Read Isolation, Consistency, and Recency

On this page

Isolation Guarantees

Read Uncommitted

In MongoDB, clients can see the results of writes before the writes are durable:

- Regardless of write concern, other clients using `"local"` (i.e. the default) readConcern can see th result of a write operation before the write operation is acknowledged to the issuing client.
- Clients using `"local"` (i.e. the default) readConcern can read data which may be subsequently ro back.

Read uncommitted is the default isolation level and applies to `mongod` standalone instances as well as t replica sets and sharded clusters.

Read Uncommitted And Single Document Atomicity

Write operations are atomic with respect to a single document; i.e. if a write is updating multiple fields in document, a reader will never see the document with only some of the fields updated.

With a standalone `mongod` instance, a set of read and write operations to a single document is serializat With a replica set, a set of read and write operations to a single document is serializable *only* in the absence of a rollback.

However, although the readers may not see a *partially* updated document, read uncommitted means tha
concurrent readers may still see the updated document before the changes are durable.

Read Uncommitted And Multiple Document Write

When a single write operation modifies multiple documents, the modification of each document is atomi
but the operation as a whole is not atomic and other operations may interleave. However, you can *isola*
single write operation that affects multiple documents using the `$isolated` operator.

Without isolating the multi-document write operations, MongoDB exhibits the following behavior:

1. Non-point-in-time read operations. Suppose a read operation begins at time t_1 and starts reading documents. A write operation then commits an update to one of the documents at some later time The reader may see the updated version of the document, and therefore does not see a point-in-ti snapshot of the data.
2. Non-serializable operations. Suppose a read operation reads a document d_1 at time t_1 and a write operation updates d_1 at some later time t_3. This introduces a read-write dependency such that, if tl operations were to be serialized, the read operation must precede the write operation. But also suppose that the write operation updates document d_2 at time t_2 and the read operation subseque reads d_2 at some later time t_4. This introduces a write-read dependency which would instead requi the read operation to come *after* the write operation in a serializable schedule. There is a depende cycle which makes serializability impossible.
3. Reads may miss matching documents that are updated during the course of the read operation.

Using the `$isolated` operator, a write operation that affects multiple documents can prevent other processes from interleaving once the write operation modifies the first document. This ensures that no c sees the changes until the write operation completes or errors out.

`$isolated` does **not** work with sharded clusters.

An isolated write operation does not provide "all-or-nothing" atomicity. That is, an error during the write operation does not roll back all its changes that preceded the error.

NOTE

`$isolated` operator causes write operations to acquire an exclusive lock on the collection, *even for document-level locking storage engines* such as WiredTiger. That is, `$isolated` operator will make WiredTiger single-threaded for the duration of the operation.

SEE ALSO

Atomicity and Transactions