MongoDB PHP Library: CRUD Operations

CRUD operations *create*, *read*, *update*, and *delete* documents. The MongoDB PHP Library's `MongoDB\Collection` class implements MongoDB's cross-driver CRUD specification, providing access to methods for inserting, finding, updating, and deleting documents in MongoDB.

This document provides a general introduction to inserting, querying, updating, and deleting documents using the MongoDB PHP Library. The MongoDB Manual's CRUD Section provides a more thorough introduction to CRUD operations with MongoDB.

Insert Documents

Insert One Document

The `MongoDB\Collection::insertOne()` method inserts a single document into MongoDB and returns an instance of `MongoDB\InsertOneResult`, which you can use to access the ID of the inserted document.

The following operation inserts a document into the `users` collection in the `test` database:

```
<?php

$collection = (new MongoDB\Client)->test->users;

$insertOneResult = $collection->insertOne([
    'username' => 'admin',
    'email' => 'admin@example.com',
    'name' => 'Admin User',
]);

printf("Inserted %d document(s)\n", $insertOneResult->getInsertedCount());

var_dump($insertOneResult->getInsertedId());
```

The output would then resemble:

```
Inserted 1 document(s)
object(MongoDB\BSON\ObjectId)#11 (1) {
  ["oid"]=>
  string(24) "579a25921f417dd1e5518141"
}
```

The output includes the ID of the inserted document.

If you include an `_id` value when inserting a document, MongoDB checks to ensure that the `_id` value is unique for the collection. If the `_id` value is not unique, the insert operation fails due to a duplicate key error.

The following example inserts a document while specifying the value for the `_id`:

```
<?php

$collection = (new MongoDB\Client)->test->users;

$insertOneResult = $collection->insertOne(['_id' => 1, 'name' => 'Alice']);

printf("Inserted %d document(s)\n", $insertOneResult->getInsertedCount());

var_dump($insertOneResult->getInsertedId());
```

The output would then resemble:

```
Inserted 1 document(s)
int(1)
```

See also

`MongoDB\Collection::insertOne()`

Insert Many Documents

The `MongoDB\Collection::insertMany()` method allows you to insert multiple documents in one write operation and returns an instance of `MongoDB\InsertManyResult`, which you can use to access the IDs of the inserted documents.

The following operation inserts two documents into the `users` collection in the `test` database:

```
<?php

$collection = (new MongoDB\Client)->test->users;

$insertManyResult = $collection->insertMany([
    [
        'username' => 'admin',
        'email' => 'admin@example.com',
        'name' => 'Admin User',
    ],
    [
        'username' => 'test',
        'email' => 'test@example.com',
        'name' => 'Test User',
    ],
]);

printf("Inserted %d document(s)\n", $insertManyResult->getInsertedCount());

var_dump($insertManyResult->getInsertedIds());
```

The output would then resemble:

```
Inserted 2 document(s)
array(2) {
  [0]=>
  object(MongoDB\BSON\ObjectId)#11 (1) {
    ["oid"]=>
    string(24) "579a25921f417dd1e5518141"
  }
  [1]=>
  object(MongoDB\BSON\ObjectId)#12 (1) {
    ["oid"]=>
    string(24) "579a25921f417dd1e5518142"
  }
}
```

See also

`MongoDB\Collection::insertMany()`

Query Documents

The MongoDB PHP Library provides the `MongoDB\Collection::findOne()` and `MongoDB\Collection::find()` methods for querying documents and the `MongoDB\Collection::aggregate()` method for performing aggregation operations.

When evaluating query criteria, MongoDB compares types and values according to its own comparison rules for BSON types, which differs from PHP's comparison and type juggling rules. When matching a special BSON type the query criteria should use the respective BSON class in the driver (e.g. use MongoDB\BSON\ObjectId to match an ObjectId).

Find One Document

`MongoDB\Collection::findOne()` returns the first document that matches the query or `null` if no document matches the query.

The following example searches for the document with `_id` of `"94301"`:

```
<?php

$collection = (new MongoDB\Client)->test->zips;

$document = $collection->findOne(['_id' => '94301']);

var_dump($document);
```

The output would then resemble:

```
object(MongoDB\Model\BSONDocument)#13 (1) {
  ["storage":"ArrayObject":private]=>
  array(5) {
    ["_id"]=>
```

```
    string(5) "94301"
    ["city"]=>
    string(9) "PALO ALTO"
    ["loc"]=>
    object(MongoDB\Model\BSONArray)#12 (1) {
      ["storage":"ArrayObject":private]=>
      array(2) {
        [0]=>
        float(-122.149685)
        [1]=>
        float(37.444324)
      }
    }
    ["pop"]=>
    int(15965)
    ["state"]=>
    string(2) "CA"
  }
}
```

Note

The criteria in this example matched an `_id` with a string value of `"94301"`. The same criteria would not have matched a document with an integer value of `94301` due to MongoDB's comparison rules for BSON types. Similarly, users should use a MongoDB\BSON\ObjectId object when matching an `_id` with an ObjectId value, as strings and ObjectIds are not directly comparable.

See also

`MongoDB\Collection::findOne()`

Find Many Documents

`MongoDB\Collection::find()` returns a MongoDB\Driver\Cursor object, which you can iterate upon to access all matched documents.

The following example lists the documents in the `zips` collection with the specified city and state values:

```
<?php

$collection = (new MongoDB\Client)->test->zips;

$cursor = $collection->find(['city' => 'JERSEY CITY', 'state' => 'NJ']);

foreach ($cursor as $document) {
    echo $document['_id'], "\n";
}
```

The output would resemble:

```
07302
07304
07305
07306
07307
07310
```

See also

`MongoDB\Collection::find()`

Query Projection

By default, queries in MongoDB return all fields in matching documents. To limit the amount of data that MongoDB sends to applications, you can include a projection document in the query operation.

Note

MongoDB includes the `_id` field by default unless you explicitly exclude it in a projection document.

The following example finds restaurants based on the `cuisine` and `borough` fields and uses a projection to limit the fields that are returned. It also limits the results to 5 documents.

```
<?php

$collection = (new MongoDB\Client)->test->restaurants;

$cursor = $collection->find(
    [
        'cuisine' => 'Italian',
        'borough' => 'Manhattan',
    ],
    [
        'projection' => [
            'name' => 1,
            'borough' => 1,
            'cuisine' => 1,
        ],
        'limit' => 4,
    ]
);

foreach($cursor as $restaurant) {
   var_dump($restaurant);
};
```

The output would then resemble:

```
object(MongoDB\Model\BSONDocument)#10 (1) {
  ["storage":"ArrayObject":private]=>
```

```
  array(4) {
    ["_id"]=>
    object(MongoDB\BSON\ObjectId)#8 (1) {
      ["oid"]=>
      string(24) "576023c6b02fa9281da3f983"
    }
    ["borough"]=>
    string(9) "Manhattan"
    ["cuisine"]=>
    string(7) "Italian"
    ["name"]=>
    string(23) "Isle Of Capri Resturant"
  }
}
object(MongoDB\Model\BSONDocument)#13 (1) {
  ["storage":"ArrayObject":private]=>
  array(4) {
    ["_id"]=>
    object(MongoDB\BSON\ObjectId)#12 (1) {
      ["oid"]=>
      string(24) "576023c6b02fa9281da3f98d"
    }
    ["borough"]=>
    string(9) "Manhattan"
    ["cuisine"]=>
    string(7) "Italian"
    ["name"]=>
    string(18) "Marchis Restaurant"
  }
}
object(MongoDB\Model\BSONDocument)#8 (1) {
  ["storage":"ArrayObject":private]=>
  array(4) {
    ["_id"]=>
    object(MongoDB\BSON\ObjectId)#10 (1) {
      ["oid"]=>
      string(24) "576023c6b02fa9281da3f99b"
    }
    ["borough"]=>
    string(9) "Manhattan"
    ["cuisine"]=>
    string(7) "Italian"
    ["name"]=>
    string(19) "Forlinis Restaurant"
  }
}
object(MongoDB\Model\BSONDocument)#12 (1) {
  ["storage":"ArrayObject":private]=>
  array(4) {
    ["_id"]=>
    object(MongoDB\BSON\ObjectId)#13 (1) {
      ["oid"]=>
      string(24) "576023c6b02fa9281da3f9a8"
    }
    ["borough"]=>
    string(9) "Manhattan"
    ["cuisine"]=>
```

```
    string(7) "Italian"
    ["name"]=>
    string(22) "Angelo Of Mulberry St."
  }
}
```

Limit, Sort, and Skip Options

In addition to projection criteria, you can specify options to limit, sort, and skip documents during queries.

The following example uses the `limit` and `sort` options to query for the five most populous zip codes in the United States:

```
<?php

$collection = (new MongoDB\Client)->test->zips;

$cursor = $collection->find(
    [],
    [
        'limit' => 5,
        'sort' => ['pop' => -1],
    ]
);

foreach ($cursor as $document) {
    printf("%s: %s, %s\n", $document['_id'], $document['city'],
$document['state']);
}
```

The output would then resemble:

```
60623: CHICAGO, IL
11226: BROOKLYN, NY
10021: NEW YORK, NY
10025: NEW YORK, NY
90201: BELL GARDENS, CA
```

Regular Expressions

Filter criteria may include regular expressions, either by using the MongoDB\BSON\Regex class directory or the `$regex` operator.

The following example lists documents in the `zips` collection where the city name starts with "garden" and the state is Texas:

```
<?php

$collection = (new MongoDB\Client)->test->zips;

$cursor = $collection->find([
```

```
    'city' => new MongoDB\BSON\Regex('^garden', 'i'),
    'state' => 'TX',
]);

foreach ($cursor as $document) {
   printf("%s: %s, %s\n", $document['_id'], $document['city'],
$document['state']);
}
```

The output would then resemble:

```
78266: GARDEN RIDGE, TX
79739: GARDEN CITY, TX
79758: GARDENDALE, TX
```

An equivalent filter could be constructed using the `$regex` operator:

```
[
    'city' => ['$regex' => '^garden', '$options' => 'i'],
    'state' => 'TX',
]
```

See also

$regex in the MongoDB manual

Although MongoDB's regular expression syntax is not exactly the same as PHP's PCRE syntax, preg_quote() may be used to escape special characters that should be matched as-is. The following example finds restaurants whose name starts with "(Library)":

```
<?php

$collection = (new MongoDB\Client)->test->restaurants;

$cursor = $collection->find([
    'name' => new MongoDB\BSON\Regex('^' . preg_quote('(Library)')),
]);
```

Complex Queries with Aggregation

MongoDB's Aggregation Framework allows you to issue complex queries that filter, transform, and group collection data. The MongoDB PHP Library's `MongoDB\Collection::aggregate()` method returns a Traversable object, which you can iterate upon to access the results of the aggregation operation. Refer to the `MongoDB\Collection::aggregate()` method's behavior reference for more about the method's output.

The following example lists the 5 US states with the most zip codes associated with them:

```
<?php

$collection = (new MongoDB\Client)->test->zips;
```

```
$cursor = $collection->aggregate([
    ['$group' => ['_id' => '$state', 'count' => ['$sum' => 1]]],
    ['$sort' => ['count' => -1]],
    ['$limit' => 5],
]);

foreach ($cursor as $state) {
    printf("%s has %d zip codes\n", $state['_id'], $state['count']);
}
```

The output would then resemble:

```
TX has 1671 zip codes
NY has 1595 zip codes
CA has 1516 zip codes
PA has 1458 zip codes
IL has 1237 zip codes
```

See also

MongoDB\Collection::aggregate()

Update Documents

Update One Document

Use the MongoDB\Collection::updateOne() method to update a single document matching a filter. MongoDB\Collection::updateOne() returns a MongoDB\UpdateResult object, which you can use to access statistics about the update operation.

Update methods have two required parameters: the query filter that identifies the document or documents to update, and an update document that specifies what updates to perform. The MongoDB\Collection::updateOne() reference describes each parameter in detail.

The following example inserts two documents into an empty `users` collection in the `test` database using the MongoDB\Collection::insertOne() method, and then updates the documents where the value for the `state` field is `"ny"` to include a `country` field set to `"us"`:

```
<?php

$collection = (new MongoDB\Client)->test->users;
$collection->drop();

$collection->insertOne(['name' => 'Bob', 'state' => 'ny']);
$collection->insertOne(['name' => 'Alice', 'state' => 'ny']);
$updateResult = $collection->updateOne(
    ['state' => 'ny'],
    ['$set' => ['country' => 'us']]
);
```

```
printf("Matched %d document(s)\n", $updateResult->getMatchedCount());
printf("Modified %d document(s)\n", $updateResult->getModifiedCount());
```

Since the update operation uses the `MongoDB\Collection::updateOne()` method, which updates the first document to match the filter criteria, the results would then resemble:

```
Matched 1 document(s)
Modified 1 document(s)
```

It is possible for a document to match the filter but *not be modified* by an update, as is the case where the update sets a field's value to its existing value, as in this example:

```
<?php

$collection = (new MongoDB\Client)->test->users;
$collection->drop();

$collection->insertOne(['name' => 'Bob', 'state' => 'ny']);
$updateResult = $collection->updateOne(
    ['name' => 'Bob'],
    ['$set' => ['state' => 'ny']]
);

printf("Matched %d document(s)\n", $updateResult->getMatchedCount());
printf("Modified %d document(s)\n", $updateResult->getModifiedCount());
```

The number of matched documents and the number of *modified* documents would therefore not be equal, and the output from the operation would resemble:

```
Matched 1 document(s)
Modified 0 document(s)
```

See also

- `MongoDB\Collection::updateOne()`
- `MongoDB\Collection::findOneAndUpdate()`

Update Many Documents

`MongoDB\Collection::updateMany()` updates one or more documents matching the filter criteria and returns a `MongoDB\UpdateResult` object, which you can use to access statistics about the update operation.

Update methods have two required parameters: the query filter that identifies the document or documents to update, and an update document that specifies what updates to perform. The `MongoDB\Collection::updateMany()` reference describes each parameter in detail.

The following example inserts three documents into an empty `users` collection in the `test` database and then uses the `$set` operator to update the documents matching the filter criteria to include the `country` field with value `"us"`:

```
<?php

$collection = (new MongoDB\Client)->test->users;
$collection->drop();

$collection->insertOne(['name' => 'Bob', 'state' => 'ny', 'country' =>
'us']);
$collection->insertOne(['name' => 'Alice', 'state' => 'ny']);
$collection->insertOne(['name' => 'Sam', 'state' => 'ny']);
$updateResult = $collection->updateMany(
    ['state' => 'ny'],
    ['$set' => ['country' => 'us']]
);

printf("Matched %d document(s)\n", $updateResult->getMatchedCount());
printf("Modified %d document(s)\n", $updateResult->getModifiedCount());
```

If an update operation results in no change to a document, such as setting the value of the field to its current value, the number of modified documents can be less than the number of *matched* documents. Since the update document with `name` of `"Bob"` results in no changes to the document, the output of the operation therefore resembles:

```
Matched 3 document(s)
Modified 2 document(s)
```

See also

`MongoDB\Collection::updateMany()`

Replace Documents

Replacement operations are similar to update operations, but instead of updating a document to include new fields or new field values, a replacement operation replaces the entire document with a new document, but retains the original document's `_id` value.

The `MongoDB\Collection::replaceOne()` method replaces a single document that matches the filter criteria and returns an instance of `MongoDB\UpdateResult`, which you can use to access statistics about the replacement operation.

`MongoDB\Collection::replaceOne()` has two required parameters: the query filter that identifies the document or documents to replace, and a replacement document that will replace the original document in MongoDB. The `MongoDB\Collection::replaceOne()` reference describes each parameter in detail.

Important

Replacement operations replace all of the fields in a document except the `_id` value. To avoid accidentally overwriting or deleting desired fields, use the `MongoDB\Collection::updateOne()` or `MongoDB\Collection::updateMany()` methods to update individual fields in a document rather than replacing the entire document.

The following example inserts one document into an empty `users` collection in the `test` database, and then replaces that document with a new one:

```
<?php

$collection = (new MongoDB\Client)->test->users;
$collection->drop();

$collection->insertOne(['name' => 'Bob', 'state' => 'ny']);
$updateResult = $collection->replaceOne(
    ['name' => 'Bob'],
    ['name' => 'Robert', 'state' => 'ca']
);

printf("Matched %d document(s)\n", $updateResult->getMatchedCount());
printf("Modified %d document(s)\n", $updateResult->getModifiedCount());
```

The output would then resemble:

```
Matched 1 document(s)
Modified 1 document(s)
```

See also

- `MongoDB\Collection::replaceOne()`
- `MongoDB\Collection::findOneAndReplace()`

Upsert

Update and replace operations support an upsert option. When `upsert` is `true` *and* no documents match the specified filter, the operation creates a new document and inserts it. If there *are* matching documents, then the operation modifies or replaces the matching document or documents.

When a document is upserted, the ID is accessible via `MongoDB\UpdateResult::getUpsertedId()`.

The following example uses `MongoDB\Collection::updateOne()` with the `upsert` option set to `true` and an empty `users` collection in the `test` database, therefore inserting the document into the database:

```
<?php

$collection = (new MongoDB\Client)->test->users;
```

```
$collection->drop();

$updateResult = $collection->updateOne(
    ['name' => 'Bob'],
    ['$set' => ['state' => 'ny']],
    ['upsert' => true]
);

printf("Matched %d document(s)\n", $updateResult->getMatchedCount());
printf("Modified %d document(s)\n", $updateResult->getModifiedCount());
printf("Upserted %d document(s)\n", $updateResult->getUpsertedCount());

$upsertedDocument = $collection->findOne([
    '_id' => $updateResult->getUpsertedId(),
]);

var_dump($upsertedDocument);
```

The output would then resemble:

```
Matched 0 document(s)
Modified 0 document(s)
Upserted 1 document(s)
object(MongoDB\Model\BSONDocument)#16 (1) {
  ["storage":"ArrayObject":private]=>
  array(3) {
    ["_id"]=>
    object(MongoDB\BSON\ObjectId)#15 (1) {
      ["oid"]=>
      string(24) "57509c4406d7241dad86e7c3"
    }
    ["name"]=>
    string(3) "Bob"
    ["state"]=>
    string(2) "ny"
  }
}
```

Delete Documents

Delete One Document

The `MongoDB\Collection::deleteOne()` method deletes a single document that matches the filter criteria and returns a `MongoDB\DeleteResult`, which you can use to access statistics about the delete operation.

If multiple documents match the filter criteria, `MongoDB\Collection::deleteOne()` deletes the first matching document.

`MongoDB\Collection::deleteOne()` has one required parameter: a query filter that specifies the document to delete. Refer to the `MongoDB\Collection::deleteOne()` reference for full method documentation.

The following operation deletes the first document where the `state` field's value is `"ny"`:

```
<?php

$collection = (new MongoDB\Client)->test->users;
$collection->drop();

$collection->insertOne(['name' => 'Bob', 'state' => 'ny']);
$collection->insertOne(['name' => 'Alice', 'state' => 'ny']);
$deleteResult = $collection->deleteOne(['state' => 'ny']);

printf("Deleted %d document(s)\n", $deleteResult->getDeletedCount());
```

The output would then resemble:

```
Deleted 1 document(s)
```

See also

MongoDB\Collection::deleteOne()

Delete Many Documents

MongoDB\Collection::deleteMany() deletes all of the documents that match the filter criteria and returns a MongoDB\DeleteResult, which you can use to access statistics about the delete operation.

MongoDB\Collection::deleteMany() has one required parameter: a query filter that specifies the document to delete. Refer to the MongoDB\Collection::deleteMany() reference for full method documentation.

The following operation deletes all of the documents where the `state` field's value is `"ny"`:

```
<?php

$collection = (new MongoDB\Client)->test->users;
$collection->drop();

$collection->insertOne(['name' => 'Bob', 'state' => 'ny']);
$collection->insertOne(['name' => 'Alice', 'state' => 'ny']);
$deleteResult = $collection->deleteMany(['state' => 'ny']);

printf("Deleted %d document(s)\n", $deleteResult->getDeletedCount());
```

The output would then resemble:

```
Deleted 2 document(s)
```