To retrieve documents in reverse insertion order, issue `find()` along with the `sort()` method with the `$natural` parameter set to `-1`, as shown in the following example:

```
db.cappedCollection.find().sort( { $natural: -1 } )
```

Check if a Collection is Capped Use the `isCapped()` method to determine if a collection is capped, as follows:

```
db.collection.isCapped()
```

Convert a Collection to Capped You can convert a non-capped collection to a capped collection with the `convertToCapped` command:

```
db.runCommand({"convertToCapped": "mycoll", size: 100000});
```

The `size` parameter specifies the size of the capped collection in bytes.

Warning: This command obtains a global write lock and will block other operations until it has completed.

Changed in version 2.2: Before 2.2, capped collections did not have an index on `_id` unless you specified `autoIndexId` to the `create`, after 2.2 this became the default.

Automatically Remove Data After a Specified Period of Time For additional flexibility when expiring data, consider MongoDB's *TTL* indexes, as described in *Expire Data from Collections by Setting TTL* (page 211). These indexes allow you to expire and remove data from normal collections using a special type, based on the value of a date-typed field and a TTL value for the index.

TTL Collections (page 211) are not compatible with capped collections.

Tailable Cursor You can use a *tailable cursor* with capped collections. Similar to the Unix `tail -f` command, the tailable cursor "tails" the end of a capped collection. As new documents are inserted into the capped collection, you can use the tailable cursor to continue retrieving documents.

See *Create Tailable Cursor* (page 123) for information on creating a tailable cursor.

Expire Data from Collections by Setting TTL

New in version 2.2.

This document provides an introduction to MongoDB's "*time to live*" or *TTL* collection feature. TTL collections make it possible to store data in MongoDB and have the `mongod` automatically remove data after a specified number of seconds or at a specific clock time.

Data expiration is useful for some classes of information, including machine generated event data, logs, and session information that only need to persist for a limited period of time.

A special *TTL index property* (page 503) supports the implementation of TTL collections. The TTL feature relies on a background thread in `mongod` that reads the date-typed values in the index and removes expired *documents* from the collection.