**Chapter 16: Recovery System**

**Implementation of Atomicity and Durability**

Recovery-management가 atomicity, durability를 책임짐.

Shadow-database scheme:

하나의 트랜잭션만 한 번에 실행된다고 가정.

Db\_pointer는 항상 현재 consistent copy를 가리킴.

모든 업데이트는 shadow copy에 이루어짐.

Partial commit 단계로 들어서면 모든 업데이트가 디스크로 flush 됨. All을 보장.

fail되면, db\_pointer가 old copy를 가리킴. 그러면 nothing을 보장.

하지만 큰 DB에서는 매우 비효율적임!

**Failure Classification**

트랜잭션 실패

Logical error

System error

시스템 크래쉬

전기가 나가거나… 시스템 바깥의 문제. DBMS의 책임은 상태를 consistent로 되돌려 놓는 것.

Disk failure

Fail-stop assumption: non-volatile 저장 장치는 system crash에 의해 corrupt 되지 않음을 가정

**Storage Structure**

Volatile storage

System crash에서 살아남지 못함. (메인 메모리, 캐시 메모리)

Non-volatile storage

System crash에서 살아 남음. 디스크 등.

Stable storage

모든 failure에서 살아 남는 이론적인 형태.

여러 non-volatile media의 다양한 copy를 유지함으로써 근접하게 구현.

**Data Access**

Data blocks

Physical blocks: 디스크에 있는 블락

Buffer blocks: 메인 메모리에 있는 블락

각 트랜잭션은 private work-area가 있음.

Block movement

Input(B): physical block B to main memory

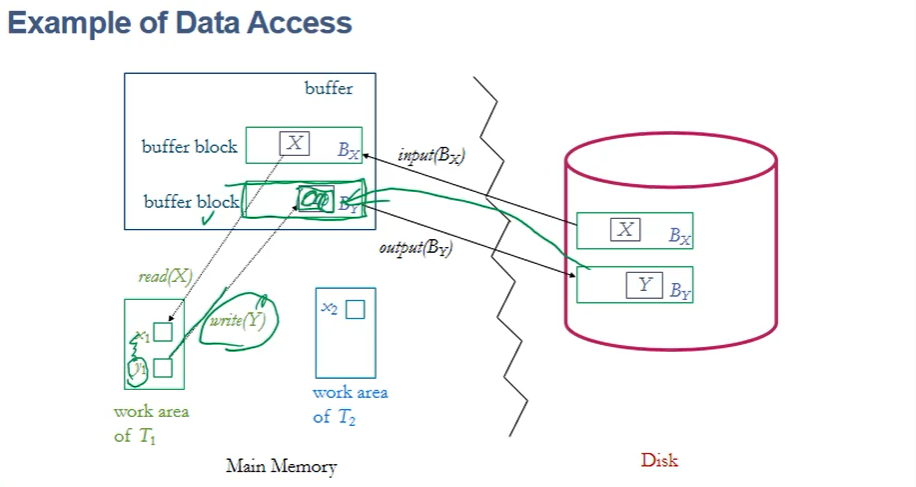
Output(B): main memory to physical block B

Transaction 연산

Read(X): X가 메모리에 없으면 input(B\_x)를 수행.

Write(X): B\_x가 메모리에 없으면 input(B\_x). 버퍼 블락에 X\_i 값 할당.

Output(B\_x)가 바로 write(X)를 따를 필요는 없음.



buffer에만 업데이트 돼도 DBMS에서 업데이트라고 마크함. 그리고 책임을 져야 함.

**Recovery Algorithms**

두 파트가 있음

1. Normal transaction에서 recovery할 수 있게 충분한 정보를 가져야 함
2. 실제로 failure 이후 액션을 취함.

**Log-Based Recovery**

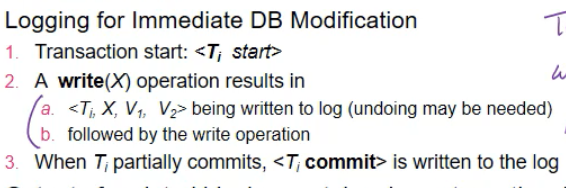
Log – 데이터베이스의 활동을 기록하는 log records의 시퀀스. Stable storage에 보관됨.

Log records – Start, commit, abort를 기록함. 그리고 write(X)를 한다면 V1 (업데이트 이전), V2 (업데이트 이후) 값을 모두 씀.

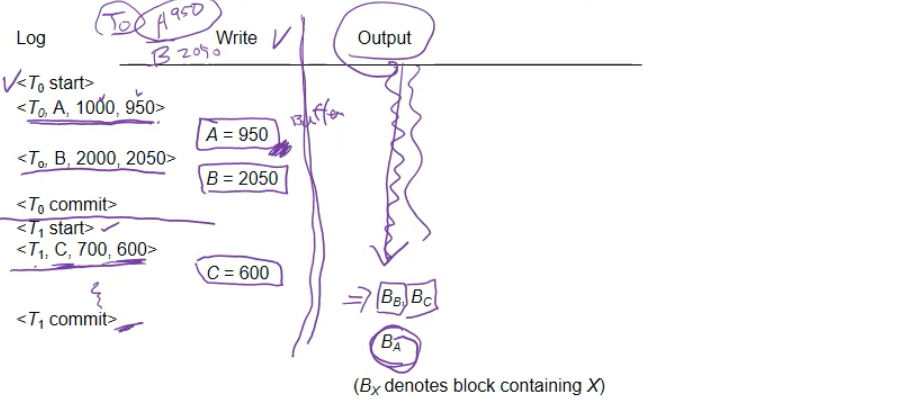
각 트랜잭션은 serial하게 수행되고, log records는 발생하는 즉시 stable storage에 쓰인다고 가정.

**Immediate Database Modification**

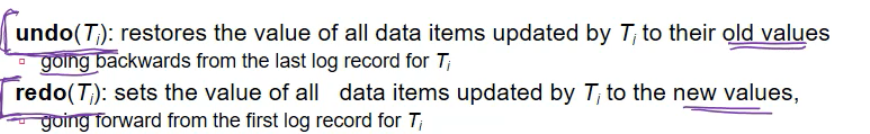
Write가 issue될 때 데이터베이스가 uncommitted transaction을 업데이트하는 것을 허용.



업데이트된 블락의 output은 트랜잭션이 커밋되기 전이나 후의 아무 때나 반영될 수 있음.

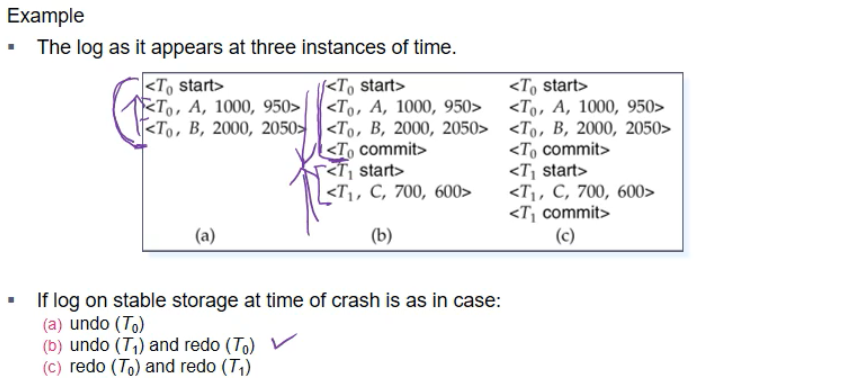


Fail이 나면 로그에서 값을 읽음. 예를 들어 T0 A 1000 950 후에 에러나면 A 블락을 버퍼로 읽어 와서 1000인지 950인지 확인.

**Undo & Redo Operations** 

두 operation은 idempotent 해야 함. (멱등성. 여러 번 수행해도 결과가 같아야 함)

**Recovery Logic**



(b)에서 undo를 먼저 해줘야 함.

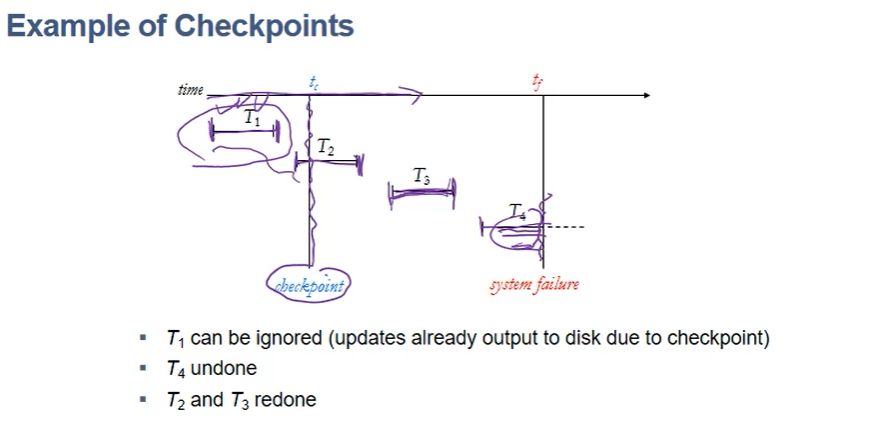
**Checkpoints**

전체 log를 찾는 것은 낭비가 심함.

Checkpoint는 recovery overhead를 줄임.

Checkpoint process

1. 현재 메인 메모리에 존재하는 모든 log record를 stable storage에 써줌
2. 모든 modified buffer block을 디스크에 써줌
3. Log record <checkpoint>를 stable storage에 씀.



**Log Record Buffering**

Log record는 main memory에 buffer됨.

바로바로 stable storage에 쓰지 않음.

Log record는

1. Buffer가 full인 경우
2. Log force 연산이 commit에 수행되면, 트랜잭션의 모든 record가 stable storage에 써짐.

**Write-Ahead Logging (WAL)**

1. Log record는 생성된 순서로 stable storage에 output 되어야 함.
2. 트랜잭션은 log record가 stable storage에 반영이 되었을 때만 commit state로 들어섬.
3. 메인 메모리의 a block of data가 database에 쓰이기 전에, 그 블락의 데이터를 담고 있는 모든 로그 레코드는 stable storage에 쓰여야 함.

=> write-ahead logging 혹은 WAL이라고 함.