Deadlock 현상 해결

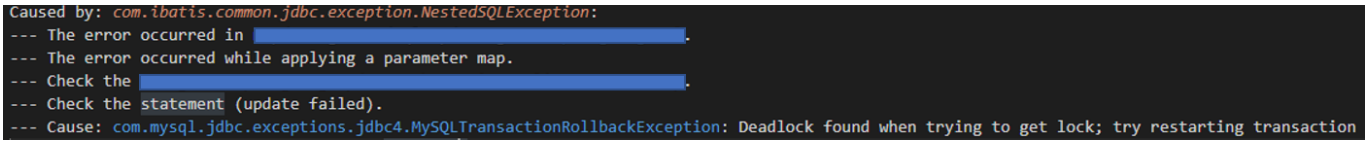
고객사로부터 특정 요청이 간헐적으로 실패가 난다는 이슈가 접수되었다. 로그를 확인해보니 Deadlock exception 발생하고 있었고 이슈를 해결하며 찾아보며 알게된 것들을 정리한다.

환경

DB	MariaDB
Engine	InnoDB
Isolation Level	Repeatable Read

현상

톰캣 catalina 로그를 확인해보면 아래와 같은 exception 나고 있었다.



exception내용 : *Deadlock found when trying to get lock; try restarting transaction*

일단 exception메시지를 보면 *lock을 가져오려 하는데 Deadlock이 발생하여 트랜잭션을 재시작*하겠다 정도로 해석할 수 있다.

당장 떠오른 해결방법은 두가지 였고, 장단점은 명확했다.

-	해결법	장점	단점
1	문제가 되는 요청을 호출 하지 않도록 변경하자 (이슈를 접수한 고객사는 다행히 해당 기능을 사용하지 않는 고객이었다.)	바로 해결 가능	향후에 해당 기능 사용할 때 혹은 다른 고객사에서도 동일한 이슈가 발생할 수 있다.
2	exception 나는 원인을 파악 후 해결하자	근본적인 해결책	원인을 파악하는데 어느 정도의 시간이 걸릴 것이고, 고객은 그 때까지 계속 exception 나는 상태로 사용해야 한다.

물론 누가봐도 2번을 선택해서 문제를 해결 해야했지만 당장의 issue 해결도 중요했기에 일단 임시방편으로 고객사에 1번 조치를 하고 근본적인 원인을 분석 후 이슈가 해결된 버전을 전달하기로 했다.

그럼 지금부터 **어떤 상황에서 어떤 쿼리가 lock을 가져오려 했고, 왜 가져오지 못해 Deadlock이 발생**했는지 알아보자.

분석

Deadlock exception 난 곳을 확인해보니 A 프로시저 수행 중 이 프로시저에서 select 하는 테이블을(정확히 말하면 update 문 안에 select문이 있다) 다른 프로시저 B에서 그 테이블을 update 하는 쿼리 수행 중 exception 발생하고 있었다. 발생 패턴

을 찾은 후 exception발생 재현을 성공했고, 본격적으로 분석에 들어갔다.

처음 의문이 든 점은 update문이 수행될 때 어떤 lock을 거는지 였다. update가 대상 테이블에 어떤 lock을 걸길래, 또 그 테이블에 어떤 lock이 걸려있길래 deadlock이 걸리는걸까?

Update문이 거는 lock

처음엔 구글링을 통해 알아봤지만, 좀 더 정확한 정보를 위해 mysql 문서내용을 가져왔다.

> *UPDATE 또는 DELETE는 일반적으로 SQL 문 처리에서 스캔되는 모든 인덱스 레코드에 레코드 잠금을 설정한다 ... ... , WHERE절에 적절한 인덱스를 사용하지 않으면 MySQL은 쿼리를 수행하기 위해 전체 table에 lock을 건다*

보통은 `update`되는 대상 row에 `record lock`이 걸리지만, where절에 인덱스를 적절히 태우지 않거나 인덱스가 없는 경우는 **테이블 전체에 lock**이 걸린다!

확인 결과 B프로시저 update문은 index를 타고 있지 않았고 쿼리 수행할 때 table 전체에 lock을 걸고 있었다. 그리고 테이블에 index를 생성하여 update대상의 row만 record lock이 걸리도록 변경했다. 해결했다는 기쁨을 가지고 테스트 했지만 예상은 보기좋게 빗나갔다. 발생 빈도는 줄었지만 동일한 exception 발생했다.

무엇이 문제였을까

Deadlock에 관해 대충만 알고있어서 이런 생각을 하게 된것 같다. Where절에 index를 태움으로서 lock의 범위가 table에서 record로 줄어들긴 했지만 Deadlock이 생길만한 요소를 제거했다고 보기 어려웠다. 다시 분석에 들어갔다.

Deadlock 관련 로그 분석

검색을 통해 알게되었는데, `show engine innodb status`를 통해 가장 최근에 발생한 deadlock에 대한 정보를 알 수 있었다.

로그는

```
------------------------
LATEST DETECTED DEADLOCK
------------------------
2020-01-10 18:16:32 7fee7a600700
*** (1) TRANSACTION:
TRANSACTION 92465172, ACTIVE 0 sec fetching rows
mysql tables in use 1, locked 1
LOCK WAIT 3 lock struct(s), heap size 360, 6 row lock(s), undo log entries
1
MySQL thread id 949396, OS thread handle 0x7fef8cfb6700, query id 28309277
127.0.0.1 root updating
UPDATE    {tbl_a}   SET    {date_col} = now()   WHERE    {client_key} =
'{value}'
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 981 page no 3 n bits 128 index `PRIMARY` of table
`{tbl_a}` trx id 92465172 lock_mode X waiting
*** (2) TRANSACTION:
TRANSACTION 92465171, ACTIVE 0 sec starting index read
mysql tables in use 36, locked 36
5 lock struct(s), heap size 1184, 3 row lock(s)
MySQL thread id 949674, OS thread handle 0x7fee7a600700, query id 28309288
```

```
localhost root Sending data
update {tbl_b} b,
        (
            SELECT
                {columns...}
                (SELECT {id} FROM {tbl_a} a WHERE {client_key} =
'{value}') as ID,
                {columns...}
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 981 page no 3 n bits 128 index `PRIMARY` of table
{tbl_a} trx id 92465171 lock mode S locks rec but not gap
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 981 page no 3 n bits 128 index `PRIMARY` of table
{tbl_a} trx id 92465171 lock mode S waiting
*** WE ROLL BACK TRANSACTION (1)
```

이렇게 생겼는데, 데드락이 발생한 원인에 대한 정보를 가지고 있다. 분석을 해보면

1. 92465172번 트랜잭션이 {tbl_a} 테이블을 update를 하기 위해 space id 981 page no 3 부분에 lock_mode X를 거려고 기다리고 있고
2. 92465171번 트랜잭션이(tranjaction 번호가 더 빠른것으로 보아 먼저 실행된 것을 알 수 있다.) space id 981 page no 3 부분에 mode S locks을 걸고 있고, space id 981 page no 3 부분에 또 lock mode S를 걸기 위해 기다리고 있다.(shared lock은 기본적으로 lock을 공유하지만 shared lock이 걸린 row에 쓰기를 허용하지 않는다.)
3. 92465171번 트랜잭션이 먼저 shared lock을 걸고있고, 92465172번 트랜잭션이 update를 위해 exclusive lock을 걸기 위해 기다리고 있는 상태에서 92465171번 트랜잭션이 다시 shared lock을 거려고 하는 상태에서
4. 로그 마지막 부분을 보면 교착상태를 해소하기 위해 92465172번 트랜잭션을 rollback 한다고 나와있다.
5. 추가적으로, 92465171번 트랜잭션이 mode S locks을 건 row에 왜 한번 더 mode S locks을 걸려고 하는지 분석이 필요해 보인다.

> ## Shared and Exclusive Locks
>
> The two standard row-level locks are share locks(S) and exclusive locks(X).
>
> A shared lock is obtained to read a row, and allows other transactions to read the locked row, but not to write to the locked row. Other transactions may also acquire their own shared locks.
>
> An exclusive lock is obtained to write to a row, and stops other transactions from locking the same row. It's specific behavior depends on the isolation level; the default (REPEATABLE READ), allow other transactions to read from the exclusively locked row.(https://mariadb.com/kb/en/innodb-lock-modes/)

해결에 대한 희망이 조금씩 보이기 시작했다. 일단 92465171 트랜잭션 안의 select문을 제거하고 외부에서 변수를 전달해주는 방식으로 변경하고 다시 테스트를 진행했다.

해결

해결이 될것이라 강하게 믿었지만 여전히 exception은 발생하고 있었다. 조금 멘붕에 빠져서 팀원들에게 조언을 구하던 중 문제가 됐던 트랜잭션 이후에 수행되는 트리거가 있다는 것을 알게 되었고, 역시나 그 트리거에서도 문제의 테이블을 접근하고 있었다.

결국 관련된 모든 프로시저에서 문제의 테이블 select 하는 부분을 제거하였고 다시 테스트를 돌려본 결과 이젠 진짜 문제가 해결되었다는 것을 확인하였다.

정리를 하자면

- update문 실행 시 보통 where절에 걸리는 row만 lock이 걸리지만 쿼리가 index를 타지 못하는 경우 테이블 전체에 lock이 걸린다.
- shared lock은 기본적으로 lock을 공유하지만 shared lock이 걸린 row에 쓰기를 허용하지 않는다.
- 프로시저에 update문이 있고 그 안에 select문은 사용하지 않는 것이 좋다.
- DB Engine의 Isolation Level에 따라 동작방식이 달라지는데 읽어보다 그냥 패스했다... 기회가 되면 정리해봐야겠다!!
- `show engine innodb status` 쿼리를 통해 deadlock 발생 정보를 확인할 수 있다. (가장 최근 1건에 대해서만)
- 트랜잭션과 트리거(function도 추가)에 대한 의문이... 디버깅을 할 수도 없고 의존성(어디서 어떤 테이블 참조하고, 어떤 트리거가 발생하는지)을 찾기도 힘들다. 물어보니 속도가 빨라서 사용한다고 하시던데 과연 이게 최선의 선택일까.