

Database System 2020-2

Final Report

ITE2038-11800

2019044711

김찬웅

Table of Contents

Overall Layered Architecture	3 p.
Concurrency Control Implementation	7 p.
Crash-Recovery Implementation	13 p.
In-depth Analysis	17 p.

Overall Layered Architecture

- DBMS란 DB(Database: collection of structured data)를 관리하는 시스템으로 사용자와 SQL interface를 통해서 communication을 한다. 또한 이러한 DBMS는 여러 layer로 나뉘어서 각자의 고유한 일만 하도록 하는 **Layered Architecture**로 이루어져 있다.

■ Layered Architecture

- ◆ 각자 layer은 자신만의 고유한 일을 한다.
- ◆ Layer들 간에는 api를 이용해서 일을 넘긴다.
- ◆ 장점
 - 각각의 단계가 할 일만 하기 때문에 자신의 일만 신경 쓰면 된다. (다른 layer들의 결과는 신경을 쓰지 않는다.)
- DBMS의 Layered Architecture은 아래와 같이 이루어져 있다.



- **Query Parsing & Optimization**

- **Relational Operators**

- **Files and Index Management**

- **Buffer Management**

- **Disk Space Management**

- 이제 아래에서 하나씩 설명하도록 하겠다.

- **Query Parsing & Optimization**

- 어떻게 하면 주어진 query를 빠른 시간 안에 수행할 수 있는지를 맡는 layer이다. SQL interface에서 받아온 SQL code를 machine friendly code로 변환을 하는 역할을 한다.

- Query optimizer가 하위 layer에게 relation operator로 변경해달라고 요청하고 다시 받아서 optimization을 진행을 한다.

- 다양한 query plan들을 세우고 각각에 대해서 cost estimation을 진행을 하고 cost가 가장 작은 plan으로 진행을 하게 된다. (다음 단계로 해당 plan을 넘겨준다.)

- **Relational Operators**

- Optimization을 위해서 parsing된 query를 relational algebra를 이용하여 relational operator로 만들어서 다시 상위 layer로 보낸다.

- **아래의 layer들은 Database의 접근과 관련된 layer들이다.**

- **Files and Index Management**

- Plan을 기반으로 Database에 접근을 하기 위한 layer

- Logical한 table을 어떻게 physical하게 구성을 할 것인지 알고 있는

layer이다. → pagination

- DB의 접근을 위해서 반드시 필요한 layer이며, 나중에 필요한 정보를 찾거나 탐색이 용이하도록 file과 index를 구성을 하는 layer이다.
- 정보를 요약하는 header page, 비어 있는 공간, record가 있는 공간을 잘 구성해야는 그런 part를 맡고 있는 layer이다.

- **Buffer Management**

- Database paginated architecture에서 Database에 있는 정보들을 buffer에 담아서 빠른 read / write가 가능하도록 하는 layer이다.
- 사실상 없어도 correctness에는 문제가 없는 layer다. memory (ram)와/과 disk의 속도 차이를 해소할 수 있도록 즉 files and index management와 disk space management의 속도 차이를 해소하기 위한 완충제 역할로, efficiency에 많은 영향을 주게 된다. (거의 필요할 정도)
- 이 Layer에서는 상위 layer에서 요청한 페이지를 전달을 하게 되는데, 이 때 요청한 page가 현재 buffer에 없을 경우 하위 layer에게서 page를 요청을 하는 역할을 한다.
- Buffer의 용량에는 한계가 있으므로 요청한 것들을 전부 담아낼 수 없다. 그래서 page replacement policy가 존재한다.
- Page replacement policy
 - ◆ LRU : Least Recently Used
 - ◆ MRU : Most Recently Used

- **Disk Space Management**

- Disk Space Management는 Database와 직접 연결이 되어 있는 layer이며, DBMS에서 최하위 layer이고, disk의 space를 관리한다.
- 주요 기능은 다음과 같다.

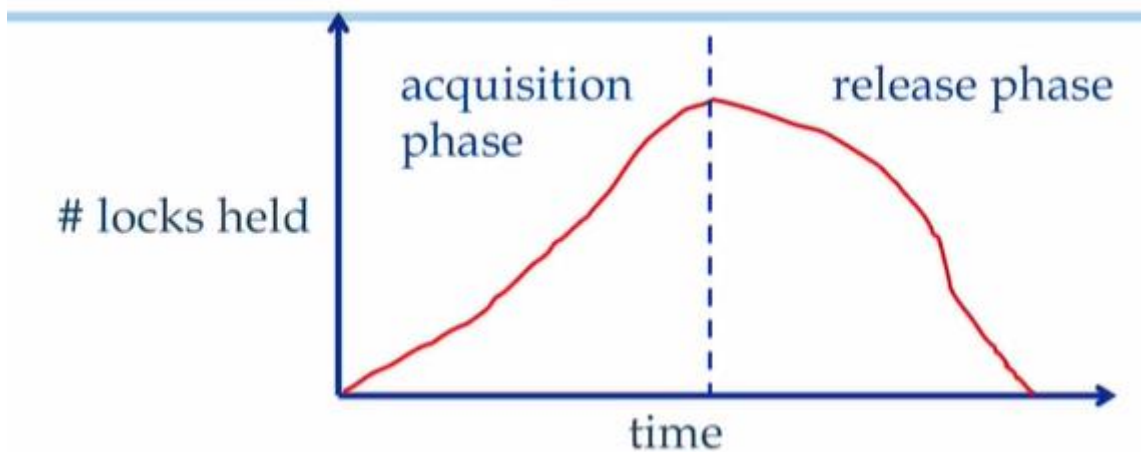
- ◆ Mapping pages to locations on disk (page#에서 file#, offset으로)
- ◆ Loading pages from disk to memory
- ◆ Saving pages back to disk & ensuring writes
- 상위 level에서는 이 layer에게 다음과 같은 기능(api)를 호출한다.
 - ◆ Read/write a page
 - ◆ Allocate/de-allocate logical pages

Concurrency Control Implementation

- **Concurrency Control**이란 다수의 사용자가 Database에 접근해서 어떤 연산을 수행하더라도 그 Data의 접근을 빠르고 올바르게 결과를 낼 수 있도록 하는 것이 Concurrency Control이다.
- **Concurrency Control**은 실제 연산결과는 순차적 수행된 것과 같이 그리고 Disorderly processing한 것처럼 보이도록 해야 한다.
- **Concurrent Execution의 중요성**
 - **The throughput argument** : cpu의 여러 core를 사용해서 multiple transaction을 한다면 throughput이 향상, 즉 성능이 향상되기 때문이다.
 - **The latency argument** : 상관이 없는 데이터를 접근하면 다른 실행 중이던 latency에 영향을 받지 않기 때문에 또한 성능이 향상된다.
- 중요 개념
 - Transaction : Collection of operations that form a single logical unit
 - ◆ A sequence of many actions considered to be one atomic unit of work
 - ◆ 사용자 기반 즉, user defined atomic unit of work이다.
 - ACID Guarantee
 - ◆ Atomicity : All actions in the Xact happen, or none happen.
 - ◆ Consistency : If the DB starts out consistent, it ends up consistent at the end of Xact
 - ◆ Isolation : Execution of each Xact is isolated from that of other Xacts
 - ◆ Durability : If a Xact commits, its effects persist.

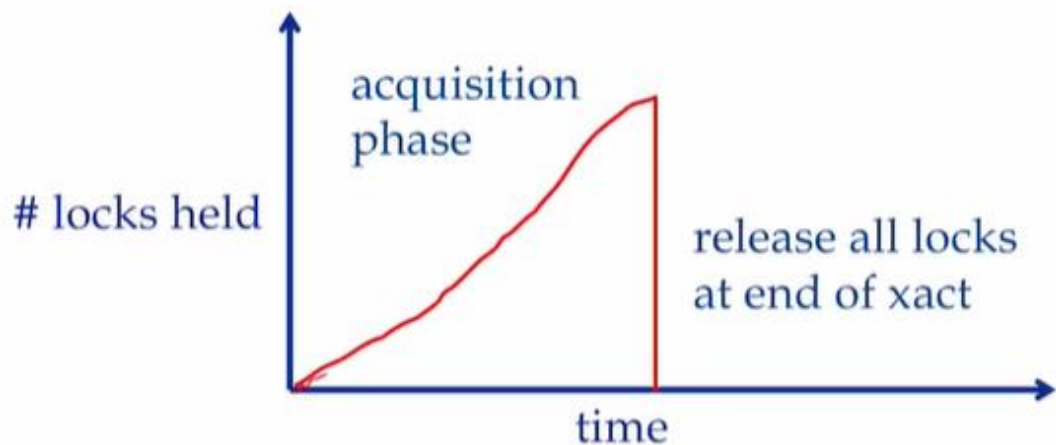
- **** Isolation Property (Concurrency) :** 여러 사용자가 Xact를 수행 중이더라도 혼자 돌고 있는 것처럼 해야 한다는 것. (혼자 serial order 수행 중이라고 느낄 수 있도록)
- Serial Schedules
 - 정의 : transaction set을 펼쳐서 하나씩 수행한 것들
 - Schedules are equivalent if they
 - ◆ 같은 transactions 포함
 - ◆ 각 transaction들의 action들의 순서가 동일
 - ◆ 둘의 schedule의 실행 후 결과의 상태가 동일한 상태
 - Schedule S is serializable if
 - ◆ S가 any serial schedule과 equivalent할 때
- Conflicting Operations
 - 정의
 - ◆ 서로 다른 transaction에서 수행
 - ◆ 같은 object를 대상으로 수행
 - ◆ 둘 중 적어도 하나가 write일 경우
- **CC (Concurrency Control)**
 - 크게 2가지로 나뉨
 - 1. PCC (Pessimistic CC)
 - ◆ Conflict가 일어날 것이다.
 - ◆ 미리 조치를 취함 – Locking (Conflict detecting)
 - ◆ **2PL (Two Phase Locking) – Locking 기법**

- 어떤 output schedule s 에 대해 그리고 s 의 모든 transaction에 대해서 어떤 q_i 이 첫 ou_i step 뒤에 나오면 안된다는 기법을 말한다. q, o 는 (r 또는 w)
- Lock object contains - $\langle rid, S / X \text{ (Lock mode) } , TXN_id \rangle$**



→ Cascading abort 발생 가능성이 있다. (중간에 abort시)

◆ Strict 2PL (Cascading abort방지)



- 모든 transaction이 끝나면 그 때 lock들을 release하는 것 (abort commit 결정 후 unlock하는 것)

◆ Lock management

- Lock을 관리하기 위한 module
- Hash table & linked list를 이용, rid를 key로
- Conflict lock
 - 존재하지 않으면 – 바로 request lock에 넣어주고 proceed
 - 존재하면 – wait queue에 넣음, sleep 시킴

◆ Deadlocks

- transaction들이 서로가 서로를 붙잡고 있는데 cycle이 생길 때
- 방법
- 1. Prevention / Avoidance
 - 누가 붙잡고 있는지, 우선순위가 누가 더 높은 지
 - T_i 가 T_j 보다 뒤에 들어온 경우
 - **Wait-Die** : 만약 T_i 가 우선순위가 높으면 T_j 를 기다리고 그렇지 않으면 T_i abort
 - **Wound-Wait** : 만약 T_i 가 더 높으면 T_j 를 abort 아니면 T_i 기다림
 - 2가지 방법 모두 우선순위가 높아지는 방향 또는 낮아지는 방향으로만 wait하므로 Deadlock이 발생하지 않음
 - 장점 : 안전한 wait cycle을 만들기 때문에 반드시 수행 가능
 - 단점 : 생기지 않는 상황에도 priority를 자꾸 abort를 시키는 상황이 발생
- 2.Detection and Resolution

- Wait-for graph를 생성시킴

- ◆ 단점 : graph traversal을 자주 해야함

- → 주기적으로 측정해보자 (adaptive하게)

- ◆ Locking Granularity

- Fine granularity : high concurrency, high locking overhead
 - Coarse granularity : low locking overhead, low concurrency
 - New Lock Modes

- Concurrency 향상을 위해

- IS

- IX

- SIX : S & IX (Select for update)

- 2. OCC (Optimistic CC)

- ◆ Conflict가 안 일어날 것이다.

- ◆ 3가지 phase

- 1. Read phase

- Record 읽어서 private workspace에 씴 (DB에서 copy)

- 2. Validation phase

- CSR 검사 (conflict serializable)

- RS(t), WS(t)으로 validation검사

- ◆ BOCC : t_j 를 이전의 모든 t_i 와 비교

- ◆ FOCC : t_j 모든 active transaction set과 비교

- 3. Write phase
- ◆ Multiversion CC (MVCC)
 - **Snapshot Isolation**

Crash-Recovery Implementation

- Recovery : 어떤 종류의 failure가 일어 났던지 간에 결과가 항상 올바르게 있어야 한다는 것
- Recovery Manager : ACID properties 중 Atomicity & Durability를 관여하고 있고 이를 지키기 위해 logging을 하고 있음
 - Atomicity : Transactions may abort ("Rollback")
 - Durability : 중간에 꺼져도 올바른 상태로 남아 있어야함 (올바르게 commit한 결과가 오직 DB에 반영) 진행 중인 것이 종료가 되었으면 rollback을 시켜야 한다.
 - ◆ Redo 필요
 - Undo / Redo : transaction이 제대로 끝났는지 끝나기 전에 종료가 되었는지에 따라 나뉨
- 가정 : Strict 2PL, Updates are happening in place
 - Important component : Buffer management가 Buffer pool에 어떤 정책을 가지고 있는지.
 - ◆ Force policy : transaction 종료 시 변경 된 내용을 무조건 DB에 반영되는 policy → syncing
 - Durability가 지켜 짐, redo logging이 필요하지 않음
 - Performance에 안 좋은 영향이 있음
 - ◆ No STEAL policy : transaction이 완전히 끝날 때 (commit) 까지 buffer에 있는 내용이 절대로 disk에 반영 (write) 이 되지 못하도록 하는 policy
 - Atomicity가 지켜진다, undo logging이 필요하지 않음
 - Performance에 안 좋은 영향이 있음

◆ Steal/No-Force (성능을 위해서)

- 성능이 향상이 되나 위의 policy 보다는 복잡함
- 추가의 recovery action이 필요함
- No Force (Durability와 연관) : Durability를 보장해야 하므로 Redo action이 필요함 (새로 들어온 값을 기억해야 하므로)
- Steal (Atomicity와 연관) : Atomicity를 위해서 Undo가 필요 (old값을 기억해야 하므로)

	No Steal	Steal		No Steal	Steal
No Force		Fastest	No Force	No UNDO REDO	UNDO REDO
Force	Slowest		Force	No UNDO No REDO	UNDO No REDO
Performance Implications			Logging/Recovery Implications		

■ 아래에는 필요한 function & component들이다.

■ Logging

◆ Log : An ordered list of log records to allow REDO / UNDO

- Log record contains
 - <XID, pageID, offset, length, old data (for undo), new data (for redo)> 기본적으로 들어가야할 것들
- 페이지를 변경할 때 마다 log가 발생함
- Write-Ahead Logging Protocol (log를 write하는 규칙)

- Must force the log record for an update before the corresponding data page gets to the DB disk
 - ◆ Atomicity를 위해서 – Undo
- Must force all log records for a Xact before commit
 - ◆ Durability – Redo
- WAL기법으로 log를 쓴다고 했을 때에는 LSN이 발급이 됨
 - LSN (Log Sequence Number) : $\text{pageLSN} \leq \text{flushedLSN}$ 이것은 잘 지켜줘야 함 아니면 recovery 불가
- Log Records
 - ◆ 변경하는 작업에 대해서 발급이 됨
 - ◆ Fields
 - LSN : 현재 record의 log sequence number
 - prevLSN : 이전 log, 없으면 NIL (NULL)
 - XID
 - Type
 - Update 등
 - 아래는 필수정보이다.
 - pageID, length, offset : 장소
 - before-image : 이전의 값 (old)
 - after-image : 덮어 쓴 내용 (new)
- transaction table
 - ◆ contains XID, status, lastLSN

- Dirty Page Table

- ◆ Dirty로 만든 첫번째 log record를 담고 있음
- ◆ 너무 오래된 것들은 최우선적으로 반영을 시키기 위해 만들어져 있다.

- Checkpointing

- ◆ Flush dirty pages
- ◆ Log – 복구를 위한 정보, dirty page를 disk에 반영이 되었다면 log를 truncate 시켜도 됨
- ◆ Log를 truncate 시키지 않으면 복구하는데 매우 많은 시간이 걸림, recovery time을 줄일 수 있음 – foreground Xact와 trade off

- 요약

- Recovery는 크게 2가지 동작으로 볼 수 있다. (Undo / Redo)
- Log Records, Data pages, Xact Table, Dirty Page Table, Log tail, Buffer pool로 Recovery (Redo, Undo)가 동작하고 있다.

In-depth Analysis

1. Workload with many concurrent non-conflicting read-only transactions.
2. Workload with many concurrent non-conflicting write-only transactions.