Atomicity and Transactions

On this page

In MongoDB, a write operation is atomic on the level of a single document, even if the operation modifie multiple embedded documents *within* a single document.

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 *isolat* single write operation that affects multiple documents using the `$isolated` operator.

`$isolated` Operator

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.

The `$isolated` operator does **not** work on sharded clusters.

For an example of an update operation that uses the `$isolated` operator, see `$isolated`. For an exam of a remove operation that uses the `$isolated` operator, see Isolate Remove Operations.

Transaction-Like Semantics

Since a single document can contain multiple embedded documents, single-document atomicity is suffic for many practical use cases. For cases where a sequence of write operations must operate as if in a si transaction, you can implement a two-phase commit in your application.

However, two-phase commits can only offer transaction-*like* semantics. Using two-phase commit ensure data consistency, but it is possible for applications to return intermediate data during the two-phase com or rollback.

For more information on two-phase commit and rollback, see Perform Two Phase Commits.

Concurrency Control

Concurrency control allows multiple applications to run concurrently without causing data inconsistency conflicts.

One approach is to create a unique index on a field that can only have unique values. This prevents insertions or updates from creating duplicate data. Create a unique index on multiple fields to force uniqueness on that combination of field values. For examples of use cases, see update() and Unique Index and findAndModify() and Unique Index.

Another approach is to specify the expected current value of a field in the query predicate for the write operations. The two-phase commit pattern provides a variation where the query predicate includes theapplication identifier as well as the expected state of the data in the write operation.

SEE ALSO

Read Isolation, Consistency, and Recency