Spring transaction isolation level tutorial

Introduction

Transaction isolation level is a concept that is not exclusive to the Spring framework. It is applied to transactions in general and is directly related with the ACID transaction properties. Isolation level defines how the changes made to some data repository by one transaction affect other simultaneous concurrent transactions, and also how and when that changed data becomes available to other transactions. When we define a transaction using the Spring framework we are also able to configure in which isolation level that same transaction will be executed.

Usage example

Using the **@Transactional** annotation we can define the isolation level of a Spring managed bean transactional method. This means that the transaction in which this method is executed will run with that isolation level:

Isolation level in a transactional method

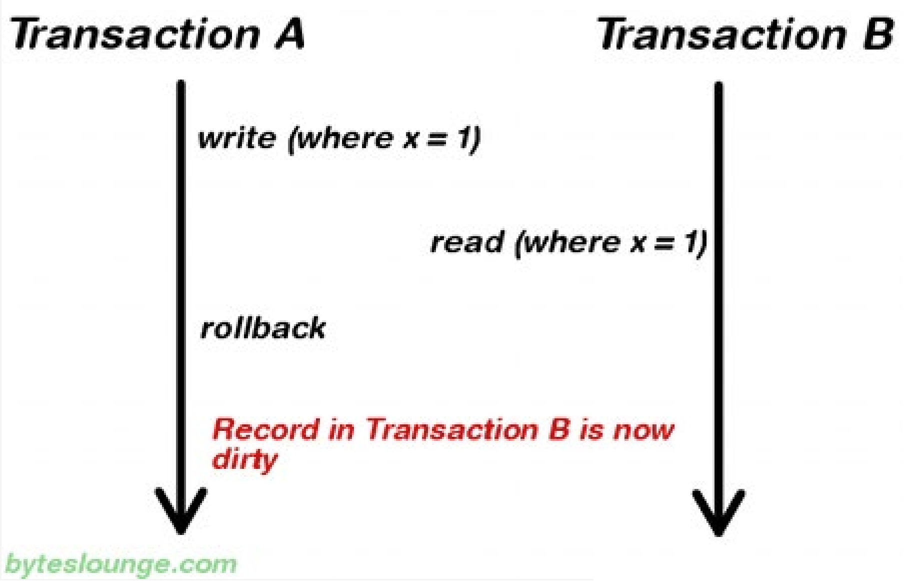
```
@Autowired

private TestDAO testDAO;



@Transactional(isolation=Isolation.READ_COMMITTED)

public void someTransactionalMethod(User user) {



  // Interact with testDAO



}
```

We are defining this method to be executed in a transaction which isolation level is **READ_COMMITTED**. We will see each isolation level in detail in the next sections.

READ_UNCOMMITTED

READ_UNCOMMITTED isolation level states that a transaction **may** read data that is still **uncommitted** by other transactions. This constraint is very relaxed in what matters to transactional concurrency but it may lead to some issues like **dirty reads**. Let's see the following image:

Dirty read



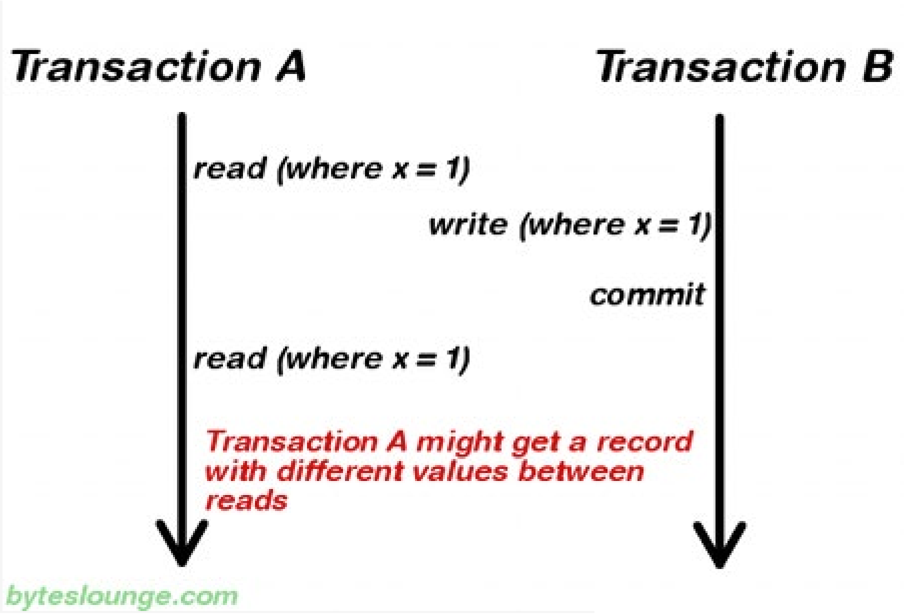
In this example **Transaction A** writes a record. Meanwhile **Transaction B** reads that same record before **Transaction A** commits. Later **Transaction A** decides to rollback and now we have changes in **Transaction B** that are inconsistent. This is a **dirty read**. **Transaction B** was running in **READ_UNCOMMITTED** isolation level so it was able to read **Transaction A** changes before a commit occurred.

Note: READ_UNCOMMITTED is also vulnerable to **non-repeatable reads** and **phantom reads**. We will also see these cases in detail in the next sections.

READ_COMMITTED

READ_COMMITTED isolation level states that a transaction can't read data that is **not** yet committed by other transactions. This means that the **dirty read** is no longer an issue, but even this way other issues may occur. Let's see the following image:

Non-repeatable read



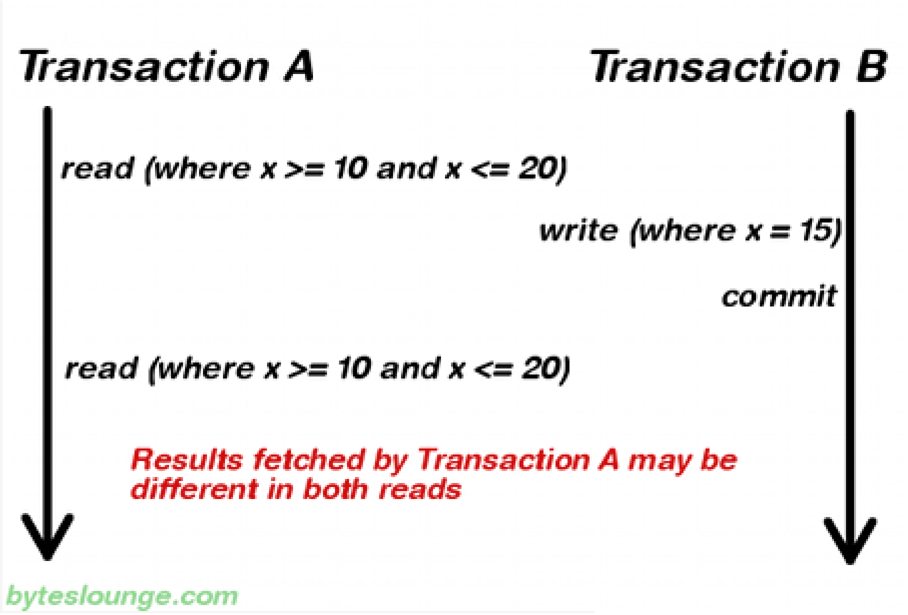
In this example **Transaction A** reads some record. Then **Transaction B** writes that same record and commits. Later **Transaction A** reads that same record again and may get different values because **Transaction B** made changes to that record and committed. This is a **non-repeatable read**.

Note: READ_COMMITTED is also vulnerable to **phantom reads**. We will also see this case in detail in the next section.

REPEATABLE_READ

REPEATABLE_READ isolation level states that if a transaction reads one record from the database multiple times the result of all those reading operations must always be the same. This eliminates both the **dirty read** and the **non-repeatable read** issues, but even this way other issues may occur. Let's see the following image:

Phantom read


Transaction A
Transaction B
read (where x >= 10 and x <= 20)
write (where x = 15)
commit
read (where x >= 10 and x <= 20)
Results fetched by Transaction A may be different in both reads
byteslounge.com

In this example **Transaction A** reads a **range** of records. Meanwhile **Transaction B** inserts a new record in the same range that **Transaction A** initially fetched and commits. Later **Transaction A** reads the same range again and will also get the record that **Transaction B** just inserted. This is a **phantom read**: a transaction fetched a range of records multiple times from the database and obtained different result sets (containing phantom records).

SERIALIZABLE

SERIALIZABLE isolation level is the most restrictive of all isolation levels. Transactions are executed with locking at all levels (**read**, **range** and **write** locking) so they appear as if they were executed in a serialized way. This leads to a scenario where **none** of the issues mentioned above may occur, but in the other way we don't allow transaction concurrency and consequently introduce a **performance penalty**.

DEFAULT

DEFAULT isolation level, as the name states, uses the default isolation level of the datastore we are actually connecting from our application.

Summary

To summarize, the existing relationship between isolation level and read phenomena may be expressed in the following table:

	dirty reads	non-repeatable reads	phantom reads
READ_UNCOMMITTED	yes	yes	yes
READ_COMMITTED	no	yes	yes

REPEATABLE_READ	no	no	yes
SERIALIZABLE	no	no	no

JPA

If you are using Spring with JPA you may come across the following exception when you use an isolation level that is different the default:

```
InvalidIsolationLevelException: Standard JPA does not support custom isolation levels - use a special JpaDialect for your JPA implementation
at
org.springframework.orm.jpa.DefaultJpaDialect.beginTransaction(DefaultJpaDialect.java:67)
at
org.springframework.orm.jpa.JpaTransactionManager.doBegin(JpaTransactionManager.java:378)
at
org.springframework.transaction.support.AbstractPlatformTransactionManager.getTransaction(AbstractPlatformTransactionManager.java:372)
at
org.springframework.transaction.interceptor.TransactionAspectSupport.createTransactionIfNecessary(TransactionAspectSupport.java:417)
at
org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:255)
at
org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:94)
at
org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:172)
at
org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:204)
```

To solve this problem you must implement a custom JPA dialect which is explained in detail in the following article: Spring - Change transaction isolation level example.