제9장 트랜잭션

9장. 트랜잭션

- 트랜잭션(transaction)
  - ✓ 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
  - ✓ 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

9장. 트랜잭션(계속)

- 트랜잭션(transaction)(계속)
  - ✓ 회복(recovery)
    - 데이터베이스를 갱신하는 도중에 시스템이 고장 나도 데이터베이스의 일관성을 유지함

9.1 트랜잭션 개요

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

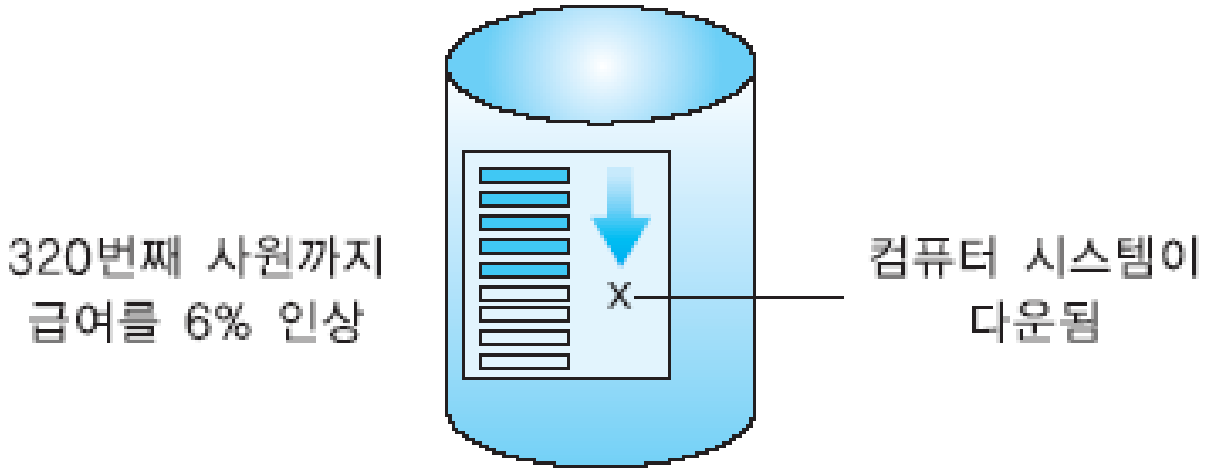
예 1 : 전체 사원의 급여를 6% 인상

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

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

- ✓ 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)



[그림 9.1] 데이터베이스를 갱신하는 중에 컴퓨터 시스템의 다운

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?
- ✓ 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

9.1 트랜잭션 개요(계속)

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

예 3 : 항공기 예약

여행사에서 고객의 요청에 따라 항공기를 예약하려고 한다. 아래의 응용 프로그램은 고급 프로그래밍 언어로 작성한 프로그램 내에 세 개의 SQL문을 내포시킨 것이다. 이 응용 프로그램에서는 두 개의 릴레이션을 사용한다. FLIGHT 릴레이션은 각 항공기편마다 FNO(항공기편 번호), DATE(출발일), SOURCE(출발지), DESTINATION(목적지), SEAT_SOLD(팔린 좌석수), CAPACITY(총좌석수) 등의 애트리뷰트를 갖는다. RESERVED 릴레이션은 각 예약 고객마다 FNO(항공기편 번호), DATE(출발일), CUST_NAME(고객 이름), SPECIAL(비고) 등의 애트리뷰트를 갖는다. FLIGHT 릴레이션에서 고객이 원하는 날짜의 항공기편에 빈 좌석이 남아 있으면 팔린 좌석수를 1만큼 증가시키고, RESERVED 릴레이션에 예약 고객에 관한 투플을 삽입한다.

FLIGHT (FNO, DATE, SOURCE, DESTINATION, SEAT_SOLD, CAPACITY)
RESERVED (FNO, DATE, CUST_NAME, SPECIAL)

```
Begin_transaction Reservation
  begin
    input(flight_no, date, customer_name);

    EXEC SQL SELECT SEAT_SOLD, CAPACITY
             INTO   temp1, temp2
             FROM   FLIGHT
             WHERE  FNO=flight_no AND DATE = date;        (1)

    if temp1 = temp2
      then output("빈 좌석이 없습니다");
          Abort
      else

    EXEC SQL UPDATE FLIGHT
             SET    SEAT_SOLD = SEAT_SOLD + 1            (2)
             WHERE  FNO = fight_no  AND DATE = date;

    EXEC SQL INSERT
             INTO   RESERVED(FNO, DATE, CUST_NAME,
                    SPECIAL)                              (3)
             VALUES (flight_no, date,
                     customer_name, null);

    Commit
    output("예약이 완료되었습니다");
  endif
end. {Reservation}
```

9.1 트랜잭션 개요(계속)

- ✓ 만일 SQL문 (2)를 수행하고 SQL문 (3)을 수행하기 전에 컴퓨터 시스템이 다운되고 재기동한 후에 DBMS가 어떻게 대응해야 하는가?
- ✓ 위의 세 개의 SQL문이 모두 완전하게 수행되거나 하나도 수행되어서는 안되도록, 즉 하나의 트랜잭션(단위)처럼 DBMS가 취급해야 함
- ✓ DBMS는 각 SQL문의 의미를 알 수 없으므로 하나의 트랜잭션으로 취급해야 하는 SQL문들의 범위를 사용자가 명시적으로 표시해야 함

9.1 트랜잭션 개요(계속)

❑ 트랜잭션의 특성(ACID 특성)

✓ 원자성(Atomicity)

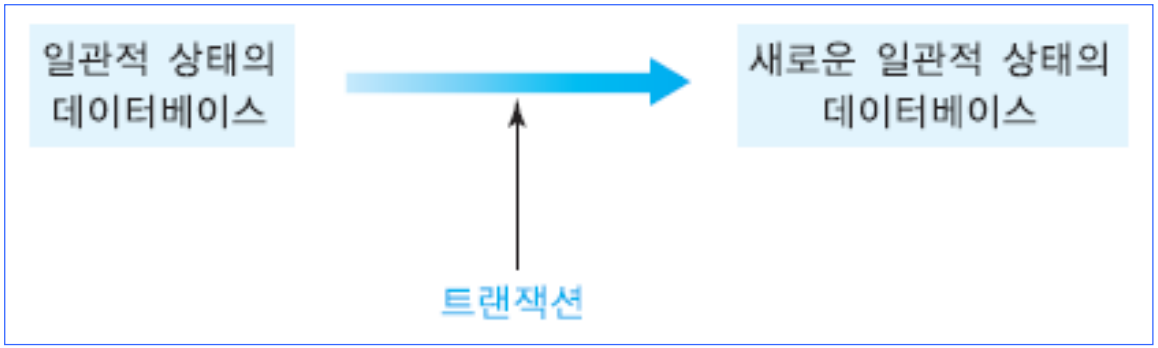
- 한 트랜잭션 내의 모든 연산들이 완전히 수행되거나 전혀 수행되지 않음(all or nothing)을 의미
- DBMS의 회복 모듈은 시스템이 다운되는 경우에, 부분적으로 데이터베이스를 갱신한 트랜잭션의 영향을 취소함으로써 트랜잭션의 원자성을 보장함
- 완료된 트랜잭션이 갱신한 사항은 트랜잭션의 영향을 재수행함으로써 트랜잭션의 원자성을 보장함

9.1 트랜잭션 개요(계속)

❑ 트랜잭션의 특성(ACID 특성)(계속)

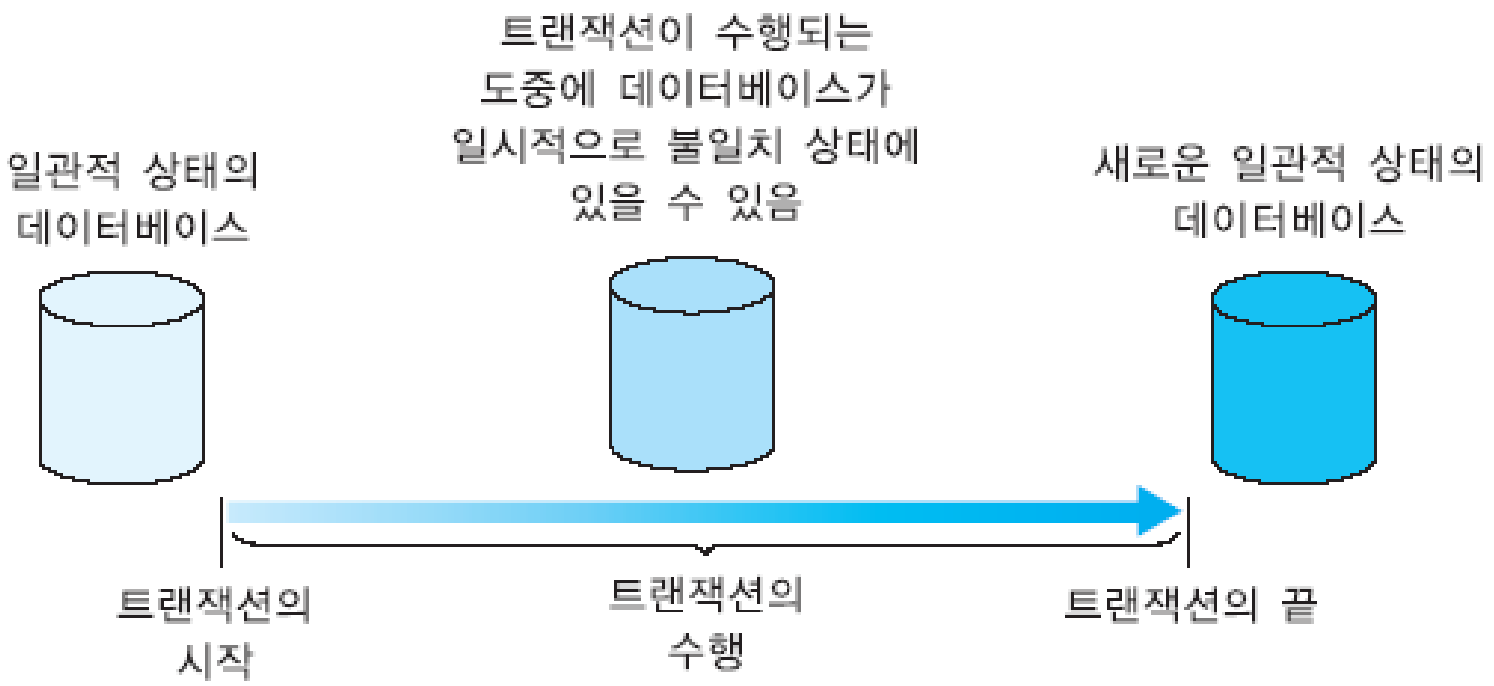
✓ 일관성(Consistency)

- 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

9.1 트랜잭션 개요(계속)



[그림 9.2] 데이터베이스의 일시적인 불일치 상태

9.1 트랜잭션 개요(계속)

❑ 트랜잭션의 특성(ACID 특성)(계속)

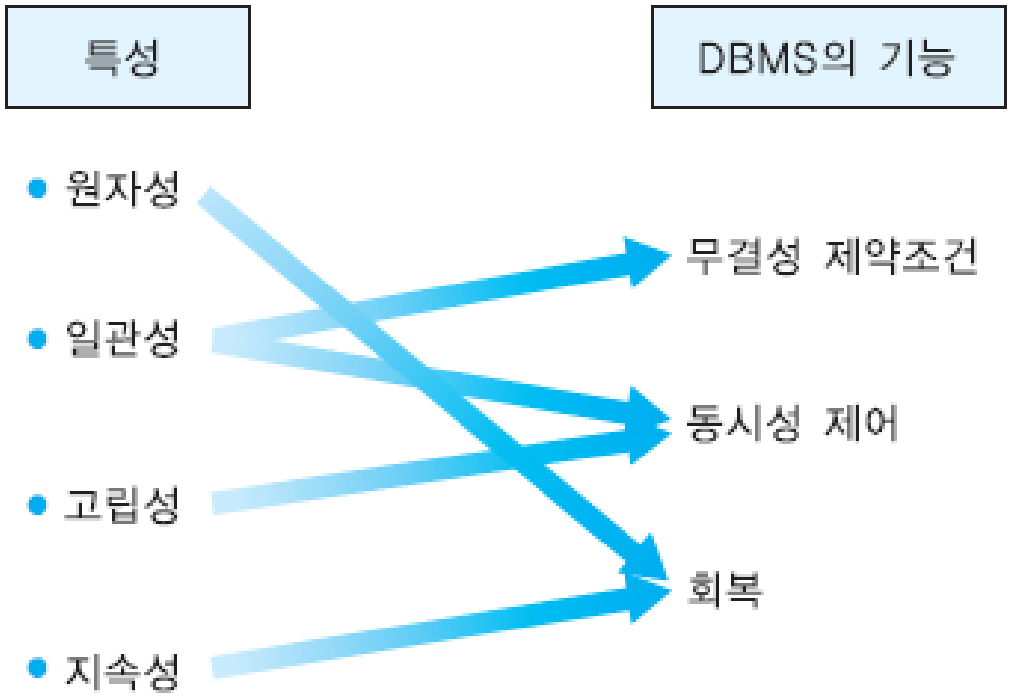
✓ 고립성(Isolation)

- 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

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

9.1 트랜잭션 개요(계속)



[그림 9.3] 트랜잭션의 네 가지 특성과 DBMS의 기능과의 관계

9.1 트랜잭션 개요(계속)

- 트랜잭션의 완료(commit)
  - ✓ 트랜잭션에서 변경하려는 내용이 데이터베이스에 완전하게 반영됨
  - ✓ SQL 구문상으로 COMMIT WORK

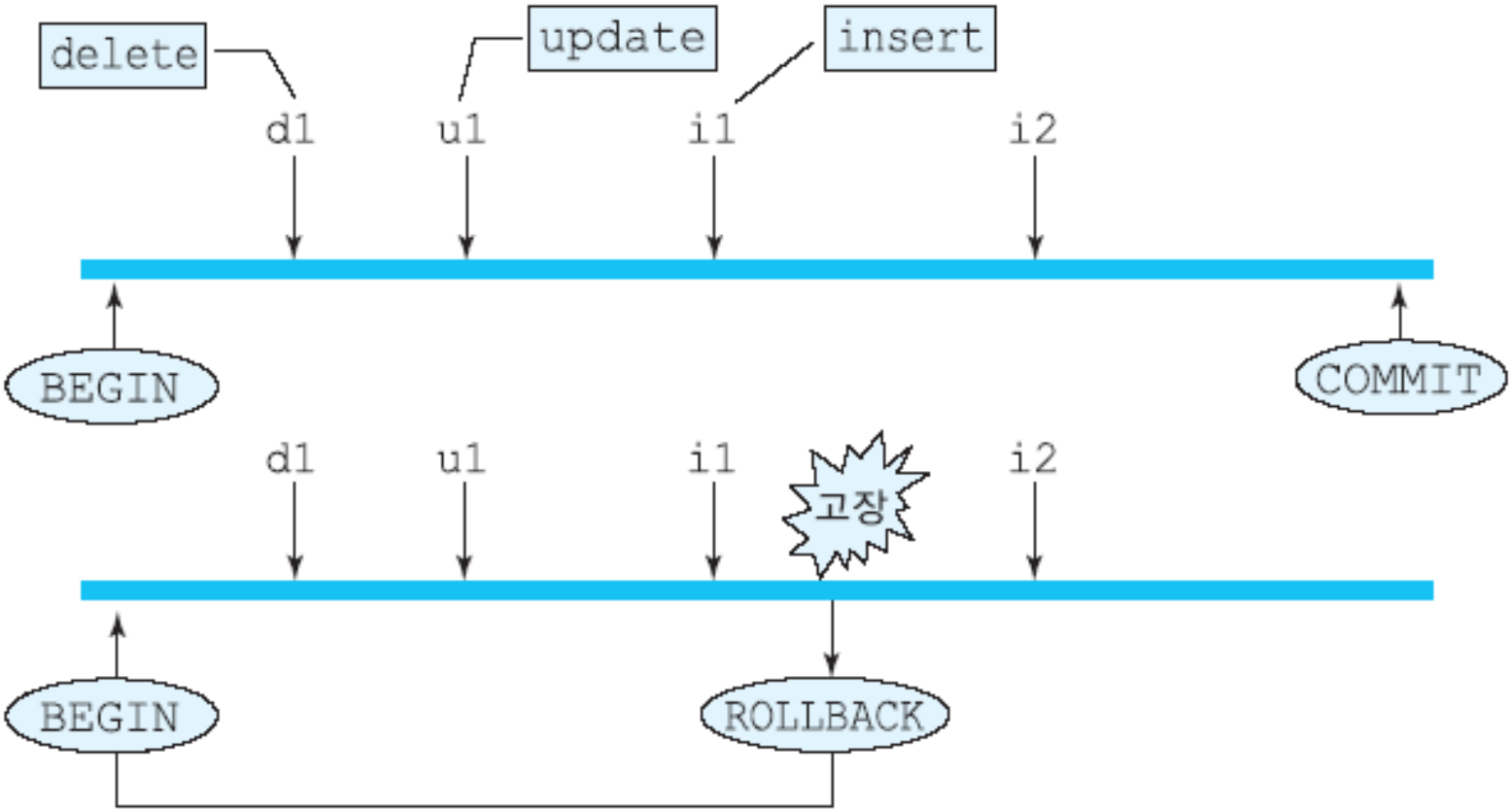
- 트랜잭션의 철회(abort)
  - ✓ 트랜잭션에서 변경하려는 내용이 데이터베이스에 일부만 반영된 경우에는 원자성을 보장하기 위해서, 트랜잭션이 갱신한 사항을 트랜잭션이 수행되기 전의 상태로 되돌림
  - ✓ SQL 구문상으로 ROLLBACK WORK

9.1 트랜잭션 개요(계속)

〈표 9.1〉 COMMIT과 ROLLBACK의 비교

연산	COMMIT	ROLLBACK
의미	완료(성공적인 종료)	철회(비성공적인 종료)
DBMS의 트랜잭션 관리 모듈에게 알리는 사항	• 트랜잭션이 성공적으로 끝났음 • 데이터베이스는 새로운 일관된 상태를 가짐 • 트랜잭션이 수행한 갱신을 데이터베이스에 반영해야 함	• 트랜잭션의 일부를 성공적으로 끝내지 못했음 • 데이터베이스가 불일치 상태를 가질 수 있음 • 트랜잭션이 수행한 갱신이 데이터베이스에 일부 반영되었다면 취소해야 함

9.1 트랜잭션 개요(계속)



[그림 9.4] COMMIT과 ROLLBACK

9.1 트랜잭션 개요(계속)

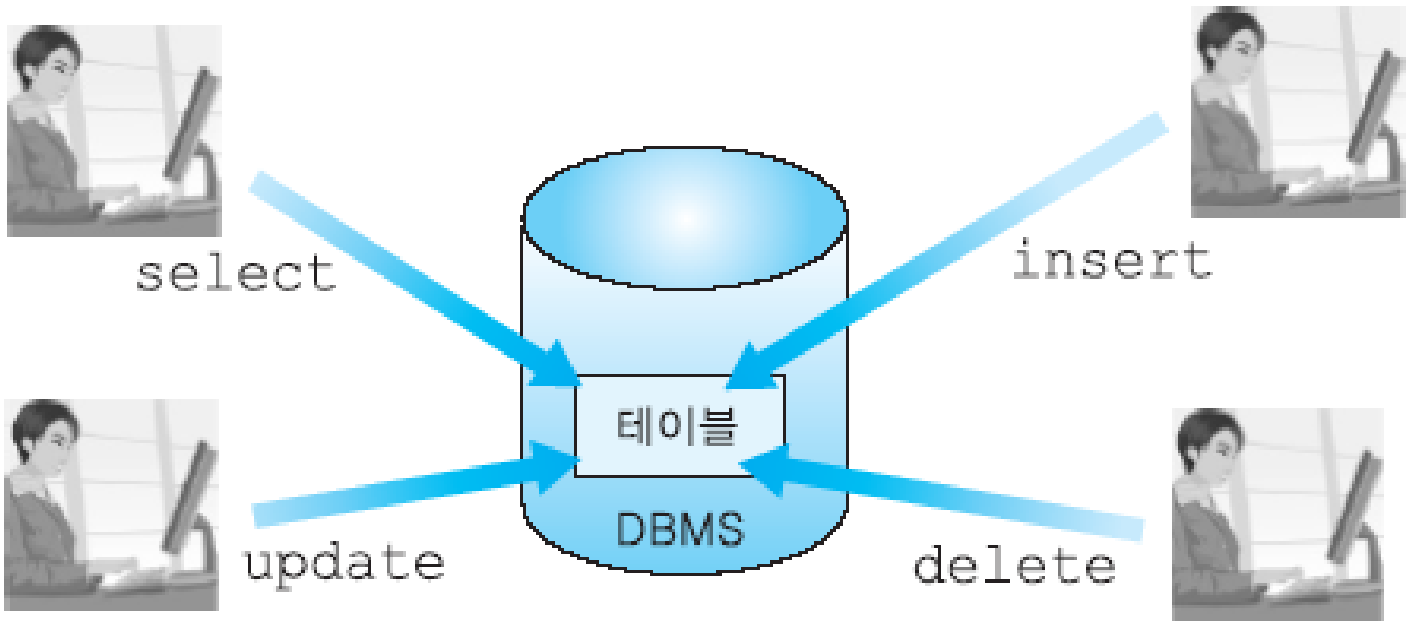
- 트랜잭션이 성공하지 못하는 원인
  - ✓ 시스템(사이트) 고장
    - 중앙 처리 장치, 주기억 장치, 전원 공급 장치 등이 고장남
  - ✓ 트랜잭션 고장
    - 트랜잭션 고장은 트랜잭션이 수행되는 도중에 철회됨
  - ✓ 매체 고장
    - 디스크 헤드, 디스크 콘트롤러 등이 고장 나서 보조 기억 장치의 전부 또는 일부 내용이 지워짐
  - ✓ 통신 고장
  - ✓ 자연적 재해
  - ✓ 부주의 또는 고의적인 고장

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
- ✓ 동시성 제어 기법은 여러 사용자들이 다수의 트랜잭션들을 동시에 수행하는 환경에서 부정확한 결과를 생성할 수 있는, 트랜잭션들 간의 간섭이 생기지 않도록 함

9.2 동시성 제어



[그림 9.5] 다수 사용자의 동시 접근

9.2 동시성 제어

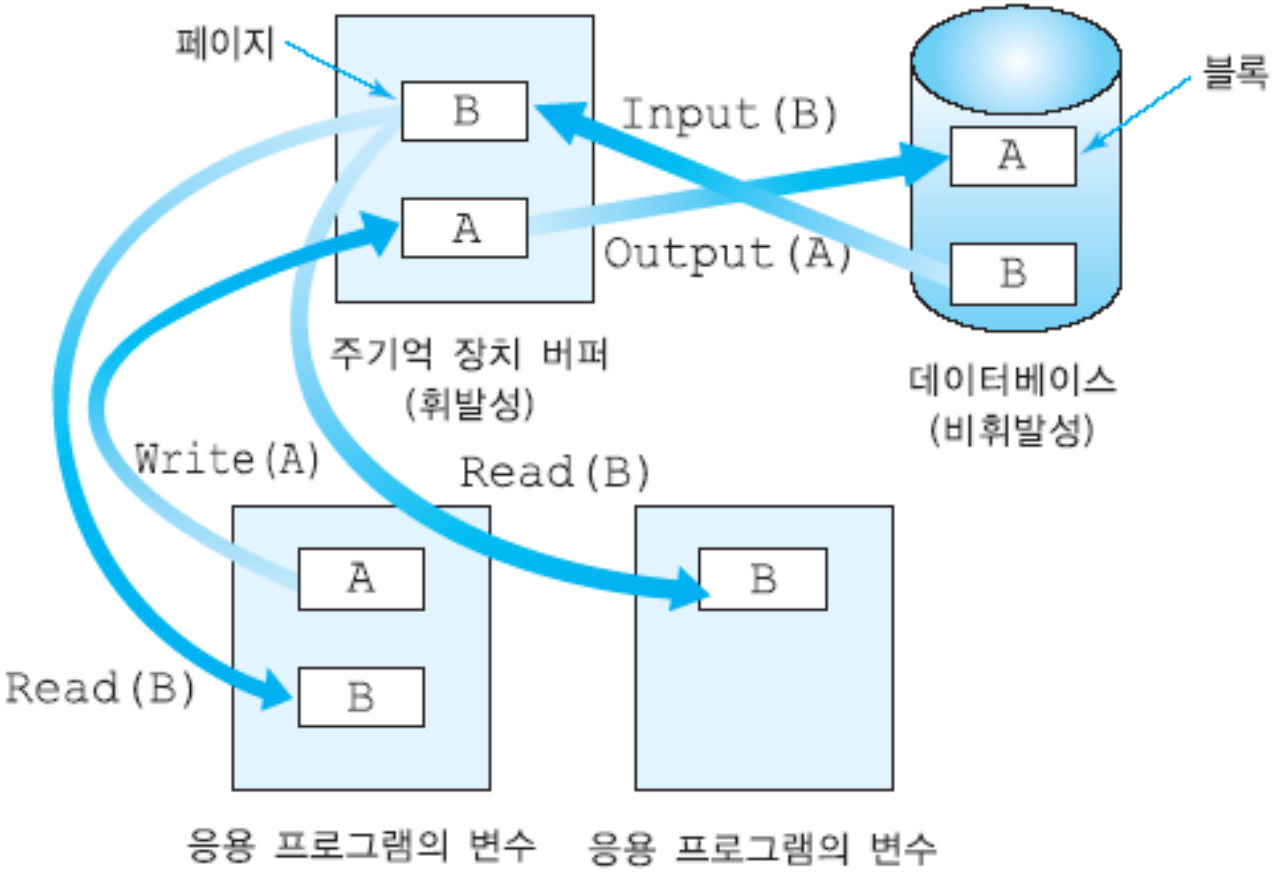
- 직렬 스케줄(serial schedule)
  - ✓ 여러 트랜잭션들의 집합을 한 번에 한 트랜잭션씩 차례대로 수행함
- 비직렬 스케줄(non-serial schedule)
  - ✓ 여러 트랜잭션들을 동시에 수행함
- 직렬가능(serializable)
  - ✓ 비직렬 스케줄의 결과가 어떤 직렬 스케줄의 수행 결과와 동등함

9.2 동시성 제어(계속)

❑ 데이터베이스 연산

- ✓ Input(X) 연산은 데이터베이스 항목 X를 포함하고 있는 블록을 주기억 장치의 버퍼로 읽어들임
- ✓ Output(X) 연산은 데이터베이스 항목 X를 포함하고 있는 블록을 디스크에 기록함
- ✓ read_item(X) 연산은 주기억 장치 버퍼에서 데이터베이스 항목 X의 값을 프로그램 변수 X로 복사함
- ✓ write_item(X) 연산은 프로그램 변수 X의 값을 주기억 장치 내의 데이터베이스 항목 X에 기록함

9.2 동시성 제어(계속)



[그림 9.6] 값의 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 씀으로써 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)

예: 갱신 손실

하나의 SQL문은 DBMS 내에서 여러 개의 명령들로 나뉘어 수행된다. 다수 사용자 환경에서는 여러 사용자들이 동시에 요청한 트랜잭션의 명령들이 섞여서 수행될 수 있다. 트랜잭션 T1은 X에서 Y로 100000을 이체하고, 트랜잭션 T2는 X의 값에 50000을 더하려고 한다. 두 트랜잭션이 수행되기 전의 X와 Y의 초기값이 각각 300000과 600000이라고 가정하면 T1의 수행을 먼저 완료하고 T2의 수행을 완료하던지, T2의 수행을 먼저 완료하고 T1의 수행을 완료하던지 관계 없이 X의 최종값은 250000, Y의 최종값은 700000이 되어야 한다.

9.2 동시성 제어(계속)

T1	T2	X와 Y의 값
read_item(X); X=X-100000;		X=300000 Y=600000
	read_item(X); X=X+50000;	X=300000 Y=600000
write_item(X); read_item(Y);		X=200000 Y=600000
	write_item(X);	X=350000 Y=600000
Y=Y+100000; write_item(Y);		X=350000 Y=700000

[그림 9.7] 갱신 손실

9.2 동시성 제어(계속)

예: 오손 데이터 읽기

그림 9.8에서 트랜잭션 T1이 정미림의 잔액을 100000원 감소시킨 후에 트랜잭션 T2는 모든 계좌의 잔액의 평균값을 검색하였다. 그 이후에 T1이 어떤 이유로 철회되면 T1이 갱신한 정미림 계좌의 잔액은 원래 상태로 되돌아간다. 따라서 T2는 완료되지 않은 트랜잭션이 갱신한 데이터, 즉 틀린 데이터를 읽었다.

9.2 동시성 제어(계속)

T1	T2
UPDATE account **SET** balance=balance-100000 **WHERE** cust_name='정미림';	
	SELECT AVG(balance) **FROM** account;
ROLLBACK;	
	COMMIT;

[그림 9.8] 오손 데이터 읽기

9.2 동시성 제어(계속)

예: 반복할 수 없는 읽기

그림 9.9에서 먼저 트랜잭션 T2는 모든 계좌의 잔액의 평균값을 검색하였다. 트랜잭션 T2가 완료되기 전에 트랜잭션 T1이 정미림의 잔액을 100000원 감소시키고 완료되었다. 트랜잭션 T2가 다시 모든 계좌의 잔액의 평균값을 검색하면 첫 번째 평균값과 다른 값을 보게 된다. 동일한 읽기 연산을 여러 번 수행할 때 매번 서로 다른 값을 보게 될 수 있다.

9.2 동시성 제어(계속)

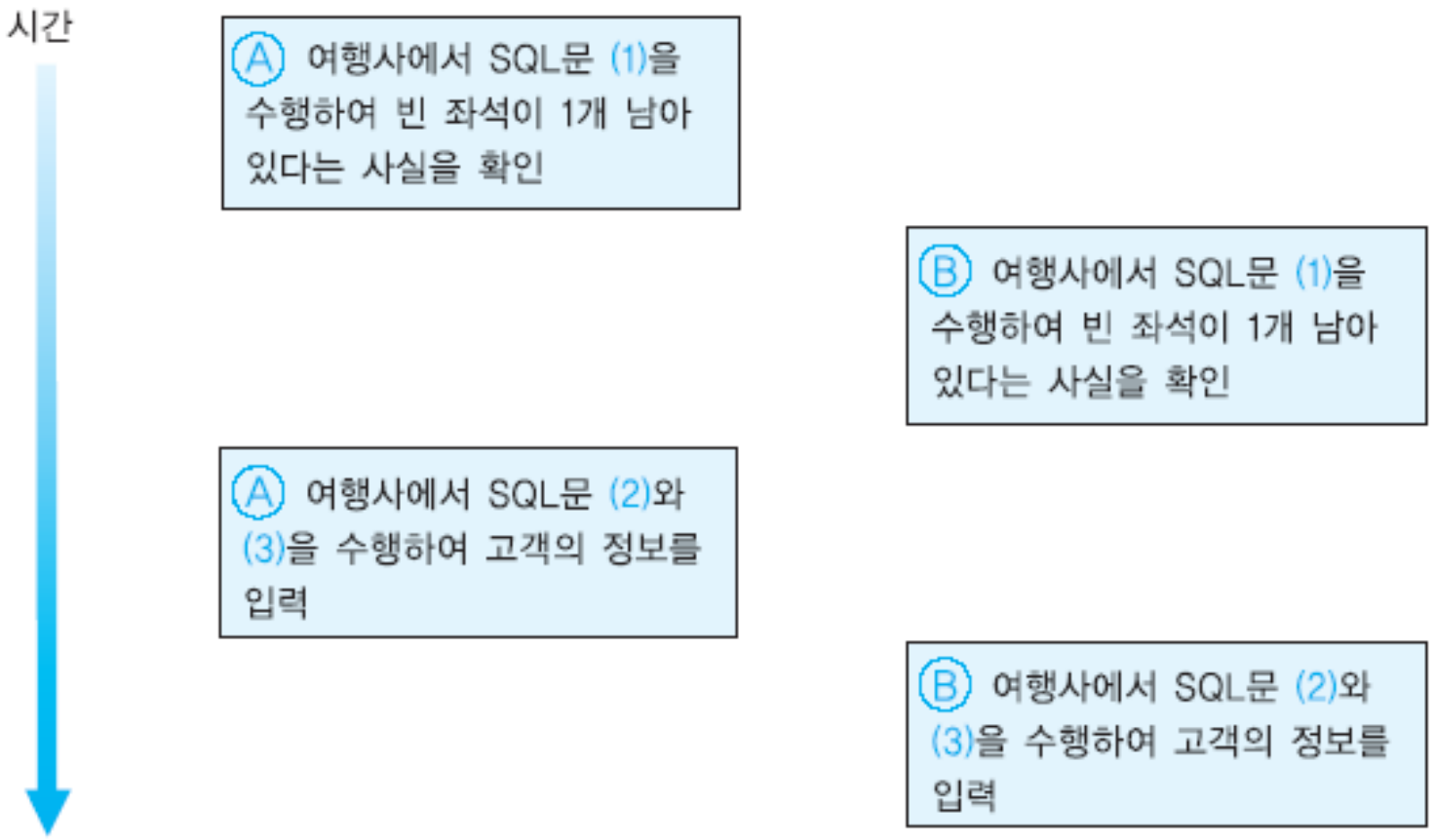
T1	T2
	SELECT AVG(balance) **FROM** account;
UPDATE account **SET** balance=balance-100000 **WHERE** cust_name='정미림'; **COMMIT**;	
	SELECT AVG(balance) **FROM** account; **COMMIT**;

[그림 9.9] 반복할 수 없는 읽기

9.2 동시성 제어(계속)

- 9.1절의 항공기 예약 트랜잭션
  - ✓ 여러 여행사에서 동시에 고객들의 요청에 따라 동일한 날짜에 출발하는 항공기의 빈 좌석 유무를 검사할 수 있음
  - ✓ 그림 9.10의 순서와 같이 만일 두 여행사에서 각각 트랜잭션을 수행하는 과정에 SQL문 (1)의 수행 결과로 특정 항공기에 빈 좌석이 1개 남아 있다는 사실을 확인하고 동시에 SQL문 (2)와 (3)을 수행하여 팔린 좌석수를 1만큼씩 증가시키고 자신의 고객의 정보를 항공사 데이터베이스에 입력하려 할 때 DBMS가 아무런 조치를 취하지 않으면, 1개 남은 좌석에 두 명의 고객이 배정되는 결과를 초래하게 됨

9.2 동시성 제어(계속)



[그림 9.10] 한 좌석을 두 고객에게 배정

9.2 동시성 제어(계속)

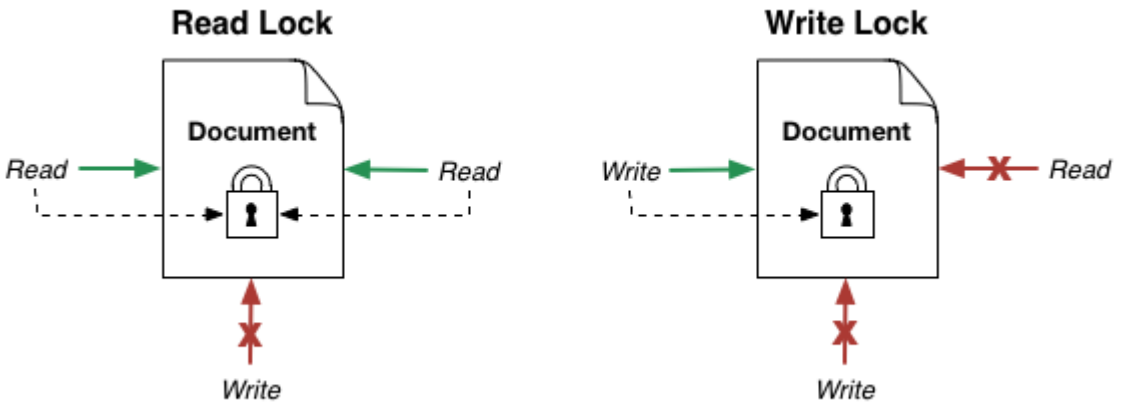
❑ 로킹(locking)

- ✓ 데이터 항목을 로킹하는 개념은 동시에 수행되는 트랜잭션들의 동시성을 제어하기 위해서 가장 널리 사용되는 기법
- ✓ 로크(lock)는 데이터베이스 내의 각 데이터 항목과 연관된 하나의 변수
- ✓ 각 트랜잭션이 수행을 시작하여 데이터 항목을 접근할 때마다 요청한 로크에 관한 정보는 로크 테이블(lock table) 등에 유지됨
- ✓ 트랜잭션에서 데이터 항목을 접근할 때 로크를 요청하고, 접근을 끝낸 후에 로크를 해제(unlock)함
  - ✓ 갱신 목적 접근: 독점 로크(X-lock, eXclusive lock)를 요청함
  - ✓ 판독(읽기) 목적 접근: 공유 로크(S-lock, Shared lock)를 요청함

9.2 동시성 제어(계속)

〈표 9.2〉 로크 양립성 행렬

		현재 걸려 있는 로크		
		공유 로크	독점 로크	로크가 걸려 있지 않음
요청 중인 로크	공유 로크	허용	대기	허용
	독점 로크	대기	대기	허용



9.2 동시성 제어(계속)

- 2단계 로킹 프로토콜(2-phase locking protocol)
  - ✓ 로크를 요청하는 것과 로크를 해제하는 것이 2단계로 이루어짐
  - ✓ 로크 확장 단계가 지난 후에 로크 수축 단계에 들어감
  - ✓ 일단 로크를 한 개라도 해제하면 로크 수축 단계에 들어감

9.2 동시성 제어(계속)

T1	T2

X-lock(A);
read_item(A);
A=A+1;
write_item(A);
unlock(A);	

	X-lock(A);
	read_item(A);
	A=A * 2;
	write_item(A);
	unlock(A);
	X-lock(B);
	read_item(B);
	B=B * 2;
	write_item(B);
	unlock(B);

X-lock(B);
read_item(B);
B=B+1;
write_item(B);
unlock(B);	

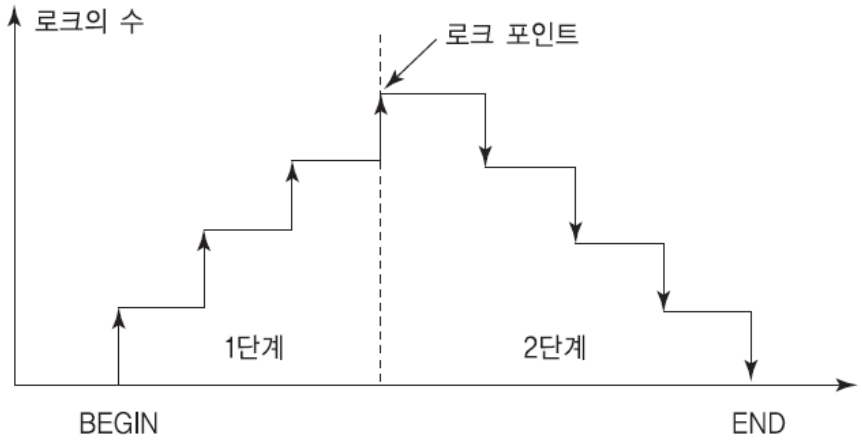
[그림 9.11] 로크를 일찍 해제

9.2 동시성 제어(계속)

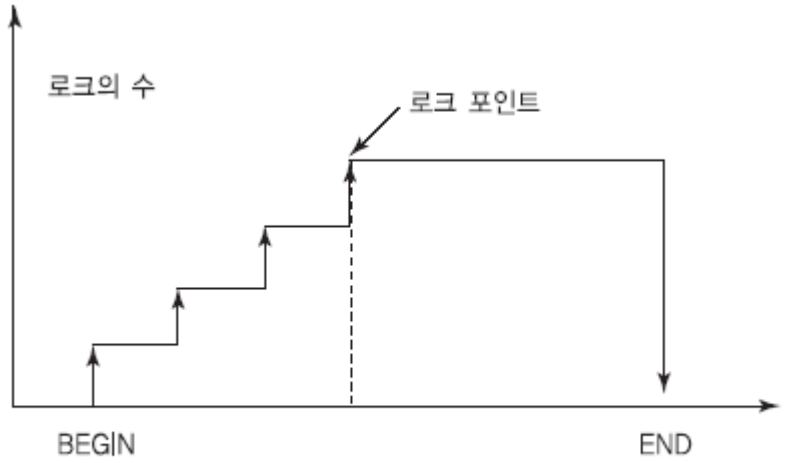
❑ 2단계 로킹 프로토콜(계속)

- ✓ 로크 확장 단계(1단계)
  - 로크 확장 단계에서는 트랜잭션이 데이터 항목에 대하여 새로운 로크를 요청할 수 있지만 보유하고 있던 로크를 하나라도 해제할 수 없음
- ✓ 로크 수축 단계(2단계)
  - 로크 수축 단계에서는 보유하고 있던 로크를 해제할 수 있지만 새로운 로크를 요청할 수 없음
  - 로크 수축 단계에서는 로크를 조금씩 해제할 수도 있고(그림 9.12), 트랜잭션이 완료 시점에 이르렀을 때 한꺼번에 모든 로크를 해제할 수도 있음(그림 9.13)
  - 일반적으로 한꺼번에 해제하는 방식이 사용됨
- ✓ 로크 포인트(lock point)는 한 트랜잭션에서 필요로 하는 모든 로크를 걸어놓은 시점

9.2 동시성 제어(계속)



[그림 9.12] 로크를 조금씩 해제



[그림 9.13] 로크를 한꺼번에 해제

9.2 동시성 제어(계속)

- 데드록(deadlock)
  - ✓ 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)

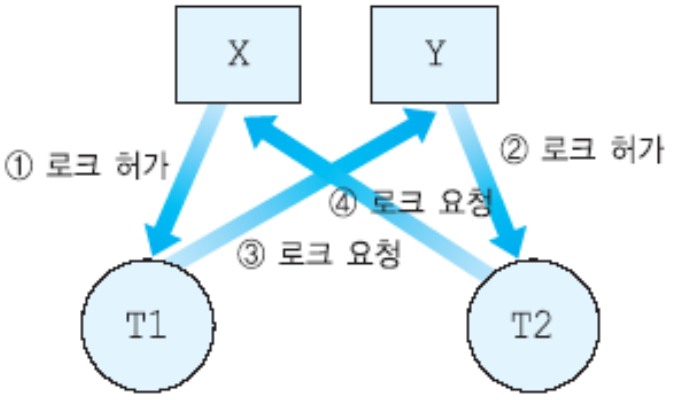
❑ 데드록(계속)

① T1이 X에 대해 독점 로크를 요청하여 허가받음

② T2이 Y에 대해 독점 로크를 요청하여 허가받음

③ T1이 Y에 대해 공유 로크나 독점 로크를 요청하면 로크가 해제될 때까지 기다리게 됨

④ T2가 X에 대해 공유 로크나 독점 로크를 요청하면 로크가 해제될 때까지 기다리게 됨



[그림 9.14] 데드록의 예

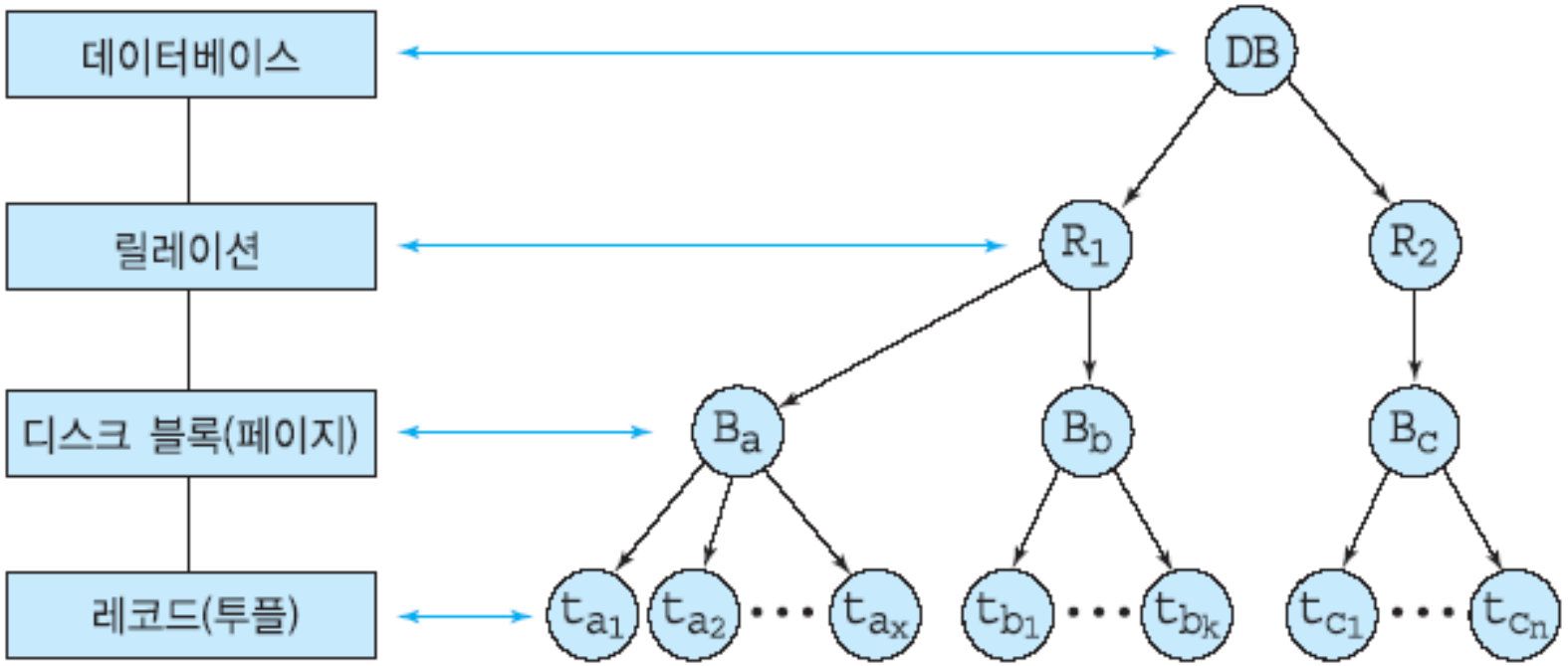
9.2 동시성 제어(계속)

- 다중 로크 단위(multiple granularity)
  - ✓ 대부분의 트랜잭션들이 소수의 투플들을 접근하는 데이터베이스 응용에서는 투플 단위로 로크를 해도 로크 테이블을 다루는 시간이 오래 걸리지 않음
  - ✓ 트랜잭션들이 많은 투플을 접근하는 데이터베이스 응용에서 투플 단위로만 로크를 한다면 로크 테이블에서 로크 충돌을 검사하고, 로크 정보를 기록하는 시간이 오래 걸림
  - ✓ 트랜잭션이 접근하는 투플의 수에 따라 로크를 하는 데이터 항목의 단위를 구분하는 것이 필요함
  - ✓ 한 트랜잭션에서 로크할 수 있는 데이터 항목이 두 가지 이상 있으면 다중 로크 단위라고 말함
  - ✓ 데이터베이스에서 로크할 수 있는 단위로는 데이터베이스, 릴레이션, 디스크 블록, 투플 등

9.2 동시성 제어(계속)

- 다중 로크 단위(계속)
  - ✓ 일반적으로 DBMS는 각 트랜잭션에서 접근하는 투플 수에 따라 자동적으로 로크 단위를 조정함
  - ✓ 로크 단위가 작을수록 로킹에 따른 오버헤드가 증가함
  - ✓ 로크 단위가 작을수록 동시성의 정도는 증가함

9.2 동시성 제어(계속)

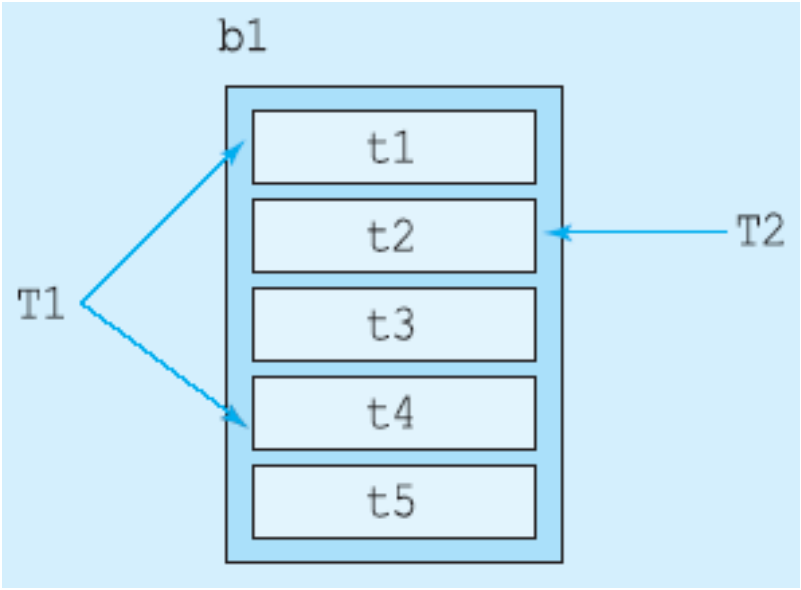


[그림 9.15] 다중 로크 단위

9.2 동시성 제어(계속)

예: 다중 단위 로크

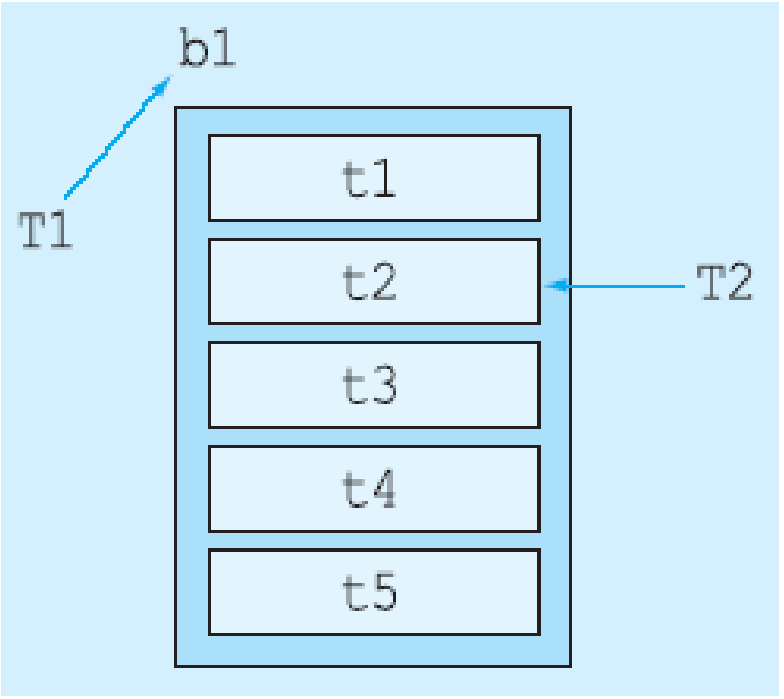
EMPLOYEE 릴레이션에 속하는 디스크 블록 b1에 다섯 개의 투플 t1, t2, t3, t4, t5가 있다고 가정하자. 또한 트랜잭션 T1은 이 중에서 투플 t1과 t4를 갱신하고, 트랜잭션 T2는 투플 t2를 검색한다고 가정하자. 만일 로크 단위가 투플이라면 두 트랜잭션이 접근하는 투플들이 서로 상이하므로 해당 투플에 로크를 걸고 두 트랜잭션이 동시에 수행될 수 있다.



9.2 동시성 제어(계속)

예: 다중 단위 로크

트랜잭션 T1은 블록 단위, T2는 투플 단위로 로크를 하는 경우에, 먼저 T1이 블록 b1에 대해 독점 로크를 요청하여 허가를 받으면 이 블록에 들어 있는 투플 다섯 개에도 모두 독점 로크가 걸린다. 그 다음에 트랜잭션 T2가 투플 t2에 대해 공유 로크를 요청하면 트랜잭션 T1이 로크를 풀 때까지 기다려야 한다.



9.2 동시성 제어(계속)

- 팬텀 문제(phantom problem)
  - ✓ 두 개의 트랜잭션 T1과 T2가 그림 4.8의 EMPLOYEE 릴레이션에 대해서 아래와 같은 순서대로 수행된다고 가정 (그림 9.16)
  - ✓ 트랜잭션 T1: EMPLOYEE 릴레이션에서 1번 부서에 근무하는 사원들의 이름을 검색하는 동일한 SELECT문을 두 개 포함
  - ✓ 트랜잭션 T2: 1번 부서에 근무하는 사원 투플을 한 개 삽입하는 INSERT문을 포함

9.2 동시성 제어(계속)

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)(계속)

- ✓ 시간 1에 트랜잭션 T1의 SELECT문이 수행되면 1번 부서에 근무하는 사원들의 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 문제라고 부름

9.3 회복

❑ 회복의 필요성

- ✓ 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

- ✓ 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

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

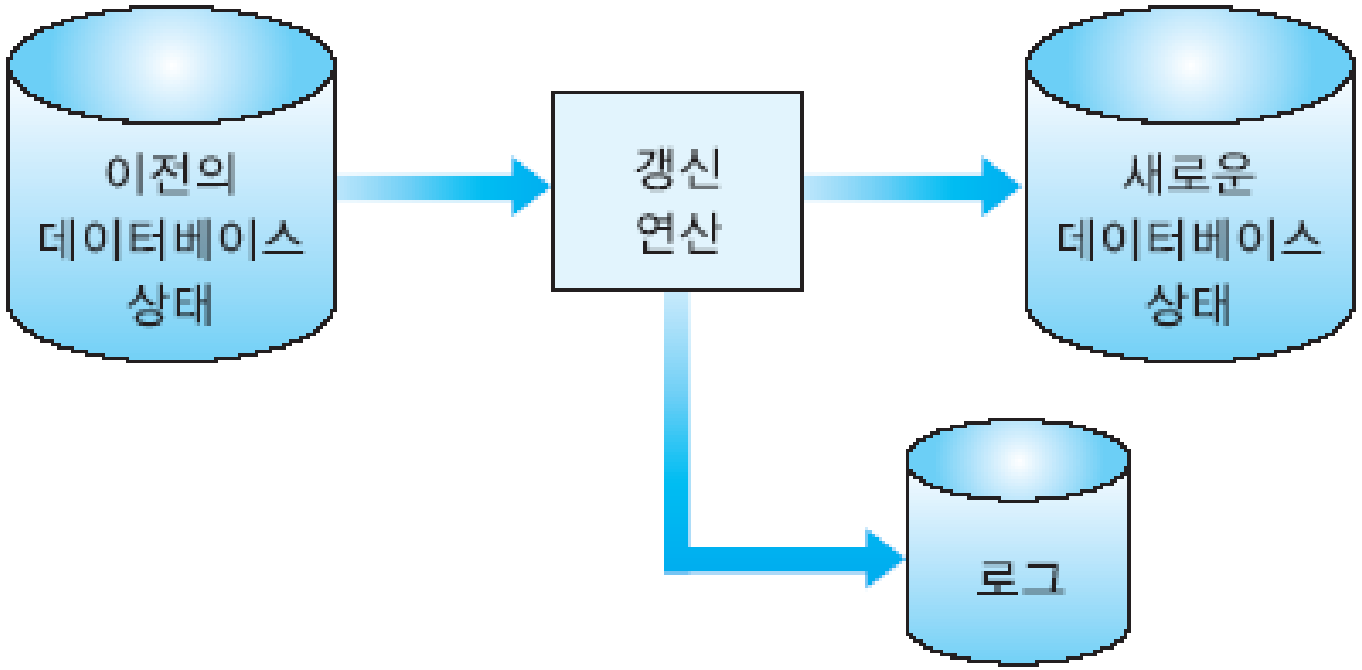
- 그 이외의 고장
- 대부분의 회복 알고리즘들은 비재해적 고장에 적용됨
- 로그를 기반으로 한 즉시 갱신, 로그를 기반으로 한 지연 갱신, 그림자 페이징(shadow paging) 등 여러 알고리즘
- 대부분의 상용 DBMS에서 로그를 기반으로 한 즉시 갱신 방식을 사용

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 로그 (log)라고 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

9.3 회복(계속)



[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)

❑ 로그 레코드의 유형

- ✓ [Trans-ID, start]
  - 한 트랜잭션이 생성될 때 기록되는 로그 레코드
- ✓ [Trans-ID, X, old_value, new_value]
  - 주어진 Trans_ID를 갖는 트랜잭션이 데이터 항목 X를 이전값(old_value)에서 새값(new_value)로 수정했음을 나타내는 로그 레코드
- ✓ [Trans-ID, commit]
  - 주어진 Trans_ID를 갖는 트랜잭션이 데이터베이스에 대한 갱신을 모두 성공적으로 완료하였음을 나타내는 로그 레코드
- ✓ [Trans-ID, abort]
  - 주어진 Trans_ID를 갖는 트랜잭션이 철회되었음을 나타내는 로그 레코드

9.3 회복(계속)

예 : 트랜잭션의 로그 레코드

아래의 두 트랜잭션 T1과 T2를 고려해 보자. T1 다음에 T2가 수행되고, 데이터베이스 항목 A, B, C, D, E의 초기값은 각각 100, 300, 5, 60, 80이라고 가정한다. 이 두 트랜잭션을 수행하면 표 9.3과 같은 로그 레코드들이 생성된다. 표 9.3에서 2번 로그 레코드는 트랜잭션 T1이 데이터베이스 항목 B를 이전값 300에서 새값 400으로 갱신했음을 나타낸다. 일단 이 로그 레코드가 디스크의 로그에 기록된 후에는 B가 새값으로 고쳐진 주기억 장치의 버퍼가 언제든지 디스크의 데이터베이스에 기록될 수 있다.

```
T1:   read_item(A);          T2:   read_item(A);
      A = A + 30;                  A = A + 10;
      read_item(B);                write_item(A);
      B = B + 100;                 read_item(D);
      write_item(B);               D = D - 10;
      read_item(C);                read_item(E);
      C = 2 * C;                   read_item(B);
      write_item(C);               E = E + B;
      A = A + B + C;               write_item(E);
      write_item(A);               D = D + E;
                                   write_item(D);
```

9.3 회복(계속)

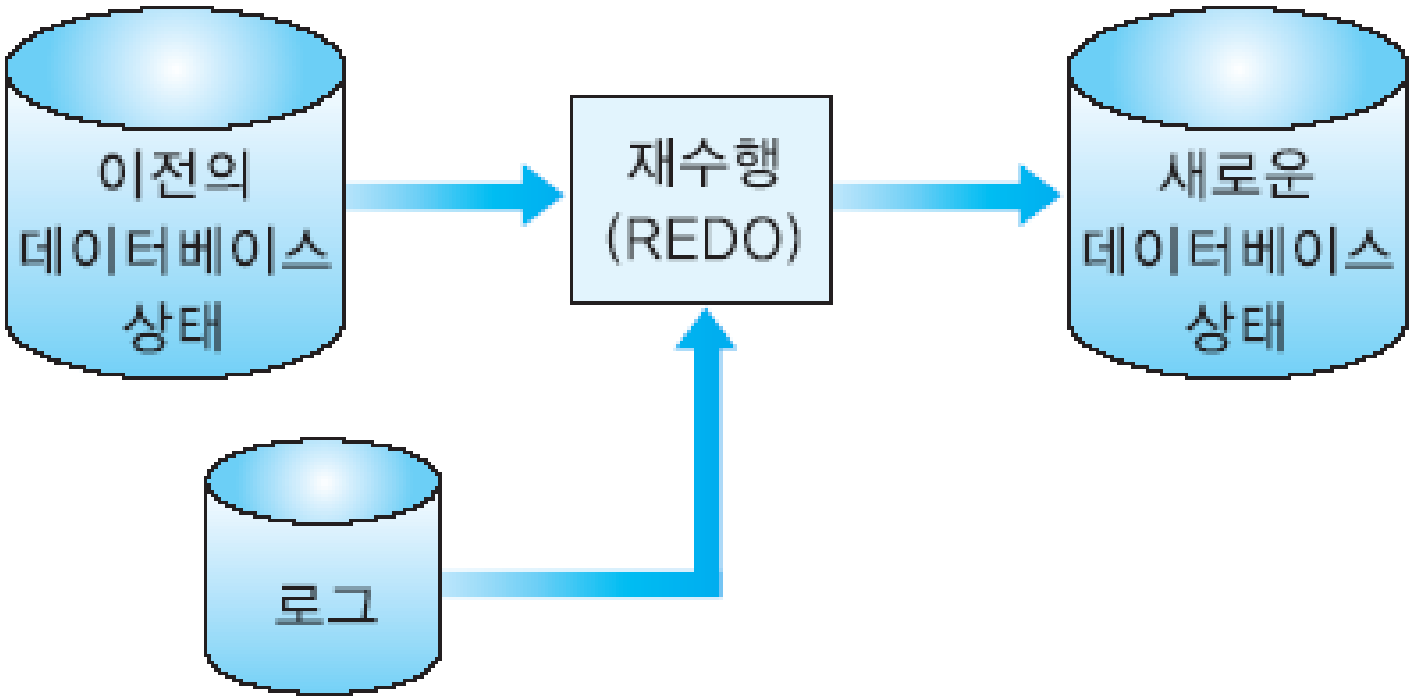
〈표 9.3〉 로그 레코드의 예

로그 순서 번호	로그 레코드
1	[T1, start]
2	[T1, B, 300, 400]
3	[T1, C, 5, 10]
4	[T1, A, 100, 540]
5	[T1, commit]
6	[T2, start]
7	[T2, A, 540, 550]
8	[T2, E, 80, 480]
9	[T2, D, 60, 530]
10	[T2, commit]

9.3 회복(계속)

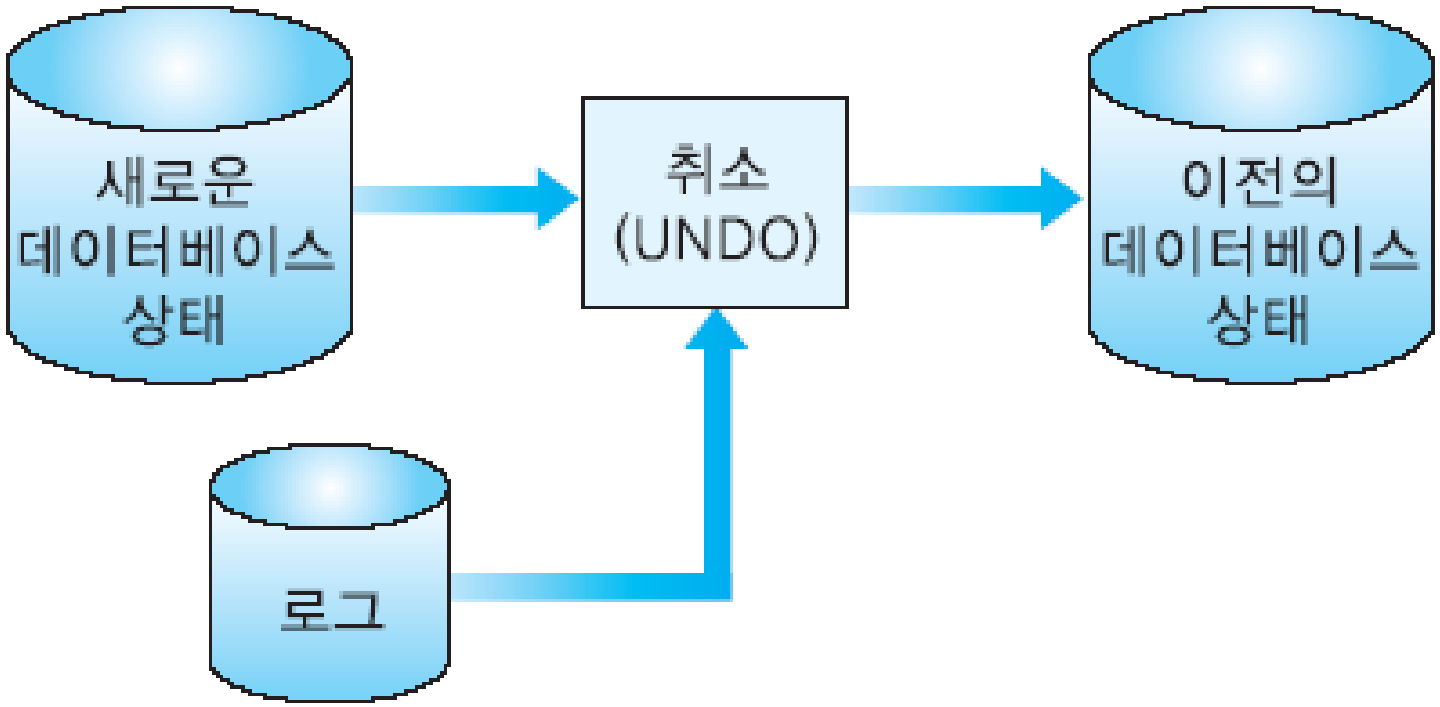
- 트랜잭션의 완료점(commit point)
  - ✓ 한 트랜잭션의 데이터베이스 갱신 연산이 모두 끝나고 데이터베이스 갱신 사항이 로그에 기록되었을 때
  - ✓ DBMS의 회복 모듈은 로그를 검사하여 로그에 [Trans-ID, start] 로그 레코드와 [Trans-ID, commit] 로그 레코드가 모두 존재하는 트랜잭션들은 재수행
  - ✓ [Trans-ID, start] 로그 레코드는 로그에 존재하지만 [Trans-ID, commit] 로그 레코드가 존재하지 않는 트랜잭션들은 취소

9.3 회복(계속)



[그림 9.17] 트랜잭션의 재수행

9.3 회복(계속)



[그림 9.18] 트랜잭션의 취소

9.3 회복(계속)

〈표 9.4〉 재수행할 트랜잭션과 취소할 트랜잭션

마지막으로 기록된 로그 순서 번호	작업	결과
$l = 0$	아무 작업도 필요 없음	T1과 T2가 수행을 시작하기 전
$1 \le l \le 4$	– T1을 취소함: 1부터 l까지 T1이 생성한 로그 레코드의 이전값을 사용하여 데이터베이스 항목의 값을 되돌림	T1이 수행을 하지 않은 것과 같음
$5 \le l \le 9$	– T1을 재수행: 1부터 4까지 T1이 생성한 로그 레코드의 새값을 사용하여 데이터베이스 항목의 값을 기록함 – T2를 취소함: 5부터 l까지 T2가 생성한 로그 레코드의 이전값을 사용하여 데이터베이스 항목의 값을 되돌림	– T1의 수행이 완료됨 – T2는 수행을 하지 않은 것과 같음
10	T1과 T2를 재수행	T1과 T2의 수행이 완료됨

9.3 회복(계속)

- 로그 먼저 쓰기(WAL: Write-Ahead Logging)
  - ✓ 트랜잭션이 데이터베이스를 갱신하면 주기억 장치의 데이터베이스 버퍼에 갱신 사항을 기록하고, 로그 버퍼에는 이에 대응되는 로그 레코드를 기록함
  - ✓ 만일 데이터베이스 버퍼가 로그 버퍼보다 먼저 디스크에 기록되는 경우에는 로그 버퍼가 디스크에 기록되기 전에 시스템이 다운되었다가 재기동되었을 때 주기억 장치는 휘발성이므로 데이터베이스 버퍼와 로그 버퍼의 내용은 전혀 남아 있지 않음
  - ✓ 로그 레코드가 없어서 이전값을 알 수 없으므로 트랜잭션의 취소가 불가능함
  - ✓ 따라서 데이터베이스 버퍼보다 로그 버퍼를 먼저 디스크에 기록해야 함

9.3 회복(계속)

- 체크포인트(checkpoint) 필요성
  - ✓ 시스템이 다운된 시점으로부터 오래 전에 완료된 트랜잭션들이 데이터베이스를 갱신한 사항은 이미 디스크에 반영되었을 가능성이 큼
  - ✓ DBMS가 로그를 사용하더라도 어떤 트랜잭션의 갱신 사항이 주기억 장치 버퍼로부터 디스크에 기록되었는가를 구분할 수 없음
  - ✓ 따라서 DBMS는 회복시 재수행할 트랜잭션의 수를 줄이기 위해서 주기적으로 체크포인트를 수행함
- 체크포인트 전략
  - ✓ 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)

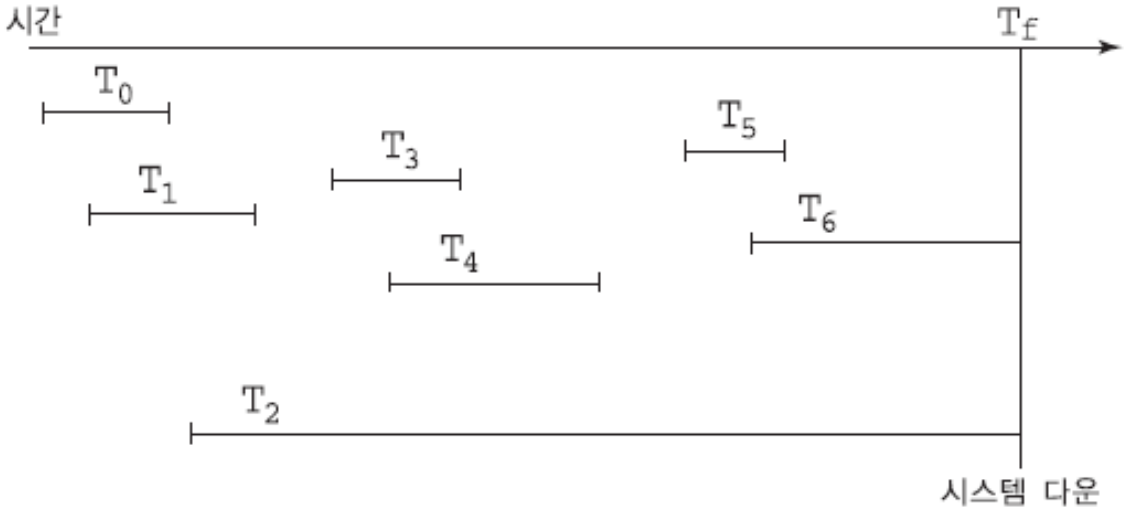
❑ 체크포인트를 할 때 수행되는 작업

- ✓ 수행 중인 트랜잭션들을 일시적으로 중지시킴. 회복 알고리즘에 따라서는 이 작업이 필요하지 않을 수 있음
- ✓ 주기억 장치의 로그 버퍼를 디스크에 강제로 출력
- ✓ 주기억 장치의 데이터베이스 버퍼를 디스크에 강제로 출력
- ✓ [checkpoint] 로그 레코드를 로그 버퍼에 기록한 후 디스크에 강제로 출력
- ✓ 체크포인트 시점에 수행 중이던 트랜잭션들의 ID도 [checkpoint] 로그 레코드에 함께 기록
- ✓ 일시적으로 중지된 트랜잭션의 수행을 재개

9.3 회복(계속)

예: 체크포인트를 하지 않았을 때

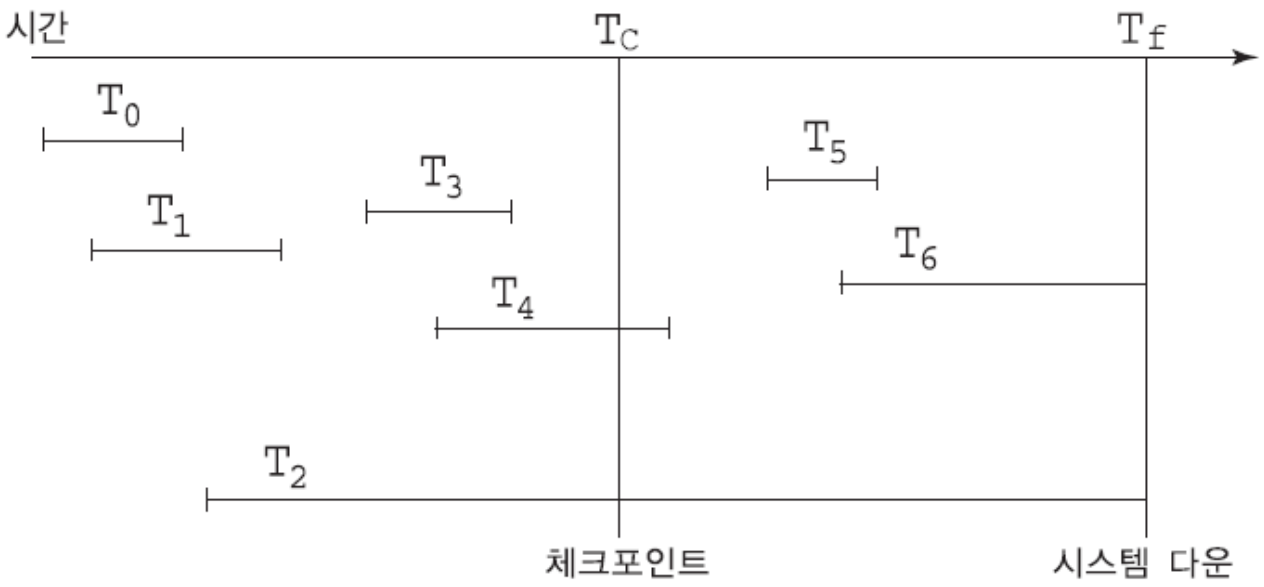
그림 9.19는 시스템이 다운된 후에 재기동되었을 때 회복 모듈이 디스크에 저장되어 있는 로그 레코드를 조사하여 얻은 것. 트랜잭션 T0, T1, T3, T4, T5는 재수행. 트랜잭션 T2, T6은 취소.



[그림 9.19] 체크포인트가 없는 로그

9.3 회복(계속)

예: 체크포인트를 했을 때
그림 9.20도 디스크에 저장되어 있는 로그 레코드를 조사하여 얻은 것. 트랜잭션 T0, T1, T3은 무시. 트랜잭션 T4, T5는 재수행. 트랜잭션 T2, T6은 취소.



[그림 9.20] 체크포인트가 있는 로그

9.3 회복(계속)

- 데이터베이스 백업과 재해적 고장으로부터의 회복
  - ✓ 아주 드물지만, 데이터베이스가 저장되어 있는 디스크의 헤드 등이 고장나서 데이터베이스를 읽을 수 없는 경우가 발생함
  - ✓ 이런 경우에 데이터베이스를 회복하는 한 가지 방법은 주기적으로 자기 테이프에 전체 데이터베이스와 로그를 백업하고, 자기 테이프를 별도의 공간에 안전하게 보관하는 것
  - ✓ 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

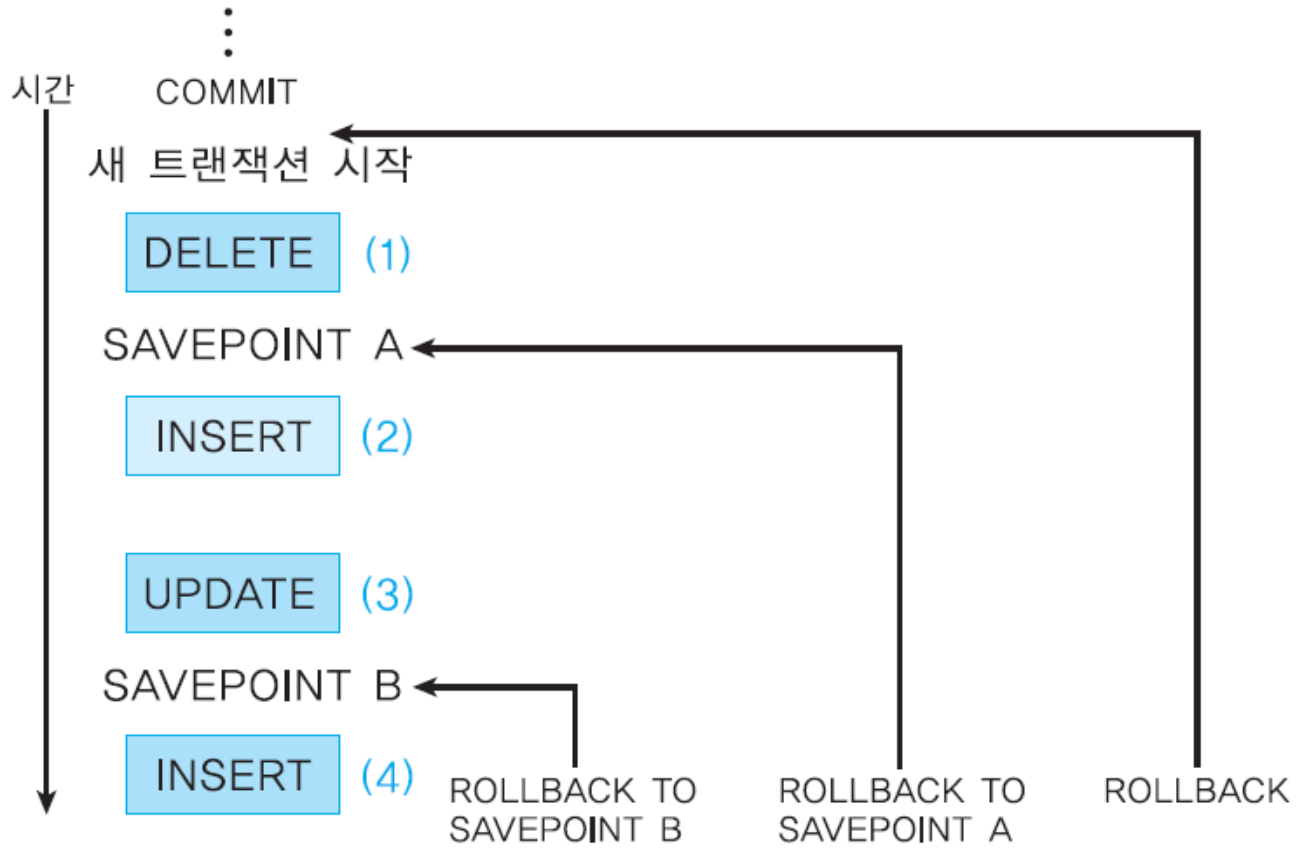
- ✓ 오라클에서 한 트랜잭션은 암시적으로 끝나거나 명시적으로 끝날 수 있음
- ✓ 한 트랜잭션은 실행 가능한 첫 번째 SQL문이 실행될 때 시작되어 데이터 정의어를 만나거나, 데이터 제어어를 만나거나, COMMIT이나 ROLLBACK 없이 Oracle SQL Developer를 정상적으로 종료했을 때는 수행 중이던 트랜잭션이 암시적으로 완료(commit)됨

9.4 PL/SQL의 트랜잭션(계속)

❑ 트랜잭션의 시작과 끝(계속)

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

9.4 PL/SQL의 트랜잭션(계속)



[그림 9.22] 트랜잭션 제어

9.4 PL/SQL의 트랜잭션(계속)

❑ 트랜잭션의 시작과 끝(계속)

✓ SQL*Plus에서는 묵시적으로 한 트랜잭션은 데이터 정의어나 데이터 제어어 이전까지 입력한 여러 개의 데이터 조작어로 이루어짐

✓ set 명령을 사용하여 각 데이터 조작어를 한 트랜잭션으로 처리할 수 있음

```
set auto on;
```

또는

```
set autocommit on;
```

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';
```

9.4 PL/SQL의 트랜잭션(계속)

❑ 트랜잭션의 속성(계속)

✓ 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, UPDATE문을 모두 수행할 수 있음

```
SET TRANSACTION READ WRITE;
UPDATE   EMPLOYEE
SET      SALARY = SALARY * 1.06;
```

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 (다른 트랜잭션의 고립 수준에 영향을 미치지 않음)

9.4 PL/SQL의 트랜잭션(계속)

- 오라클에서 제공하는 몇 가지 고립 수준
  - ✓ READ UNCOMMITTED
    - 가장 낮은 고립 수준
    - 트랜잭션 내의 질의들이 공유 로크를 걸지 않고 데이터를 읽음
    - 따라서 오손 데이터를 읽을 수 있음
    - 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

- 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

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

9.4 PL/SQL의 트랜잭션(계속)

- 오라클에서 제공하는 몇 가지 고립 수준(계속)
  - ✓ REPEATABLE READ
    - 질의에서 검색되는 데이터에 대해 공유 로크를 걸고, 트랜잭션이 끝날 때까지 보유함
    - 한 트랜잭션 내에서 동일한 질의를 두 번 이상 수행할 때, 이전에 읽은 값이 항상 동일하게 유지됨
    - 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

- 가장 높은 고립 수준
- 질의에서 검색되는 투플들 뿐만 아니라 인덱스에 대해서도 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
- SERIALIZABLE은 SQL2의 디폴트 고립 수준

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

9.4 Transact-SQL의 트랜잭션(계속)

❑ REPEATABLE READ vs. SERIALIZABLE

SELECT col1 FROM A WHERE col1 BETWEEN 1 AND 10;

✓ REPEATABLE READ

- 이 범위에 해당하는 데이터가 2건이 있는 경우(col1=1과 5) 다른 사용자가 col1이 1이나 5인 Row에 대한 UPDATE가 불가능
- col1이 1과 5를 제외한 나머지 이 범위에 해당하는 Row를 INSERT하는 것은 가능

✓ SERIALIZABLE

- 영역에 해당되는 데이터에 대한 수정 및 입력이 불가능
- SQL의 결과가 항상 동일함

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