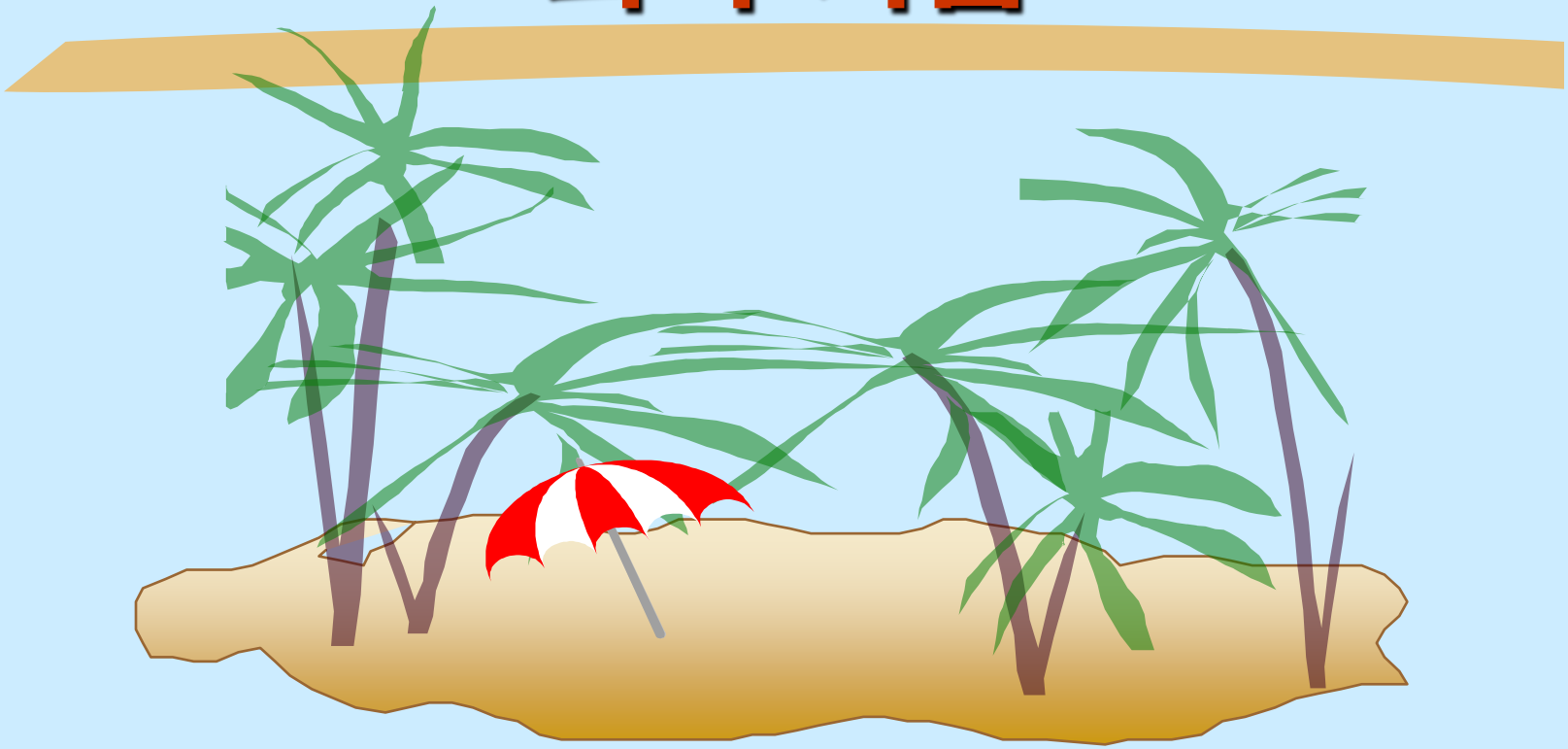
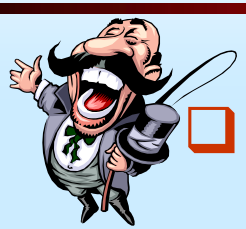
회복 기법



안동대학교 정보과학교육과

순서



- 장애와 회복
- 로그 이용 회복
- 검사시점 회복
- 미디어 회복

장애와 회복(I)

- 회복
  - 데이터베이스를 장애발생 이전의 일관된 상태로 복원시키는 것
- 일관된 상태(consistent state)
  - 데이터베이스에 오류가 없는 상태, 데이터베이스의 내용에 모순이 없는 상태
- 장애
  - 시스템이 정해진 명세대로 작동하지 않는 상태
  - 원인 : 하드웨어 결함, 소프트웨어의 논리오류, 사람의 실수
- 장애의 유형
  - 트랜잭션 장애 : 논리적 오류 입력 데이터의 불량
  - 시스템 장애 : 하드웨어의 오동작
  - 미디어 장애 : 디스크 헤드 붕괴 또는 고장

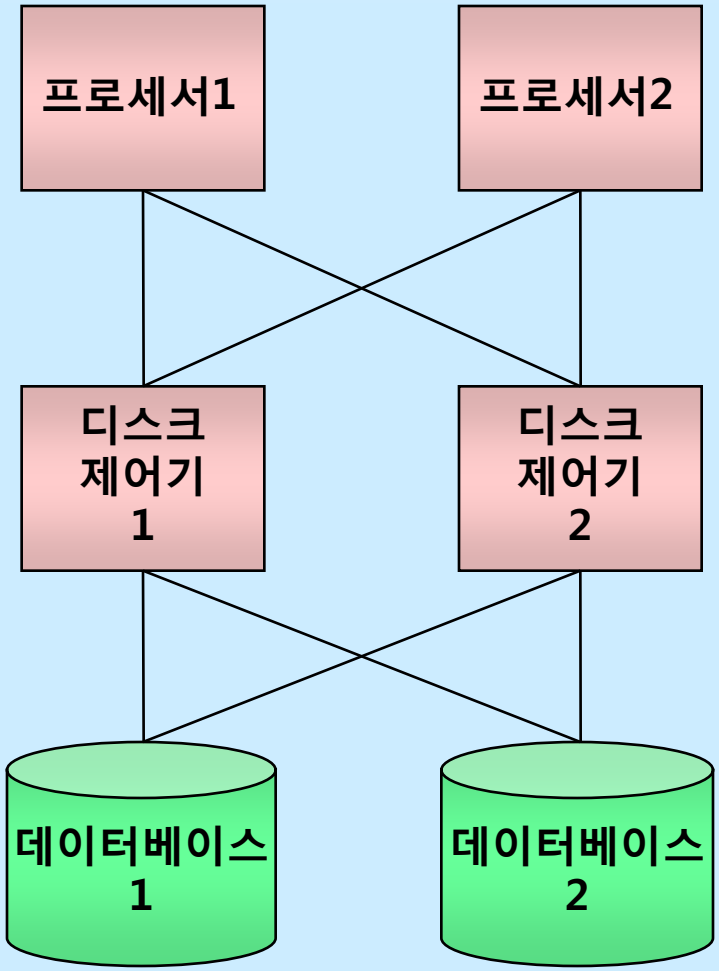
장애와 회복 (II)

- 회복관리자(Recovery manager)
  - DBMS 코드의 10% 이상을 차지
  - 신뢰성 있는 회복을 책임
- 회복의 기본원리 : 중복(redundancy)
  - 덤프(dump) : 다른 저장장치로 복제(archive)
  - 로그(log, journal) : 데이터 아이템의 옛 값과 새 값(old/new values)을 별도의 파일에 기록
- 회복을 위한 조치
  - REDO : 복제본 + 로그

    ➔ 데이터베이스 복원
  - UNDO : 로그 + 모든 변경들을 취소

    ➔ 원래의 데이터베이스 상태로 복원

이중 데이터베이스 시스템



회복 관리자

- 기능
  - 장애 탐지
  - 데이터베이스 복원
- 회복 작업
  - 손상된 부분만을 포함하는 최소의 범위
  - 최단시간 내
  - 트랜잭션 기반 회복
  - 회복자료의 보장
  - 시스템레벨의 자동조치

데이터베이스 저장연산

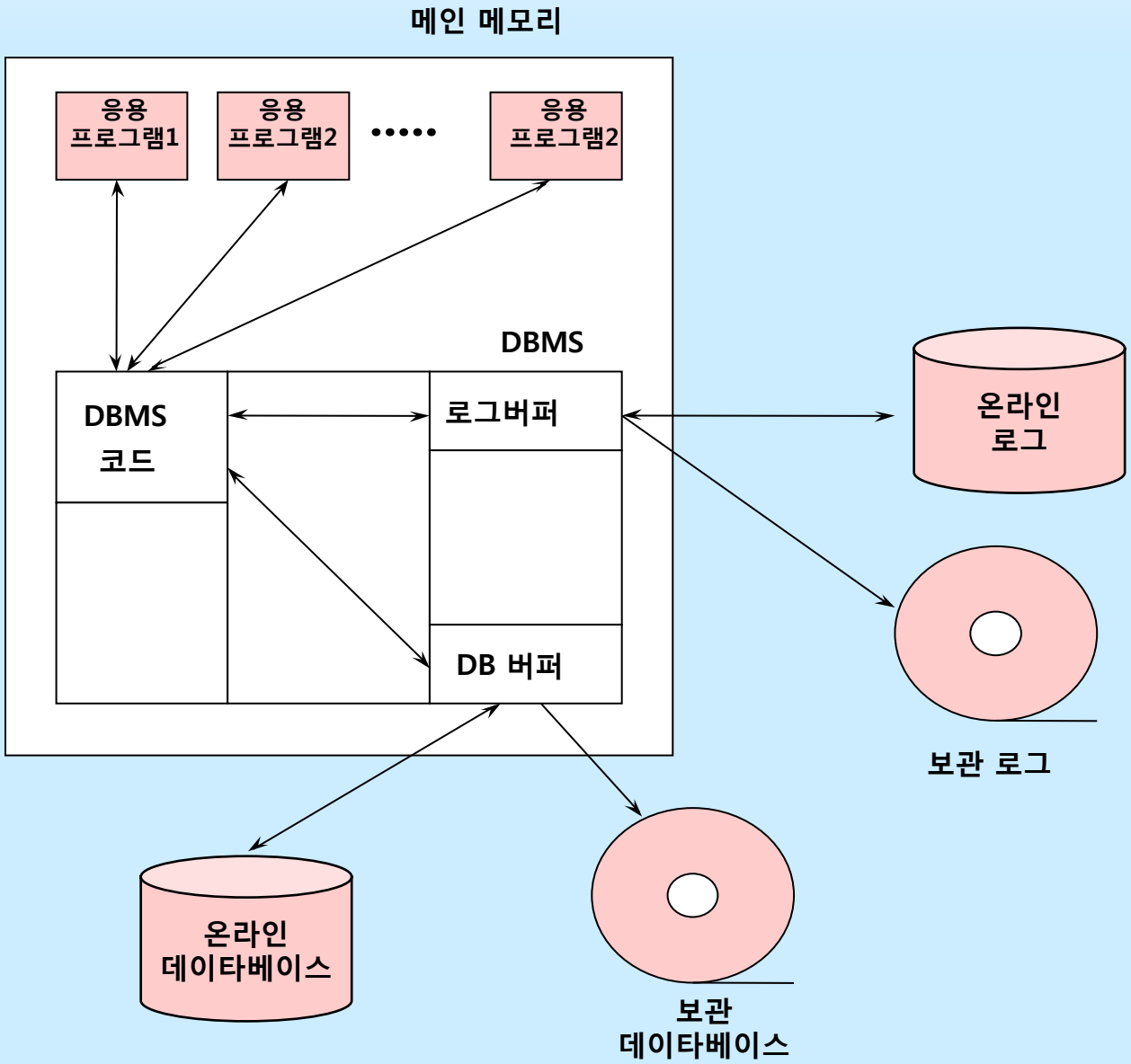
- 저장장치의 타입 : 속도, 용량, 장애시의 탄력성

1) 휘발성 저장장치(volatile storage)
   - 시스템 붕괴 이후에 소멸
   - 메인 메모리, 캐시 메모리
   - 빠르고 직접적인 접근

2) 비휘발성 저장장치(nonvolatile storage)
   - 시스템 붕괴 이후에도 생존
   - 디스크 : 온라인 저장장치
   - 자기 테이프 : 아카이브 저장장치(archival storage)

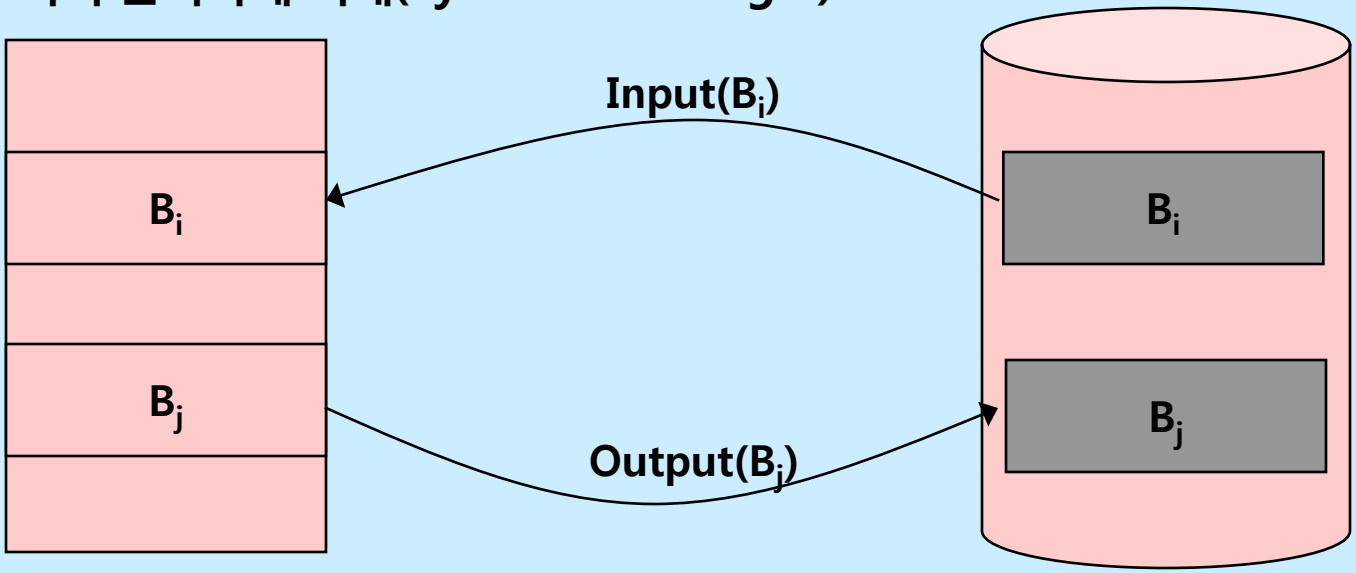
3) 안정적인 저장장치
   - 영구적 생존(never lost)
   - RAID
   - 원격의 다중 비휘발성 저장장치(multiple nonvolatile storage media at a remote site)

DBMS의 저장 구조


메인 메모리
응용 프로그램1
응용 프로그램2
•••••
응용 프로그램2
DBMS
DBMS 코드
로그버퍼
DB 버퍼
온라인 로그
보관 로그
온라인 데이타베이스
보관 데이타베이스

데이터베이스 저장연산

- 디스크와 메인 메모리 사이의 블록 이동
  - Input(B_i)
    - 데이터 B_i가 포함되어 있는 디스크 블록을 메인 메모리로 이동
    - 요청에 의해(On demand)
  - Output(B_j)
    - 데이터 B_i가 포함되어 있는 버퍼 블록을 디스크 블록에 이동시켜 기록
    - 버퍼 관리자에 의해(by Buffer Manager)



데이터베이스 저장연산

- 프로그램과 데이터베이스 사이의 데이터 이동

  X : 데이터 아이템 이름

  x : 프로그램 변수(local variable)

  - Read(X, x)
    - 만일 데이터 아이템 X가 버퍼 블록에 없으면 Input(X)를 실행
    - x ← X
  - Write(X, x)
    - 만일 데이터 아이템 X가 버퍼블록에 없으면 Input(X)를 실행
    - X ← x
    - Output(X)는 나중에 수행될 수 있다(force-output)

트랜잭션

- 다음과 같은 연산자의 시퀀스
  - T : $S_i \rightarrow S_j$, $S_i, S_j \in S$

    S : 데이터베이스의 일관된 상태의 집합
- 작업의 논리적 단위

```
Begin_Trans
...
End_Trans
```

- 트랜잭션의 특성(ACID)
  - 원자성(Atomicity)
    - 전무(All or Nothing)
  - 일관성(Consistency)
    - 트랜잭션 실행 후에도 일관성 유지
  - 격리성(Isolation)
    - 트랜잭션 실행 중 연산의 중간 결과에 다른 트랜잭션이 접근할 수 없다
  - 영속성(Durability)
    - 트랜잭션이 일단 성공적으로 실행되면 그 결과는 영속적이다

트랜잭션

- 프로그램 : 하나 이상의 트랜잭션을 포함
  - 프로그램의 성공적인 수행

    → 모든 트랜잭션의 성공적인 완료
- 트랜잭션의 예
  - 계좌 A에서 계좌 B로 100원을 이체

```
BEGIN_TRANS;
    UPDATE ACCOUNT
    SET BAL = BAL - 100
    WHERE ACCNT = 'A';

    IF ERROR
        THEN GOTO UNDO;

    UPDATE ACCOUNT
    SET BAL = BAL + 100
    WHERE ACCNT = 'B';

    IF ERROR
        THEN GOTO UNDO;
    COMMIT TRANS;
    GO TO FINISH;

UNDO:
    ROLLBACK TRANS;

FINISH:
    RETURN;

END_TRANS;
```

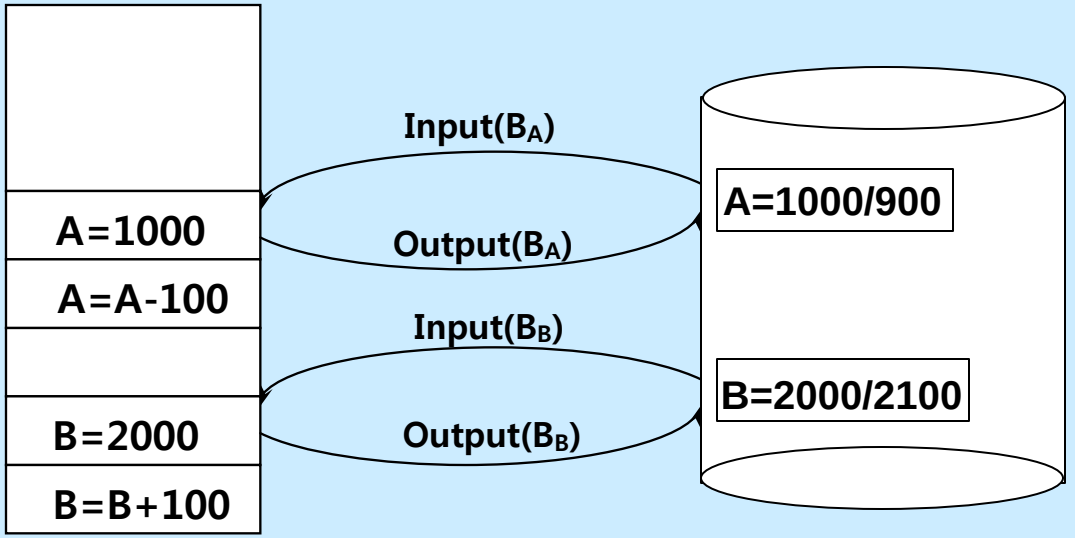
- 트랜잭션 회복(Transaction Recovery)
  - 트랜잭션 : 회복의 논리적 단위

- 트랜잭션의 원자성을 위한 연산
  - COMMIT
    - 트랜잭션의 성공적인 실행
      → 일관성 있는 데이터베이스 상태
    - 영구적인 갱신
    - 갱신된 데이터의 영속성을 보장
  - ROLLBACK
    - 트랜잭션 실행의 실패
      → 모순된 데이터베이스 상태
    - 수행한 모든 연산 결과의 UNDO

원자성을 위한 연산(1)

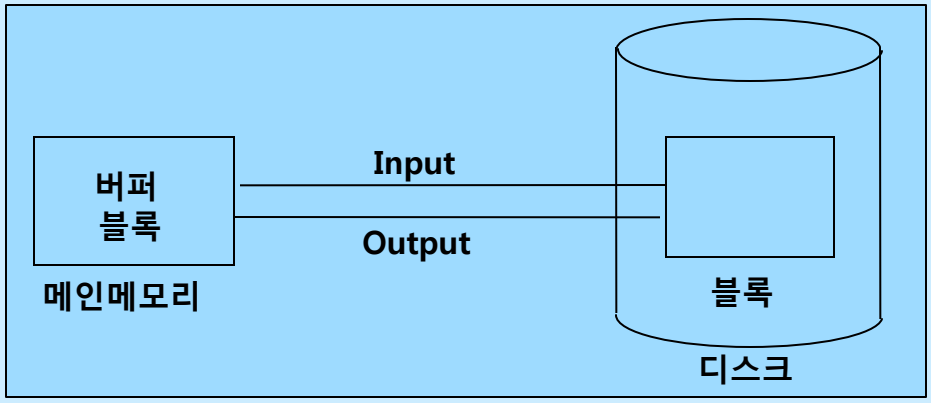
- 예
  - 트랜잭션 T_0
    - 계좌 A에서 계좌 B로 100원을 이체(A=1000, B=2000)

T_0 :
Read(A)
A := A - 100
Write(A)
Read(B)
B := B + 100
Write(B)

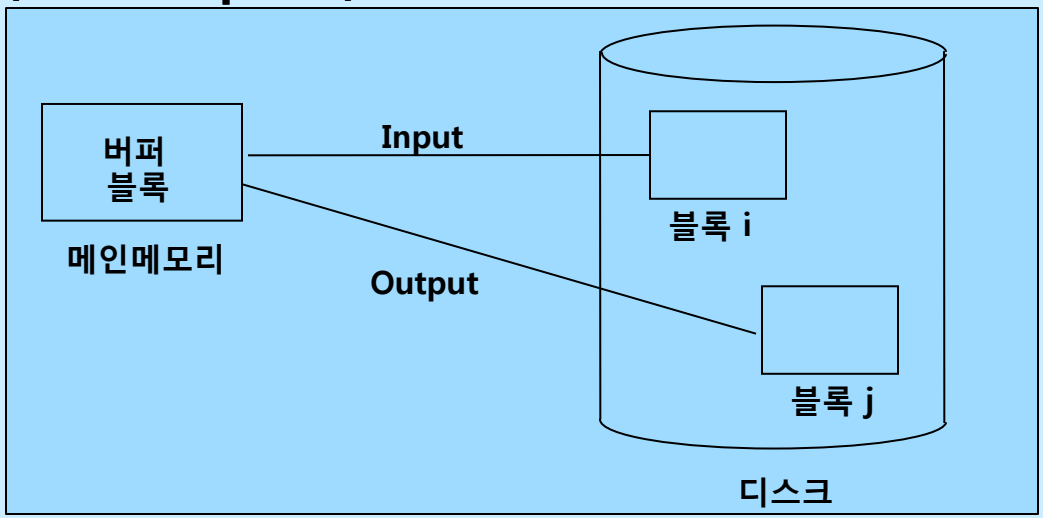


원자성을 위한 연산(2)

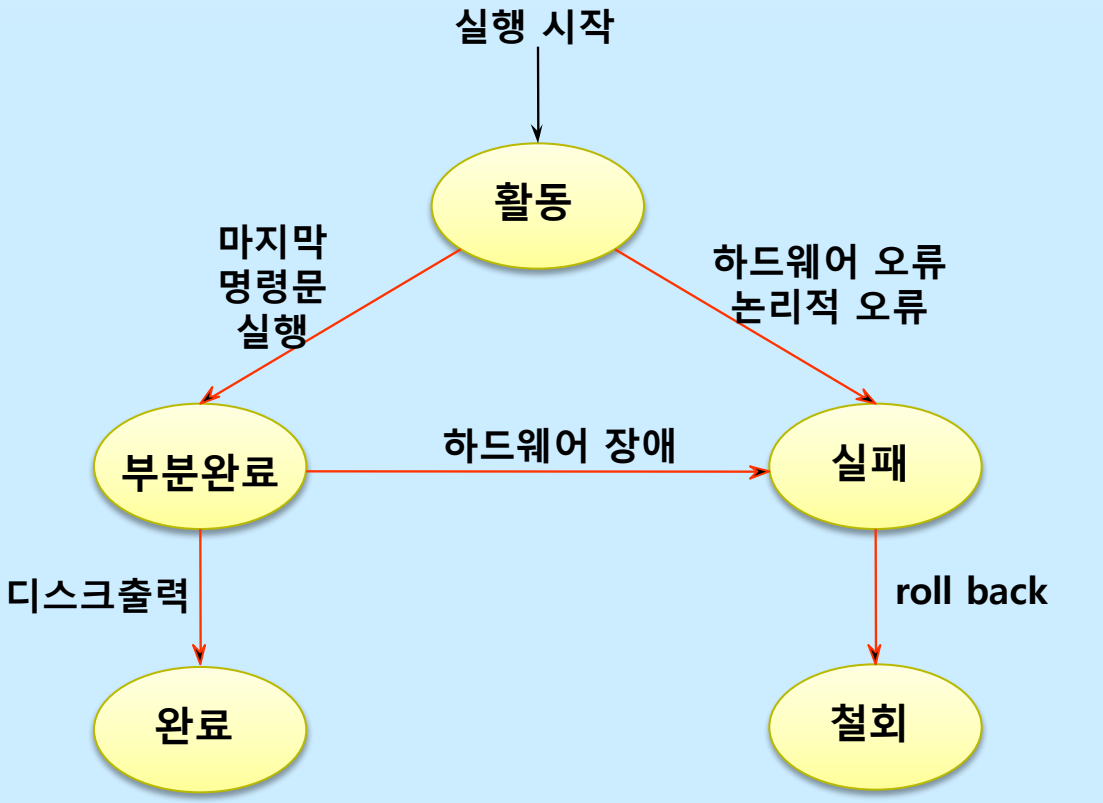
- 디스크 블록 갱신
  - 제자리 갱신(update in place)



  - 간접 갱신(indirect update)



트랜잭션 상태


실행 시작
활동
마지막
명령문
실행
하드웨어 오류
논리적 오류
부분완료
하드웨어 장애
실패
디스크출력
roll back
완료
철회
-완료
-트랜잭션의 재시작
- 트랜잭션의 대기

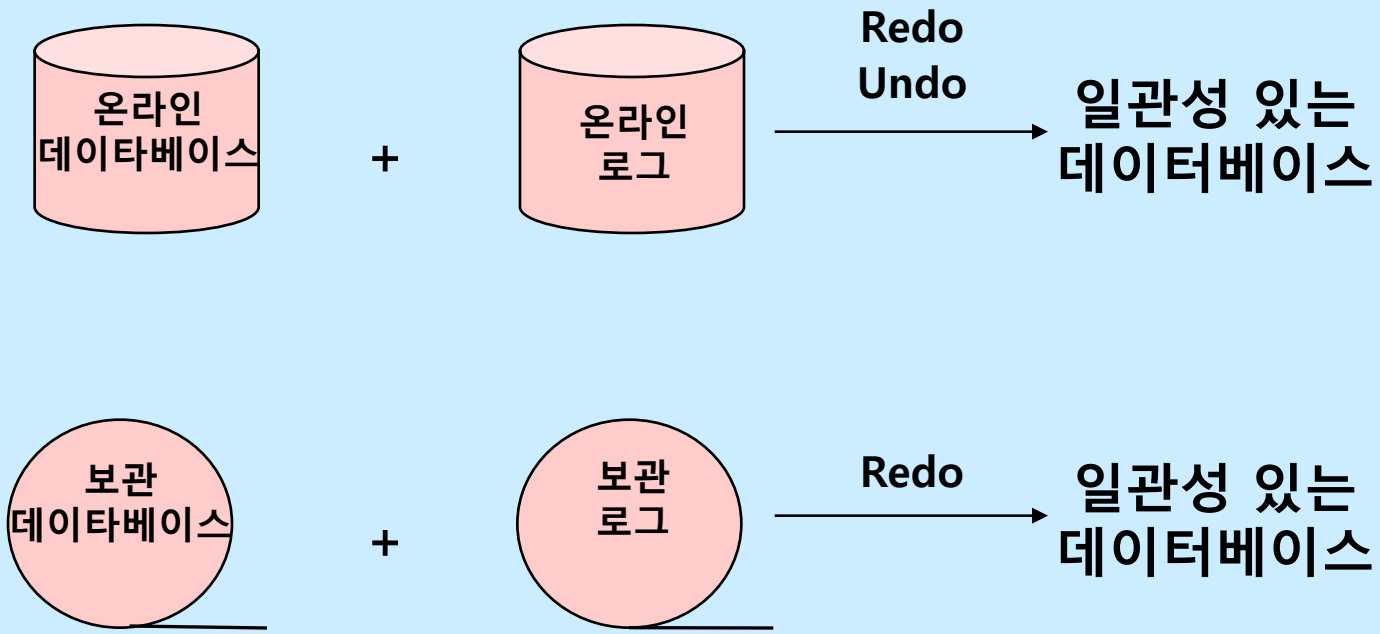
순서

- 장애와 회복

- 로그 이용 회복
- 검사시점 회복
- 미디어 회복

로그 이용 회복

❑ 로그의 이용



데이터베이스 로그(I)

- 로그(log, journal) : 대용량
  - 온라인 로그(on-line log) : 디스크
  - 보관 로그(archival log) : 테이프

- 로그 레코드
  - <T_i, starts>
  - <T_i, data-object, old-value, new-value>
  - <T_i, commit>

데이터베이스 로그(II)

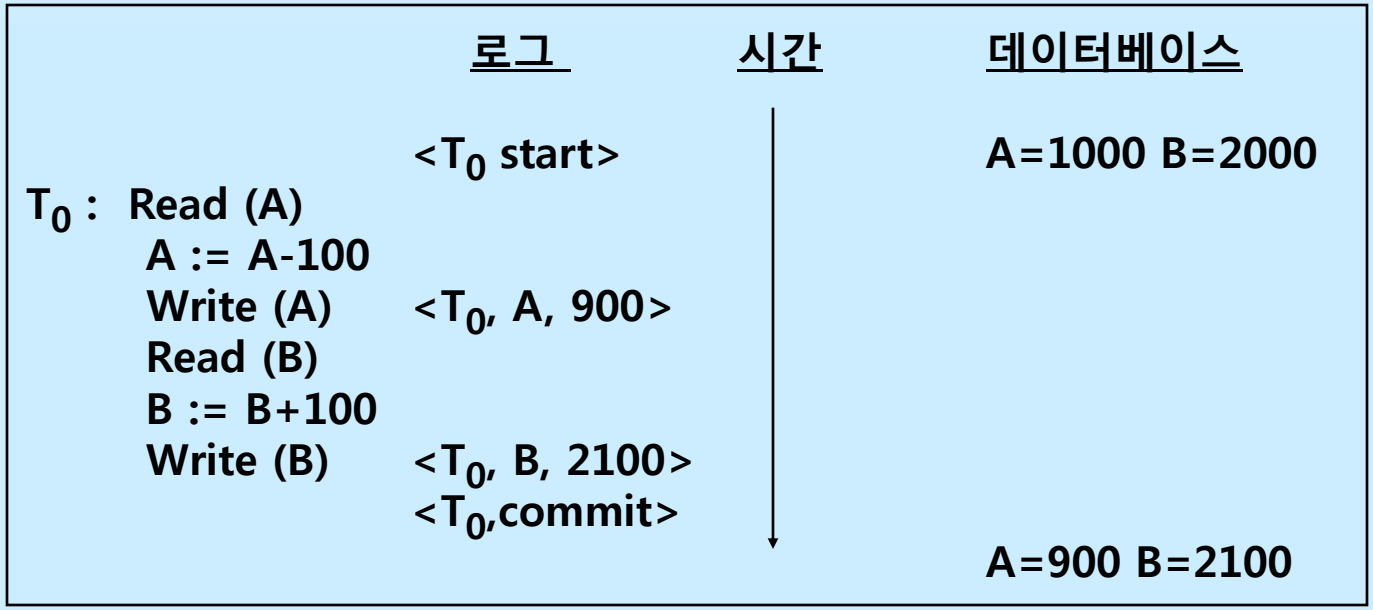
- 로그 압축
  - 저장장치의 효율성, 신속한 회복
  - 실패한 트랜잭션은 로그 불필요 - 트랜잭션이 이미 rollback되었음
  - 성공한 트랜잭션의 갱신 전 데이터는 불필요
    - REDO를 위해 새로운 값만 필요
  - 하나의 데이터 아이템이 여러 트랜잭션에 의해 여러 번 갱신되었다면 가장 마지막 데이터 값만 필요
    - REDO시 중간 과정의 데이터 값은 불필요

지연갱신(deferred update)의 회복

- 부분완료 때까지 모든 output연산을 지연
- 모든 데이터베이스의 변경을 로그에 기록
- 안전한 저장소에 <T_i, Commit>를 포함하는 로그레코드를 기록한 후에 데이터베이스 갱신
  - <T_i, Commit> 는 부분적으로 기록
- UNDO연산자 불필요
- 로그 레코드(REDO)

  <트랜잭션 ID, 데이터 아이템, 변경된 값>

지연 갱신의 회복



- <T_i commit> 후의 장애
  - REDO(Ti) : T_i에 의해 갱신된 모든 값들을 로그의 값들로 갱신
- <T_i commit> 전의 장애
  - 로그를 무시하고 T_i를 재시작

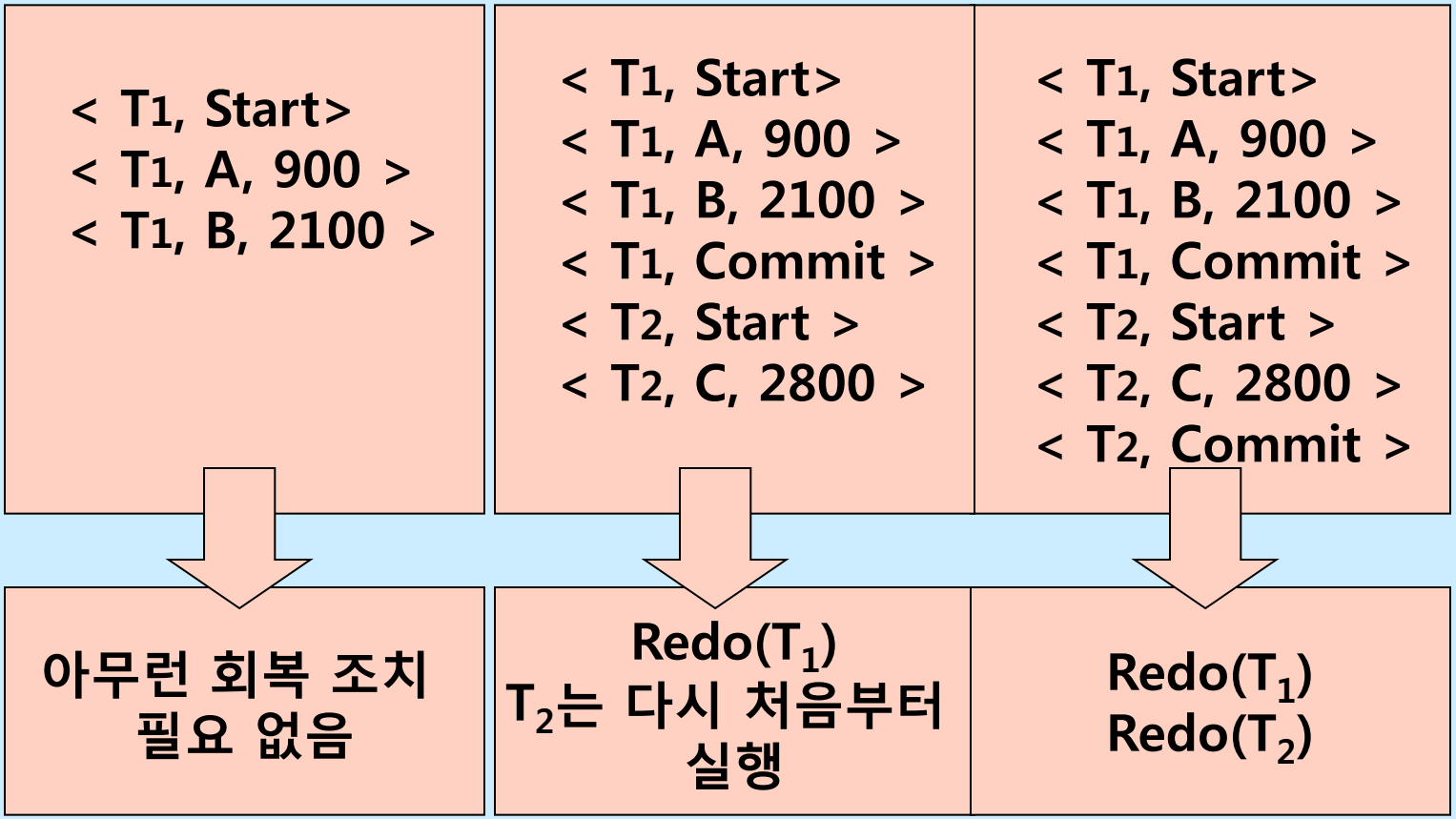
REDO의 실행

- 트랜잭션 T_i가 변경한 모든 데이타 아이템 값들을 로그 파일의 순서에 따라 다시 로그에 있는 새로운 값으로 재지정한다

- REDO 연산의 성질
  - 멱등성(idempotent) : REDO를 여러 번 실행해도 그 결과는 동등하여야 함

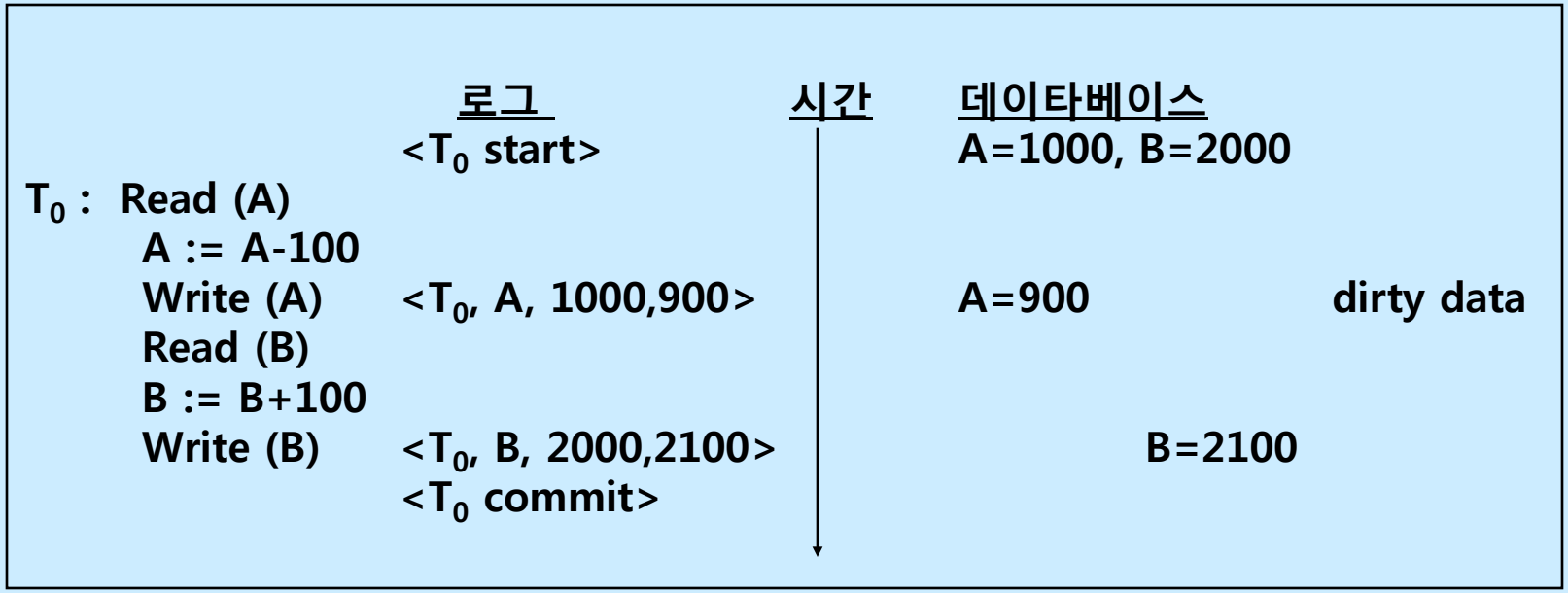
    REDO(REDO(… (REDO(X)))…) = REDO(X)

지연 갱신 회복의 예


< T1, Start>
< T1, A, 900 >
< T1, B, 2100 >
< T1, Start>
< T1, A, 900 >
< T1, B, 2100 >
< T1, Commit >
< T2, Start >
< T2, C, 2800 >
< T1, Start>
< T1, A, 900 >
< T1, B, 2100 >
< T1, Commit >
< T2, Start >
< T2, C, 2800 >
< T2, Commit >
아무런 회복 조치 필요 없음
Redo(T1)
T2는 다시 처음부터 실행
Redo(T1)
Redo(T2)

즉시 갱신의 회복

- 로그 레코드(UNDO)
  - <트랜잭션 ID, 데이터 아이템, 변경 전 값, 변경된 값>



- $<T_i$ commit> 후의 장애
  - UNDO(Ti) : 로그의 변경 전 값으로 환원
- $<T_i$ commit> 전의 장애
  - REDO(Ti) : 로그의 변경 후 값으로 갱신

UNDO의 실행

- UNDO 연산의 성질
  - 멱등성(Idempotent) : UNDO를 여러 번 실행해도 그 결과는 동등하여야 함
  - UNDO(UNDO(… (UNDO(X))) …) = UNDO(X)

즉시 갱신 회복 예

로그	데이터베이스
< T1, Start >	
< T1, A, 1000, 900 >	
	A = 900
< T1, B, 2000, 2100 >	
	B = 2100
< T1, Commit >	
< T2, Start >	
< T2, C, 3000, 2800 >	
	C = 2800
< T2, Commit >	

- 회복 기법 적용의 예
  - T_1이 Commit하기 직전에 시스템 붕괴
    - Undo(T_1) 실행
  - T_2가 Commit하기 직전에 시스템 붕괴
    - Undo(T_2)를 먼저 실행, 다음 Redo(T_1) 실행
  - T_2가 < T_2, Commit> 로그 레코드 출력 직후 시스템 붕괴
    - Redo(T_1), Redo(T_2) 실행

순서

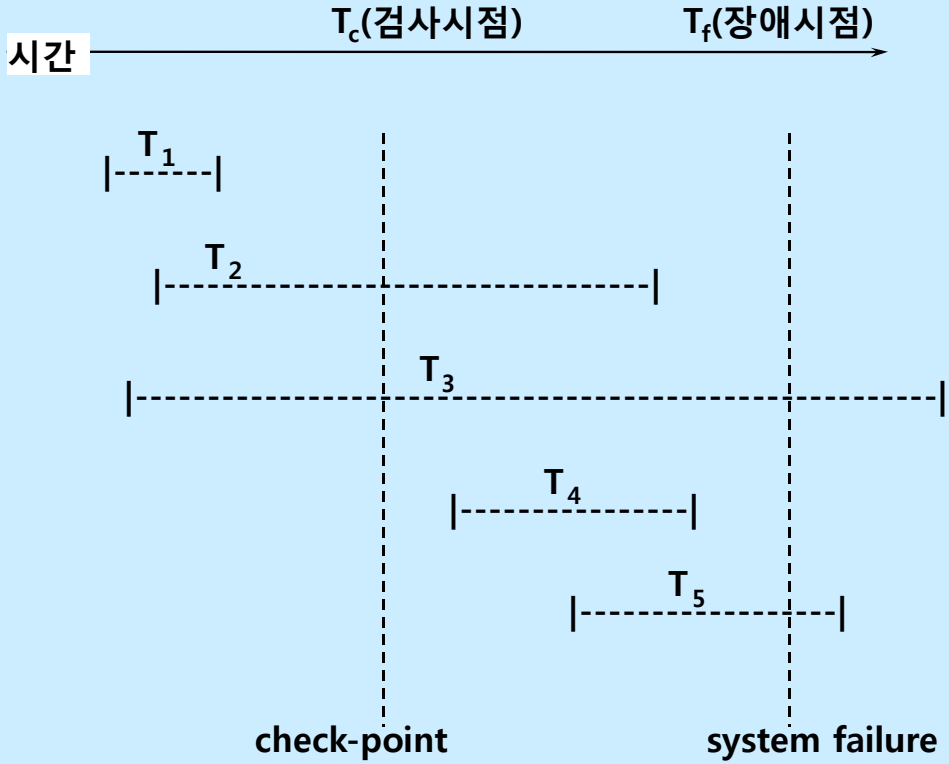
- 장애와 회복
- 로그 이용 회복

- 검사시점 회복
- 미디어 회복

검사시점 회복(1)

- 주기적으로 검사시점을 설정
  - 단계 1 : 메인 메모리의 모든 로그레코드를 안정 저장소로출력
  - 단계 2 : 변경된 버퍼 블록을 모두 디스크로 출력
  - 단계 3 : 검사시점 로그레코드<checkpoint L>를 안정 저장소로 출력
    - L : 현재 실행중인 트랜잭션 리스트

- 트랜잭션 유형
  - T2, T4 : REDO
  - T3, T5 : UNDO
  - T1 : 회복작업과 무관



검사시점 회복(2)

- 어떻게 결정할 것인가?
  - 검사시점 설정 당시에 활동중인 트랜잭션은 전부 UNDO-list에 삽입한다
    REDO-list = □로 설정
  - 검사시점부터 앞에서부터 로그를 탐색한다
    <Ti start> → Ti를 UNDO-list에 첨가한다
    <Ti commit> → Ti를 UNDO-list에서는 삭제하고 REDO-list에 첨가한다

- UNDO/REDO의 수행
  - Undo : UNDO-list의 모든 트랜잭션들에 대해 로그에 기록된 역순으로 UNDO수행 – 후진 회복(backward recovery)
  - Redo : REDO-list의 모든 트랜잭션들에 대해 로그에 기록된 순으로 REDO수행 - 전진 회복(forward recovery)

- 모든 UNDO연산 수행 후 REDO를 수행

그림자 페이징 기법(Shadow Paging)

- 두 개의 페이지 테이블을 유지
  - 현 페이지 테이블(Current page table)
  - 그림자 페이지 테이블(Shadow page table)
- 트랜잭션 실행 중에는 현 페이지 테이블만 사용
- 페이지 i를 갱신하기 위해
  - 페이지 i를 읽는다
  - 페이지 i를 갱신한다
  - 새로 할당된 페이지 공간에 페이지 i를 기록한다
  - 현재의 페이지 테이블을 변경한다

그림자 페이징 기법(Shadow Paging)

- 트랜잭션을 커밋(commit)하기 위해
  1. 모든 수정된 버퍼페이지를 디스크에 기록

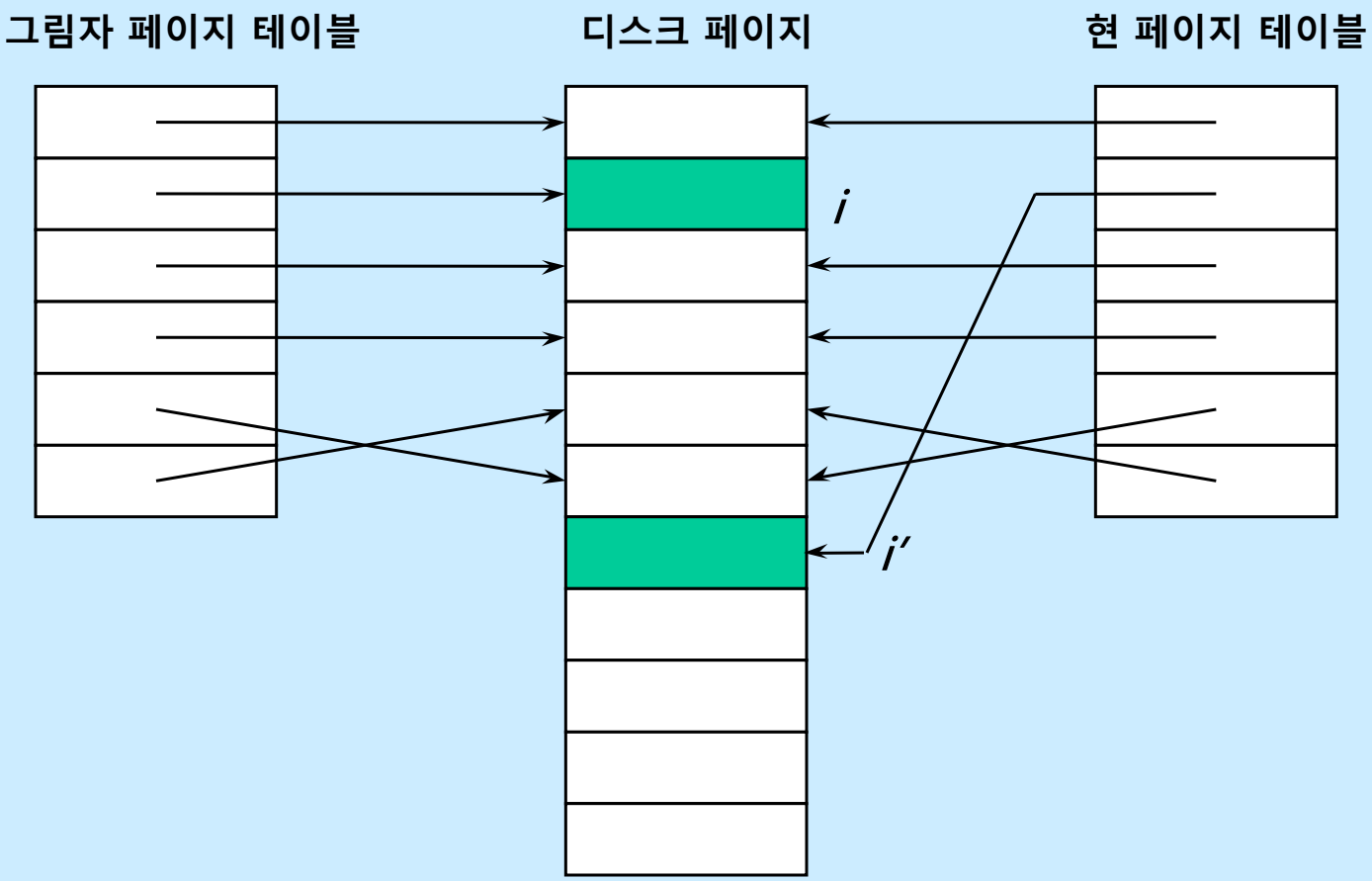
     (그림자 페이지테이블내의 어떤 엔트리에 의해 지시된 데이터베이스 페이지는 변화가 없음)
  2. 현 페이지 테이블을 디스크에 기록

     (그림자 페이지 테이블에 덮어쓰지 않음)
  3. 그림자 페이지 테이블의 주소를 포함한 안정한 저장소 내의 고정된 위치에 현 페이지의 디스크 주소를 기록

     (현 페이지 테이블이 그림자 페이지 테이블이 된다)

그림자 페이징 기법(Shadow Paging)

- 두 번째 페이징 Write연산을 수행한 경우



그림자 페이징 기법(Shadow Paging)

- 트랜잭션이 성공하면, 현 페이지 테이블을 그림자 페이지 테이블로 대체
- 트랜잭션이 실패하면, 현 페이지 테이블을 버린다
  - 로그가 불필요
  - Undo가 불필요
- 결점(Drawbacks)
  - 커밋 오버헤드(commit overhead)
  - 데이터의 단편화(data fragmentation)
  - 쓰레기 수집(garbage collection)
  - 병행 트랜잭션(concurrent transaction) 지원이 곤란

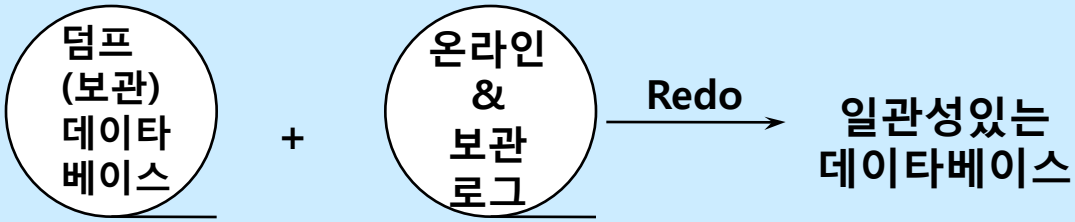
순서

- 장애와 회복
- 로그 이용 회복
- 검사시점 회복

- 미디어 회복

미디어 회복

- 디스크 붕괴 : 비소멸성 저장 장치의 내용이 손상
- 주기적인 덤프
  - 메인 메모리에 있는 모든 로그 레코드를 안정 저장소에 출력
  - 변경된 버퍼 블록들을 모두 디스크에 출력
  - 데이터베이스의 내용을 안정 저장장치에 복사
  - 로그 레코드 <dump>를 안정 저장소에 출력시켜 덤프를 표시
- 회복
  - 가장 최근의 덤프를 이용해서 디스크에 데이터베이스를 적재
  - 로그를 이용해서 이 덤프 이후에 완결된 트랜잭션들을 재실행(REDO)



회복 기법의 구현(I)

- 로그 레코드 버퍼링
  - 출력 회수를 줄임

- 로그 우선 기록 규약
  - 트랜잭션Ti는 <Ti commit> 로그 레코드를 안정 저장장치에 출력시켜야만 완료상태로 들어갈 수 있다
  - <Ti commit> 로그 레코드를 출력시키기 위해서는 먼저 이 Ti에 관련된 모든 로그 레코드를 안정 저장장치에 출력시켜야 된다
  - 데이터베이스 버퍼 블록을 출력시키기 위해서는 먼저 이 버퍼 블록의 데이터와 관련된 모든 로그 레코드가 안정 저장장치에 출력되어야 한다

회복 기법의 구현(II)

- 데이터베이스 버퍼링
  - 디스크블록 B_2가 B_1으로 적재될 때 만약 B_1이 갱신되어야 한다면
    - 데이터베이스 버퍼 블록 B_1에 관련된 모든 로그 레코드를 안정 저장소에 출력
    - 버퍼 블록 B_1을 디스크에 출력
    - 디스크 블록 B_2를 메인 메모리 B_1의 자리에 적재

    그러나 대부분의 운영체제는 write-ahead logging을 지원하지 않는다.

  - 메인 메모리 일부를 운영체제가 아니라 DBMS가 관리하는 버퍼로 예약해 놓고 데이타베이스블록 이동을 DBMS가 관리

  - DBMS는 운영체제의 가상 기억장치 속에 데이타베이스 버퍼를 구현하고 데이터베이스 자체는 운영체제의 파일 시스템 속에 저장, 그리고 데이터베이스 파일과 이 가상 기억장치의 버퍼 사이의 이동은 DBMS가 관리하여 변경된 블록보다 로그 레코드가 먼저 출력되도록 함

요약

- 장애와 회복
- 로그 이용 회복
- 검사시점 회복
- 미디어 회복

데이터베이스(DATABASE)

데이터베이스관리시스템(DBMS)