Database System Report

Spring, 2020

School of Software, CAU

- IV -

20150804

정민준

1. Oracle, SQL Server, MySQL 중 하나를 선택하여 설치 및 구동하고 그 시스템의 대화식 SQL 도구 로 아래에 답하시오.

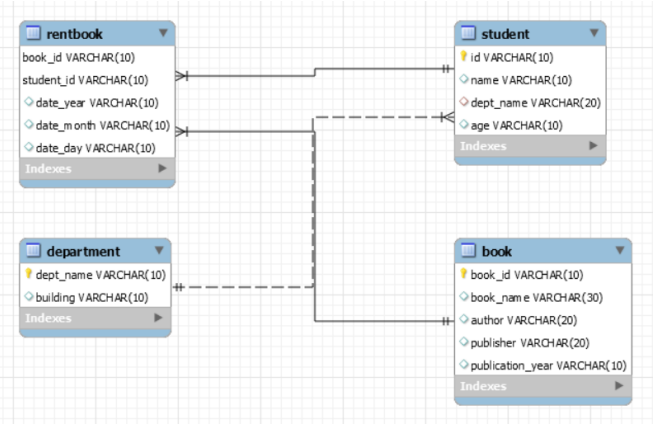
(a) 자신의 시스템 OS, 선택한 DBMS 제품명 및 에디션/버전, 대화식 SQL 도구(제품명 및 버전정보 등)을 기술하시오.

- Windows10, MySQL8.0, MySQL 8.0 Command line Client, MySQL workbench

(b) SQL문을 실행할 때 DBMS의 질의처리기가 어떤 질의처리 플랜 (또는 전략)을 사용하였는지 또 그러한 질의처리의 성능에 대한 정보 (처리 시간 등)를 대화식 SQL 도구로 확인할 수 있다. 어떤 질의 q가 있다고 하자. q를 자신이 사용한 실제 DB 시스템에서 처리한 후 대화식 SQL 도구로 확인했을 때 q의 질의 처리 플랜은 p이고 그 비용은 c였다고 하자. 교재 15장 (6판 12장) query processing에서 각 연산의 비용을 추정(estimate)하는 방법, 16장 (6판 13장) query optimization에서는 각 연산의 결과 크기를 추정하는 방법을 공부했는데 누군가로부터 다음과 같은 질문을 받았다고 하자. "q를 처리하는 플랜 p의 비용을 이번학기에 교재에서 공부한 방법들로 추정한 비용은 c와 어느 정도 일치합니까?" 이 질문에 대해 명확하고 정확한 답변을 하기는 힘들지 모른다. 왜냐하면 교재의 추정 방법에서 사용하는 정보와 자신이 사용하는 대화식 SQL 도구가 제공한 질의처리 플랜 및 비용에 대한 정보에 차이가 있을 수 있기 때문이다. 이러한 제약을 감안한 상태에서 아래의 방법을 준수하여 이번학기 DB 시스템 수강자로서 위 질문에 대한 각자 나름의 답변을 제시하시오.

- 자신의 설계과제(개인 프로젝트)의 응용분야에서 작성한 다양한 질의문들과 테스트 DB를 예로 이용하고 해당 질의문들을 대화식 SQL 도구로 실행해서 얻은 정보를 토대로 한다. 자신의 (i) DB 스키마 (ii) 질의문들 (iii) 생성된 테스트 DB가 위 질문에 대한 답변의 논지 전개에 부족하다고 판단할 경우 왜 그러한지 이유를 쓰고 (i),(ii),(iii) 등을 확장하고 어떻게 확장하였는지 설명한 후 예로 이용한다. 예로 사용한 질의 문들에 대해서는 설계과제에서 부여했던 고유식별자 (예: S1-1, S2-3, ... 등)를 명시한다.
- 자신의 대화식 SQL 도구가 제공하는 질의처리 플랜 및 비용에 대한 정보가 교재의 추정 방법들이 사용하는 정보 대비 부족한 부분에 대해서는 관련 자료 조사 및 자신의 가정(assumption)에 따라 논지를 전개한다.

1) DB 스키마



- 먼저 MySQL 실행계획을 살펴보겠습니다. MySQL에서는 쿼리를 최적으로 실행하기 위해 각 테이블의 데이터가 어떤 분포로 저장돼 있는지 통계 정보를 활용하며, 그러한 기본 데이터를 비교해 최적의 실행계획을 수립하는 작업을 합니다. 이는 DBMS의 옵티마이저가 담당합니다. MySQL에서 쿼리가 실행되는 과정은 크게 3가지로 나눌 수 있습니다. 1) SQL 파싱, 2) 어떤 요소를 참조하여 실행할지에 대한 결정, 3) 두번째 단계에서 수립된 실행계획으로 읽어온 레코드를 조인하거나 정렬하는 작업.

- 저는 MySQL에서 쿼리 실행 계획을 분석하기 위해 EXPLAIN 명령과 MySQL workbench의 Visual Explain 기능을 사용했습니다. 수업에서 배운 내용을 활용한 비용 추정과 설계한 데이터베이스 시스템 내에서 연산 수행결과를 다음 질의문들을 통해 비교하겠습니다.

2) Queries

- 단순 SQL :

```
select count(id)
from student
```

- 인덱스를 활용한 SQL :

```
select count(id)
from student
where dept_name = 'Comp.sci';
```

- 자연조인 SQL :

```
select count(book_id)
from book natural join rentbook
```

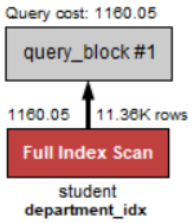
- 중첩 SQL :

```
select count(book_id)
from rentbook
where student_id in (
select id
from student
where dept_name = 'Comp.sci')
```

※ 인덱스를 이용하지 않는 SQL문은 where 절을 제거하였습니다. 이유는 where절의 여부와 관계없이 검사 유형이 ALL, 즉 모든 레코드를 검사하기 때문에 제외했습니다. 첫번째 연산을 기준으로 하여 나머지 연산들을 비교해 보겠습니다.

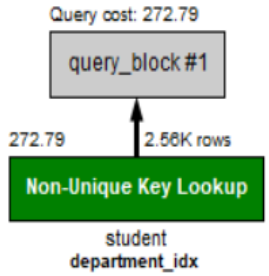
3) 테스트

① 단순 SQL Test:



- 레코드 수 = 11360, 쿼리 cost = 1160.
- select 연산 cost estimation, 1160 = br, 1160개의 블록이 있고 한 블록당 약 9.7개 튜플이 저장되어 있다고 가정하겠습니다.

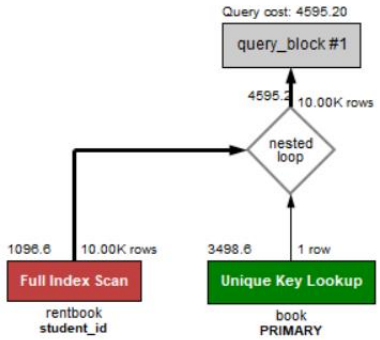
② 인덱스를 활용한 SQL:



- 저희는 key 가 아닌 속성에 대한 index scan 에서 Cost 는 트리의 높이와 블록의 개수의 합으로 정의하고 있습니다. 대용량 데이터베이스 경우에도 Depth 가 4~5 이상으로 깊어지지 않고 4 이하라고 가정했을 때 작은 수임으로 블록의 개수로 추정이 가능합니다.

- 위의 가정으로 Cost를 계산해 보면 Comp.Sci로 매칭되는 블록수 + 트리의 높이로 계산이 됩니다. 학생은 13000명, 학과는 5학과로 13000 / 5 면 2600개의 칼럼이 존재합니다. 이를 9.7로 나누면 268.04가 계산되고 트리 높이를 더하면 270으로 추정할 수 있습니다. MySQL workbench를 통해 얻은 Cost 272.79와 비슷한 Cost가 계산되었음을 확인할 수 있습니다.

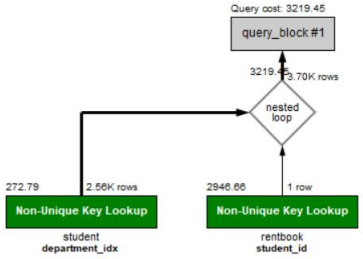
③ 자연조인 SQL:



- 저희는 Join 연산의 경우 튜플의 개수 * 블록 개수 + 블록 개수로 Cost를 정의하고 있습니다. 이대로 계산한 Cost는 10000 * 10000/9.7 + 15000/9.7 = 10,310,778이 계산됩니다. 하지만 MySQL 에서는 테이블 생성시 PK에 대한 인덱스가 생성되고 위와 같은 Cost 계산방식은 적절하지 않다고 판단했습니다.

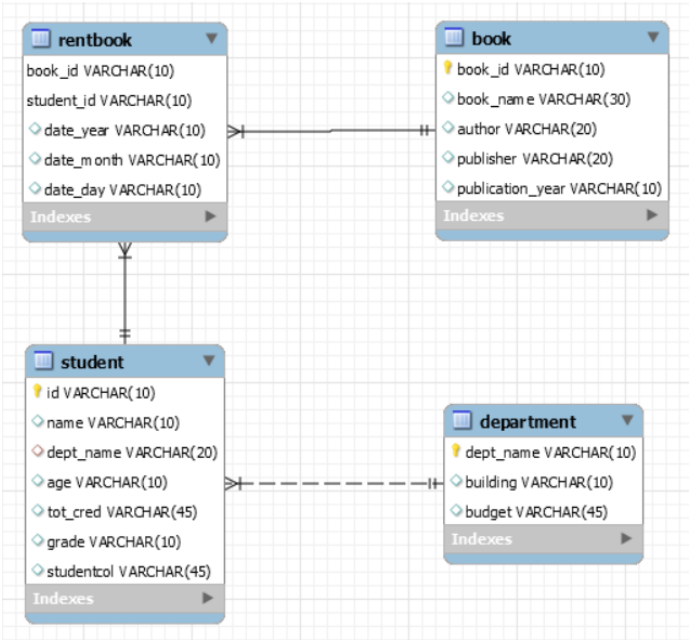
- rentbook은 book 테이블을 참조하고 있기 때문에 book table을 전부 비교할 필요가 없습니다. 키값에 대한 select 연산과 같다고 생각하면 (15000/9.7/2 + 10000 / 9.7) * 3 = 5319로 코스트를 추정할 수 있습니다. 실제 MySQL 에서는 4595라고 계산했는데 이는 블록당 튜플 개수를 모든 테이블에 일정하게 가정하고 실제 트리 높이와 엔트리를 알지 못하기에 난 오차라고 생각했습니다.

④ **중첩 SQL:**



- student table에서 department 는 5개로 인덱스를 선언해 놓은 상태입니다. student_id 가 같은걸 비교하니 join 연산 Cost 계산과 같게 했습니다. 먼저 괄호안의 select문을 계산하면 13000 / 5 면 2600개의 칼럼이 존재합니다. 이를 9.7로 나누면 268.04가 계산되고 트리 높이를 더하면 270으로 Cost를 추정할 수 있습니다. 그리고 rentbook에 대해 Cost를 추정해보면 10000 / 9.7 * 3 = 3092로 total Cost는 약 3362로 추정할 수 있습니다. MySQL Workbench 에서는 3219.45로 Cost를 추정하였으며 해당 Cost보다 조금 높게 추정했음을 확인할 수 있었습니다.

2. 자신의 설계과제(개인 프로젝트)의 응용분야에서 작성한 다양한 트랜잭션들을 예로 이용하여 아래에 답하시오. 단, 자신의 (i) DB 스키마 (ii) 트랜잭션들이 아래 질문에 대해 답하기에 부족하다고 판단할 경우 왜 그러한지 이유를 쓰고 (i), (ii) 등을 확장하고 어떻게 확장하였는지 설명한 후 예로 이용한다. 아래 각 문항에서 예로 사용한 트랜잭션들에 대해서는 설계과제에서 부여했던 고유식별자(예: T1, T2, ... 등) 를 명시한 후 응용의 어떤 기능을 수행하기 위한 것인지 기술한다. 기타 필요한 가정이 있으면 기술한다.



- 이전 스키마에서 확장된 내용을 설명드리겠습니다. 먼저 department 테이블에서 budget, 예산 속성이 추가되었고 student 테이블에 학기, 학점 속성인 grade, tot_cred 가 추가되었습니다.

(a) 서로 다른 두 트랜잭션의 연산 간에 read-write 충돌(conflict) 관계인 예를 제시하시오.

- student와 department에 대한 두 트랜잭션이 있습니다. T1은 A라는 id를 가진 학생이 전과를 하여 소속 학과와 건물 속성값이 교체되는 트랜잭션입니다. T2는 컴퓨터공학과가 이전을 해서 건물 속성값을 교체한다고 가정하겠습니다. 만약 학과 트랜잭션이 진행전에 학생 정보가 교체된다면 이전 건물 번호가 입력됨으로 충돌이 발생하게 됩니다.

(b) 서로 다른 두 트랜잭션의 연산 간에 write-write 충돌(conflict) 관계인 예를 제시하시오.

- department에 대한 두 트랜잭션이 있습니다. T1은 물리학과의 예산을 100만원 추가하고 싶고 T2는 물리학과의 예산을 두배 증가시키고 싶다고 가정하겠습니다. 만약 기존 물리학과 예산이 100만원이라고 가정하면 T1 -> T2, T2 - > T1 순서의 트랜잭션으로 진행된다고 가정하면 400만원, 300만원으로 충돌이 발생하게 됩니다.

(c) 연산 간 충돌이 존재하는 서로 다른 두 트랜잭션의 serializable한 concurrent (즉, 비직렬 (non-serial)) 스케쥴의 예를 제시하고 serializable한 이유를 설명하시오. 두 트랜잭션이 접근 및 변경하는 데이터의 상기 스케쥴 시작 전 디스크에서의 초기값을 예로 들고 이들 데이터 값이 상기 스케쥴대 로 수행되는 과정에서 버퍼 및 디스크에서 어떻게 변해가는지 설명하시오. 기타 필요한 가정이 있으면 기술하시오.

- T1은 A라는 학생이 1학기동안 20학점을 수강한 정보를 갱신하고 T2는 A라는 학생이 2학기에 18학점을 수강한 정보를 갱신하는 트랜잭션 입니다.

- 여기서 tot_cred를 T, grade를 G로 가정하겠습니다. 하드디스크에 학생 A의 정보는 신입생이며 이전까지 수강한 과목이 없다는 상태로 가정하겠습니다. (grade : 0, tot_cred : 0) T1, T2를 시간의 흐름으로 표현하고 버퍼에 저장된 값을 순서대로 적겠습니다. 그리고 serializable한 스케줄임을 확인하겠습니다.

1)

- T1 : read(T), write(T + 20) || read(G), write(G + 1) ||
- T2 : || read(T), write(T + 18) || read(G), write(G + 1)

① A : G = 0, T = 20, ② A : G = 1, T = 20, ③ A : G = 1, T = 38, ④ A : G = 2, T = 38

2)

- T1 : read(T), write(T + 20) || || read(G), write(G + 1) ||
- T2 : || read(T), write(T + 18) || || read(G), write(G + 1)

① A : G = 0, T = 20, ② A : G = 0, T = 38, ③ A : G = 1, T = 38, ④ A : G = 2, T = 38

- 다음과 같이 똑같은 결과를 보임을 확인할 수 있었습니다.

(d) 연산 간 충돌이 존재하는 서로 다른 두 트랜잭션의 serializable하지 않은 concurrent (즉, 비직렬 (non-serial)) 스케쥴의 예를 제시하고 serializable하지 않은 이유를 설명하시오. 두 트랜잭션이 접근 및 변경하는 데이터의 상기 스케쥴 시작 전 디스크에서의 초기값을 예로 들고 이들 데이터 값이 상기 스케쥴대로 수행되는 과정에서 버퍼 및 디스크에서 어떻게 변해가는지 설명하시오. 기타 필요한 가정이 있으면 기술하시오.

- T1 트랜잭션은 컴퓨터 공학과의 예산을 100만원 지원하고 T2 트랜잭션은 모든 학과의 예산을 두배 증가시키는 기능을 수행합니다.

- 학과는 총 5개입니다. 각 학과의 예산을 가정하고 진행하겠습니다. 각 학과의 첫 글자 알파벳으로 표현하겠습니다. Economics 는 E1, English 는 E2. (E1.B는 경영학과의 예산)

- Comp.sci : 100, Physics : 200, Economics : 100, Biology : 150, English : 120.

1) **T1** **T2**

read(C), write(C + 100) ||
|| read(C), write(C * 2)
|| read(P), write(P * 2)
|| read(E1), write(E1 * 2)
|| read(B), write(B * 2)
|| read(E2), write(E2 * 2)

① C .B = 200, ② C.B = 400, ③ P.B = 400, ④ E1.B = 200, ⑤ B.B = 300, ⑥ E2.B = 240

2) **T1** **T2**

|| read(C), write(C * 2)
read(C), write(C + 100) ||
|| read(P), write(P * 2)
|| read(E1), write(E1 * 2)
|| read(B), write(B * 2)
|| read(E2), write(E2 * 2)

① C.B = 200, ② C.B = 300, ③ P.B = 400, ④ E1.B = 200, ⑤ B.B = 300, ⑥ E2.B = 240

- 다음 Comp.sci 학과의 예산의 결과가 다르게 나온걸 확인할 수 있습니다.

(e) 연산 간 충돌이 존재하는 서로 다른 두 트랜잭션의 스케쥴로서 serializable하지만 recoverable하지 않은 스케쥴의 예를 제시하고 이유를 설명하시오.

- Physics 학과의 예산을 100, 300을 증가시키는 트랜잭션 T1, T2가 있습니다.

- T1 : read(P), write(P + 100) ||
- T2 : || read(P), commit, write(P + 300)

- 위와 같은 경우에서 T2가 T1보다 먼저 commit을 해버렸기 때문에 복구가 불가능한 스케줄이 되어 버립니다. 복구가능한 스케줄이려면 T1이 T2보다 먼저 commit을 해야할 것입니다.

(f) 연산 간 충돌이 존재하는 서로 다른 두 트랜잭션의 스케쥴로서 serializable and recoverable한 스케쥴의 예를 제시하고 이유를 설명하시오.

- (e) 에서 설명한 트랜잭션을 수정하여 설명하겠습니다.

- T1 : read(P), write(P + 100), commit ||
- T2 : || read(P), write(P + 300)

- 다음과 같은 경우 먼저 read-write연산을 수행한 T1이 T2보다 먼저 commit 명령어를 수행함으로써 해당 스케줄은 복구가능한 스케줄임을 확인할 수 있습니다.

(g) 연산 간 충돌이 존재하는 서로 다른 두 트랜잭션의 스케쥴로서 cascading rollback이 발생할 수 있는 스케쥴의 예를 제시하고 이유를 설명하시오.

- Comp.sci 의 예산을 증가시키는 두 트랜잭션을 보겠습니다.

T1		T2
read(C)	||	
write(C + 100)	||	
	||	read(C)
	||	write(C * 2)
Abort	||	

- 다음과 같이 T1이 100만큼 증가시키고 T2가 두배 증가시킨 후 T1의 rollback에 의해 T2도 같이 rollback하는 cascading rollback이 발생하게 됩니다.

(h) 연산 간 충돌이 존재하는 서로 다른 두 트랜잭션의 스케쥴로서 cascadeless 스케쥴의 예를 제시하 고 cascadeless한 이유를 설명하시오.

- 위의 예산을 증가시키는 두 트랜잭션을 변형하여 설명하겠습니다.

T1		T2
read(C)	||	
write(C + 100)	||	
commit	||	
	||	read(C)
	||	write(C * 2)
Abort	||	

- 다음과 같은 스케줄링에선 rollback의 경우 커밋 이전의 쓰기작업을 허용하기 때문에 cascadeless한 스케줄입니다.

(i) 연산 간 충돌이 존재하는 서로 다른 두 트랜잭션 각각이 2PL 프로토콜을 준수할 경우 생성될 수 있는 스케줄의 예를 제시하고 생성 과정을 설명하시오.

- C 학생의 수강한 학점을 반영하는 트랜잭션 T1, T2가 있습니다.

T1	T2
Lock(C)	||
read(C)	||
write(C + 15)	||
Unlock(C)	||
	|| Lock(C)
	|| read(C)
	|| write(C + 20)
	|| Unlock(C)

- two-phase locking protocol을 따르는 트랜잭션은 read, update 연산을 수행하기 전에 반드시 lock을 설정해야 하며, 트랜잭션종료 전에는 반드시 unlock을 하여 다른 트랜잭션이 lock을 얻을 수 있도록 해야 합니다.

- 다음 T1 트랜잭션에서는 C 학생의 학점을 갱신하기 전, 락을 획득하고 write 이후 언락을 해주었습니다. 따라서 이후에 실행되는 T2 트랜잭션이 락을 획들할 수 있었고 write 연산 이후 언락을 실행한 스케줄입니다.

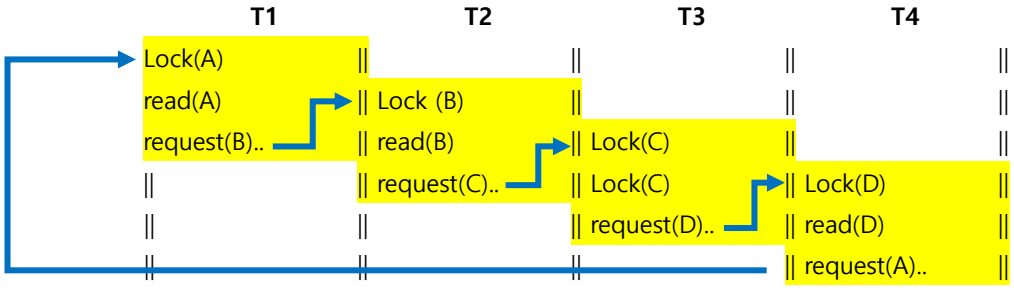
(j) 연산 간 충돌이 존재하는 서로 다른 두 트랜잭션 각각이 strict 2PL 프로토콜을 준수할 경우 생성될 수 있는 스케줄의 예를 제시하고 생성 과정을 설명하시오.

- A학생과 B학생의 총 학점을 반영하는 트랜잭션 T1, T2가 있습니다.

T1	T2
Lock(A)	||
read(A)	||
write(A + 15)	||
Unlock(A)	||
Lock(B)	||
read(B)	||
write(B + 10)	||
	|| Lock(A)
	|| read(A)
	|| write(A + 20)

- 2PL 프로토콜과 동일하지만 전체적인 락을 트랜잭션이 성공적으로 commit, abort하기까지 가지고 있습니다. 이 경우에 cascadeless 복구를 보장할 수 있습니다.

(k) 2PL 프로토콜로 동시성 제어가 수행된다고 할 때, 네 개의 서로 다른 트랜잭션 간에 deadlock이 발생하는 예를 제시하고 설명하시오.



- 다음과 같이 A,B,C,D의 학생에 대해 read-write 연산을 수행하는 트랜잭션 T1, T2, T3, T4가 있습니다. 각 트랜잭션이 순차적으로 학생의 정보를 읽어야 하는데 각 트랜잭션이 락을 걸어놓은 상태이고 상호간 락이 해제되길 기다리는 deadlock 현상이 발생하게 됩니다. 이를 해결하기 위해서 나온 방법은 Wait-die, TimeOut 등이 있습니다.

(l) 네 개의 서로 다른 트랜잭션을 고려하자. 이들 간에는 연산 충돌이 없다고 가정하자. 이들이 strict 2PL 프로토콜에 의한 동시성 제어로 concurrent 스케쥴로 수행되다가 system crash가 발생했다고 하자. 넷 중 한 트랜잭션은 고장 전에 commit하였고 하나는 abort하였다고 하자. 교재의 immediate DB modification (즉, steal policy)가 사용되었고 commit되지 않은 변경이 디스크로 output된 경우가 최 소 1회 있었다고 가정한다. 고장 전까지의 concurrent 스케쥴 및 그에 따른 로그의 예를 제시하시오. 네 트랜잭션이 접근 및 변경하는 데이터의 상기 스케쥴 시작 전 디스크에서의 초기값을 예로 들고 이들 데이터 값이 상기 스케쥴대로 고장 전까지 수행되는 과정에서 버퍼 및 디스크에서 어떻게 변해가는지 보이시오. 이후 상기 로그로 교재의 복구 알고리즘이 수행되는 과정을 복구 대상 데이터의 버퍼 값 변화를 보이면서 설명하시오. 기타 필요한 가정이 있으면 기술하시오.

- 각 트랜잭션 수정이력, log와 write, output 세 칼럼으로 나누어 스케줄링을 나타내었습니다.

Log		Write		OUTPUT	
T1, start	||		||		||
T1, A: 100-> 50	||		||		||
	||	A = 50	||		||
	||		||	A	||
T2, start	||		||		||
T2, B: 50-> 100	||		||		||
	||	B = 100	||		||
T2, abort	||		||		||
T3, start	||		||		||
T3, C: 100-> 70	||		||		||
	||	C = 70	||		||
T3, commit	||		||		||
T4, start	||		||		||
T4, D: 100-> 0	||		||		||
	||	D = 0	||		||

System Crash!

	||		||		||
	||		||		||
			.		
			.		
			.		

- Undo(T1) : A가 100으로 저장, but 디스크에 50으로 이미 내려가있는 상태
- ~~Undo(T2)~~ : abort된 연산이니 신경쓰지 않습니다.
- Redo(T3) : C를 70으로 변경.
- Undo(T4) : D를 100으로 저장.

- 순서대로 살펴보겠습니다.

1) Undo(T4)

A : B : C : D : 100

2) Redo(T3)

A : B : C : 100 D : 100

A : B : C : 70 D : 100

3) ~~Undo(T2)~~

A : B : 50 C : D : 100

4) Redo(T1)

A : 100 B : 50 C : 100 D : 100

- 현재 문제에선 연산 충돌이 없는 경우를 가정하였습니다. 지금 T1이 commit하기 전에 A 아웃풋이 발생했는데 만약 연산충돌이 있는 경우, 즉 이후 트랜잭션에서 A의 값을 수정한 이력이 있었고 이의 경우에서는 잘못된 값으로 복구될 수 있습니다. 트랜잭션은 버퍼를 수정하는 작업이며 바로 디스크를 수정하는 아웃풋 연산을 하지 않습니다. 아웃풋 시점은 시스템이 적절한 시기를 정하는 것인데 이런 경우를 방지하기 위해 커밋하기 전까지는 아웃풋을 지연시켜야 할 것입니다.

3. 자신의 설계과제(개인 프로젝트)로 개발한 응용의 어떤 테이블 R의 관계 레코드들을 MapReduce를 지원하는 빅데이타 저장 시스템인 key-value store에 저장했다고 하자. 자신의 응용이 제공하는 R에 대한 검색 기능 중 MapReduce로 병렬처리 가능한 것을 제시하시오. 테이블 R의 이름 및 스키마 그리고 해당 검색 기능이 무엇인지 설명한 후 해당 map 및 reduce 함수를 수업시간에 공부한 것과 같이 슈도코드로 기술하고 설명하시오. 만약 자신의 응용에서 (i) DB 스키마 및 (ii) 검색 기능 중 MapReduce로 병렬처리 가능한 것이 없다고 판단할 경우 왜 그러한지 이유를 쓰고 (i),(ii) 등을 수정 또는 확장하고 어떻게 수정 또는 확장하였는지 설명한 후 예로 이용하시오.

- 제가 설계한 테이블 중 제시할 테이블은 student table 입니다. 학생의 나이, 학과, 학년의 정보를 수집하여 final result를 생성하는 프로그램입니다. student 테이블의 칼럼은 병렬처리가 가능하고 다른 테이블 보다 더 활용도가 높을거라 예상되어 선정하게 되었습니다.

1) def Map(input_data) :

Result_list = [] <- 결과를 담을 리스트를 선언합니다.
Result_set = {} <- 결과를 담을 딕셔너리를 선언합니다.
For each data in input_data <- 인풋 데이터를 하나씩 불러옵니다.
    If new data <- 만약 새로운 정보가 들어온다면
        Set Result[col] = 1
    else<- 이전 정보와 같은 정보가 들어온다면 잘못된 데이터일 확률이 아주 높기에 예외처리 하겠습니다.
        Continue
    Add Result_list <- 결과를 리스트에 추가합니다.
return Result_list <- 인풋 데이터에 대한 정보를 전부 담으면 이대로 반환!

2) def Reuce(input_data) :

Reduce_set = {} <- 결과를 담을 딕셔너리를 선언합니다.
For each data in input_data <- 인풋 데이터를 하나씩 불러옵니다.
    If new data <- 만약 새로운 정보가 들어온다면 1로 초기화해줍니다.
        Set Result[col] = 1
    else<- 이전 정보와 같은 정보가 들어온다면 횟수를 추가해줍니다.
        Set Result[col] ++
return Result_set <- 결과를 반환합니다.