7장 트랜잭션과 동시성 제어

요약

- `트랜잭션`: 복수의 쿼리를 연달아 사용해 일관성 있는 형태의 한 단위로 처리해야 하는 경우가 많은데 이 단위를 트랜잭션이라고 한다.

- DBMS 트랜잭션의 성질 ACID
    - `Atomicity` `원자성`:
        - 트랜잭션과 관련된 작업들이 부분적으로 실행되다가 중단되지 않는 것을 보장하는 능력
    - `Consistency` `일관성`:
        - 트랜잭션이 실행을 성공적으로 완료하면 언제나 일관성 있는 데이터베이스 상태로 유지하는 것
    - `Isolation` `고립성` `격리성`:
        - 트랜잭션을 수행 시 다른 트랜잭션의 연산 작업이 끼어들지 못하도록 보장하는 것
    - `Durability` `지속성`:
        - 성공적으로 수행된 트랜잭션은 영원히 반영되어야 함

- ANSI가 정의하는 격리 수준
    - 커밋되지 않은 읽기
    - 커밋된 읽기
    - 반복 읽기
    - 직렬화 가능

- `더티 읽기`: 어떤 트랜잭션이 커밋되기 전에 다른 트랜잭션에서 데이터를 읽는 현상
- `애매한 읽기`: 어떤 트랜잭션이 이전에 읽어 들인 데이터를 다시 읽어들일 때 2회 이후의 결과가 1회 때와 다른 현상
- `팬텀 읽기`: 어떤 트랜잭션을 읽을 때 선택할 수 있는 데이터가 나타나거나 사라지는 현상

- 잠금 타임아웃, 교착 상태, 오토 커밋

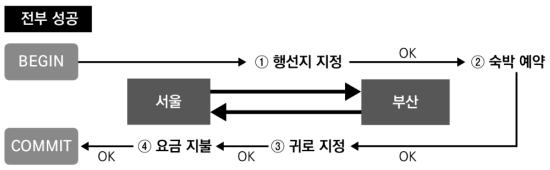
- 주의해야 할 트랜잭션 처리
    - 오토 커밋: 쿼리 단위로 커밋하는 설정 (1 쿼리 당 1 커밋)
    - 긴 트랜잭션

트랜잭션이란

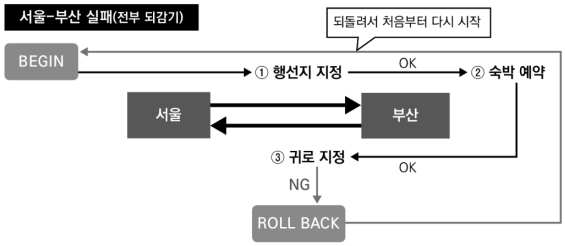
- 복수 쿼리를 한 단위로 묶은 것
- MySQL 테이블 종류
    - MyISAM: 트랜잭션을 사용할 수 없는 단순한 구조
    - InnoDB: 일반적인 DBMS와 똑같은 트랜잭션 구조 사용 가능
- 트랜잭션의 특징: ACID

트랜잭션의 특징 1 - 원자성 Atomicity

- All or Nothing
- 트랜잭션과 관련된 작업들이 부분적으로 실행되다가 중단되지 않는 것을 보장하는 능력



모든 절차가 다 성공 하면 커밋을 실행하여 영구적으로 저장한다



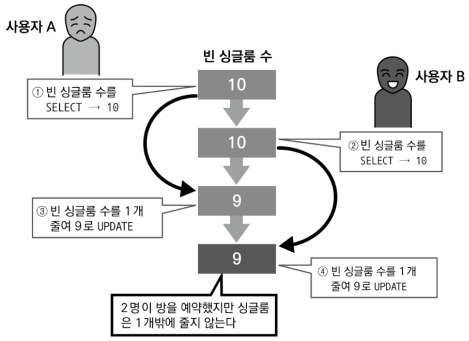
좌석이 부족하던지, 예약금이 부족한 상황 등 처리 도중 오류가 발생하는 경우 일부만 실행되지 않고 처음으로 돌아가야 한다

트랜잭션의 특징 2 - 일관성 Consistency

- 트랜잭션이 실행을 성공적으로 완료하면 언제나 일관성 있는 데이터베이스 상태로 유지하는 것
- 일관성 있는 상태란? 도메인의 유효범위, 무결성 제약조건 등의 제약조건을 위배하지 않는 정상적인 상태
  - 돈은 0보다 커야 한다.
  - 사용자의 일련번호는 단 하나만 존재해야 한다. (Unique key)

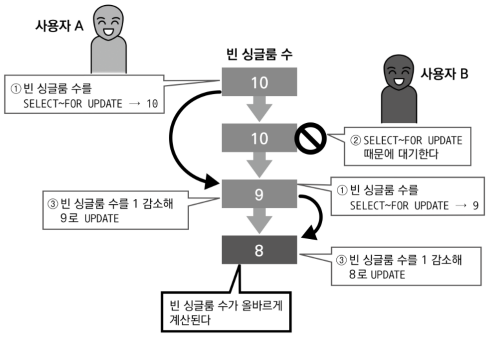
트랜잭션의 특징 3 - 고립성, 격리성 Isolation

- 트랜잭션을 수행 시 다른 트랜잭션의 연산 작업이 끼어들지 못하도록 보장하는 것
- 일련의 데이터 조작을 복수 사용자가 동시에 실행해도 각각의 처리가 모순없이 실행되는 것을 보장하는 것
  - 모순 없는 상태? 복수의 트랜잭션이 순서대로 실행되는 경우와 같은 결과를 얻을 수 있는 상태



두명이 방을 예약했으면 룸은 2개가 줄어야 한다.

- Isolation 을 유지하기 위해 테이블에 대해 잠금을 걸어 후속 처리를 블록하는 방법을 사용할 수 있다.
- 잠금 단위? 테이블 전체, 블록, 행



(1) 실행시 SELECT 행에 잠금이 걸리고, 따라서 사용자 B 는 잠금이 해제 될 때까지 기다려야 한다. 잠금은 커밋이나 롤백시 해제 된다.

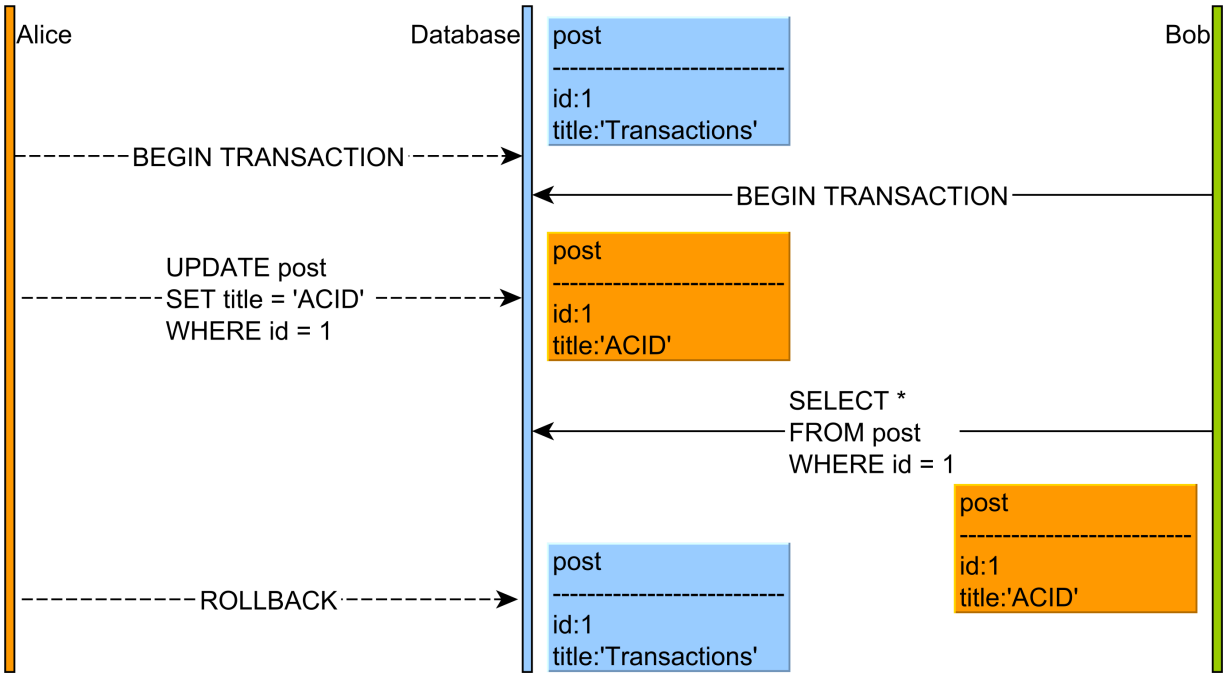
▼ InnoDB 형의 테이블의 경우

- MVCC 구조로 동작
- 위의 예에서 사용자 B 가 단순히 값을 참조하는 경우에는 읽기가 블록되지 않는다.
- 테이블을 갱신하는 사용자가 소수고 참조하는 사용자가 많은 경우에는 사용자의 동시성, 병렬성이 높아진다.

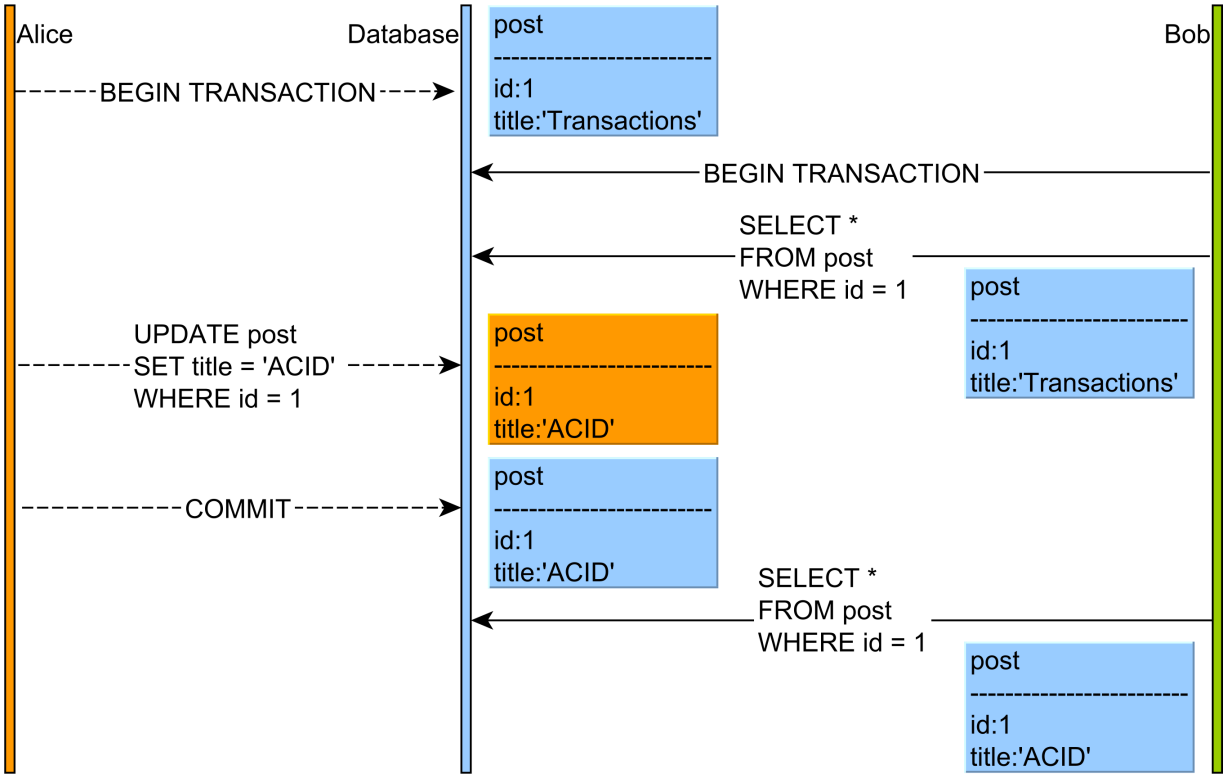
- DBMS 는 고립성을 격리수준으로 구현하며 ANSI 에서는 격리수준을 4 단계로 정의했다.
  - 커밋되지 않은 읽기 Read Uncommited (가장 느슨)
  - 커밋된 읽기 Read Committed
  - 반복 읽기 Repeatable Read
  - 직렬화 가능 Serializable (가장 엄격)

격리 수준	더티 읽기	애매한 읽기	팬텀 읽기
커밋되지 않은 읽기	○	○	○
커밋된 읽기	×	○	○
반복 읽기	×	×	○
직렬화 가능	×	×	×

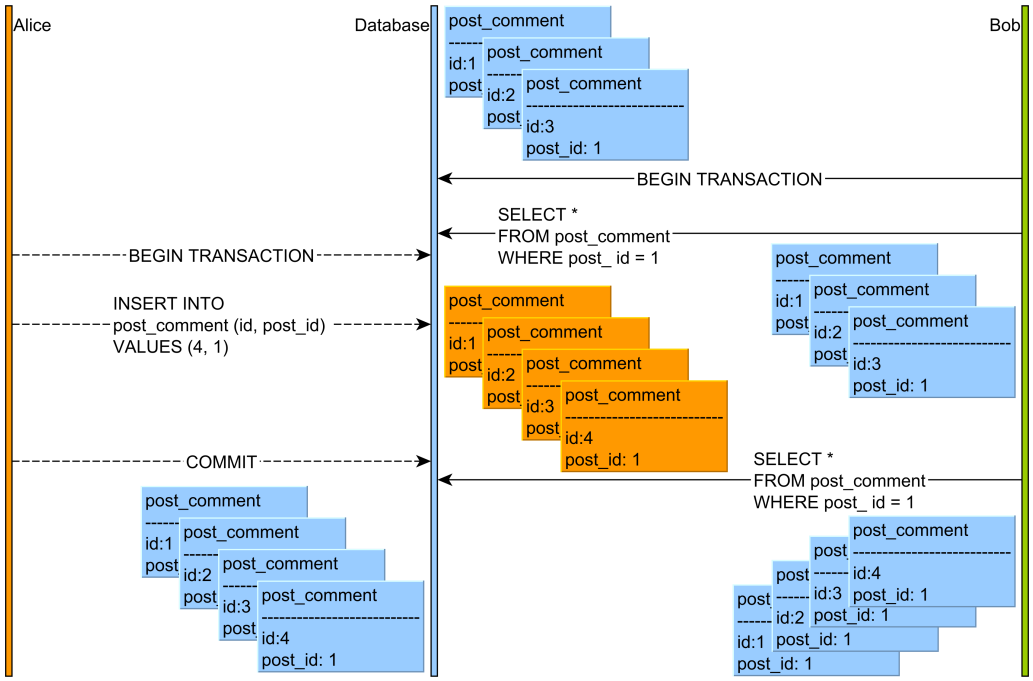
▼ 더티 읽기 Dirty Read: 어떤 트랜잭션이 커밋되기 전에 다른 트랜잭션에서 데이터를 읽는 현상



▼ 애매한 읽기 Fuzzy / Non repeatable Read : 어떤 트랜잭션이 이전에 읽어 들인 데이터를 다시 읽어들일 때 2회 이후의 결과가 1회 때와 다른 현상



▼ 팬텀 읽기: 어떤 트랜잭션을 읽을 때 선택할 수 있는 데이터가 나타나거나 사라지는 현상



💡 동시 접근을 제한하기 위하여 Lock을 설정할 수 있다. Lock을 건다는 것은 그만큼 동시처리량이 줄어든다는 의미이기 때문에 과도하게 사용하면 성능에 문제가 생길 수 있다. 따라서 상황별로 꼭 필요한 수준의 Lock을 걸어서 정합성을 유지하고 최대한의 성능을 내기 위해서는 동시성 이슈가 발생할 수 있는 상황들을 정확히 정의하여 구분하는 것이 매우 중요하다.

트랜잭션의 특징 4 - 지속성 Durability

- 성공적으로 수행된 트랜잭션은 영원히 반영되어야 함
- 시스템이 정상일 때 뿐 아니라, 데이터베이스나 OS의 이상 종료도 견딜 수 있다는 뜻
- 많은 데이터베이스는 하드 디스크에 로그를 기록하고 있는데 시스템에 이상이 발생하여 롤백이 필요한 경우 전에 기록한 로그를 사용해 이상 발생 이전 상태까지 복원함으로써 지속성을 실현한다

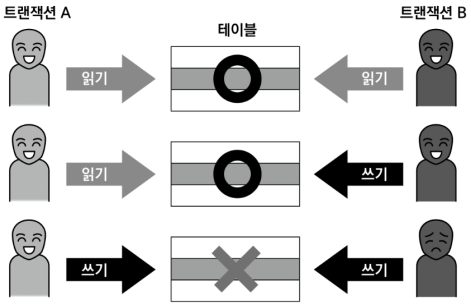
어떤 커넥션에서 데이터베이스 조작 시 다른 커넥션에서는 어떻게 보일 것인가?

- (MySQL이나 오라클에서는) DDL 실행 시 암묵적인 커밋이 발행한다.
  - 따라서 한 커넥션에서 `CREATE TABLE` 과 같은 DDL 실행을 성공하면 다른 커넥션에서도 새로 생성한 테이블을 참조할 수 있게 된다.

- 트랜잭션 개시가 명시적으로 지정`(BEGIN TRANSACTION. START TRANSACTION. SET TRANSACTION)` 되지 않았을 때 트랜잭션을 구분하는 방식
  1. 하나의 SQL 을 하나의 트랜잭션으로 구분 -
  2. 사용자가 COMMIT 또는 ROLLBACK 을 실행할 때 까지를 하나의 트랜잭션으로 구분

MVCC를 사용하는 MySQL의 특징

그림 7-4 MVCC에서의 잠금(같은 행에 대한 처리)



실습

```
create table t1 (

  i1 INT not null primary key,
  v2 varchar(30)

) ENGINE=InnoDB;
```

- 읽기를 수행할 경우 갱신 중이라도 블록되지 않는다 (읽기와 읽기도 서로 블록되지 않는다)
- 읽기 내용은 격리 수준에 따라 내용이 바뀌는 경우가 있다.

복수의 읽기가 중복되지 않는 상황

```
Transaction A> use test;
Transaction A> start transaction;




Transaction A> select * from t1;
+------+----------+
| i1   | v2       |
+------+----------+
|    1 | Firdbird |
+------+----------+
1 row in set (0.00 sec)









Transaction A> rollback;
```

```
Transaction B> use test;
Transaction B> start transaction;









Transaction B> select * from t1;
+------+----------+
| i1   | v2       |
+------+----------+
|    1 | Firdbird |
+------+----------+
1 row in set (0.00 sec)


Transaction B> rollback;
```

읽기와 갱신이 블록되지 않는 상황 + 반복읽기 (더티 읽기 발생하지 않고, 커밋되기 전 데이터는 못 읽는다)

```
Transaction A> use test;
Transaction A> start transaction;









Transaction A> select * from t1;
Empty set (0.00 sec)



Transaction A> select * from t1;
Empty set (0.00 sec)
```

```
Transaction B> use test;
Transaction B> start transaction;


Transaction B> insert into t1 values (1, 'PostgreSQL');



Transaction B> insert into t1 values (2, 'MySQL');



Transaction B> insert into t1 values (3, 'Oracle');



Transaction B> select * from t1;
+----+------------+
| i1 | v2         |
```

```
Transaction A> rollback;
```

```
+----+------------+
|  1 | PostgreSQL |
|  2 | MySQL      |
|  3 | Oracle     |
+----+------------+
3 rows in set (0.00 sec)


Transaction B> rollback;
```

갱신과 갱신은 블락된다.

- 갱신 시 **배타적 잠금(insert, update, delete)** 을 얻는다. 잠금은 기본적으로 행 단위로 얻으며 트랜잭션이 종료될 때까지 유지된다.
  격리 수준이나 InnoDB의 설정에 따라 실제 잠금 하는 행의 범위가 다른 경우도 있다.
- 갱신과 갱신은 나중에 온 트랜잭션이 잠금을 획득하려고 할 때 블록된다.
  일정 시간을 기다리며 그 사이에 잠금을 획득할 수 없는 경우에는 잠금 타임아웃이 된다.

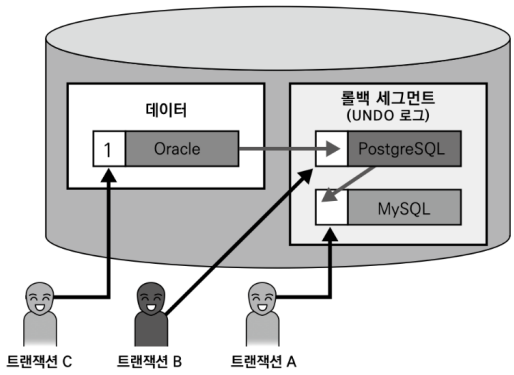
```
Transaction A> start transaction;




Transaction A> insert into t1 values (4, 'Oracle');
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction


Transaction A> rollback;
```

```


Transaction B> start transaction;

Transaction B> insert into t1 values (4, 'JavaDB');




Transaction B> rollback;
```

- 갱신하는 경우 갱신 전의 데이터를 UNDO 로그로 롤백 세그먼트라는 영역에 유지한다.
- UNDO 로그의 용도
  1. 갱신하는 트랜잭션의 롤백 시 갱신 전으로 되돌릴 때 사용
  2. 복수의 트랜잭션으로부터 격리 수준에 따라 대응하는 갱신 데이터를 참조할 때 사용

읽기 내용은 격리 수준에 따라 내용이 바뀌는 경우가 있다.

그림 7-5 ⑤ select * from t1 where i1 =1로 보이는 행



- 반복 읽기 상태의 트랜잭션에서는 (MySQL 트랜잭션 격리 수준의 기본 값) - 트랜잭션 A
  - 최초 쿼리를 실행한 시점에 커밋된 데이터를 읽어 들인다.
  - 같은 쿼리를 복수회 실행하면 최초 읽은 내용의 결과 세트가 반환된다.
  - 복수 회의 쿼리 실행 사이에 다른 트랜잭션이 커밋되어도 그 내용은 반영되지 않는다.

- 커밋된 읽기 상태의 트랜잭션에서는 - 트랜잭션 B
  - 쿼리를 실행한 시점에서 커밋된 데이터를 읽어 들인다.
  - 같은 쿼리를 복수 회 실행하는 사이 다른 트랜잭션에서 커밋을 하게 되면 최신 쿼리의 실행 개시 시점에서 커밋된 데이터를 읽어 들인다.

- 갱신을 수행하는 트랜잭션 자신 - 트랜잭션 C
  - 자신의 트랜잭션 격리 수준에 상관 없이 자신이 수행한 갱신을 즉시 볼 수 있다.

```
A> set transaction isolation level repeatable read;
A> start transaction;




A> select * from t1;
+----+-------+
| i1 | v2    |
+----+-------+
|  1 | MySQL |
+----+-------+
1 row in set (0.00 sec)




A> select * from t1;
+----+-------+
| i1 | v2    |
+----+-------+
|  1 | MySQL |
+----+-------+
1 row in set (0.00 sec)




A> rollback;
```

```
B> set transaction isolation level read committed;
B> start transaction;




B> select * from t1;
+----+----------+
| i1 | v2       |
+----+----------+
|  1 | MySQL    |
+----+----------+
1 row in set (0.00 sec)




B> select * from t1 where i1 = 1;
+----+------------+
| i1 | v2         |
+----+------------+
|  1 | PostgreSQL |
+----+------------+
1 row in set (0.00 sec)




B> rollback;
```

```
C> start transaction;


C> update t1 set v2='MySQL' where i1 = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

C> commit;









C> start transaction;
C> update t1 set v2='PostgreSQL' where i1 = 1;
C> commit;


C> start transaction;
C> update t1 set v2='Oracle' where i1 = 1;









C> select * from t1 where i1 = 1;
+----+--------+
| i1 | v2     |
+----+--------+
|  1 | Oracle |
+----+--------+
1 row in set (0.00 sec)



C> rollback;
```

커밋되지 않는 읽기가 사용되지 않는 이유

?????? 더티 리드 더티 라이트 때문에 쓰지 않는다고 한다.

잠금 타임 아웃

- 갱신과 참조는 서로를 블록하지 않지만
- 갱신과 갱신은 나중에 온 갱신이 잠금 대기 상태가 된다.
- 잠금을 건 쪽이 언제 잠금을 풀 지 모르기 때문에 작금 해제를 기다리고 있는 쪽에서는
  1. 계속 잠금을 기다리거나
  2. 기다리지 않거나 (MySQL 에서는 없는 기능)
  3. 기다린다면 얼마 동안 기다릴지를

  설정할 수 있습니다.

```
mysql> set innodb_lock_wait_timeout = 1
```

잠금 대기로 타임아웃이 발생하면 롤백이 된다. 롤백 되는 단위로는

1. 해당 트랜잭션 전체를 롤백하는 방식

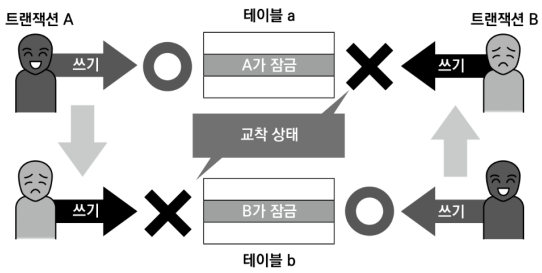
2. 오류가 발생한 쿼리만 롤백하는 방식 (MySQL 기본값)

두 가지 방식이 있다.

> 💡 MySQL에서 잠금 타임아웃으로 해당 트랜잭션 전체를 롤백하고 싶다면

교착 상태

그림 7-7 교착 상태



1. 트랜잭션 A 가 테이블 a 의 잠금을 얻는다
2. 트랜잭션 B 가 테이블 b 의 잠금을 얻는다
3. 트랜잭션 A 가 테이블 b 에서 잠금이 필요한 처리를 실행하거나, 반대의 경우

아무리 기다려도 아무도 잠금을 얻을 수 없는 교착 상태가 된다

교착 상태의 빈도를 낮추는 대책

- 잠금 타임아웃은 일정 시간 기다리면 잠금을 건 곳에서 잠금을 풀어서 잠금을 얻게 될 수 있는 개선의 여지가 있지만
- 교착 상태는 아무리 기다려도 상황이 개선될 가능성이 없다.
- 따라서 DBMS 에서는 교착 상태를 독자적으로 검출해 교착 상태를 보고한다.

  MySQL 의 경우 시스템에 영향이 적은 쪽의 트랜잭션을 트랜잭션 개시 시점까지 롤백한다.
- 어플리케이션 쪽에서 교착 상태를 일으켜 롤백되는 경우에 트랜잭션을 재실행 할 수 있는 구조로 만들어야 한다.

1. 트랜잭션을 자주 커밋한다.
2. 테이블을 액세스 할 수 있는 순서를 정한다 (무조건 테이블 a 이후 테이블 b 를 액세스 할 수 있도록)
3. 필요 없는 경우 읽기 잠금 획득의 사용을 피한다.
4. 잠금 범위를 좁혀라. 되도록 격리 수준을 커밋된 읽기로 한다. (왜지???)
5. 동시성과 교착 상태의 트레이드 오프. 애초에 테이블 자체를 잠궈버려서 한 테이블 내 복수 행을 갱신하다가 부딪히는 교착 상태를 피할 수도 있다.
6. 테이블에 적절한 인덱스를 추가하여 쿼리가 이를 이용하게 한다.
   인덱스가 사용되지 않는 경우는 필요한 행의 잠금이 아닌 스캔한 행 전체에 잠금이 걸리게 된다.
   (필요한 행의 갯수가 스캔한 행 전체 갯수보다 적다)

주의해야 할 트랜잭션 종류

1. 오토 커밋

   일정 수 이상의 갱신을 수행하는 처리는 적당히 여러개 모아서 한번의 트랜잭션으로 처리해야 커밋에 걸리는 부하를 줄일 수 있다.
2. 긴 트랜잭션

   긴 트랜잭션은 트랜잭션의 동시성이나 자원의 유효성을 저하한다.

   같은 테이블과 행을 갱신하려는 다른 테이블을 블록하고 장시간 이어지면 타임아웃까지 시키기 때문.

   트랜잭션이 길어지면 교착 상태에 빠질 가능성도 크다.