

Lock 심화

Database Lock



목적: 락의 주요 목적은 동시성 제어입니다. 여러 사용자나 프로세스가 동시에 데이터베이스의 동일한 자원 (예: 행, 테이블)에 액세스하려고 할 때, 데이터의 일관성과 무결성을 보장하기 위해 락을 사용합니다.

1. 단어 그대로 "잠금"
2. 하나의 데이터를 동시에 여러명이 조작할 수 없도록
 - a. 락은 특정 자원에 대한 액세스를 제한합니다. 예를 들어, 한 트랜잭션이 특정 행에 대한 락을 보유하고 있으면, 다른 트랜잭션은 해당 행을 수정할 수 없습니다.
3. 동시성(concurrency)을 보장함
4. MySQL 엔진락 vs InnoDB 락
5. Database Lock 종류

MySQL 엔진락 vs InnoDB 락

• MySQL 엔진 락 (MySQL Server-Level Locking):

1. **글로벌 락 (Global Lock):** 전체 데이터베이스에 대한 락으로, 주로 백업 또는 특정 관리 작업을 수행할 때 사용됩니다. `FLUSH TABLES WITH READ LOCK` 명령을 사용하여 글로벌 락을 획득할 수 있습니다.
2. **테이블 락 (Table Lock):** 특정 테이블 전체에 대한 락입니다. MyISAM과 같은 일부 스토리지 엔진에서는 행 수준 락을 지원하지 않기 때문에 테이블 락을 주로 사용합니다. `LOCK TABLES` 및 `UNLOCK TABLES` 명령을 사용하여 테이블 락을 관리할 수 있습니다.

• InnoDB 락 (InnoDB Locking):

1. **행 수준 락 (Row-Level Lock):** InnoDB의 주요 특징 중 하나는 행 수준의 락을 지원한다는 것입니다. 이는 특정 행에 대한 동시 액세스를 세밀하게 제어할 수 있게 해주며, 동시성을 크게 향상시킵니다.
2. **인텐션 락 (Intention Locks):** InnoDB는 행 락과 테이블 락 사이의 호환성을 관리하기 위해 인텐션 락을 사용합니다. 인텐션 락은 세션이 특정 유형의 락(예: 공유 락 또는 배타적 락)을 획득하려는 의도를 나타냅니다.
3. **공유 및 배타적 락 (Shared and Exclusive Locks):** InnoDB에서는 행에 대한 읽기 또는 쓰기 액세스를 제어하기 위해 공유(S) 락과 배타적(X) 락을 사용합니다.

4. **레코드 락 (Record Locks)**: 특정 행에 대한 락입니다.
5. **갭 락 (Gap Locks)**: 행 간의 간격에 대한 락으로, Phantom Read 문제를 방지하기 위해 사용됩니다.
6. **넥스트-키 락 (Next-Key Locks)**: 레코드 락과 갭 락의 조합으로, 특정 행과 그 앞의 갭에 대한 락을 의미합니다.

요약하면, MySQL 서버 레벨에서는 주로 글로벌 락과 테이블 락을 제공하며, InnoDB 스토리지 엔진에서는 행 수준의 락 및 관련 락 메커니즘을 제공합니다. InnoDB의 락 메커니즘은 동시성을 최대화하면서 데이터의 일관성과 무결성을 보장하기 위해 설계되었습니다.

Global Lock

1. 스토리지 엔진 단에서 락을 잡는 것

- a. ROW 에 lock을 획득한다고 보면 됨
- b. 정확히는 index에 lock을 잡음

```
mysql > FLUSH TABLES WITH READ LOCK;
```

- 위의 명령어를 통해 글로벌 락을 걸음

```
UNLOCK TABLES;
-- unlock을 하게 되면 read가 아닌 operation을 사용할 수 있게 됨
```

2. 말그대로 글로벌. 전역 LOCK

3. 특징

- a. READ만 가능하고 수정이 불가능
 - i. 수정하게 되면 대기에 걸림
- b. 요즘은 TRANSACTION 적용돼서 잘 안씀

4. 그렇다면 쓰는 경우가 존재하지 않나?

- a. MySQL을 덤프할 때, 어떠한 오퍼레이션이 일어나지 않도록 Global Lock을 걸 수 있음

Table Lock

1. 테이블을 잠그는 것

- a. 유효 범위 : 테이블

2. 특징

- a. **READ LOCK** - 내가 READ 할거니까 아무도 수정하지 말아라

```
LOCK TABLES {table name} READ;
UNLOCK TABLES;
```

- b. **WRITE LOCK** - 내가 WRITE 할거니까 아무도 READ하지 말아라

```
LOCK TABLES {table name} WRITE;  
UNLOCK TABLES;
```

- 동시성 보장을 위해 UPDATE 불가

c. 동시성을 보장한다고 생각하면 됨

i. 수정하려고 하는데 누군가 읽어버린다면?

- 만약 락이 없다면, 읽는 트랜잭션은 수정 중인 "미완료"의 데이터를 읽게 될 수 있습니다. 이는 일관성 없는 데이터를 반환할 수 있습니다.
- 테이블 락 또는 더 세분화된 락(예: 행 락)을 사용하면, 수정 중인 데이터에 대한 읽기 작업을 잠시 대기 상태로 만들어 이러한 문제를 방지할 수 있습니다.

ii. 읽으려고 하는데 누군가 수정해버린다면?

- **READ LOCK** : 읽기 작업이 먼저 시작되었을 경우, 수정 작업은 해당 데이터에 대한 락이 해제될 때까지 대기해야 합니다. 이렇게 하면 읽기 작업이 일관된 데이터를 얻을 수 있습니다.
- **WRITE LOCK** : 반대로, 수정 작업이 먼저 시작되었을 경우, 읽기 작업은 수정이 완료될 때까지 대기하게 됩니다.

3. MyISAM이나 MEMORY에서 자동으로 사용되는 LOCK

- a. InnoDB의 경우에는 트랜잭션을 활용하기 때문에 불필요
- b. task가 나뉘어서 작동하게 됨

Named Lock



MySQL에서 제공하는 애플리케이션 수준의 동기화 메커니즘입니다. 네임드 락 자체는 데이터베이스의 특정 테이블이나 행에 직접적인 락을 걸지 않습니다. 대신, 애플리케이션 로직에서 동시성을 제어하기 위해 사용됩니다.

- 명령어

```
GET_LOCK(), RELEASE_LOCK(), IS_USED_LOCK(), 및 IS_FREE_LOCK()
```

1. 네임드 락은 MySQL 세션 간에 동기화 메커니즘으로 작동. 한 세션에서 획득한 락은 다른 세션에서는 획득할 수 없음. 그러나 동일한 세션 내에서는 동일한 락을 여러 번 획득 가능.
2. 그냥 내가 임의로 LOCK을 잡는것 (명시적 락)

```
SELECT GET_LOCK('wanted', 10);
```

- 락의 이름 (**'wanted'** 와 같은 문자열)
- 대기 시간 (초 단위)
 1. 10초 내에 락을 획득할 수 있다면 함수는 **1**을 반환
 2. 0은 사용 중

여기서 **10**은 대기 시간을 나타냅니다.

```
SELECT RELEASE_LOCK('wanted'); -- 획득 가능하게 release

SELECT IS_FREE_LOCK('wanted');
```

위와 같이 특정 문자열에 해당하는 어플리케이션 로직에 락을 걸어 해당 로직에 대한 동시성 제어를 수행할 수 있게 됩니다.

3. 다양한 로직 처리에 유리할 수 있음

- 네임드락의 범위는 글로벌 → 샤딩(여러 데이터베이스), 테이블 여러 개
- Named Lock은 서버 내 공유 자원에 대한 접근을 동기화하는데 유용하지만, 분산 시스템에서 적합하지 않을 수 있으므로 다른 분산 락 메커니즘을 고려해야 합니다.

4. 네임드 락은 MySQL 서버 전체의 모든 세션 간에 동기화를 제공하는 글로벌 락.

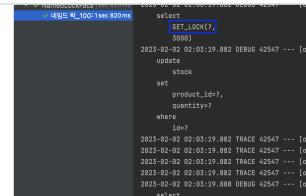
특정 테이블이나 행에 연결되지 않으며, 주로 애플리케이션 로직의 동기화를 위해 사용

- 네임드 락은 애플리케이션 수준의 동기화를 위한 것이며, 데이터베이스의 특정 테이블이나 행에 직접적인 락을 걸지 않습니다. 따라서, 네임드 락을 획득한 상태에서도 다른 사용자는 **UPDATE** 쿼리를 실행할 수 있습니다.
- 특정 테이블이나 행에 대한 동시 액세스를 제한하려면, InnoDB의 행 수준 락(row-level locking) 또는 테이블 락을 사용해야 합니다. 예를 들어, **SELECT ... FOR UPDATE** 쿼리를 사용하면 특정 행에 대한 락을 획득할 수 있습니다, 이로 인해 다른 사용자는 해당 행을 수정할 수 없게 됩니다.

5. 애플리케이션 로직에 따라 네임드 락을 사용하여 특정 작업의 동시 실행을 제어하려는 경우, 다른 세션에서 해당 작업을 수행하기 전에 동일한 네임드 락을 획득하려고 시도할 것이므로, 락을 획득할 수 없게 됩니다.

동시성 문제 해결하기2 - 네임드 락(Named Lock) & Redis를 활용한 분산 락(Distributed lock)
네임드 락(Named Lock)과 Redis를 활용한 분산 락(Distributed Lock) 구현을 통한 동시성 문제 해결하기

<https://velog.io/@hyojhand/named-lock-distributed-lock-with-redis>

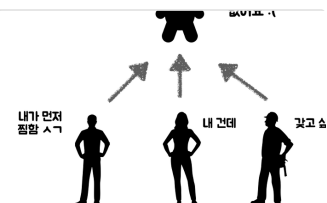


- 우아한형제들 기술블로그에서 광고시스템은 Named Lock을 활용해 분산락을 구현
- 분산락이란 : 멀티 스레드 환경에서 공유 자원에 접근할 때, **데이터의 정합성을 지키기 위해** 사용하는 기술

동시성 문제를 해결해보자! 분산락 구현하기 - 1편 (네임드락 - Named Lock 활용하기)

🌱 들어가기 전 DB 공부하다가 '네임드락'에 대해서 알게 되었는데, 네임드락을 사용하면 분산락을 구현할 수 있다는 글을 보고 한 번 테스트해보고 싶어서 글을 작성해보고자 한다. 전체 소스코드는 여기에서 확인 가능하다. (원가 테스트용 레포 만들기 애매해서 그냥 안 쓰는 레포에다가 하려

<https://cl8d.tistory.com/112>



6. Redis를 이용해서 MySQL 분산 락 구현이 가능하다. 분산 락을 구현하기 위해 Lock에 대한 정보를 Redis에 보관하고, 분산 환경의 서버는 공통된 Redis를 통해 임계 영역(critical section)에 접근할 수 있는지 확인한다.

7. Named Lock은 MySQL 인스턴스 내부의 메모리를 기반으로 동작하기 때문에 분산 시스템에 적합하지 않습니다. 분산 시스템의 장애를 관리하기 위해서는 분산 락 관리자(Zookeeper 등등)를 사용할 필요가 있습니다.

Metadata Lock

- 테이블 정보를 수정할 때 자동으로 획득
 - 테이블 이름
 - 컬럼 이름이나 컬럼 정보 등을 수정할 때 자동으로

Record Lock

- 스토리지 엔진단에서 락을 잡는 것
 1. ROW 에 lock을 획득한다고 보면 됨
 2. 정확히는 index에 lock을 잡음

Auto Increment Lock


- 여러 클라이언트 동시에 데이터를 추가하려고 할 때를 대비함
 1. Auto increment 설정 해놨는데 동시에 데이터를 추가하면
 2. 같은 pk를 가진 row들이 여러개가 될 수도 있음

참고

원티드 백엔드

동시성 문제 해결하기2 - 네임드 락(Named Lock) & Redis를 활용한 분산 락(Distributed lock)

네임드 락(Named Lock)과 Redis를 활용한 분산 락(Distributed Lock) 구현을 통한 동시성 문제 해결하기

 <https://velog.io/@hyojhand/named-lock-distributed-lock-with-redis>

```
mysql> select
      GET_LOCK('
      lock')
2023-02-02 02:03:19.882 DEBUG 42547 --- [sql-1-1-thr
update
  stock
set
  product_id?,
  quantity?
where
  id=?
2023-02-02 02:03:19.882 TRACE 42547 --- [sql-1-1-thr
2023-02-02 02:03:19.882 TRACE 42547 --- [sql-1-1-thr
2023-02-02 02:03:19.882 TRACE 42547 --- [sql-1-1-thr
2023-02-02 02:03:19.882 TRACE 42547 --- [sql-1-1-thr
2023-02-02 02:03:19.888 DEBUG 42547 --- [sql-1-1-thr
select
```