class MongoDB::Collection

Operations on collections in a MongoDB database

Table of Contents

1 Synopsis
2 Readonly attributes
2.1 database
2.2 name
2.3 full-collection-name
2.4 read-concern
3 Methods
3.1 new
3.2 find


```
unit package MongoDB;
class Collection { ... }
```

Synopsis

```
# Initialize
my MongoDB::Client $client .= new(:uri('mongodb://'));
my MongoDB::Database $database = $client.database('contacts');
my MongoDB::Collection $collection = $database.collection('perl_users');

# Find everything
for $collection.find -> BSON::Document $document {
  $document.perl.say;
}

# Or narrow down using conditions.
my MongoDB::Cursor $cursor = $collection.find(
  :$criteria(nick => 'camelia'), $number-to-return(1)
);
$cursor.fetch.perl.say;
```

Class to help accessing and manipulating collections in MongoDB databases.

Readonly attributes

database

```
has DatabaseType $.database;
```

Get the database object of this collection. It is set by MongoDB::Database when a collection object is created.

name

```
has Str $.name;
```

Get the name of the current collection. It is set by MongoDB::Database when a collection object is created.

full-collection-name

```
has Str $.full-collection-name;
```

Get the full representation of this collection. This is a string composed of the database name and collection name separated by a dot. E.g. *person.address* means collection *address* in database *person*.

read-concern

```
has BSON::Document $.read-concern;
```

The read-concern is a structure to have some control over the read operations to which server the operations are directed to. Default is taken from the database. The structure will be explained elsewhere.

Methods

new

```
submethod BUILD (
  DatabaseType:D :$database, Str:D :$name, BSON::Document :$read-concern
)
```

Example

```
my MongoDB::Database $database .= new( :$client, :name<contacts>);
my MongoDB::Collection $collection .= new( :$database, :name<perl_users>);
```

Creates a new Collection object. However, it is better to call collection on the database or client object as shown here;

```
my MongoDB::Database $database = $client.database('contacts');
my MongoDB::Collection $collection = $database.collection('perl_users');
```

or

```
my MongoDB::Collection $collection = $client.collection('contacts.perl_users');
my MongoDB::Database $database = $collection.database;
```

find

```
multi method find (
  List :$criteria where all(@$criteria) ~~ Pair = (),
  List :$projection where all(@$projection) ~~ Pair = (),
  Int :$number-to-skip = 0, Int :$number-to-return = 0,
  QueryFindFlags :@flags = Array[QueryFindFlags].new,
  List :$read-concern
  --> MongoDB::Cursor
)
```

Call method find using lists of pairs. An example;

```
my MongoDB::Cursor $c = $collection.find(
  :criteria(nick => 'MARTIMM',), :projection(_id => 0,)
);
```

Mind the comma's! When only one pair is entered in the list, it is coerced to a pair instead of a list of pairs.

```
multi method find (
  BSON::Document :$criteria = BSON::Document.new,
  BSON::Document :$projection?,
  Int :$number-to-skip = 0, Int :$number-to-return = 0,
  QueryFindFlags :@flags = Array[QueryFindFlags].new,
  BSON::Document :$read-concern
  --> MongoDB::Cursor
)
```

Call method find using the BSON::Document class.

```
my MongoDB::Cursor $c = $collection.find(
  :criteria( BSON::Document.new( nick => 'MARTIMM',)),
  :projection( BSON::Document.new( _id => 0,))
);
```

Find documents in the database. When $criteria is not provided all documents are returned. There are 2 options and some flags to affect the search. $projection is used to select the fields to be returned. It looks like **(field => 1,)** or **(field => 0,)**. When 1 the field is included, when 0 it will be excluded. The _id field is always included unless explicitly excluded like **(_id => 0,)**. The method returns a MongoDB::Cursor.

- **:number-to-skip**. Sets the number of documents to omit - starting from the first document in the resulting dataset - when returning the result of the query.
- **:number-to-return**. Limits the number of documents in the first OP_REPLY message to the query. However, the database will still establish a cursor and return the cursorID to the client if there are more results than number-to-return. If number-to-return is 0, the db will use the default return size. If the number is negative, then the database will return that number and close the cursor. No further results for that query can be fetched. If number-to-return is 1 the server will treat it as -1 (closing the cursor automatically).
- **:flags**. This is an array variable which is filled with QueryFindFlags values defined in MongoDB. An example;

```
my $c = $collection.find(
  :flags(Array[QueryFindFlags].new(C-QF-SLAVEOK))
);
```

or

```
my QueryFindFlags @flags = C-QF-SLAVEOK, C-QF-TAILABLECURSOR;
my $c = $collection.find(:@flags);
```

First example looks complex but that might change in the future, this is the situation at 2016-11-9 with rakudo version 2016.10-249-gb84158c built on MoarVM version 2016.10-37-gf769569 implementing Perl 6.c.

  - **C-QF-TAILABLECURSOR**: corresponds to TailableCursor. Tailable means cursor is not closed when the last data is retrieved. Rather, the cursor marks the final object's position. You can resume using the cursor later, from where it was located, if more data were received. Like any 'latent cursor', the cursor may become invalid at some point (CursorNotFound) for example if the final object it references were deleted.
  - **C-QF-SLAVEOK**: corresponds to SlaveOk.Allow query of replica slave. Normally these return an error except for namespace 'local'.
  - **C-QF-OPLOGREPLAY**: corresponds to OplogReplay. Internal replication use only - driver should not set.
  - **C-QF-NOCURSORTIMOUT**: corresponds to NoCursorTimeout. The server normally times out idle cursors after an inactivity period (10 minutes) to prevent excess memory use. Set this option to prevent that. When used, the cursor must be removed explicitly using $cursor.kill().
  - **C-QF-AWAITDATA**: corresponds to AwaitData. Use with TailableCursor. If we are at the end of the data, block for a while rather than returning no data. After a timeout period, we do return as normal.

- **C-QF-EXHAUST**: corresponds to Exhaust. Stream the data down full blast in multiple 'more' packages, on the assumption that the client will fully read all data queried. Faster when you are pulling a lot of data and know you want to pull it all down. Note: the client is not allowed to not read all the data unless it closes the connection.
- **C-QF-PORTAIL**: corresponds to Partial. Get partial results from a mongos if some shards are down (instead of throwing an error)