Difference between “read commited” and “repeatable read”

Asked 10 years, 5 months ago Active 7 months ago Viewed 142k times

274

I think the above isolation levels are so alike. Could someone please describe with some nice examples what the main difference is ?

sql sql-server isolation-level

172

Share Improve this question Follow

edited May 18 '20 at 15:46
vinzee
10.2k 9 32 50

asked Oct 27 '10 at 15:35
Fore
4,568 7 20 35

3 You should expand on the question and add tags for what "isolation level" you're referring to (Java, etc). "isolation level" is a somewhat ambiguous term, and you're obviously asking for an answer for a specific environment. – jesup Oct 27 '10 at 15:39

8 Answers

Active Oldest Votes

621

Read committed is an isolation level that guarantees that any data read was *committed* at the moment is read. It simply restricts the reader from seeing any intermediate, uncommitted, 'dirty' read. It makes no promise whatsoever that if the transaction re-issues the read, will find the *Same* data, data is free to change after it was read.

Repeatable read is a higher isolation level, that in addition to the guarantees of the read committed level, it also guarantees that any data read *cannot change*, if the transaction reads the same data again, it will find the previously read data in place, unchanged, and available to read.

The next isolation level, serializable, makes an even stronger guarantee: in addition to everything repeatable read guarantees, it also guarantees that *no **new** data* can be seen by a subsequent read.

Say you have a table T with a column C with one row in it, say it has the value '1'. And consider you have a simple task like the following:

```
BEGIN TRANSACTION;
SELECT * FROM T;
WAITFOR DELAY '00:01:00'
SELECT * FROM T;
COMMIT;
```

That is a simple task that issue two reads from table T, with a delay of 1 minute between them.

- under READ COMMITTED, the second SELECT may return *any* data. A concurrent transaction may update the record, delete it, insert new records. The second select will always see the *new* data.
- under REPEATABLE READ the second SELECT is guaranteed to display at least the rows that were returned from the first SELECT *unchanged*. New rows may be added by a concurrent transaction in that one minute, but the existing rows cannot be deleted nor changed.
- under SERIALIZABLE reads the second select is guaranteed to see *exactly* the same rows as the first. No row can change, nor deleted, nor new rows could be inserted by a concurrent transaction.

If you follow the logic above you can quickly realize that SERIALIZABLE transactions, while they may make life easy for you, are always *completely blocking* every possible concurrent operation, since they require that nobody can modify, delete nor insert any row. The default transaction isolation level of the .Net `System.Transactions` scope is serializable, and this usually explains the abysmal performance that results.

And finally, there is also the SNAPSHOT isolation level. SNAPSHOT isolation level makes the same guarantees as serializable, but not by requiring that no concurrent transaction can modify the data. Instead, it forces every reader to see its own version of the world (it's own 'snapshot'). This makes it very easy to program against as well as very scalable as it does not block concurrent updates. However, that benefit comes with a price: extra server resource consumption.

Supplemental reads:

- Isolation Levels in the Database Engine
- Concurrency Effects
- Choosing Row Versioning-based Isolation Levels

Share Improve this answer Follow

edited Apr 23 '20 at 8:18
raven
2,168 1 15 41

answered Oct 27 '10 at 17:44
Remus Rusanu
272k 38 406 536

27 I think there is a mistake above for REPEATABLE READ: You say existing rows cannot be deleted nor changed, but I think they can be deleted or changed because repeatable read simply reads a "snapshot" not the actual data. From the docs dev.mysql.com/doc/refman/5.0/en/…: "All consistent reads within the same transaction read the snapshot established by the first read." – Derek Litz Jan 17 '13 at 16:40

3 @Derek Litz Am I right in that you are saying: The data CAN/MAY be changed from a third party, whilst the transaction is taking place, but the reads will still see the 'old' original data as if the change had not taken place (the snapshot). – Programster Nov 8 '13 at 17:04

5 @Cornstalks. Yes, Phantom reads can occur from deletes (or inserts). Yes, phantom reads can occur in repeatable read isolation (from inserts only). No, Phantom reads from deletes cannot occur in repeatable read isolation. Test it. What I am saying is not contradicted by the documentation you have quoted. – AndyBrown Sep 19 '14 at 15:00

6 @Cornstalks NP. I only mentioned it at all because I wasn't 100% sure myself and had to deep-dive to be sure who was right! And I didn't want future readers to be mislead. Re keeping the comments,

probably best to keep as suggested. I'm sure anyone else interested in that level of fine detail will be particular enough to read all the comments!! – AndyBrown Sep 19 '14 at 15:35

15 Thanks for not deleting your comments. The discussion helps connect more dots. – Josh Dec 15 '15 at 1:49

73

Repeatable Read

The state of the database is maintained from the start of the transaction. If you retrieve a value in session1, then update that value in session2, retrieving it again in session1 will return the same results. Reads are repeatable.

```
session1> BEGIN;
session1> SELECT firstname FROM names WHERE id = 7;
Aaron

session2> BEGIN;
session2> SELECT firstname FROM names WHERE id = 7;
Aaron
session2> UPDATE names SET firstname = 'Bob' WHERE id = 7;
session2> SELECT firstname FROM names WHERE id = 7;
Bob
session2> COMMIT;

session1> SELECT firstname FROM names WHERE id = 7;
Aaron
```

Read Committed

Within the context of a transaction, you will always retrieve the most recently committed value. If you retrieve a value in session1, update it in session2, then retrieve it in session1again, you will get the value as modified in session2. It reads the last committed row.

```
session1> BEGIN;
session1> SELECT firstname FROM names WHERE id = 7;
Aaron

session2> BEGIN;
session2> SELECT firstname FROM names WHERE id = 7;
Aaron
session2> UPDATE names SET firstname = 'Bob' WHERE id = 7;
session2> SELECT firstname FROM names WHERE id = 7;
Bob
session2> COMMIT;

session1> SELECT firstname FROM names WHERE id = 7;
Bob
```

Makes sense?

Share Improve this answer Follow

edited May 21 '13 at 14:01
BenMorel
30.4k 39 159 283

answered Jan 18 '13 at 7:23
Hazel_arun
1,467 2 12 16

I tried Repeatable Read in SQL Server 2008 with "set isolation level repeatable read". Created two sql query windows. But did not work. Why? – Aditya Bokade Aug 17 '13 at 1:57

1 Why would the second session1 still read out Aaron? Isn't session2's transaction finished and commited? I know this old, but maybe someone can shed some light. – Sonny Childs May 20 '15 at 12:27

10 I think the Repeatable Read will block the second session until the first session committed. So the example is wrong. – Nighon Jul 22 '16 at 9:22

4 In case of Repeatable Read, when session 1 reads the row, it puts a shared lock, which would not allow any Exclusive lock (to session 2) for update, hence the data cannot be updated. – Taher Mar 20 '17 at 16:58

I think SQL server and MySQL behave differently when it comes to update of shared rows between two transactions – user2488286 Jun 28 '19 at 6:46

31

Simply the answer according to my reading and understanding to this thread and @remus-rusanu answer is based on this simple scenario:

There are two transactions A and B. Transaction B is reading Table X Transaction A is writing in table X Transaction B is reading again in Table X.

- **ReadUncommitted**: Transaction B can read uncommitted data from Transaction A and it could see different rows based on B writing. **No lock at all**
- **ReadCommitted**: Transaction B can read ONLY committed data from Transaction A and it could see different rows based on COMMITTED only B writing. **could we call it Simple Lock?**
- **RepeatableRead**: Transaction B will read the same data (rows) whatever Transaction A is doing. But Transaction A can change other rows. **Rows level Block**
- **Serialisable**: Transaction B will read the same rows as before and Transaction A cannot read or write in the table. **Table-level Block**
- **Snapshot**: every Transaction has its own copy and they are working on it. **Each one has its own view**

Share Improve this answer Follow

edited Aug 31 '20 at 11:59

answered Jul 29 '16 at 3:53
Mo Zaatar
522 5 11

1 This is the best – Riding Cave Aug 10 '20 at 12:22

Very concise and to the point. I would replace the word 'Process' by 'transaction' – Junaed Aug 27 '20 at 10:21

17

Old question which has an accepted answer already, but I like to think of these two isolation levels in terms of how they change the locking behavior in SQL Server. This might be helpful for those who are debugging deadlocks like I was.

READ COMMITTED (default)

Shared locks are taken in the SELECT and then released **when the SELECT statement completes**. This is how the system can guarantee that there are no dirty reads of uncommitted data. Other transactions can still change the underlying rows after your SELECT completes and before your transaction completes.

REPEATABLE READ

Shared locks are taken in the SELECT and then released **only after the transaction completes**. This is how the system can guarantee that the values you read will not change during the transaction (because they remain locked until the transaction finishes).

Share Improve this answer Follow

answered Nov 5 '14 at 20:27

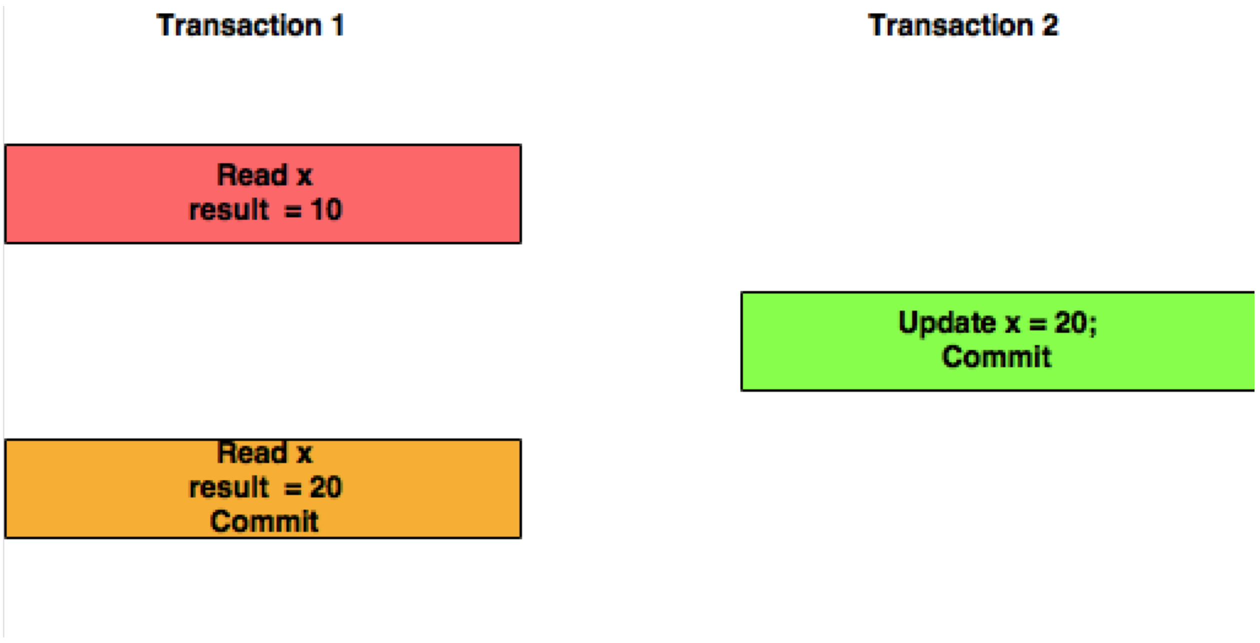
Chris Gillum

12.9k 4 40 49

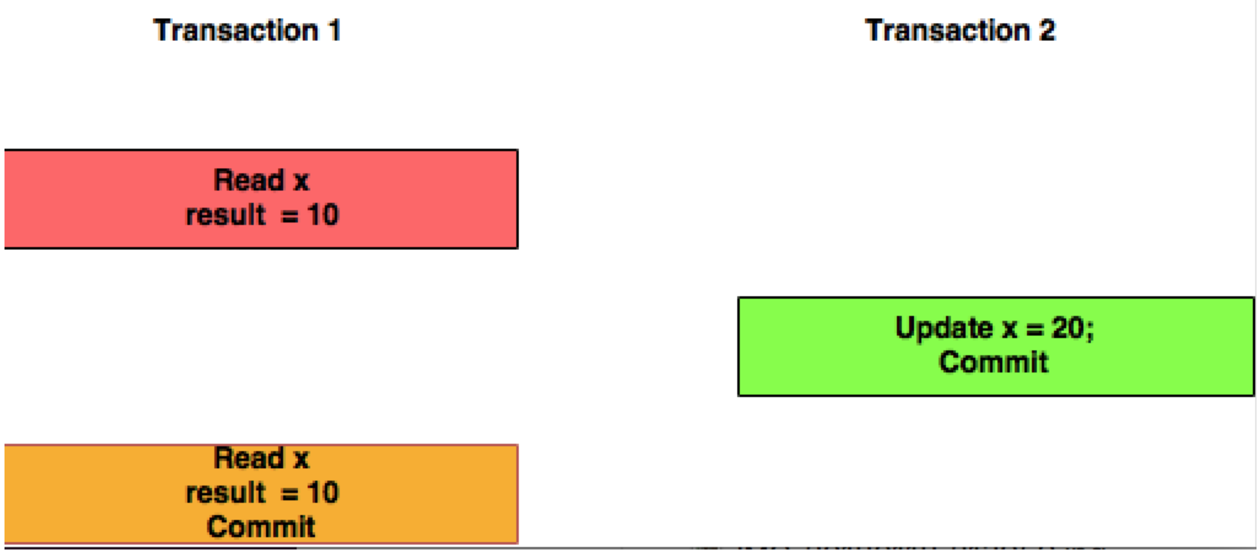
17

Trying to explain this doubt with simple diagrams.

Read Committed: Here in this isolation level, Transaction T1 will be reading the updated value of the X committed by Transaction T2.



Repeatable Read: In this isolation level, Transaction T1 will not consider the changes committed by the Transaction T2.



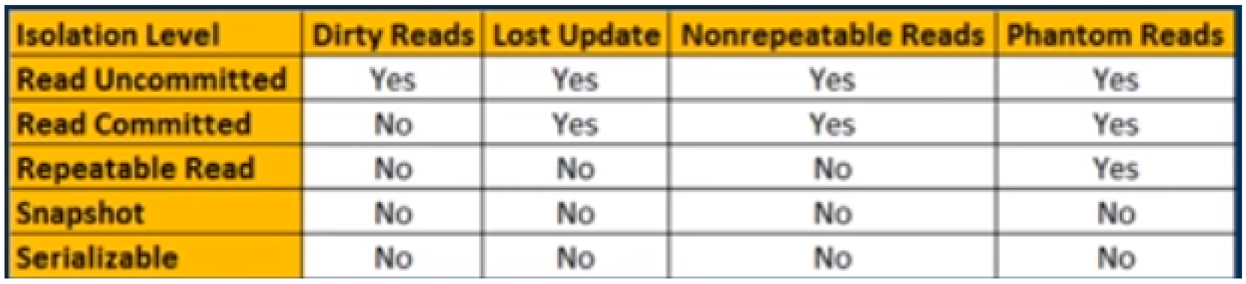
Share Improve this answer Follow

edited Jun 20 '20 at 9:12
Community ♦
1 1

answered Feb 15 '18 at 11:35
vkrishna17
787 8 17

2

I think this picture can also be useful, it helps me as a reference when I want to quickly remember the differences between isolation levels (thanks to kudvenkat on youtube)



Isolation Level	Dirty Reads	Lost Update	Nonrepeatable Reads	Phantom Reads
Read Uncommitted	Yes	Yes	Yes	Yes
Read Committed	No	Yes	Yes	Yes
Repeatable Read	No	No	No	Yes
Snapshot	No	No	No	No
Serializable	No	No	No	No

Share Improve this answer Follow

answered Nov 16 '18 at 10:20
Ivan Pavičić
1,083 2 16 28

0

Please note that, the *repeatable* in repeatable read regards to a tuple, but not to the entire table. In ANSC isolation levels, *phantom read* anomaly can occur, which means read a table with the same where clause twice may return different return different result sets. Literally, it's not *repeatable*.

Share Improve this answer Follow

answered Apr 14 '19 at 12:37
不辞长做岭南人
401 4 7

-1

My observation on initial accepted solution.

Under RR (default mysql) - If a tx is open and a SELECT has been fired, another tx can NOT delete any row belonging to previous READ result set until previous tx is committed (in fact delete statement in the new tx will just hang), however the next tx can delete *all rows* from the table without any trouble. Btw, a next READ in previous tx will still see the old data until it is committed.

Share Improve this answer Follow

answered Mar 2 '16 at 21:04

Sanjeev Dhiman
1,021 1 8 19

2 You might want to put it in the comments section for the answerer to get notified. That way he will be able to respond to your observations and make corrections if required. – RBT Aug 24 '16 at 10:03