3.2 InnoDB 스토리지 엔진 아키텍쳐

생성일	@2021년 5월 22일 오후 3:36
태그	

InnoDB = MySQL의 스토리지 엔진 가운데 가장 많이 사용됨

- 스토리지 엔진 중 거의 유일하게 레코드 기반의 잠금을 제공 → 동시성 처리가 가능 + 안정성 높음

1. InnoDB 스토리지 엔진의 특성

프라이머리 키에 의한 클러스터링

- 프라이머리 키를 기준으로 순서대로 클러스터링 되어 저장됨
- 프라이머리 키에 의한 range scan이 빠름

잠금이 필요없는 일관된 읽기

- multi version concurrency control 기술로 락을 걸지 않고 read
- read할때 락을 걸지 않기 때문에 다른 트랜잭션이 가지고 있는 락을 기다리지도 않는다

외래키 지원

but 잠금이 여러 테이블로 전파 되어서 그로 인해 데드락이 발생할 때가 많다

자동 데드락 감지

- 그래프 기반의 데드락 체크 방식 사용 → 데드락 발생과 동시에 바로 감지됨
- 롤백이 가장 용이한 트랜잭션을 자동으로 강종 한다

자동화된 장애 복구

손실이나 장애로부터 데이터 보호하기 위한 매커니즘 탑재됨

오라클의 아키텍쳐 적용

- MVCC wprhd
- Undo 데이터가 시스템 테이블 스페이스에 관리된다는 것

2. InnoDB 버퍼 풀

- 디스크의 `데이터 파일`이나 `인덱스 정보`를 메모리에 캐시해 두는 공간
- 쓰기 작업을 지연시켜 일괄작업으로 처리 ← 버퍼역할도 함
- DML 쿼리는 랜덤 디스크 작업을 발생시키지만 버퍼풀이 변경 데이터를 모아서 처리 → 랜덤 디스크 작업 횟수가 낮아짐

버퍼 풀의 메모리

: 많은 백그라운드 작업의 기반이 되는 메모리 공간

- `innodb_buffer_pool_size` 신중하게 설정
- 각 클라이언트 스레드가 사용할 메모리도 충분히 고려해서 설정

더티 페이지(Dirty Page)

- 아직 디스크에 기록되지 않은 변경 데이터를 버퍼 풀이 가지고 있다.
- InnoDB에서 주기적으로 어떤 조건을 충족시 체크포인트 발생

  → 이때 Write스레드가 필요한 만큼의 더티페이지만 디스크로 기록 (모든 더티페이지를 기록하지 않음)

3. 언두(Undo) 로그

: 언두 영역은 UPDATE / DELETE 문장으로 데이터 변경되기 전의 데이터를 보관하는 곳

- 언두 영역에 변경이전의 데이터가 백업이 되고, 롤백시 여기에서 데이터 파일이 복구된다

용도

1. 롤백 대비용
2. 트랜잭션의 격리 수준을 유지하면서 높은 동시성 제공하는데 사용됨 → REPEATABLE READ (커밋전에는 같은 데이터를 유지하는 것

4. 인서트 버퍼(Insert Buffer)

레코드 Insert/update 시 해당 테이블에 포함된 인덱스를 업데이트 하는 작업도 필요

문제점

인덱스 업데이트 작업 → 랜덤하게 디스크를 읽는 작업이 필요

테이블에 인덱스가 많다면 이 작업은 상당히 많은 자원을 소모

InnoDB의 Insert Buffer

- 변경해야할 인덱스 페이지가 버퍼 풀에 있으면 바로 업데이트를 수행
- 디스크로부터 읽어와서 업데이트를 해야한다면 이를 즉시 실행하지 않고 임시 공간인 Insert Buffer에 저장해 두고 바로 사용자에게 결과 반환

Unique 인덱스는 인서트 버퍼 사용 불가
- 결과 전달 전 반드시 중복 여부를 체크해야하므로

인서트 버퍼 머지 스레드

- 인서트 버퍼에 임시로 저장돼 있는 인덱스 레코드 조각이 이후 백그라운드 스레드에 의해 병합되는 것
- MySQL 5.5부터 INSERT/DELETE에 의해 키를 추가하거나 삭제하는 작업에 대해서도 버퍼링이 될 수 있게 개선되었다
- innodb_change_buffering 설정 파라미터가 새로 도입되어 작업의 종류별로 인서트 버퍼를 활성화 할 수 있음

5. Redo 로그 및 로그 버퍼

쿼리 커밋시, 데이터의 ACID를 보장하기 위해 즉시 변경된 데이터의 내용을 데이터 파일로 기록해야한다

→ 이러한 작업은 랜덤하게 디스크에 기록해야하기 때문에 디스크에 부하를 줌

→ 이 부하를 줄이기 위해 Buffer pool 같은 장치로 부하를 줄인다

리두 로그

버퍼 풀과 같은 목적인 디스크 부하를 줄이기 위해 변경된 내용을 순차적으로 디스크에 기록하는 로그 파일

- DBMS 데이터는 버퍼링을 통해 한꺼번에 디스크에 변경된 애용을 처리할 수 있고, 그로 인해 성능 향상이 있다
- BUT ! 변경작업 많은 DBMS 서버의 경우에는 리두로그 기록작업이 문제가 된다 → 최대한 ACID 속성을 보장하는 수준에서 버퍼링을 함

로그 버퍼

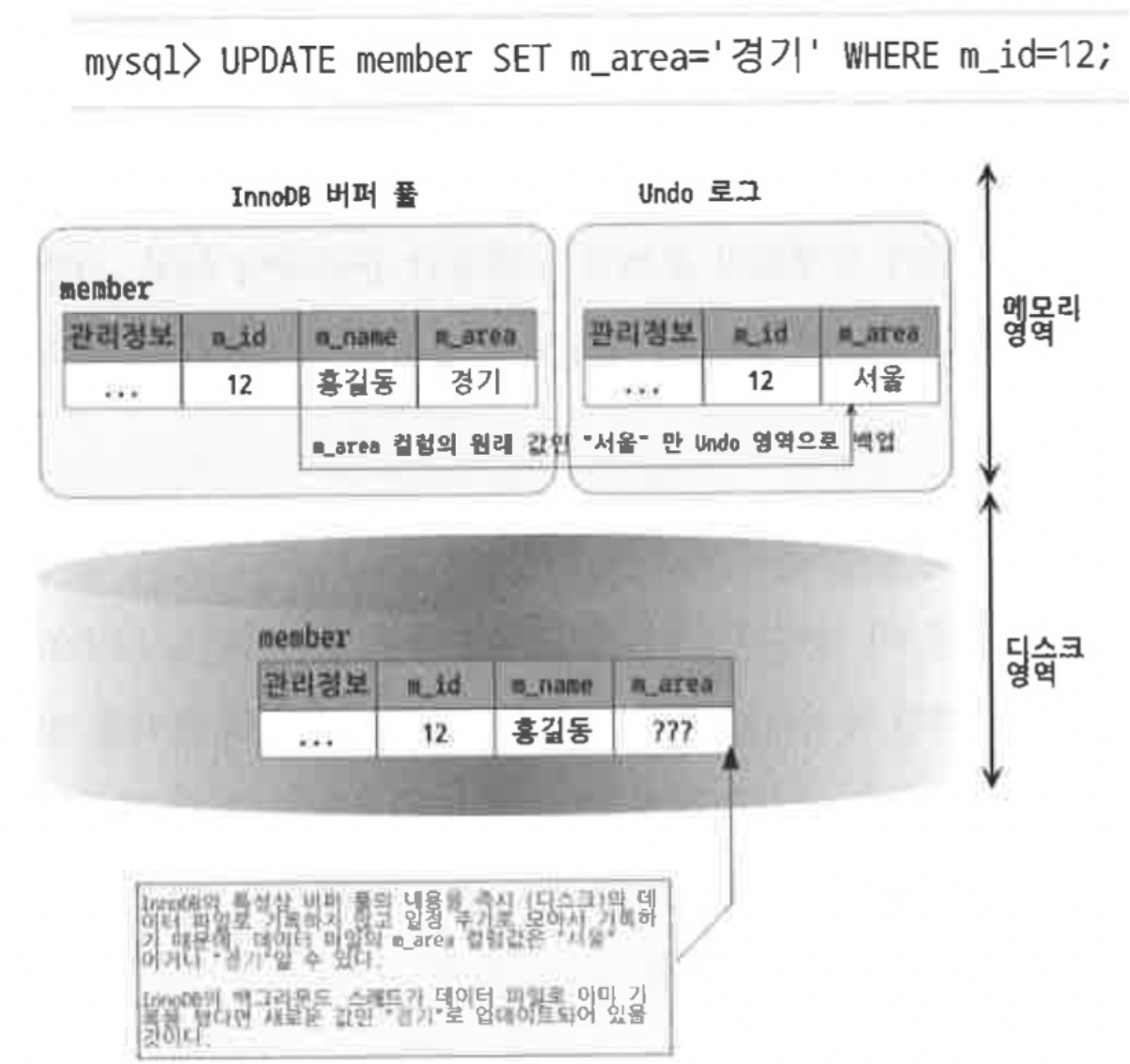
: 변경작업이 많은 DBMS에서 ACID 속성을 보장하는 수준에서 리두 로그 버퍼링을 하는데, 이때 사용되는 공간

6. MVCC (Multi Version Concurrency Control)

일반적으로 레코드 레벨의 트랜잭션을 지원하는 DBMS가 제공하는 기능

- MVCC의 가장 큰 목적 : Lock을 사용하지 않는 Repeatable read(일관된 읽기)를 제공
- InnoDB는 언두 로그를 이용해 이 기능을 구현
- MySQL의 격리 레벨은 READ_COMMITTED

여기서 업데이트 문 처리시



[그림 3-11] UPDATE 후 InnoDB 버퍼 풀과 데이터 파일 및 언두 영역의 변화

- 메모리 영역의 언두 로그에 변경 전 데이터가 들어가 있음
- 메모리 영역의 버퍼 풀에 변경 후 데이터가 들어가 있음
- 디스크 영역에서는 ***버퍼 풀 내용을 일정주기로 모아서 기록***하기 때문에 변경 전 가 들어갈 수도 있고 변경 후 데이터가 들어갈 수도 있다.
    - 백그라운드 스레드가 데이터 파일로 이미 기록을 했다면 변경 후 데이터가 들어가 있음

💡 아직 COMMIT이나 롤백이 되지 않은 상태에서 다른 사용자가 위 데이터를 조회한다면?
→ READ_UNCOMMITED라면 버퍼 풀이나 데이터 파일로부터 변경된 데이터를 읽어서 반환
→ READ_COMMITTED 인 경우, 아직 커밋되지 않아서 Undo영역의 데이터(=변경되기 이전의 내용)를 반환

📣 즉,하나의 레코드에 대해 2개의 버전이 유지되고, 상황에 따라 달라지는 구조
→ 트랜잭션이 길어지면 언두에서 관리하는 예전 데이터가 삭제안되고 오랫동안 관리 되어야 하고, system table space의 공간이 많이 늘어나야 하는 상황이 발생할 수도 있음

COMMIT 명령 실행시

InnoDB는 더 이상의 변경 작업 없이 지금의 상태를 영구적인 데이터로 만들어 버림

→ 여기서 롤백 실행시 Undo 영역에 있는 백업 데이터를 버퍼 풀로 복구하고, Undo 영역의 내용을 삭제함

7. 잠금 없는 일관된 읽기 (Non-locking consistent read)

- Read-Uncommitted
- Read-Committed
- Repeatable-Read

위 3개의 수준인 경우 INSERT와 연결되지 않은 순수한 읽기 (SELECT)작업은 다른 트랜잭션의 변경 작업과 관계 없이 항상 잠금을 대기하지 않고 바로 실행됨

⇒ Non-locking Consistent Read라고 하며 Undo 로그를 사용 (변경 전의 데이터를 읽기 위해)

오랜 시간 동안 활성 상태인 트랜잭션으로 인해 MySQL 서버가 느려지는 경우 → 보통 Consistent Read를 위해 Undo 로그를 삭제하지 못하고 계속 유지해야 하기 때문에 발생하는 문제

> 트랜잭션이 시작되었다면 ROLLBACK이나 COMMIT을 통해 트랜잭션을 완료하는 것이 좋다