PRESENTATION

17주차 주제 트랜잭션

예외상황에서의 커밋과 롤백

by mun

transaction

트랜잭션이란?

- 쪼갤 수 없는 업무 처리의 최소 단위
- 데이터베이스 관리 시스템 또는 유사한 시스템에서 상호작용의 단위

특징:ACID

Atomicity 원자성

하나의 트랜잭션은 더 이상 작게 쪼갤 수 없는 최소한의 업무 단위여야 한다

Consistency 일관성

트랜잭션이 완료된 결괏값이 일관적인 DB 상태를 유지해야 한다

Isolation 독립성

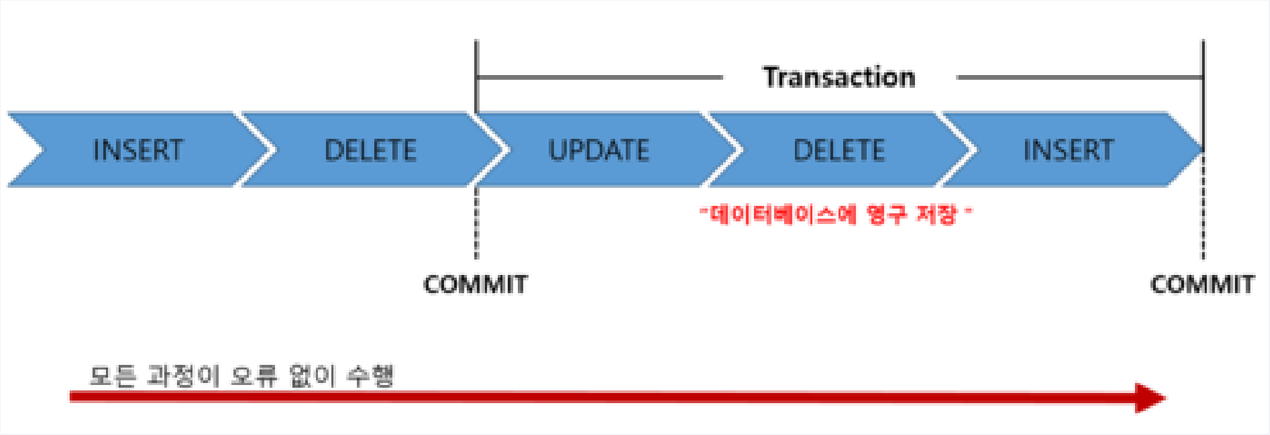
하나의 트랜잭션 수행시 다른 트랜잭션의 작업이 끼어들지 못하도록 보장되어야 한다

Durability 지속성

트랜잭션이 정상적으로 종료된 다음, DB에 영구적으로 작업의 결과가 저장되어야 한다

커밋과 롤백

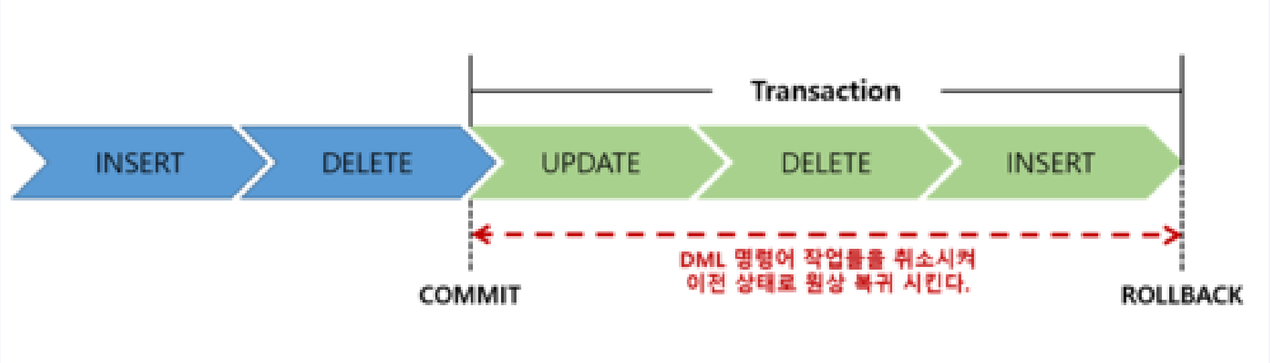
트랜잭션 과정



http://wiki.hash.kr/index.php/%ED%8A%B8%EB%9E%9C%EC%9E%AD%EC%85%98

커밋

- 모든 작업들을 정상적으로 처리하겠다고 확정하는 명령어
- 최종적으로 데이터베이스에 반영한다



http://wiki.hash.kr/index.php/%ED%8A%B8%EB%9E%9C%EC%9E%AD%EC%85%98

롤백

- 작업 중 문제가 발생하여 트랜잭션 처리과정에서 발생한 변경사항을 취소하는 명령어
- 실패했을 때 시점으로 돌아간다

롤백 확인하기

User: name=변경안됨

```
@Transactional
public void changeName() {
        User user = userRepository.findById(1).get();
        user.updateName("변경됨");
        throw new RuntimeException();
}
```

롤백 확인하기

```
@Transactional
public void changeName() {
        User user = userRepository.findById(1).get();
        user.updateName("변경됨");
        throw new RuntimeException();
}
```

User: name=변경안됨

User: name=변경안됨

롤백 확인하기

```
@Transactional
public void changeName() {
        User user = userRepository.findById(1).get();
        user.updateName("변경됨");
        throw new RuntimeException();
}
```

```
@Transactional
public void changeName() throw IOExcpetion {
        User user = userRepository.findById(1).get();
        user.updateName("변경됨");
        throw new IOException();
}
```

User: name=변경안됨

User: name=변경안됨

롤백 확인하기

```
@Transactional
public void changeName() {
        User user = userRepository.findById(1).get();
        user.updateName("변경됨");
        throw new RuntimeException();
}
```

```
@Transactional
public void changeName() throw IOExcpetion {
        User user = userRepository.findById(1).get();
        user.updateName("변경됨");
        throw new IOException();
}
```

User: name=변경안됨

User: name=변경안됨

User: name=변경됨

롤백 확인하기

```
@Transactional
public void changeName() {
        User user = userRepository.findById(1).get();
        user.updateName("변경됨");
        throw new RuntimeException();
}
```

```
@Transactional
public void changeName() throw IOExcpetion {
        User user = userRepository.findById(1).get();
        user.updateName("변경됨");
        throw new IOException();
}
```

User: name=변경안됨

User: name=변경안됨

User: name=변경됨

모든 예외상황에
롤백이 적용되지는 않는다

Exception Checked vs Unchecked

	Checked Exception	Unchecked Exception
	RuntimeException을 제외한 Exception의 하위 클래스들	RuntimeException의 하위 클래스들 실행 중에(runtime) 발생할 수 있는 예외를 의미
확인시점	컴파일 단계	실행단계
예외처리	반드시 예외를 처리해야 한다	처리를 강제하지 않는다
트랜잭션	롤백하지 않는다	롤백한다
대표예외	IOException SQLException FileNotFoundException	NullPointerException IndexOutOfBoundException

Unchecked Exception(RuntimeException)

RuntimeException

```java
@Transactional
public void testRuntimeException() {
    User user = userRepository.findById(1).get();
    log.info(user.getName());
    user.updateName( newName: "변경");
    throw new RuntimeException();
}
```

실행 전

SELECT id, name FROM "user" where id=1;

ID	NAME
1	before

(1 row, 1 ms)

실행 후

SELECT id, name FROM "user" where id=1;

ID	NAME
1	before

(1 row, 0 ms)

Unchecked Exception(CustomException)

RuntimeException

```java
@Transactional
public void testCustomException() {
    User user = userRepository.findById(1).get();
    log.info(user.getName());
    user.updateName( newName: "변경");
    throw new SpringQuestException();
}
```

실행 전

SELECT id, name FROM "user" where id=1;

ID	NAME
1	before

(1 row, 1 ms)

실행 후

SELECT id, name FROM "user" where id=1;

ID	NAME
1	before

(1 row, 0 ms)

Checked Exception(IOException)

CheckedExceptoion

```java
@Transactional
public void testCheckedException() throws IOException {
    User user = userRepository.findById(1).get();
    log.info(user.getName());
    user.updateName( newName: "변경");
    throw new IOException();
}
```

실행 전

SELECT id, name FROM "user" where id=1;

ID	NAME
1	before

(1 row, 1 ms)

실행 후

SELECT id, name FROM "user" where id=1;

ID	NAME
1	변경

(1 row, 1 ms)

어떻게 처리하면 좋을까요?

SQLException

예시 SQL 내에서 중복을 허용하지 않는 칼럼에
중복 데이터를 저장하려고 할 때

어떻게 처리하면 좋을까요?

SQLException

예시 SQL 내에서 중복을 허용하지 않는 칼럼에
중복 데이터를 저장하려고 할 때

방안 Checked 오류 발생 전에
Runtime Exception을 발생시킨다

```
userRepository.findById(id).ifPresent(it -> {
        throw new CustomException("중복된 아이디");
});
```

어떻게 처리하면 좋을까요?

SQLException

예시 SQL 내에서 중복을 허용하지 않는 칼럼에
중복 데이터를 저장하려고 할 때

방안 Checked 오류 발생 전에
Runtime Exception을 발생시킨다

```
userRepository.findById(id).ifPresent(it -> {
        throw new CustomException("중복된 아이디");
});
```

방안 rollbackFor 속성을 활용한다

```
@Transactional(rollbackFor = Exception.class)
```

rollbackFor 속성

트랜잭션을 rolblack 할 예외를 지정한다

CheckedExceptoion

```java
@Transactional(rollbackFor = IOException.class)
public void testCheckedException() throws IOException {
    User user = userRepository.findById(1).get();
    user.updateName( newName: "변경");
    throw new IOException();
}
```

실행 전

SELECT id, name FROM "user" where id=1;

ID	NAME
1	before

(1 row, 1 ms)

실행 후

SELECT id, name FROM "user" where id=1;

ID	NAME
1	before

(1 row, 0 ms)

트랜잭션을 이해하고
적절하게 적용해보아요

감사합니다

참고 :

https://devlog-wjdrbs96.tistory.com/351 , https://kghworks.tistory.com/106 ,
https://cheese10yun.github.io/checked-exception/ , https://kghworks.tistory.com/89 ,
http://wiki.hash.kr/index.php/%ED%8A%B8%EB%9E%9C%EC%9E%AD%EC%85%98