데이터베이스

회복 기법

학습목표

- 갱신 기법에 따른 로그 구조와 그에 따른 회복 기법을 작성할 수 있다.
- 비로그 기반의 회복 기법에 대하여 설명할 수 있다.

학습내용

- 로그 기반의 회복 기법
- 그림자 페이징과 다중 데이터베이스에서의 회복

로그 기반의 회복 기법

지연갱신의 회복

01 지연갱신

- 부분 완료될 때까지의 모든 Output 연산을 지연
  - 모든 데이터베이스의 변경을 로그에 기록
  - 안전한 저장소에 <T_i, Commit>를 포함하는 로그 레코드를 기록한 후에 데이터베이스를 갱신
    - 완료 상태로 감
  - <T_i, Commit>: “부분 완료(Partial Commit)“를 의미
  - UNDO 연산자: 불필요
  - 로그 레코드: REDO 연산에 대비
    - <트랜잭션 id, 데이터 아이템, 변경된 값>

로그 기반의 회복 기법

지연갱신의 회복

02 REDO 연산의 멱등성(Idempotent)

01 같은 REDO를 여러 번 실행하거나 한 번 실행한 경우

- 결과는 동등

02 REDO 작업 중 다시 장애가 일어나 REDO 연산을 또 다시 실행한 경우

- 처음 한 번 시행한 결과와 동일

T_1:Read(A) ←5000 A=A-1000 Write(A) Read(B) ←10000 B=B+1000 Write(B)	<T_1, Start> <T_1, A, 4000> <T_1, B, 11000> <T_1, Commit>
DB A=5000, B=10000	A=4000, B=11000
아무런 조치 필요 없음	***REDO(T_1)***

로그에 <T_i, Start> 레코드와 <T_i, Commit> 레코드가 모두 있는 트랜잭션 T_i에 대해서만 재실행

로그 기반의 회복 기법

즉시갱신의 회복

01 즉시갱신

- 데이터의 변경 결과를 데이터베이스에 그대로 반영
- 미완료 갱신(Uncommitted Update)
  - 활동성 완료가 안 된 트랜잭션에 의해 데이터베이스에 반영된 갱신
  - 트랜잭션 장애가 일어나면 트랜잭션이 실행되기 전 상태의 데이터 값으로 복원
  - REDO와 UNDO 필요
  - 로그 레코드
    - <트랜잭션 id, 데이터 아이템, 변경된 값>

UNDO 연산도 멱등성 성질을 가지고 있음

T_1:Read(A) ←5000 A=A-1000 Write(A) Read(B) ←10000 B=B+1000 Write(B)	<T_1, Start> <T_1, A, 5000, 4000> <T_1, B, 10000, 11000> <T_1, Commit>
DB A=5000, B=10000	A=4000, B=10000 A=4000, B=11000

로그 기반의 회복 기법

즉시갱신의 회복

02 장애 발생 시

- 장애가 발생하면 회복 관리자는 로그를 검사

만일 로그에 <T_i, Start > 레코드만 있고
< T_i, Commit> 레코드가 없다면?

- Undo(T_i)를 수행

만일 로그에 <T_i, Start > 레코드와
< T_i, Commit> 레코드가 모두 다 있다면?

- REDO(T_i)를 수행

로그 기반의 회복 기법

즉시갱신의 회복

03 회복 기법 적용의 예

- T_1이 Commit 하기 직전에 시스템이 붕괴할 경우
  - UNDO(T_1) 실행
- T_1가 <T_1, Commit> 로그 레코드 출력 직후에 시스템이 붕괴할 경우
  - REDO(T_1) 실행

<T_1, Start> < T_1, A, 5000, 4000 > < T_1, B, 10000,11000 >	<T_1, Start> < T_1, A, 5000, 4000 > < T_1, B, 10000, 11000 > < T_1, Commit >
DB A=5000, B=10000	A=4000, B=10000 A=4000, B=11000
UNDO(T_1)	***REDO(T_1)***

- 로그에 <T_i, Start> 레코드만 모두 있는 트랜잭션 T_i에 대해서 → UNDO
  - UNDO는 로그에 기록된 역순으로 변경 전 값으로 환원
- 로그에 <T_i, Start> 레코드와 <T_i, Commit> 레코드가 모두 있는 트랜잭션 T_i에 대해서 → REDO
  - REDO는 로그에 기록된 순서로 변경 후 값을 기록

로그 기반의 회복 기법

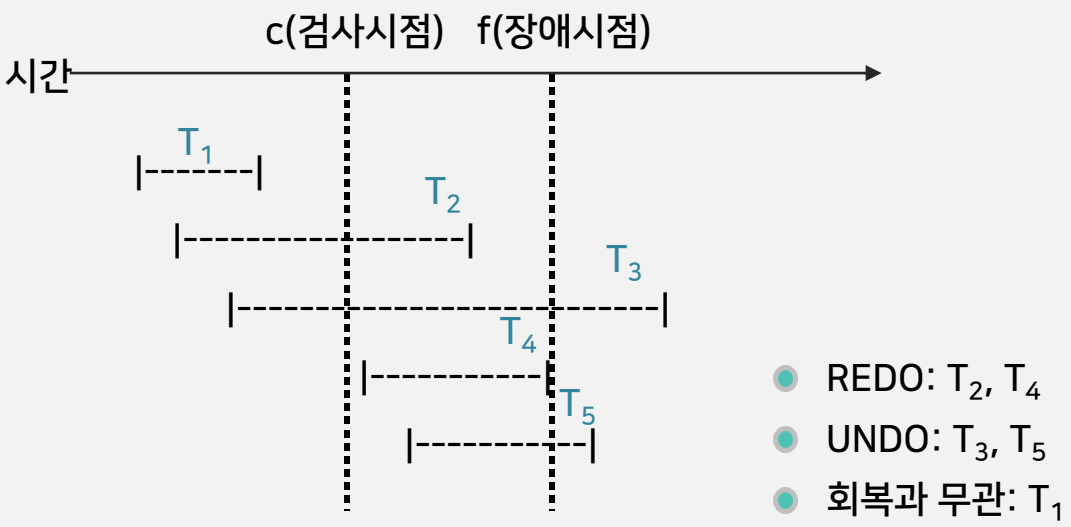
검사시점 회복

01 로그기반 회복 기법

- 시스템 장애가 일어났을 때 로그를 이용하여 회복하는 기법
  - 단점
    - REDO와 UNDO를 해야 될 트랜잭션을 결정하기 위해서는 로그 전체를 조사하기 때문에 너무 많은 시간이 걸림
    - 불필요한 REDO 발생

02 검사시점 방법

- 로그 기록 유지, 일정 시간 간격으로 검사시점을 설정
  - 메인 메모리(로그 버퍼)에 있는 모든 로그 레코드를 안정 저장소로 출력
  - 변경된 데이터 버퍼 블록을 전부 디스크로 출력
  - 검사시점 표시로써 <Checkpoint L> 로그 레코드를 안정 저장소에 출력
    - 여기서 L은 현재 실행 중에 있는 트랜잭션들의 리스트를 의미



로그 기반의 회복 기법

검사시점 회복

03 트랜잭션 목록 결정 방법

01 두 개의 빈 Undo-list와 REDO-list를 생성

02 검사시점 설정 당시 활동 중인 트랜잭션은 Undo-list에 삽입

03 로그를 차례로 검색하면서 <T_i, Start> 로그 레코드를 만나면 트랜잭션 T_i를 Undo-list에 첨가

04 로그를 차례로 검색하면서 <T_i, Commit> 로그 레코드를 만나면 트랜잭션 T_i를 Undo-list에서 삭제하고 REDO-list에 첨가

04 UNDO와 REDO의 수행

후진 회복 (Backward Recovery)	Undo-list에 있는 모든 트랜잭션들에 대해 로그에 기록된 역순으로 UNDO 연산을 수행
전진회복 (Forward Recovery)	REDO-list에 있는 트랜잭션에 대해 로그에 기록된 순서로 REDO를 수행

회복 작업이 완료될 때까지
시스템은 **새로운 트랜잭션**을 받아들일 수 없음

그림자 페이징과 다중 데이터베이스에서의 회복

그림자 페이징

01 그림자 페이징(Shadow Paging)

- 로그를 사용하지 않는 회복 기법
  - 간접 갱신: 수정된 내용이 다른 위치에 저장됨

☑ 두 개의 페이지 테이블 유지

- 현 페이지 테이블 (Current Page Table)
- 그림자 페이지 테이블 (Shadow Page Table)

트랜잭션 실행 중에는 현 페이지 테이블만 사용

02 가정

- 데이터베이스

  여러 개(N)의 고정된 크기의 디스크 페이지들로 구성

- 페이지 테이블

  i (1 < i < N)번째 엔트리는 i번째 데이터베이스 페이지를 가리킴

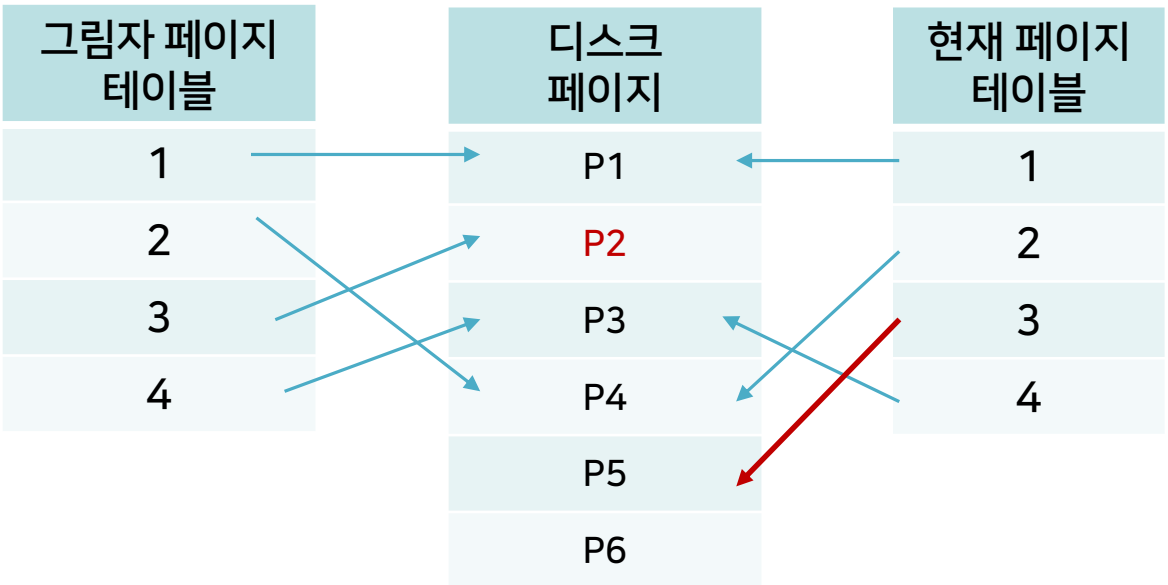
데이터베이스에 대한 모든 읽기, 쓰기 연산은 페이지 테이블을 통하여 이루어짐

그림자 페이징과 다중 데이터베이스에서의 회복

그림자 페이징

03 트랜잭션의 수행 방법

트랜잭션이 시작할 때	- 현재 페이지 테이블을 그림자 페이지테이블로 복사
write_item연산을 수행할 때	- 수정된 데이터베이스 페이지의 새 사본을 생성하고, 현재 페이지 테이블 엔트리가 새 사본을 가리키도록 수정
트랜잭션을 완료할 때	- 그림자 페이지 테이블을 폐기하고 그 그림자 페이지 테이블이 참조하는 이전 페이지들을 반환



» 예 | Write(X) 연산 수행할 때 X가 디스크 페이지 P2에 있을 경우, P5를 새로 할당 받아 P5에 기록

그림자 페이징과 다중 데이터베이스에서의 회복

그림자 페이징

04 3단계 전후의 시스템 장애

3단계 전: 시스템 장애 발생

- 현 페이지 테이블을 폐기

3단계 직후: 시스템 장애 발생

- 트랜잭션의 실행 결과에 아무런 영향 없음
- REDO 연산이 불필요함

05 장·단점

장점	• 로그 레코드를 출력하는 오버헤드가 없음 → 디스크 접근 횟수를 줄임 • UNDO 연산이 아주 간단하며, REDO 연산이 필요 없음 → 트랜잭션 실행 결과의 UNDO가 단순 장애로부터의 회복 작업은 신속함
단점	• 갱신된 데이터베이스 페이지들의 디스크 상의 위치가 변하기 때문에 클러스터링(Clustering)이 어려움 * 클러스터링(Clustering): 유사성 등의 개념에 기초하여 데이터를 몇몇의 그룹으로 분류하는 수법의 총칭 • 디렉토리가 큰 경우 오버헤드가 심각함 • 트랜잭션 완료 시 쓰레기를 수거해야 하는 문제가 있음 • 병행 트랜잭션(즉, 동시 사용자 지원)이 곤란함

그림자 페이징과 다중 데이터베이스에서의 회복

다중 데이터베이스에서의 회복

01 다중 데이터베이스 트랜잭션

- 여러 개의 데이터베이스를 액세스하는 트랜잭션으로 이 때 각각의 DBMS들은 서로 다른 회복 기법과 트랜잭션 관리자를 사용할 수 있음
- 원자성을 유지하기 위하여 2단계 완료 프로토콜(2PC: 2 phase commit)을 사용
  - 모든 참여 데이터베이스가 트랜잭션을 완료하도록 하거나 또는 어느 하나도 완료하지 않도록 함
  - 어떤 참여 데이터베이스에 고장이 발생하더라도 트랜잭션이 완료된 상태 또는 철회된 상태로의 회복은 항상 가능함

그림자 페이징과 다중 데이터베이스에서의 회복

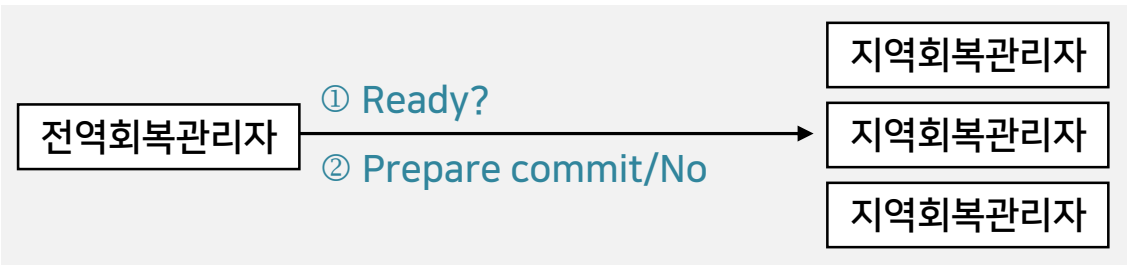
다중 데이터베이스에서의 회복

02 2단계 완료 프로토콜(Two-phase commit protocol)

전역회복관리자와 지역회복관리자로 구성된 2단계 회복기법

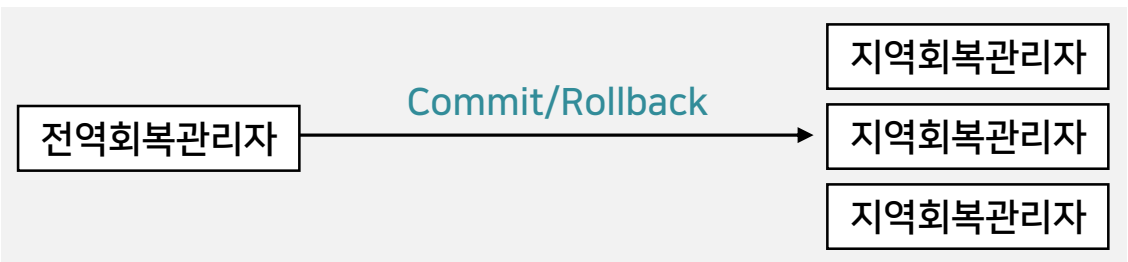
1단계 | 2단계

- 전역회복관리자가 “Ready?” 메시지를 지역회복관리자들에게 전송
- 각 참여 데이터베이스의 지역회복관리자는 자신이 담당한 트랜잭션 부분을 완료할 수 있으면 “Prepare commit” 메시지를 전역회복관리자에게 통보



1단계 | **2단계**

- 모든 지역회복관리자가 “Prepare commit”되었으면 전역회복관리자는 “Commit” 신호를 보냄
- 하나 이상의 지역회복관리자가 “No” 신호를 보내면 전역회복관리자는 “Rollback” 신호를 보냄



1 로그 기반의 회복 기법

- ✓ 지연갱신의 회복: 부분 완료될 때까지 모든 Output 연산 지연(REDO만 필요)
- ✓ 즉시갱신의 회복: 데이터베이스의 변경 결과를 데이터베이스에 그대로 반영(REDO와 UNDO 필요)
- ✓ 검사시점 회복: 불필요한 REDO를 줄이기 위하여 사용되는 회복 기법

2 그림자 페이징과 다중 데이터베이스에서의 회복

- ✓ 그림자 페이징
  - 로그를 사용하지 않는 회복 기법으로 현 페이지 테이블과 그림자 페이지 테이블 유지
  - 트랜잭션 수행 중에는 현 페이지 테이블 만을 사용
- ✓ 2Phase Commit Protocol: 다중 데이터베이스의 회복
  - 전역회복관리자와 지역회복관리자로 구성된 2단계 회복 기법