Collations

On this page

Overview

New in version 3.4.

Collations provide a set of rules which comply with the conventions of a particular language when comparing strings.

For example, in Canadian French, the last accent in a given word determines the sorting order.

Consider the following French words:

```
cote < coté < côte < côté
```

The sort order using the Canadian French collation would result in the following:

```
cote < côte < coté < côté
```

If collation is unspecified, MongoDB uses the simple binary comparison for strings. As such, the sort order of the words would be:

```
cote < coté < côte < côté
```

Usage

You can specify a default collation for collections and indexes when they are created, or specify a collation for CRUD operations and aggregations. For operations that support collation, MongoDB uses the collection's default collation unless the operation specifies a different collation.

Collation Parameters

```
'collation' => {
   'locale' => <string>,
   'caseLevel' => <bool>,
   'caseFirst' => <string>,
   'strength' => <int>,
   'numericOrdering' => <bool>,
   'alternate' => <string>,
   'maxVariable' => <string>,
   'normalization' => <bool>,
   'backwards' => <bool>
}
```

The only required parameter is `locale`, which the server parses as an ICU format locale ID. For example, set `locale` to `en_US` to represent US English or `fr_CA` to represent Canadian French.

For a complete description of the available parameters, see the MongoDB manual entry.

Assign a Default Collation to a Collection

The following example creates a new collection called `contacts` on the `test` database and assigns a default collation with the `fr_CA` locale. Specifying a collation when you create the collection ensures that all operations involving a query that are run against the `contacts` collection use the `fr_CA` collation, unless the query specifies another collation. Any indexes on the new collection also inherit the default collation, unless the creation command specifies another collation.

```
client = Mongo::Client.new([ "127.0.0.1:27017" ], :database => "test")
client[:contacts, { "collation" => { "locale" => "fr_CA" } } ].create
```

Assign a Collation to an Index

To specify a collation for an index, use the `collation` option when you create the index.

The following example creates an index on the `name` field of the `address_book` collection, with the `unique` parameter enabled and a default collation with `locale` set to `en_US`.

```
client = Mongo::Client.new([ "127.0.0.1:27017" ], :database => "test")
client[:address_book].indexes.create_one( { "first_name" => 1 },
                                          "unique" => true,
                                          "collation" => { "locale" => "en_US" }
                                         )
```

To use this index, make sure your queries also specify the same collation. The following query uses the above index:

```
client[:address_book].find({"first_name" : "Adam" },
                           "collation" => { "locale" => "en_US" })
```

The following queries do **NOT** use the index. The first query uses no collation, and the second uses a collation with a different `strength` value than the collation on the index.

```
client[:address_book].find({"first_name" : "Adam" })

client[:address_book].find({"first_name" : "Adam" },
                           "collation" => { "locale" => "en_US", "strength" => 2 })
```

Operations that Support Collation

All reading, updating, and deleting methods support collation. Some examples are listed below.

`find()` and `sort()`

Individual queries can specify a collation to use when matching and sorting results. The following query and sort operation uses a German collation with the `locale` parameter set to `de`.

```
client = Mongo::Client.new([ "127.0.0.1:27017" ], :database => "test")
docs = client[:contacts].find({ "city" => "New York" },
  { "collation" => { "locale" => "de" } }).sort( "name" => 1 )
```

`find_one_and_update()`

A collection called `names` contains the following documents:

```
{ "_id" : 1, "first_name" : "Hans" }
{ "_id" : 2, "first_name" : "Gunter" }
{ "_id" : 3, "first_name" : "Günter" }
{ "_id" : 4, "first_name" : "Jürgen" }
```

The following `find_one_and_update` operation on the collection does not specify a collation.

```
client = Mongo::Client.new([ "127.0.0.1:27017" ], :database => "test")
doc = client[:names].find_one_and_update( {"first_name" => { "$lt" => "Gunter" }},
  { "$set" => { "verified" => true } })
```

Because `Gunter` is lexically first in the collection, the above operation returns no results and updates no documents.

Consider the same `find_one_and_update` operation but with the collation specified. The locale is set to `de@collation=phonebook`.

> **NOTE:**
> Some locales have a `collation=phonebook` option available for use with languages which sort proper nouns differently from other words. According to the `de@collation=phonebook` collation, characters with umlauts come before the same characters without umlauts.

```
client = Mongo::Client.new([ "127.0.0.1:27017" ], :database => "test")
doc = client[:names].find_one_and_update( { "first_name" => { "$lt" => "Gunter" } },
  { "$set" => { "verified" => true } }, { "collation" => { "locale" => "de@collation=phonebool
      :return_document => :after } )
```

The operation returns the following updated document:

```
{ "_id" => 3, "first_name" => "Günter", "verified" => true }
```

`find_one_and_delete()`

Set the `numericOrdering` collation parameter to `true` to compare numeric string by their numeric values.

The collection `numbers` contains the following documents:

```
{ "_id" : 1, "a" : "16" }
{ "_id" : 2, "a" : "84" }
{ "_id" : 3, "a" : "179" }
```

The following example matches the first document in which field `a` has a numeric value greater than 100 and deletes it.

```
docs = numbers.find_one_and_delete({ "a" => { "$gt" => "100" } },
  { "collation" => { "locale" => "en", "numericOrdering" => true  } })
```

After the above operation, the following documents remain in the collection:

```
{ "_id" : 1, "a" : "16" }
{ "_id" : 2, "a" : "84" }
```

If you perform the same operation without collation, the server deletes the first document it finds in which the lexical value of `a` is greater than `"100"`.

```
numbers = client[:numbers]
docs = numbers.find_one_and_delete({ "a" => { "$gt" => "100" } })
```

After the above operation the document in which `a` was equal to `"16"` has been deleted, and the following documents remain in the collection:

```
{ "_id" : 2, "a" : "84" }
{ "_id" : 3, "a" : "179" }
```

`delete_many()`

You can use collations with all the various bulk operations which exist in the Ruby driver.

The collection `recipes` contains the following documents:

```
{ "_id" : 1, "dish" : "veggie empanadas", "cuisine" : "Spanish" }
{ "_id" : 2, "dish" : "beef bourgignon", "cuisine" : "French" }
{ "_id" : 3, "dish" : "chicken molé", "cuisine" : "Mexican" }
{ "_id" : 4, "dish" : "chicken paillard", "cuisine" : "french" }
{ "_id" : 5, "dish" : "pozole verde", "cuisine" : "Mexican" }
```

Setting the `strength` parameter of the collation document to `1` or `2` causes the server to disregard case in the query filter. The following example uses a case-insensitive query filter to delete all records in which the `cuisine` field matches `French`.

```
client = Mongo::Client.new([ "127.0.0.1:27017" ], :database => "test")
recipes = client[:recipes]
docs = recipes.delete_many({ "cuisine" => "French" },
                           "collation" => { "locale" => "en_US", "strength" => 1 })
```

After the above operation runs, the documents with `_id` values of 2 and 4 are deleted from the collection.

Aggregation

To use collation with an aggregation operation, specify a collation in the aggregation options.

The following aggregation example uses a collection called `names` and groups the `first_name` field together, counts the total number of results in each group, and sorts the results by German phonebook order.

```
aggregation = names.aggregate(
  [
    {
      "$group" => { "_id" => "$first_name", "name_count" => { "$sum" => 1 } }
    },
    {
        "$sort" => { "_id" => 1 }
    },

  ], { "collation" => { "locale" => "de@collation=phonebook" } }
)

aggregation.each do |doc|
  #=> Yields a BSON::Document.
end
```