12.2 SQL 문장별로 사용하는 잠금

생성일	@2021년 8월 8일 오후 4:32
태그	

InnoDB 테이블은 각 쿼리의 종류별로 사용하는 잠금 방식이 다름. 각 쿼리의 패턴별로 어떤 잠금을 사용하는지 araboja

1. SELECT 쿼리의 잠금

기본 SELCT FROM 문

기본적으로 SELECT 쿼리는 별도의 잠금을 사용하지 않는다 (INSERT .. SELECT가 아닌 이상)

- 읽어야 할 레코드가 다른 트랜잭션에 의해 변경되거나 삭제되는 중이면 InnoDB에서 관리하는 Undo log(데이터 변경이력)를 이용해 레코드를 읽는다.
  - 그래서 다른 트랜잭션에 영향 받지 않고, 대기하지도 않음
  - DDL 문장으로 테이블의 구조가 변경되는 중에도 SELECT FROM은 처리될 수 있음
- SERIALIZABLE 격리 수준인 경우는 좀 다름
  - 자동으로 LOCK IN SHARE MODE 옵션이 덧붙여져서 실행됨
  - 그래서 읽기 잠금 획득 후 읽기 실행

SELECT FROM .. LOCK IN SHARE MODE

WHERE 절에 일치하는 레코드 뿐만 아니라 접근한 모든 레코드에 대해 Shared next-key lock을 필요로 함

- 만약 읽기 잠금을 걸어야하는 레코드가 ! 다른 트랜잭션에 의해 쓰기 잠금이 걸려있을 시, 해제될때까지 기다려야함
- 위 상황에서 다른 트랜잭션이 쓰기 잠금이 아니라 읽기 잠금이 걸었다면 상호호환되어서 대기 없이 읽기 잠금 획득 가능
- COMMIT이나 ROLLBACK명령으로 트랜잭션 종료시 자동 해제

- DDL 문은 실행완료시 자동으로 트랜잭션 종료되므로 획득된 잠금도 DDL 문장 실행시 자동으로 해제됨

SELECT FROM .. FOR UPDATE

위와 차이점은.. FOR UPDATE 사용시 SELECT 쿼리는 스냅샷을 이용한 읽기를 사용하지 못하기 때문에(쓰기 잠금까지 걸어버리기 때문?!?!? 쓰기 반납될때까지 기다려야해서?!) Consistent READ가 무시됨

2. INSERT 쿼리의 잠금

INSERT 문장은 기본적으로 배타적 레코드 잠금 사용. 테이블의 프라이머리 키/유니크 키로 중복 체크를 위해 공유 레코드 잠금을 먼저 획득한다

인서트 인텐션 락

INSERT를 실행할 의도를 지닌 쿼리가 획득해야하는 잠금

- 모든 인서트 쿼리는 얘를 획득한 후 insert 실행
- insert된 레코드에 대해서는 배타적 레코드 잠금을 자동으로 획득하게 됨
- 갭 락(ㄱ나니,,? 인서트 하는 레코드가 들어갈 곳 사이사이 레코드에 거는 락)의 일종으로, 인서트 인텐션 락 끼리 서로 호환됨
  - 즉, 여러 트랜잭션이 동시에 인서트 인텐션 락을 획득할 수 있음
  - 하지만 이미 다른 트랜잭션이 레코드나 갭 락을 걸고 있다면, 인서트 인텐션 락을 걸기 위해 기다려야함

인서트 인텐션 락을 사용하는 이유

InnoDB의 갭 락으로 인한 동시성 감소를 최소화 하기 위해 사용함

인서트 인텐션 락이 없을 시

예시) tb_test

__fdpk__

1

6

8

9

여기서

....5,3,4를 insert하는데, 서로 충돌되는 값이 아님에도 순차적으로 실행돼야함

__fdpk__

1 (1️⃣ 잠금- 배타적 갭 락)

6 (1️⃣잠금- 배타적 갭 락)

8

9

__fdpk__

1 (1️⃣ 잠금- 배타적 갭 락)

←——— 2️⃣ 새로운 프라이머리 키 값(3) INSERT

6 (1️⃣잠금- 배타적 갭 락)

8

9

__fdpk__

1 (1️⃣ 잠금- 배타적 갭 락)

3 (3️⃣ 잠금 - 배타적 갭 락)

6 (1️⃣잠금- 배타적 갭 락)

8

9

__fdpk__

1 (1️⃣ 잠금- 배타적 갭 락)

3 (3️⃣ 잠금 - 배타적 갭 락)

←——— 4️⃣ 새로운 프라이머리 키 값(4) INSERT

6 (1️⃣잠금- 배타적 갭 락)

8

9

__fdpk__

1 (1️⃣ 잠금- 배타적 갭 락)

3 (3️⃣ 잠금 - 배타적 갭 락)

4 (5️⃣ 잠금 - 배타적 갭 락)

6 (1️⃣잠금- 배타적 갭 락)

8

9

.... and so on...

3개 트랜잭션은 각 작업이 끝나기 까지 기다려야하므로 직렬화되어 동시에 실행되지 못하고 순차적으로 실행

인서트 인텐션 락을 사용하면....!

3개의 insert 트랜잭션이 1부터 6 사이의 간격에 대한 인서트 인텐션 락을 동시에 획득하게 됨

__fdpk__

1 (1️⃣ 잠금- 인서트 인텐션 락)

6 (1️⃣ 잠금- 인서트 인텐션 락)

8

9

__fdpk__

1 (1️⃣ 잠금- 배타적 갭 락)

←——— 2️⃣ 새로운 프라이머리 키 값(3,4,5) INSERT

6 (1️⃣잠금- 배타적 갭 락)

8

9

__fdpk__

1 (1️⃣ 잠금- 인서트 인텐션 락) - 해제

3

4

5

6 (1️⃣ 잠금- 인서트 인텐션 락) - 해제

8

9

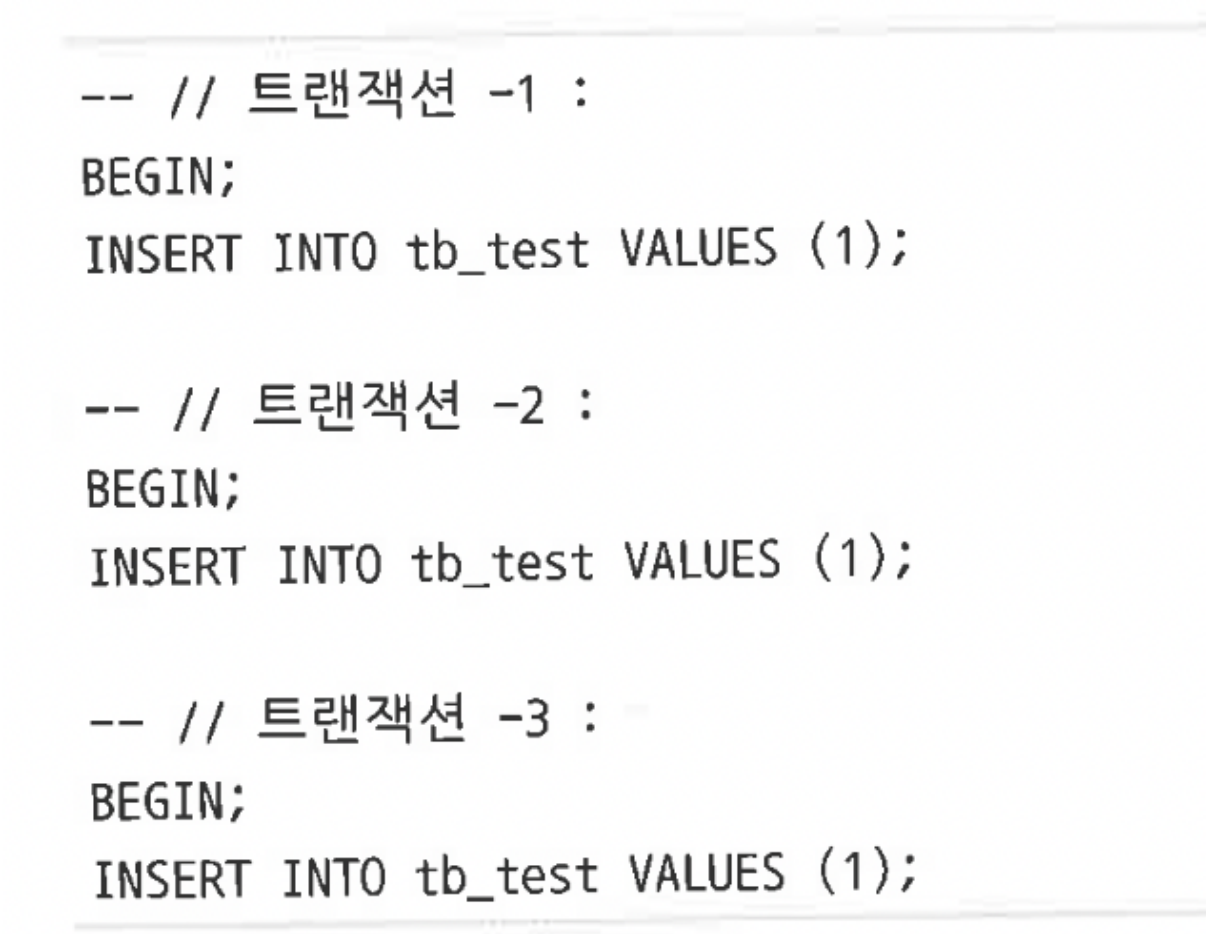
중복된 값을 인서트 하는 경우

중복이 허용되지 않는 칼럼에 중복된 값이 이미 존재한다면, InnoDB는 반드시 기존의 중복된 레코드에 **공유 레코드 락**을 걸어야 함

→ 중복 키 오류를 발생시킨 트랜잭션이 종료될때까지, 해당 중복 레코드가 다른 트랜잭션에 의해 변경되거나 삭제되면 안 되기 때문

→ 중복 키 오류시 해당 레코드에 대해 공유 잠금을 먼저 획득해야 하는 이유

(BUT DEADLOCK 원인이 될 수도...)



중복된 레코드를 삽입하려는 시도 → 2번과 3번은 공유 잠금을 획득하기 위해 대기함

- 여기서 1번 트랜잭션이 롤백이 된다면?!??!?!

2,3번 트랜잭션은 1이 없다는것을 알고 프라이머리 키 값이 1인 레코드에 공유 레코드 잠금을 걸고 동시에 배타적 잠금을 요청함

이미 2번 3번 모두 읽기 잠금을 가지고 있어서 데드락 발생... (배타적 잠금을 허용해줄 수 없다)

INSERT INTO —- ON DUPLICATE KEY UPDATE

중복된 값이 있는지 판단하기 위해 공유 잠금을 걸어야함

레코드 존재시 → 배타적 잠금 걸고 UDPATE 수행

레코드 존재X → 인서트 인텐션 락 걸고 INSERT 실행, 인서트 된 레코드에 대해서는 배타적 잠금 획득

REPLACE

중복된 레코드 존재시 배타적 잠금을 걸고 레코드 삭제

INSERT INTO ..... SELECT

읽어 오는 테이블에 공유 넥스트 키락을 설정함 (공유 잠금), 인서트 테이블엔 배타적레코드락..

→ 읽어오는 동안 원본 레코드가 변경되지 않도록 보장해 주기 위해서

→ 팬텀 레코드를 막기 위해!!!

→ 마스터 슬레이브 건수가 달라지는것을 막기 위해

REPLACE INTO ..... SELECT

위와 마찬가지로 읽어오는 테이블에는 공유잠금이, 인서트 치는 테이블에는 배타적 레코드 잠금..

3. UPDATE 쿼리의 잠금

UPDATE .. WHERE

WHERE 절에 맞는 레코드를 찾기 위해 스캔한 모든 레코드에 배타적 넥스트 키 락을 걸게 됨 (레코드의 간격까지 잠그는 이유는 팬텀 레코드 발생을 막기 위해)

- 넥스트 키 락 (레코드와 레코드 간의 갭을 동시에 잠그는 락)

- 업데이트 되는 칼럼이 포함된 테이블에는 넥스트 키 락이 걸림
- 단순 참조용으로 사용되는 테이블에는 공유 넥스트 키 락만 걸림
  - 팬텀 레코드 발생 방지하려구,,

4. DELETE 쿼리의 잠금

update와 동일

5. DDL 문장의 잠금

```
CREATE TABLE tb_new ··· SELECT ··· FROM tb_old ···
```

- 읽어오는 테이블에는 읽기 잠금이 걸림
- 새로 생기는 테이블에 INSERT되는 레코드는 배타적 레코드 락이 걸림
- 쿼리 완료됨과 동시에 자동 커밋이므로, 읽어오는 테이블의 잠금을 회피하는 방법은.....
  - 이렇게 세 쿼리로 나눠서 실행하는것...

```
CREATE TABLE tb_new···
SELECT ··· FROM tb_old INTO OUTFILE ···
LOAD DATA INFILE ··· INTO tb_new ···
```

문장 기반 복제(SBR)		레코드 기반 복제(RBR, MIXED)		
REPEATABLE-READ	SERIALIZABLE	READ-COMMITTED	REPEATABLE-READ	SERIALIZABLE
7	12	2	7	12

우왕