가능한 고립 수준(데이터베이스의 정확성)을 선택함으로써 성능을 향상시킬 수 있음
- ✓ 응용에서 명시한 고립 수준에 따라 DBMS가 사용하는 로킹 동작이 달라짐
- ✓ 한 트랜잭션에 대해 명시한 고립 수준에 따라 그 트랜잭션이 읽을 수 있는 데이터에만 차이가 있음

9.4 PL/SQL의 트랜잭션(계속)

❑ 오라클에서 제공하는 몇 가지 고립 수준

✓ READ UNCOMMITTED : 커밋되지 않은것도 읽기

- 가장 낮은 고립 수준
- 트랜잭션 내의 질의들이 공유 로크를 걸지 않고 데이터를 읽음
- 따라서 오손 데이터를 읽을 수 있음

오손 데이터 읽기(dirty read): 완료되지 않은 트랜잭션이 갱신한 데이터를 읽는 것

- 갱신하려는 데이터에 대해서는 독점 로크를 걸고, 트랜잭션이 끝날 때까지 보유함

```
SET TRANSACTION READ WRITE  
ISOLATION LEVEL READ UNCOMMITTED;
```


9.4 PL/SQL의 트랜잭션(계속)

❑ 오라클에서 제공하는 몇 가지 고립 수준(계속)

✓ READ COMMITTED : 커밋 된것만 읽기

- 트랜잭션 내의 **질의들이 읽으려는 데이터에 대해서 공유 로크를 걸고, 읽기가 끝나자마자 로크를 해제함**
- 따라서 동일한 데이터를 다시 읽기 위해 공유 로크를 다시 걸고 데이터를 읽으면, 이전에 읽은 값과 다른 값을 읽는 경우가 생길 수 있음
- 갱신하려는 데이터에 대해서는 **독점 로크를 걸고, 트랜잭션이 끝날 때까지 보유함**
- 이 고립 수준은 PL/SQL의 디폴트
- 예: 항공기 예약 트랜잭션에서 현재 빈좌석을 찾을 수 없을 때, 누가 좌석을 취소했는가를 계속 검색하기 위해서 이 고립수준을 명시 할 수 있다.

SET TRANSACTION READ WRITE

ISOLATION LEVEL READ COMMITTED;

9.4 PL/SQL의 트랜잭션(계속)

❑ 오라클에서 제공하는 몇 가지 고립 수준(계속)

✓ REPEATABLE READ : 반복적인 읽기

- 질의에서 검색되는 데이터에 대해 공유 로크를 걸고, 트랜잭션이 끝날 때까지 보유함
- 한 트랜잭션 내에서 동일한 질의를 두 번 이상 수행할 때 매번 같은 값을 포함한 결과를 검색하게 됨
- 갱신하려는 데이터에 대해서는 독점 로크를 걸고, 트랜잭션이 끝날 때까지 보유함

SET TRANSACTION READ WRITE

ISOLATION LEVEL REPEATABLE READ;

9.4 Transact-SQL의 트랜잭션(계속)

❑ 오라클에서 제공하는 몇 가지 고립 수준(계속)

✓ SERIALIZABLE(=not comparable)

- 가장 높은 고립 수준
- 질의에서 검색되는 튜플들 뿐만 아니라 인덱스에 대해서도 공유 로크를 걸고 트랜잭션이 끝날 때까지 보유함
- 갱신하려는 데이터에 대해서는 독점 로크를 걸고 트랜잭션이 끝날 때까지 보유함
- SERIALIZABLE은 SQL2의 디폴트 고립 수준

SET TRANSACTION READ WRITE

ISOLATION LEVEL SERIALIZABLE;

9.4 PL/SQL의 트랜잭션(계속)

〈표 9.5〉 고립 수준에 따른 동시성 문제

고립 수준	오손 데이터 읽기	반복할 수 없는 읽기	팬텀 문제
READ UNCOMMITTED	나타날 수 있음	나타날 수 있음	나타날 수 있음
READ COMMITTED	나타날 수 없음	나타날 수 있음	나타날 수 있음
REPEATABLE READ	나타날 수 없음	나타날 수 없음	나타날 수 있음
SERIALIZABLE	나타날 수 없음	나타날 수 없음	나타날 수 없음