Databases
트랜잭션

한국산업기술대학교
게임공학과
장 지 웅

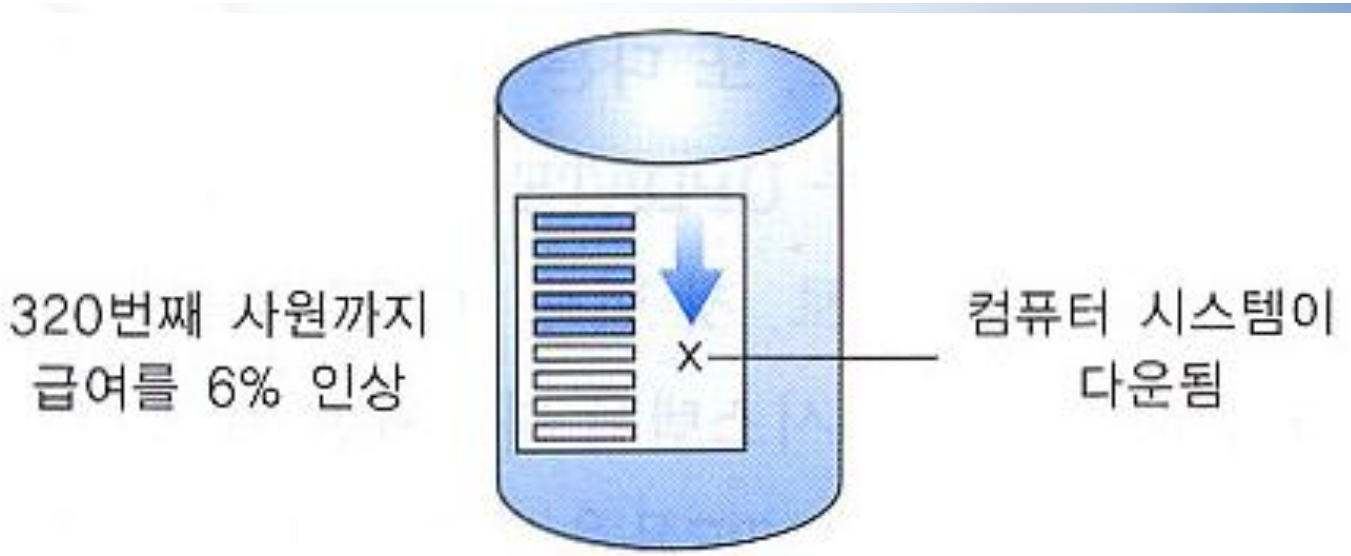
Contents

데이타베이스 응용의 오작동

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 8.1] 데이터베이스를 갱신하는 중에 컴퓨터 시스템의 다운

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';
```

두개의 질의문 중 하나만 수행된다면 어떻게 될까?

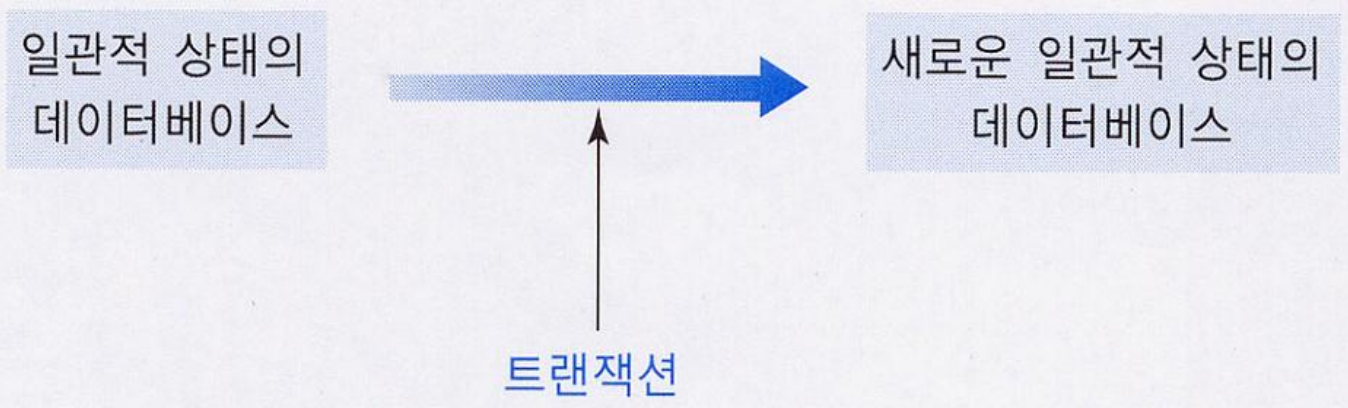

트랜잭션의
ACID 특성
A
원자성
C
일관성
I
고립성
D
지속성

원자성(Atomicity)

모두 수행되거나 혹은
전혀 수행되지 않거나…

일관성(Consistency)

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



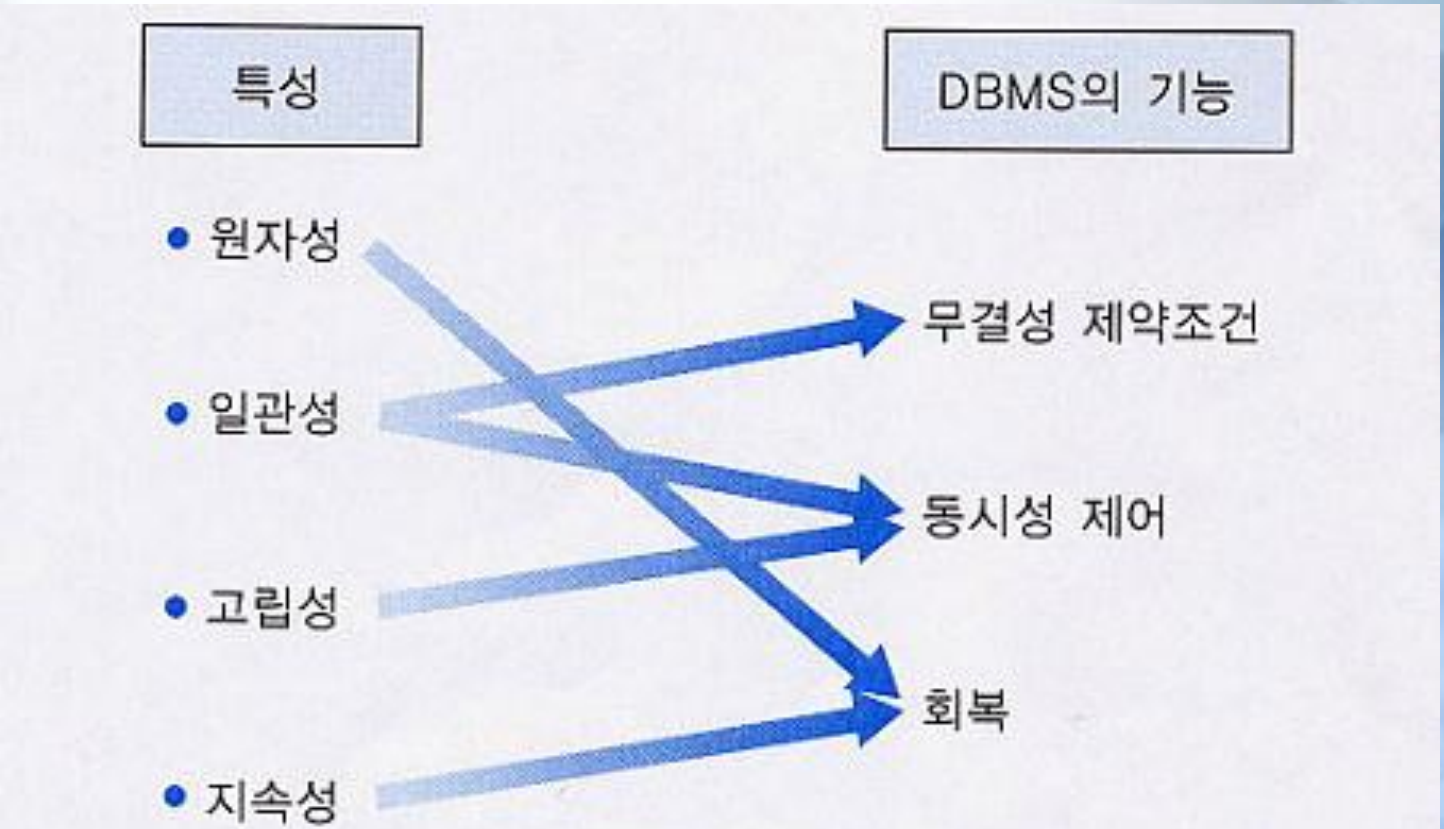
고립성(Isolation)

여러 트랜잭션이 동시에 수행되더라도 마치 혼자 수행한 것과 같아야 함

직렬가능성(serializability): 다수의 트랜잭션들이 동시에 수행되더라도 그 결과는 어떤 순서에 따라 트랜잭션들을 하나씩 차례대로 수행한 결과와 같아야 함

지속성(Durability)

트랜잭션이 완료되면 무슨일이 있어도
그 결과를 유지함

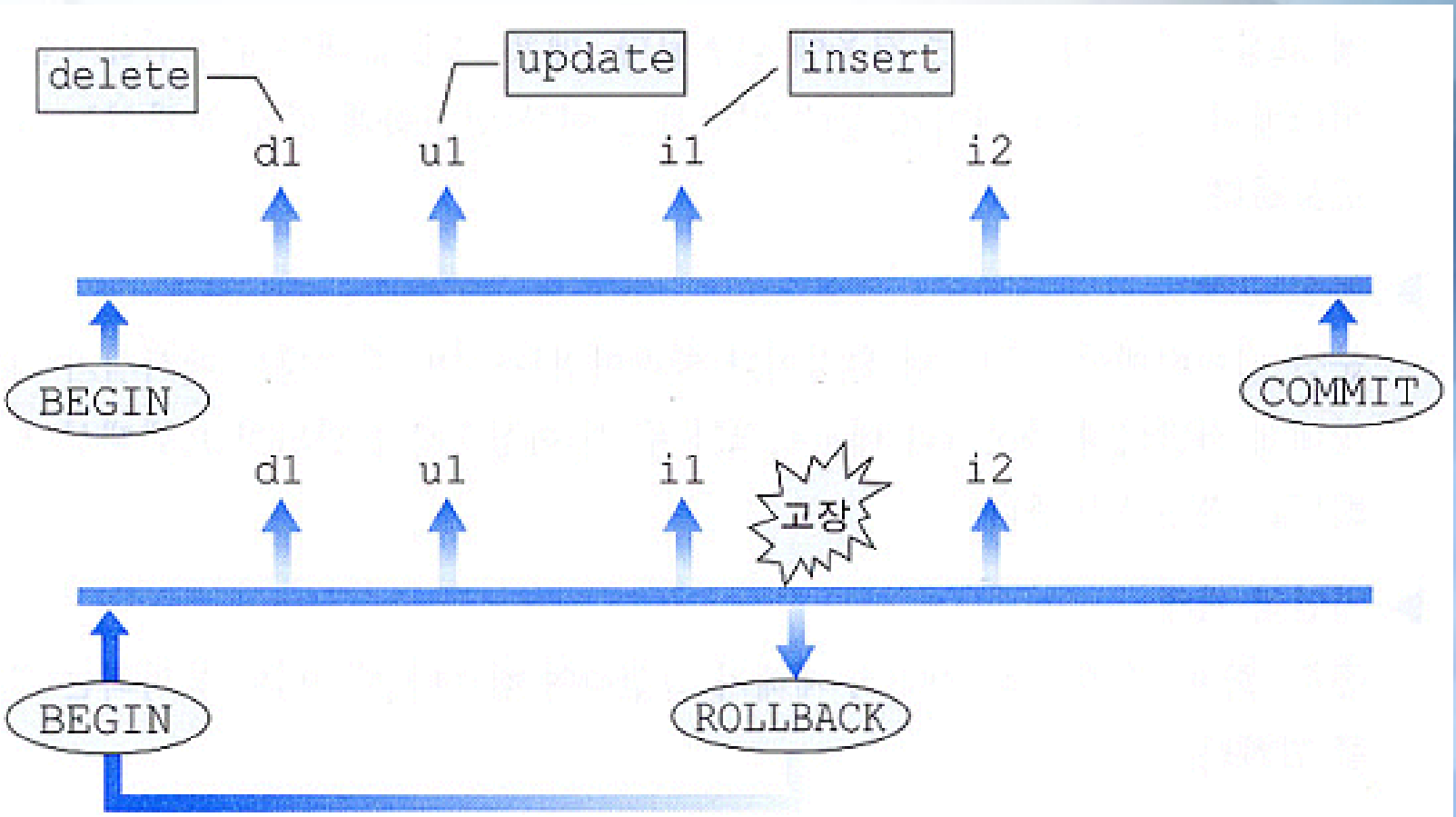


[그림 8.3] 트랜잭션의 네 가지 특성과 DBMS의 기능과의 관계

COMMIT과 ROLLBACK(ABORT)

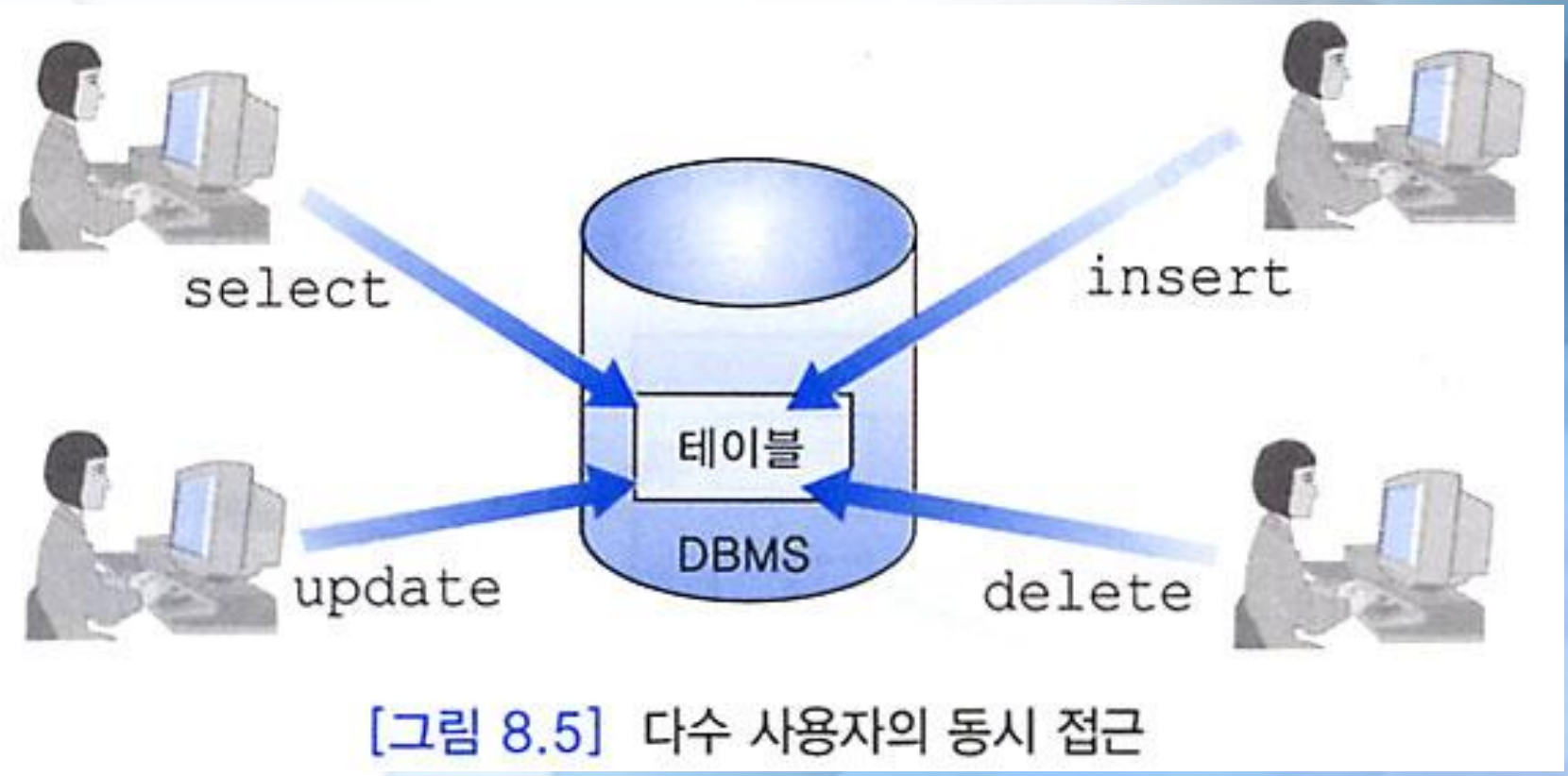
〈표 8.1〉 COMMIT과 ROLLBACK의 비교

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



[그림 8.4] COMMIT과 ROLLBACK

동시성 제어



[그림 8.5] 다수 사용자의 동시 접근

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

동시성 제어 오류

1

수행 중인 트랜잭션이 갱신한 내용을 다른 트랜잭션이 덮어씀으로써 갱신이 무효가 되는 것

갱신 손실
(lost update):

2

완료되지 않은 트랜잭션이 갱신한 데이터를 읽는 것

오손 데이터 읽기
(dirty read)

3

한 트랜잭션이 동일한 데이터를 두 번 읽을 때 서로 다른 값을 읽는 것

반복할 수 없는 읽기
(unrepeatable read)

갱신손실의 예

T1 : 100000원 이체
T2 : 50000원 입금

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 8.7] 갱신 손실

오손데이타 읽기의 예

T1 : 100000원 출금 명령 후 철회
T2 : 계좌의 평균값 검색

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

[그림 8.8] 오손 데이터 읽기

반복할 수 없는 읽기의 예

T1 : 100000원 출금
T2 : 계좌의 평균값 검색 2회 연속 수행

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 8.9] 반복할 수 없는 읽기

동시성 제어 기법 – 로킹(Locking)

- 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
  - 갱신할 때 : 독점 로크(X-lock : Exclusive lock)
  - 읽을 때 : 공유 로크(S-lock : Shared lock)
  - 트랜잭션이 데이터 항목에 대한 접근을 끝낸 후에 로크를 해제(unlock)함

〈표 8.2〉 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

로킹만으로 모든 문제가 해결될까?

T1	T2
X-lock(A); read_item(A); A=A+1; write_item(A) unlock(A)	

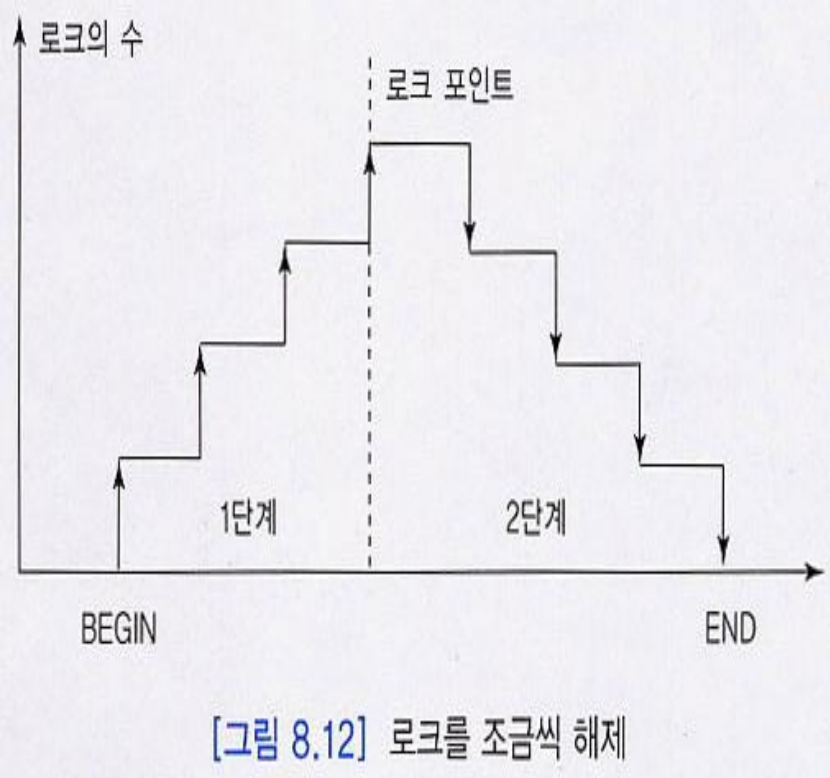
	X-lock(A);
	read_item(A);
	A=A * 2;
	unlock(A);
	X-lock(B);
	read_item(B);
	B=B * 2;
	unlock(B);

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

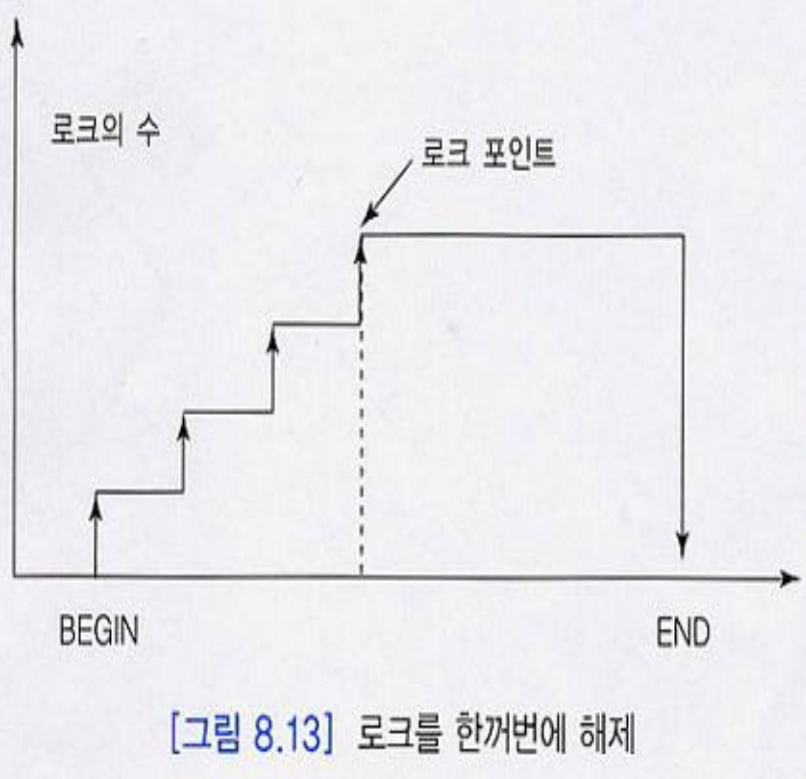
[그림 8.11] 로크를 일찍 해제

2단계 로킹 프로토콜 (2-phase locking protocol)

- 로크 확장단계 : 로크를 획득할 수만 있음
- 로크 수축단계 : 로크를 해제할 수만 있음
  - 로크를 한 개라도 해제하면 로크 수축 단계에 들어감



[그림 8.12] 로크를 조금씩 해제

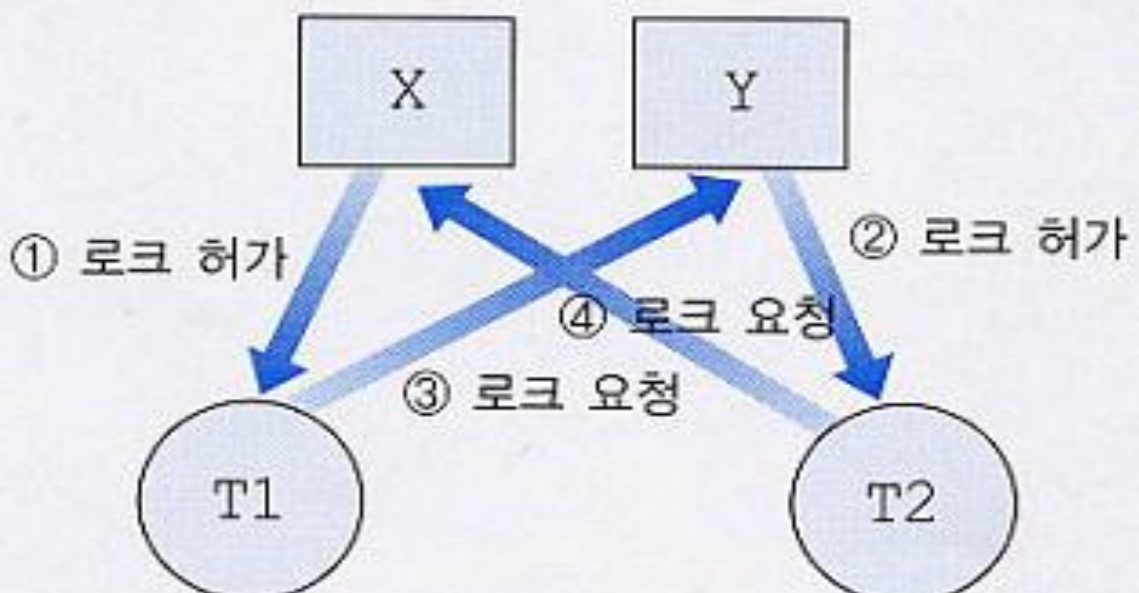


[그림 8.13] 로크를 한꺼번에 해제

2단계 로킹 프로토콜의 문제점

- **데드록(deadlock)**

  : 두 개 이상의 트랜잭션들이 서로 상대방이 보유하고 있는 로크를 요청하면서 기다리고 있는 상태

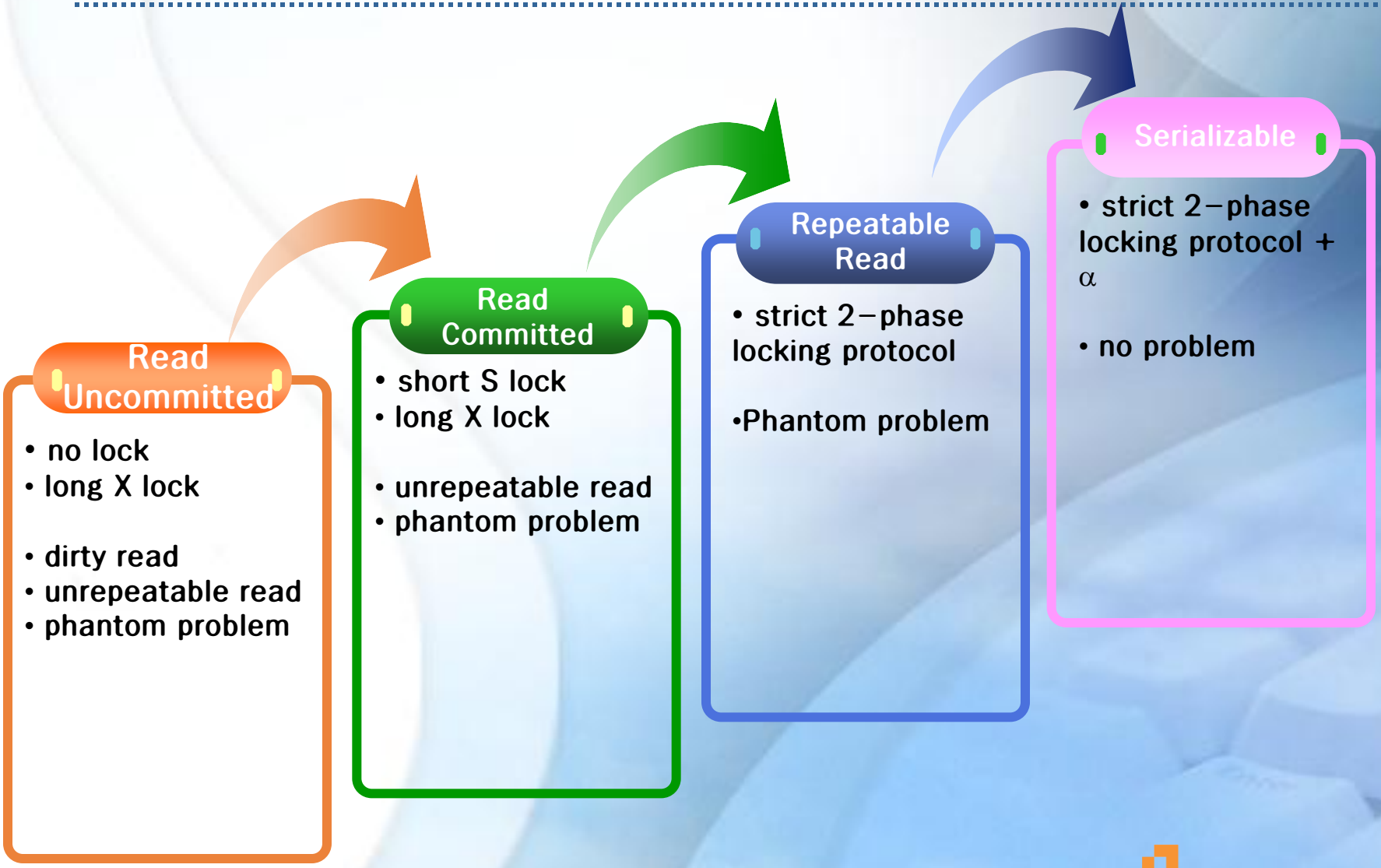


[그림 8.14] 데드록의 예

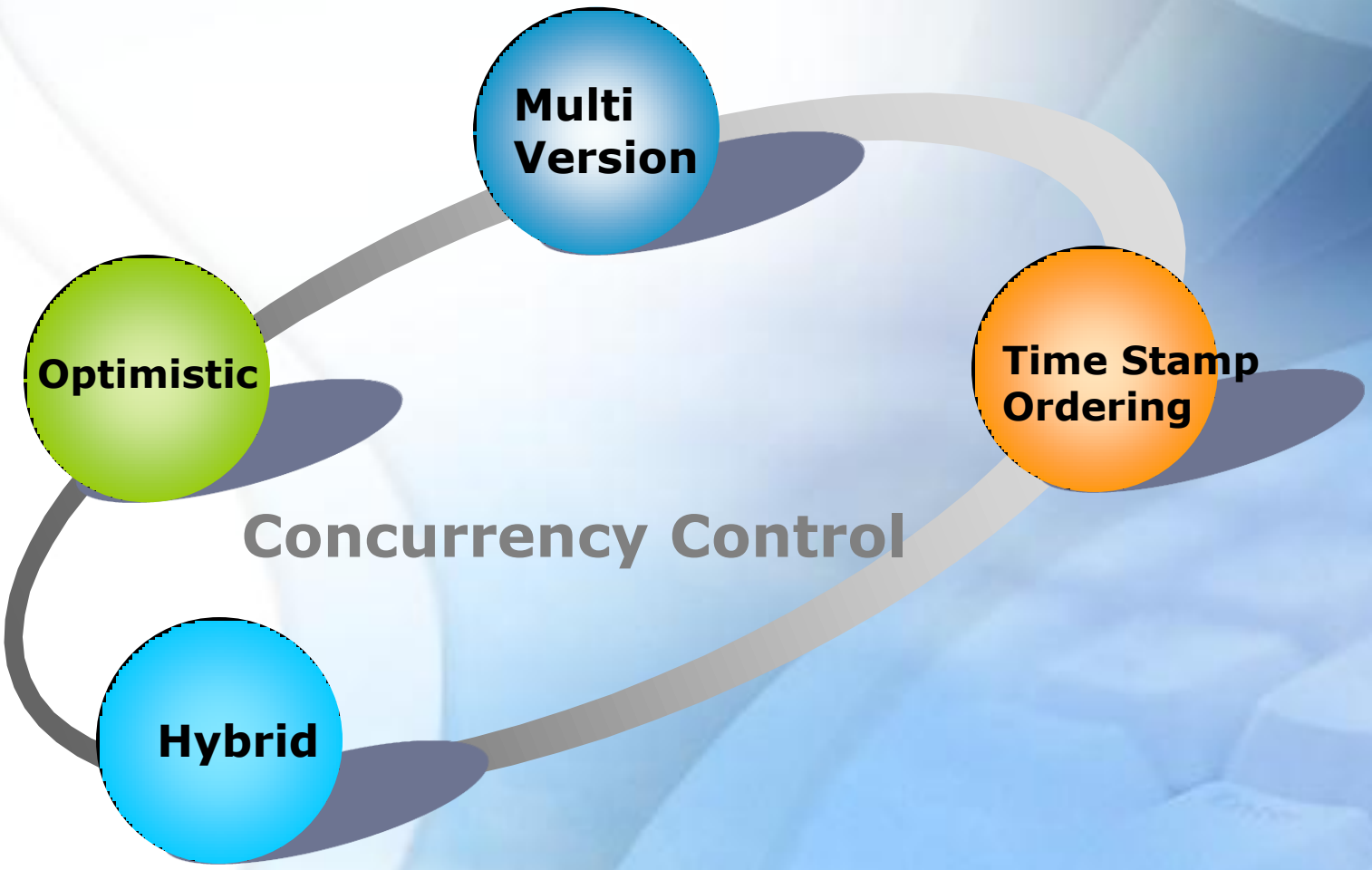
고립화 수준(Isolation Level)

한 트랜잭션이 다른 트랜잭션과 고립되는 정도
- 고립 수준이 낮으면 동시성은 높아지지만 데이터의 정확성은 떨어짐
- 고립 수준이 높으면 데이터가 정확해지지만 동시성이 저하됨

고립화 수준의 종류

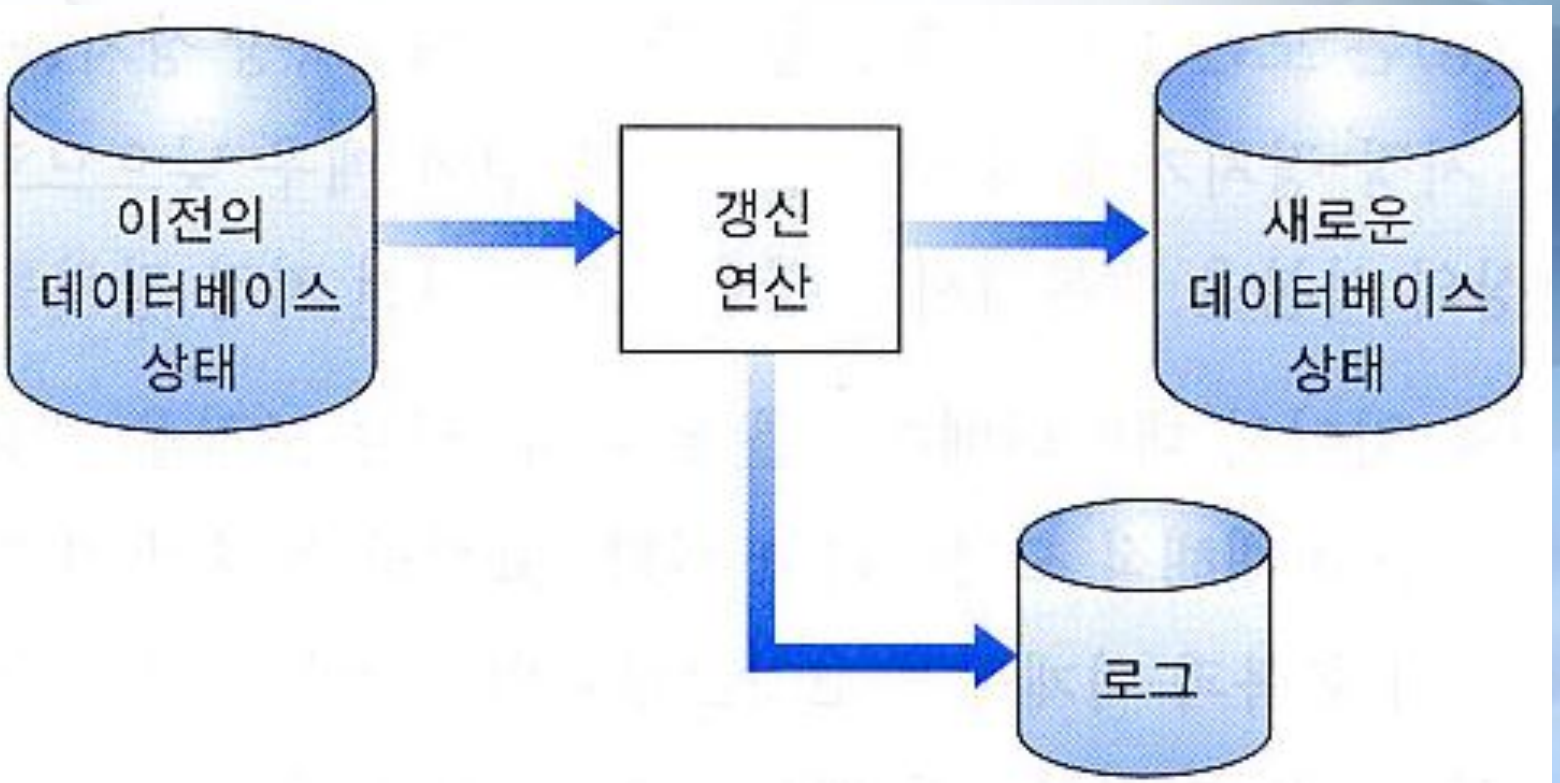

Read Uncommitted
• no lock
• long X lock
• dirty read
• unrepeatable read
• phantom problem
Read Committed
• short S lock
• long X lock
• unrepeatable read
• phantom problem
Repeatable Read
• strict 2-phase locking protocol
•Phantom problem
Serializable
• strict 2-phase locking protocol + α
• no problem

그밖의 동시성 제어 방법들



회복

- 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



[그림 8.16] 로그 생성

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 8.3과 같은 로그 레코드들이 생성된다. 표 8.3에서 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:   A = A + 30;          T2:   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(B);
      write_item(A);               D = D + E;
                                   write_item(D);
```

〈표 8.3〉 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, 570]
8	[T2, E, 80, 480]
9	[T2, 60, 530]
10	[T2, commit]

생각해 봅시다.

로그와 실제 데이터 중 어느 것을 디스크에 먼저 써야 할까?

생각해 봅시다.

1. 파손 발생 후 어느 트랜잭션을 회복해야 하는지 알 수 있을까?

2. 회복 시에 수행할 작업을 어떻게 줄일 수 있을까?

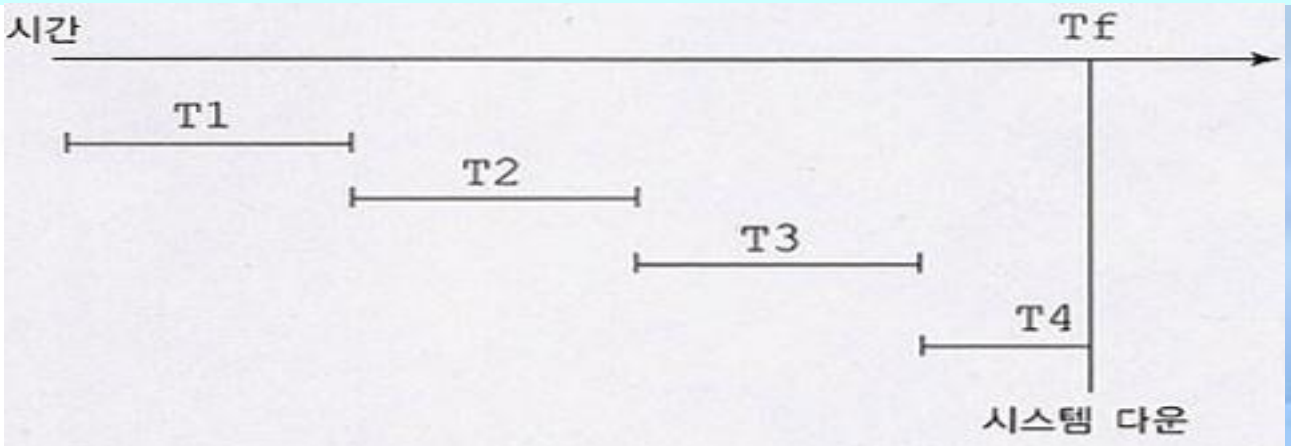
체크포인트

- 모든 버퍼의 내용을 강제로 디스크에 반영하는 작업
- 체크포인트 작업이 끝나면 로그에 [checkpoint] 로그 레코드가 기록됨

- 체크포인트를 할 때 수행되는 작업
  - 수행 중인 트랜잭션을 일시적으로 중지시킨다.
  - 주기억 장치의 로그 버퍼를 디스크에 강제로 출력한다.
  - 주기억 장치의 데이터베이스 버퍼를 디스크에 강제로 출력한다.
  - [checkpoint] 로그 레코드를 로그 버퍼에 기록한 후 디스크에 강제로 출력한다. 체크포인트 시점에 수행 중이던 트랜잭션들의 ID도 [checkpoint] 로그 레코드에 함께 기록한다.
  - 일시적으로 중지된 트랜잭션의 수행을 재개한다.

예: 체크포인트를 하지 않았을 때

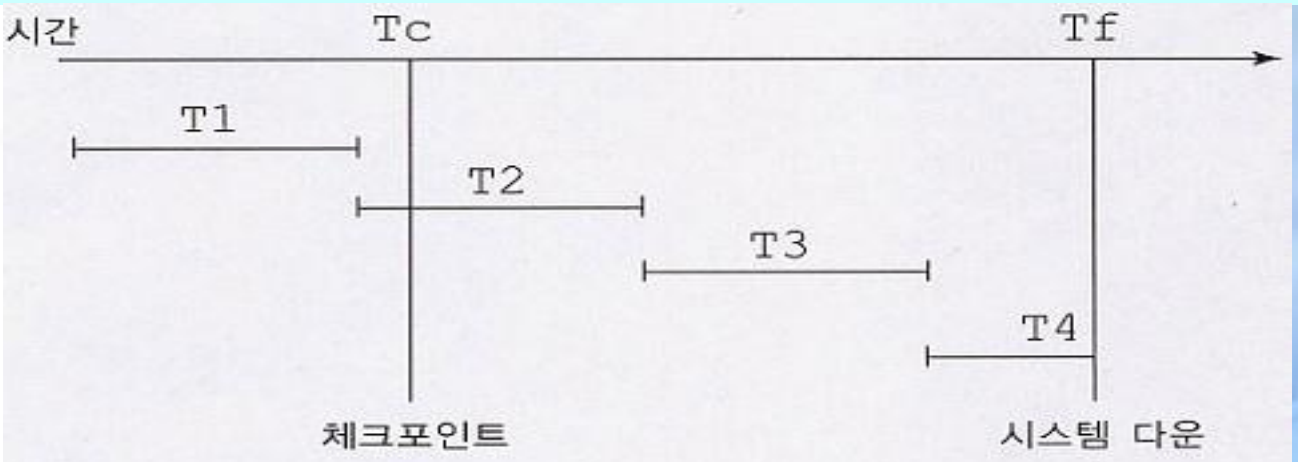
그림 8.19는 시스템이 다운된 후에 재기동되었을 때 회복 모듈이 디스크에 저장되어 있는 로그 레코드를 조사하여 얻은 것이다. 시스템이 다운되기 전에 트랜잭션 T1, T2, T3이 완료되었지만, T1, T2, T3이 재수행된다. 분명히 트랜잭션 T4는 시스템이 다운될 시점에 수행 중이었으므로 완료되지 않은 트랜잭션이다. 즉시 갱신 방식에서는 트랜잭션이 완료되기 전이라도 일단 로그 버퍼가 디스크에 기록된 후에는 언제든지 데이터베이스 버퍼가 디스크에 기록될 수 있으므로 트랜잭션 T4의 갱신 사항이 디스크의 데이터베이스에 일부 반영되었을 수 있다. 따라서 트랜잭션 T4는 취소한다.



[그림 8.19] 체크포인트가 없는 로그

예: 체크포인트를 했을 때

그림 8.20도 디스크에 저장되어 있는 로그 레코드를 조사하여 얻은 것이다. 트랜잭션 T1은 체크포인트 이전에 수행이 완료되었으므로 이미 로그 버퍼와 데이터베이스 버퍼가 디스크에 반영되었다. 따라서 회복 모듈은 트랜잭션 T1을 무시한다. 시스템이 다운되기 전에 트랜잭션 T2와 T3이 완료되었지만 트랜잭션 T2는 체크포인트 시점에 수행 중이었고 트랜잭션 T3은 체크포인트 이후에 수행을 시작했으므로 T2와 T3이 생성한 데이터베이스 버퍼가 디스크에 기록되었는지 DBMS가 알 수 없다. 따라서 T2와 T3은 재수행한다. 트랜잭션 T4는 위의 예와 같이 취소한다.



[그림 8.20] 체크포인트가 있는 로그