

5. 트랜잭션과 잠금

➤ Books

Real MySQL 8.0

1. 트랜잭션

1. MySQL에서의 트랜잭션

2. MySQL 엔진의 잠금

1. 글로벌 락

2. 테이블 락

3. 네임드 락

4. 메타데이터 락

3. InnoDB 스토리지 엔진 잠금

1. 인덱스와 잠금

4. MySQL의 격리 수준

1. READ UNCOMMITTED

2. READ COMMITTED

3. REPEATABLE READ

4. SERIALIZABLE

트랜잭션은 작업의 완전성을 보장해 주는 것이다.

잠금(Lock)과 트랜잭션은 비슷한 개념 같지만 잠금은 동시성을 제어하기 위한 기능이고, 트랜잭션은 데이터의 정합성을 보장하기 위한 기능이다.

1. 트랜잭션

1. MySQL에서의 트랜잭션

트랜잭션은 하나의 논리적인 작업 셋에 쿼리 개수에 관계 없이 100% 적용되거나 아무것도 적용되지 않아야 함을 보장해주는 것이다.

InnoDB 테이블과 MyISAM 테이블의 차이를 아래 예제로 확인 할 수 있다.

```
CREATE TABLE tb_myisam (  
    id INT NOT NULL,  
    PRIMARY KEY (id)  
) ENGINE=MYISAM;  
  
CREATE TABLE tb_innodb (  
    id INT NOT NULL,  
    PRIMARY KEY (id)  
) ENGINE=innodb;  
  
INSERT INTO tb_myisam (id) VALUES (3);  
INSERT INTO tb_innodb (id) VALUES (3);
```

2개의 테이블을 생성한 후 레코드를 각각 1건씩 저장했다.

그리고 AUTO-COMMIT 모드에서 아래 쿼리를 실행해보자.

```
INSERT INTO tb_myisam (id) VALUES (1), (2), (3);
INSERT INTO tb_innodb (id) VALUES (1), (2), (3);
```

두 테이블 모두 PK 중복 오류로 쿼리가 실패했다.

하지만 레코드를 조회해보면 InnoDB 테이블에는 '3'만 남아있지만 MyISAM 테이블에는 '1', '2', '3'이 모두 남아있다.

롤백이 발생하지 않았다고 보다는 이미 삽입된 '1'과 '2'는 그대로 두고 '3'에 대해서만 저장하지 않은 상태로 종료된 것이다.

이러한 현상을 부분 업데이트(Partial Update)라고 표현하며, 테이블 데이터의 정합성을 맞추는 데 문제가 된다.

2. MySQL 엔진의 잠금

MySQL에서 사용되는 잠금은 스토리지 엔진 레벨과 MySQL 엔진 레벨로 나눌 수 있다.

MySQL 엔진 레벨의 잠금은 모든 스토리지 엔진에 영향을 미치지만, 스토리지 엔진 레벨의 잠금은 스토리지 엔진 간 상호 영향을 미치지 않는다.

MySQL 엔진에서는 글로벌 락, 테이블 락, 메타데이터 락, 네임드 락을 제공한다.

1. 글로벌 락

MySQL에서 제공하는 잠금 중 가장 범위가 크다.

한 세션에서 글로벌 락을 획득하면 다른 세션에서 SELECT를 제외한 대부분의 DDL 문장이나 DML 문장을 실행하는 경우 글로벌 락이 해제될 때까지 해당 문장이 대기 상태로 남는다.

글로벌 락은 작업 대상 테이블이나 데이터베이스에 관계 없이 MySQL 서버 전체에 영향을 미친다.

InnoDB 스토리지 엔진은 트랜잭션을 지원하기 때문에 일관된 데이터 상태를 위해 모든 데이터 변경 작업을 멈출 필요가 없다.

그래서 InnoDB가 기본 스토리지 엔진으로 채택되면서 조금 더 가벼운 글로벌 락의 필요성이 생겼고, Xtrabackup이나 Enterprise Backup과 같은 백업 툴들의 안정적인 실행을 위해 백업 락이 도입됐다.

2. 테이블 락

테이블 락은 개별 테이블 단위로 설정되는 잠금이다.

명시적으로는

`LOCK TABLES table_name [READ | WRITE]` 명령으로 특정 테이블 락을 획득할 수 있고, 묵시적으로는 MyISAM 이나 MEMORY 테이블에 데이터를 변경하는 쿼리를 실행하면 발생한다.

InnoDB 테이블도 묵시적인 테이블 락이 설정되지만 대부분의 데이터 변경(DML) 쿼리에서는 무시되고 스키마를 변경하는 쿼리(DDL)의 경우에만 영향을 미친다.

3. 네임드 락

네임드 락은 `GET_LOCK()` 함수를 이용해 임의의 문자열에 대해 잠금을 설정할 수 있다.


네임드 락은 많은 레코드에 대해 복잡한 요건으로 레코드를 변경하는 트랜잭션에 유용하게 사용할 수 있다.

MySQL 8.0 버전부터는 네임드 락을 중첩해서 사용할 수 있게 되었다.

네임드 락 활용 사례

MySQL을 이용한 분산락으로 여러 서버에 걸친 동시성 관리 | 우아한형제들 기술블로그

우아한스터디

 <https://techblog.woowahan.com/2631/>



우아한형제들의 기술조직 이야기를 전합니다.

4. 메타데이터 락

메타데이터 락은 데이터베이스 객체의 이름이나 구조를 변경하는 경우에 획득하는 잠금이다.

명시적으로 획득하거나 해제할 수 있는 것은 아니고 테이블의 이름을 변경하는 경우 자동으로 획득한다.

3. InnoDB 스토리지 엔진 잠금

InnoDB 스토리지 엔진은 MySQL에서 제공하는 잠금과는 별개로 스토리지 엔진 내부에서 레코드 기반의 잠금 방식을 탑재하고 있다.

최근 버전에서는 InnoDB의 트랜잭션과 잠금, 그리고 잠금 대기 중인 트랜잭션의 목록을 조회할 수 있는 방법이 도입되었으며 InnoDB 스토리지 엔진의 내부 잠금(세마포어)에 대한 모니터링 방법도 추가됐다.

InnoDB는 총 8가지의 잠금을 제공한다.

- Shared / Exclusive Lock
- Intention Lock
- Record Lock
- Gap Lock
- Next-Key Lock
- Insert Intention Lock
- Auto-increment Lock
- Predicate Lock for Spatial Indexes

1. 레코드 락

레코드 자체만을 잠그는 것을 의미하며 다른 상용 DBMS의 레코드 락과 동일한 역할을 하지만 레코드 자

체가 아닌 인덱스의 레코드를 잠근다는 차이가 있다.

2. 갭 락

갭 락은 레코드 자체가 아니라 레코드와 바로 인접한 레코드 사이의 간격만을 잠그는 것을 의미한다. 레코드와 레코드 사이의 간격에 새로운 레코드가 생성되는 것을 제어하는 것이다.

3. 넥스트 키 락

레코드 락과 갭 락을 합쳐 놓은 형태의 잠금이다.

4. 자동 증가 락

AUTO_INCREMENT 칼럼이 사용된 테이블에 동시에 여러 레코드가 INSERT되는 경우, 저장되는 각 레코드는 순서대로 증가하는 일련번호 값을 가져야 한다.

InnoDB 스토리지 엔진에서는 이를 위해 내부적으로 AUTO_INCREMENT 락이라고 하는 테이블 수준의 잠금을 사요한다.

INSERT와 REPLACE 쿼리 문장과 같이 새로운 레코드를 저장하는 쿼리에서만 필요하고, UPDATE나 DELETE 등의 쿼리에서는 걸리지 않는다.

또한 명시적으로 획득하고 해제하는 방법은 없다.

1. 인덱스와 잠금

`first_name='Georgi'` 인 사원은 253명이 있고 `first_name='Georgi'`, `last_name='Klassen'` 인 사원은 1명만 있다고 가정을 해보자.

만약

`first_name` 은 인덱스가 걸려 있고 `last_name` 에는 인덱스가 걸려 있지 않을 때 `first_name='Georgi'`, `last_name='Klassen'` 인 사원의 입사 일자를 변경한다면 1건의 업데이트를 위해 253건의 레코드가 모두 잠긴다.

인덱스가 하나도 없다면 테이블을 풀 스캔하면서 UPDATE를 하기 때문에 이 과정에서 모든 레코드가 잠기게 된다.

이것이 InnoDB를 사용할 때 인덱스 설계가 중요한 이유 중 하나다.

4. MySQL의 격리 수준

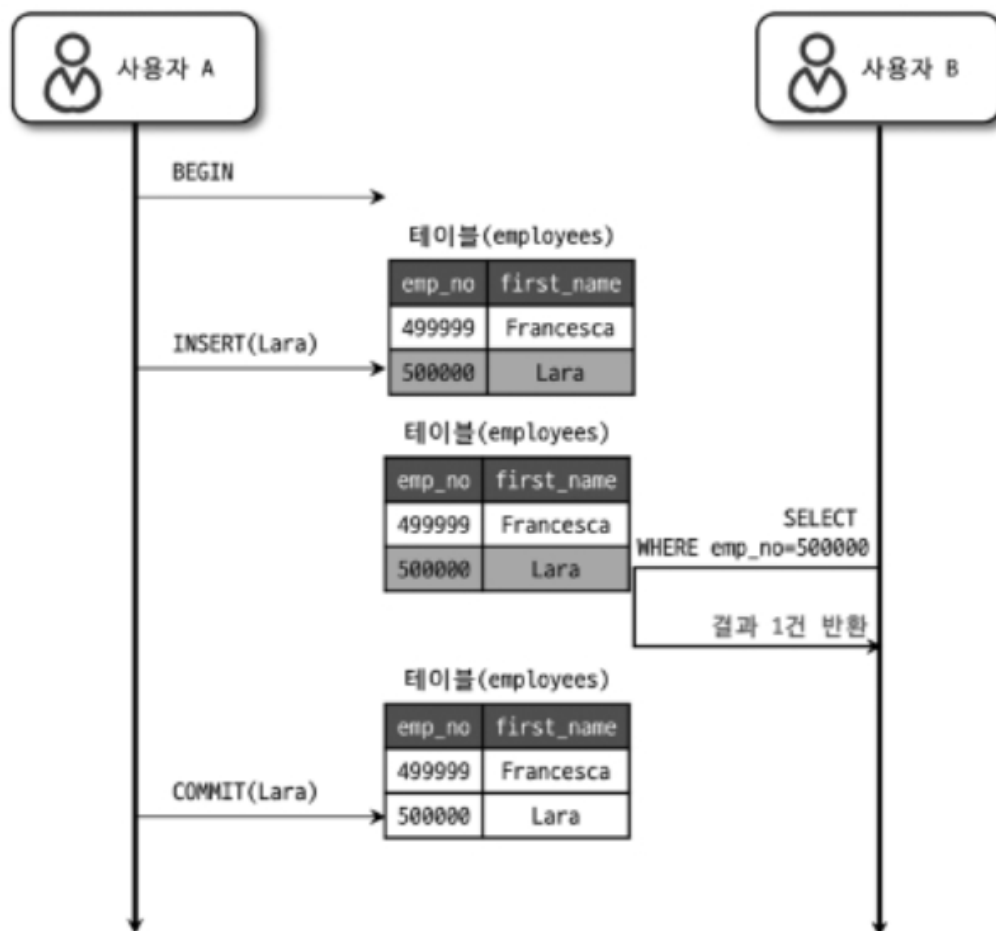
트랜잭션의 격리 수준이란 여러 트랜잭션이 동시에 처리될 때 특정 트랜잭션이 다른 트랜잭션에서 변경하거나 조회하는 데이터를 볼 수 있게 허용할지 말지를 결정하는 것이다.

격리 수준은 크게 `READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ`, `SERIALIZABLE` 4가지로 나뉜다.

4개의 격리 수준에서 순서대로 뒤로 갈수록 각 트랜잭션 간의 데이터 격리 정도가 높아지며, 동시 처리 성능도 떨어지는 것이 일반적이다.

	DIRTY READ	NON-REPEATABLE READ	PHANTOM READ
READ UNCOMMITTED	발생	발생	발생
READ COMMITTED	없음	발생	발생
REPEATABLE READ	없음	없음	발생 (InnoDB는 없음)
SERIALIZABLE	없음	없음	없음

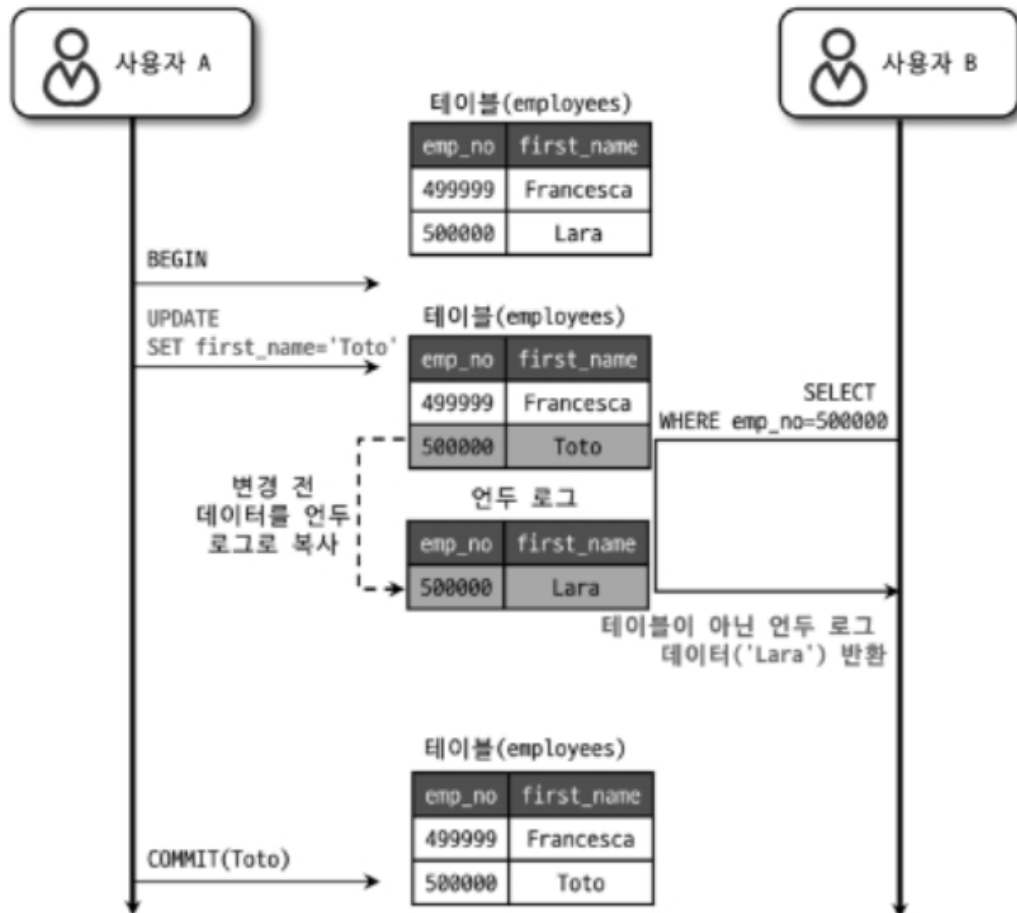
1. READ UNCOMMITTED



사용자 B가 사용자 A가 INSERT한 사원의 정보를 커밋되지 않은 상태에서도 조회가 가능한 것을 알 수 있다. 문제는 사용자 A가 처리 도중 문제가 발생해 INSERT된 내용을 롤백한다고 해도 사용자 B는 알 수 없다는 것이다.

이렇게 어떤 트랜잭션에서 처리한 작업이 완료되지 않았는데도 다른 트랜잭션에서 볼 수 있는 현상을 더티 리드(Dirty read)라고 한다.

2. READ COMMITTED

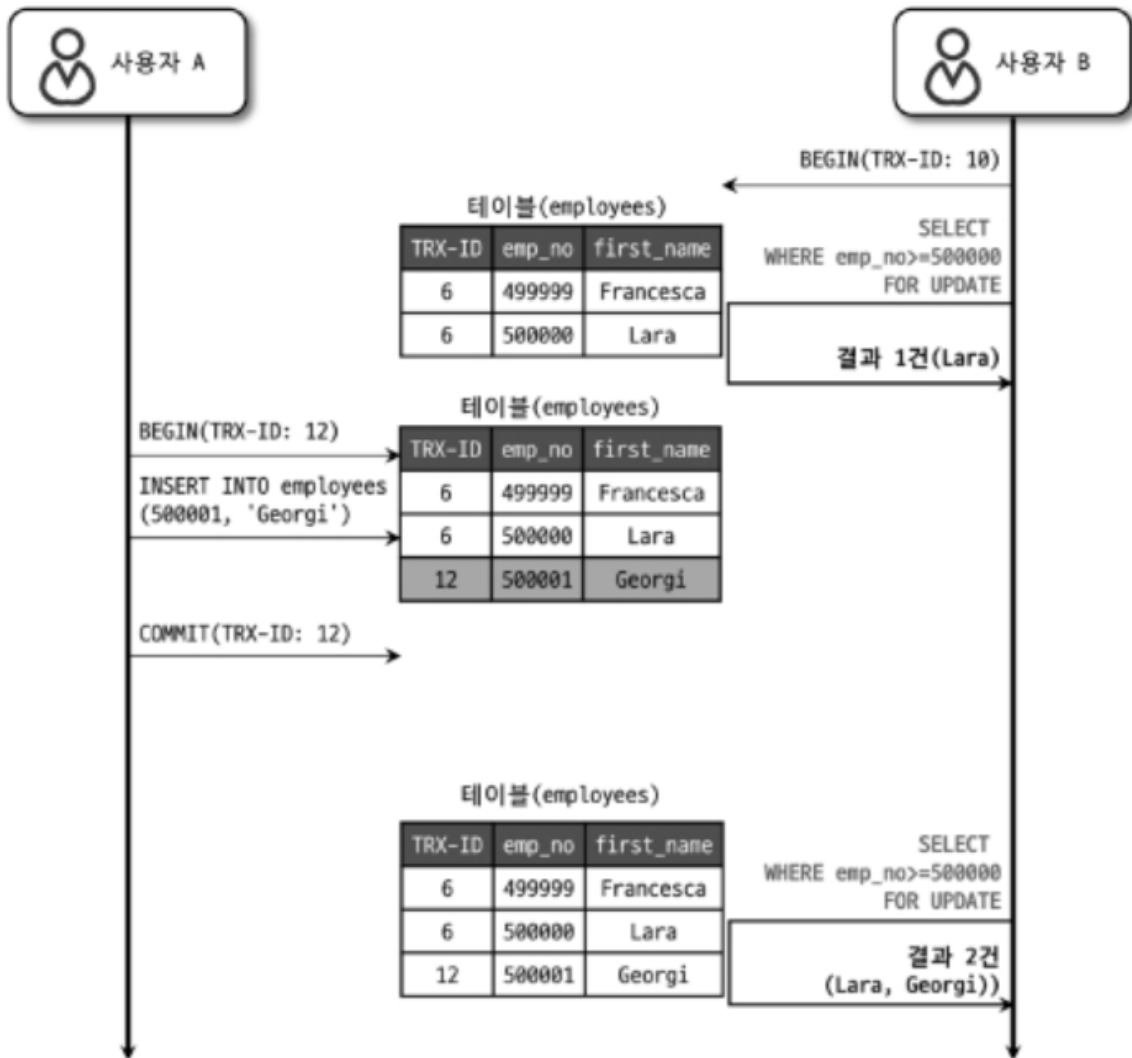


사용자 A가 COMMIT을 하기 전에 사용자 B가 emp_no=500000인 사원을 조회하면 'Lara'로 조회가 된다. 하지만 언두 영역에 백업된 레코드를 가져온 것이기 때문에 COMMIT 이후 emp_no=500000인 사원을 조회하면 'Toto'로 조회가 된다.

문제가 없어 보이지만 하나의 트랜잭션 내에서 똑같은 SELECT 쿼리를 실행했을 때 항상 같은 결과를 가져와야 한다는 REPEATABLE READ 정합성에 어긋난다. (=NON-REPEATABLE READ)

3. REPEATABLE READ

InnoDB 스토리지 엔진에서 기본으로 사용되는 격리 수준이다.



REPEATABLE READ는 MVCC를 위해 언두 영역에 백업된 이전 데이터를 이용해 동일 트랜잭션 내에서는 동일한 결과를 보여줄 수 있게 보장한다.

READ COMMITTED와의 차이는 언두 영역에 백업된 레코드의 여러 버전 가운데 몇 번째 이전 버전까지 찾아 들어가야 하느냐에 있다.

REPEATABLE READ 단계에서 실행되는 SELECT 쿼리는 자신의 트랜잭션 번호보다 작은 트랜잭션 번호에서 변경한 값만 보이게 된다.

하지만 위 그림처럼 다른 트랜잭션에서 수행한 변경 작업에 의해 레코드가 보였다 안 보였다 하는 PHANTOM READ 현상이 발생하기도 한다.

(InnoDB 스토리지 엔진에서는 갭 락과 넥스트 키 락 덕분에 발생하지 않는다.)

4. SERIALIZABLE

가장 엄격한 격리 수준이며 동시 처리 성능도 다른 격리 수준에 비해 떨어진다.

트랜잭션의 격리 수준이 SERIALIZABLE로 설정되면 읽기 작업도 공유 잠금을 획득해야 하며 동시에 다른 트랜잭션은 그러한 레코드를 변경하지 못하게 된다.



생각해 볼 내용

1. 트랜잭션 내에서 외부 환경으로 인한 예외가 발생했을 때 어떻게 해결하면 좋을까요?
2. 네임드 락은 언제 사용하면 좋을까요? 사용해본 경험이 있을까요?
3. Online DDL은 무엇일까요?
4. 왜 MySQL의 DDL은 싱글 스레드로 동작할까요?
5. 바이너리 로그란 무엇일까요?
 - 5-1. 바이너리 로그는 어떻게 존재할까요?
 1. 뮤텍스는 무엇이고 세마포어는 무엇일까요?
 - 6-1. 둘의 차이는 어디서 발생할까요?
 - 6-2. 바이너리 세마포어를 뮤텍스라고 볼 수 있을까요?
 1. 오라클의 격리수준은 왜 read committed 일까요?
 2. repeatable read는 어떤 조건을 말할까요?
 3. 왜 InnoDB에서는 Phantom Read가 발생하지 않을까요?
 4. for update는 어떤 잠금을 의미할까요?