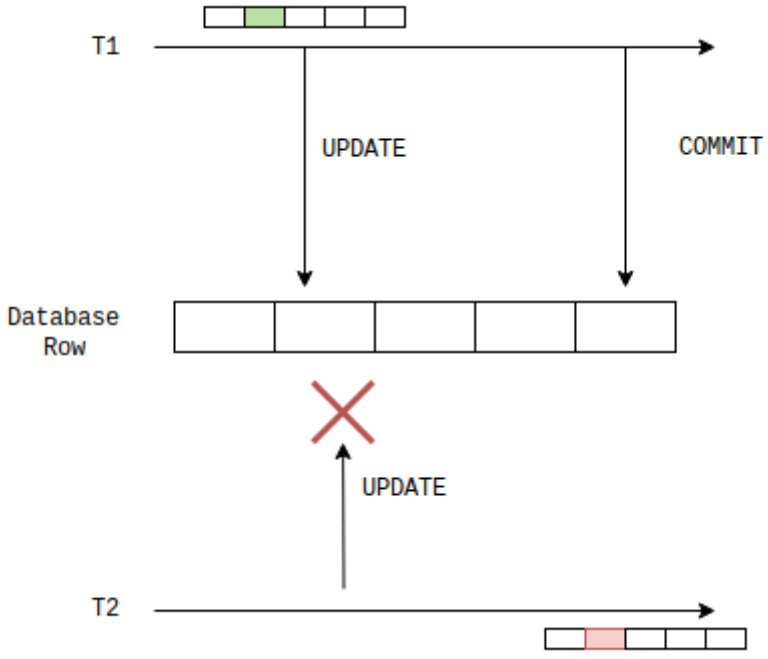
by Adam Stankowski April 1, 2019 0

Transaction Isolation Levels With Examples

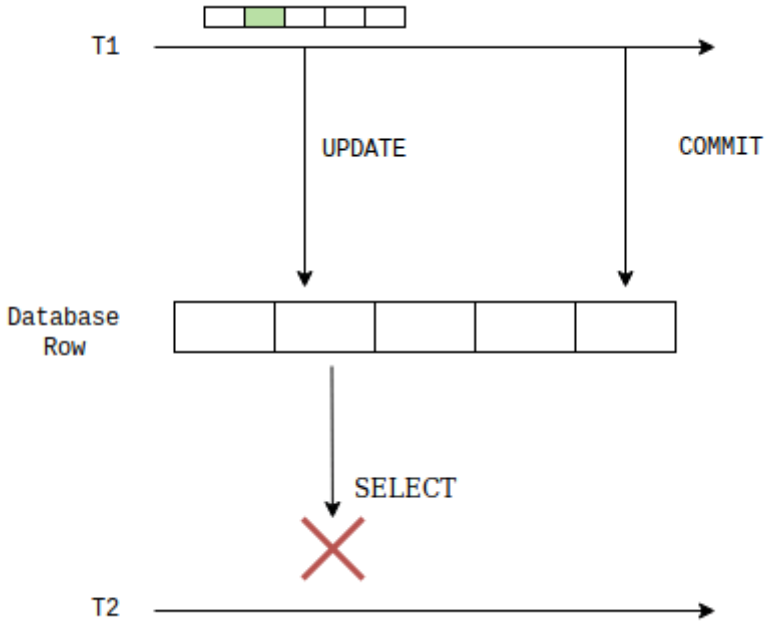
From the previous article, we know that transactions can interfere with each other, which can adversely impact what they're trying to accomplish, or rather what the code author tried to accomplish. Luckily this fact is not left out without remedy. Transactions offer a wide range of **isolation levels** which ensure some or all of the problems which we've seen are addressed. Transaction isolation levels **are defined in the ANSI standard and are not Hibernate specific**. Here's a walkthrough of what each level has to offer.

Read uncommitted

Read uncommitted isolation level makes sure **no transaction can update a database row if another transaction has already updated it and not committed.** This protects against lost updates, but won't stand in a way of dirty reads. This means that transaction two below is free to read the change made by transaction one before it commits, but transaction two won't be able to perform an update on the same database row.

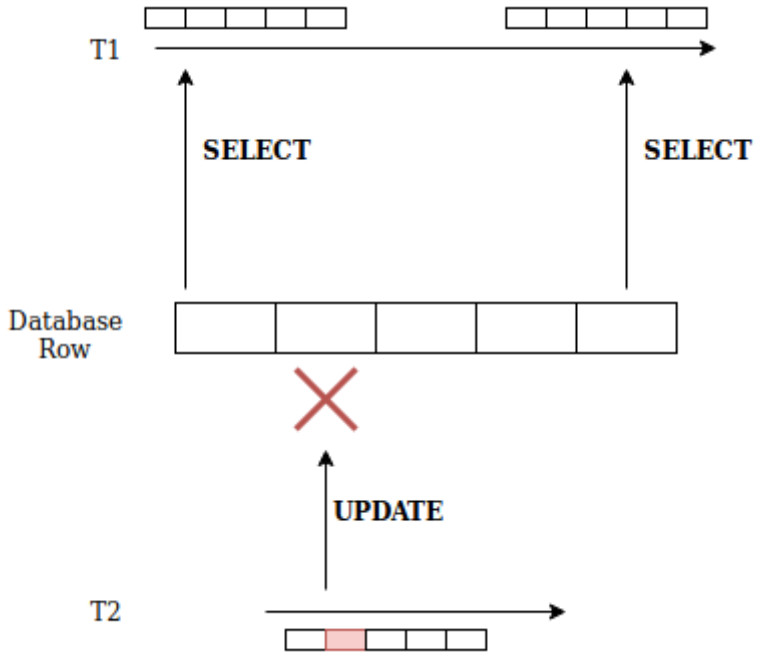


Read committed isolation **does not allow any other transaction to write or read a row to which another transaction has written to but not yet committed.** Transaction two won't be able to read the row updated by transaction one. That's how protection against lost updates and Dirty reads is realized. Unrepeatable reads are allowed with this isolation level.



Repeatable read

With Repeatable Read isolation, any **transaction which reads data from a row blocks any other writing transactions from accessing the same row.**



Serializable

In order to protect against phantom reads, serializable isolation level requires a lot more than restricting access to a single row. Typically this isolation mode would **lock the whole table**, to prevent any other transactions from inserting or reading data from it. Serializable isolation is the strictest of all the available levels and comes at the highest cost. When the entire table is locked, the probability that other transactions will have to wait is very large.

Which transaction isolation to choose

The choice of transaction isolation level is very individual and depends on the details of each specific case. Take a look at some hints which may be helpful, but please consider each situation individually.

When designing your application, you definitely want to ensure that **none of your database transactions read uncommitted changes of other transactions.** Reading uncommitted changes can easily harm your data integrity, as reverted changes in one transaction can be read and potentially accepted by another. The minimum isolation level to ensure in your application, therefore, is Read Committed.
Most of the times **you probably don't need Serializable isolation**, as it can cause big performance issues with large volume of concurrent transactions. Phantom reads are seldom an issue.
The question, therefore, comes down to choosing whether you need to ensure repeatable read and whether you'd like to handle the First Commit wins problem.
As Oracle documentation states, JDBC accepts the default transaction level provided by your DBMS. In the case of MySql it's Repeatable Read, but for most other providers it's Read Committed.
JPA specifies Read Committed as a default transaction isolation level.

- What are transactions and what are their typical isolation issues
- What is Hibernate Framework
- Complete Spring and Hibernate application example

Leave a Comment