`collection` – Collection level operations

Collection level utilities for Mongo.

`pymongo.ASCENDING = 1`

Ascending sort order.

`pymongo.DESCENDING = -1`

Descending sort order.

`pymongo.GEO2D = '2d'`

Index specifier for a 2-dimensional geospatial index.

`pymongo.GEOSPHERE = '2dsphere'`

Index specifier for a spherical geospatial index.

Added in version 2.5.

`pymongo.HASHED = 'hashed'`

Index specifier for a hashed index.

Added in version 2.5.

`pymongo.TEXT = 'text'`

Index specifier for a text index.

> See also
>
> MongoDB's Atlas Search which offers more advanced text search functionality.

Added in version 2.7.1.

`class pymongo.collection.ReturnDocument`

An enum used with `find_one_and_replace()` and `find_one_and_update()`.

`BEFORE`

Return the original document before it was updated/replaced, or `None` if no document matches the query.

`AFTER`

Return the updated/replaced or inserted document.

`class pymongo.collection.Collection(database, name, create=False, **kwargs)`

Get / create a Mongo collection.

Raises `TypeError` if *name* is not an instance of `str`. Raises `InvalidName` if *name* is not a valid collection name. Any additional keyword arguments will be used as options passed to the create command. See `create_collection()` for valid options.

If *create* is `True`, *collation* is specified, or any additional keyword arguments are present, a `create` command will be sent, using `session` if specified. Otherwise, a `create` command will not be sent and the collection will be created implicitly on first use. The optional `session` argument is *only* used for the `create` command, it is not associated with the collection afterward.

PARAMETERS:

- **database** (*Database[_DocumentType]*) – the database to get a collection from
- **name** (*str*) – the name of the collection to get
- **create** (*Optional[bool]*) – if `True`, force collection creation even without options being set
- **codec_options** (*Optional[CodecOptions[_DocumentTypeArg]]*) – An instance of `CodecOptions`. If `None` (the default) database.codec_options is used.
- **read_preference** (*Optional[_ServerMode]*) – The read preference to use. If `None` (the default) database.read_preference is used.
- **write_concern** (*Optional[WriteConcern]*) – An instance of `WriteConcern`. If `None` (the default) database.write_concern is used.
- **read_concern** (*Optional[ReadConcern]*) – An instance of `ReadConcern`. If `None` (the default) database.read_concern is used.
- **collation** – An instance of `Collation`. If a collation is provided, it will be passed to the create collection command.
- **session** (*Optional[ClientSession]*) – a `ClientSession` that is used with the create collection command
- **kwargs** (*Any*) – additional keyword arguments will be passed as options for the create collection command

Changed in version 4.2: Added the `clusteredIndex` and `encryptedFields` parameters.
Changed in version 4.0: Removed the reindex, map_reduce, inline_map_reduce, parallel_scan, initialize_unordered_bulk_op, initialize_ordered_bulk_op, group, count, insert, save, update, remove, find_and_modify, and ensure_index methods. See the PyMongo 4 Migration Guide.
Changed in version 3.6: Added `session` parameter.
Changed in version 3.4: Support the *collation* option.
Changed in version 3.2: Added the read_concern option.
Changed in version 3.0: Added the codec_options, read_preference, and write_concern options. Removed the uuid_subtype attribute. `Collection` no longer returns an instance

of `Collection` for attribute names with leading underscores. You must use dict-style lookups instead::

```
collection['__my_collection__']
```

Not:

```
collection.__my_collection__
```

See also

The MongoDB documentation on collections.

`c[name] || c.name`

Get the *name* sub-collection of `Collection` *c*.

Raises `InvalidName` if an invalid collection name is used.

`full_name`

The full name of this `Collection`.

The full name is of the form *database_name.collection_name*.

`name`

The name of this `Collection`.

`database`

The `Database` that this `Collection` is a part of.

`codec_options`

Read only access to the `CodecOptions` of this instance.

`read_preference`

Read only access to the read preference of this instance.

Changed in version 3.0: The `read_preference` attribute is now read only.

`write_concern`

Read only access to the `WriteConcern` of this instance.

Changed in version 3.0: The `write_concern` attribute is now read only.

`read_concern`

Read only access to the `ReadConcern` of this instance.

Added in version 3.2.

with_options(codec_options=None, read_preference=None, write_concern=None, read_concern=None)

Get a clone of this collection changing the specified settings.

```
>>> coll1.read_preference
Primary()
>>> from pymongo import ReadPreference
>>> coll2 = coll1.with_options(read_preference=ReadPreference.SECONDARY)
>>> coll1.read_preference
Primary()
>>> coll2.read_preference
Secondary(tag_sets=None)
```

PARAMETERS:

- **codec_options** (*CodecOptions[_DocumentTypeArg] | None*) – An instance of `CodecOptions`. If `None` (the default) the `codec_options` of this `Collection` is used.
- **read_preference** (*_ServerMode | None*) – The read preference to use. If `None` (the default) the `read_preference` of this `Collection` is used. See `read_preferences` for options.
- **write_concern** (*WriteConcern | None*) – An instance of `WriteConcern`. If `None` (the default) the `write_concern` of this `Collection` is used.
- **read_concern** (*ReadConcern | None*) – An instance of `ReadConcern`. If `None` (the default) the `read_concern` of this `Collection` is used.

RETURN TYPE:

Collection[*_DocumentType*]

bulk_write(requests, ordered=True, bypass_document_validation=False, session=None, comment=None, let=None)

Send a batch of write operations to the server.

Requests are passed as a list of write operation instances (`InsertOne`, `UpdateOne`, `UpdateMany`, `ReplaceOne`, `DeleteOne`, or `DeleteMany`).

```
>>> for doc in db.test.find({}):
...     print(doc)
...
{'x': 1, '_id': ObjectId('54f62e60fba5226811f634ef')}
{'x': 1, '_id': ObjectId('54f62e60fba5226811f634f0')}
>>> # DeleteMany, UpdateOne, and UpdateMany are also available.
...
>>> from pymongo import InsertOne, DeleteOne, ReplaceOne
>>> requests = [InsertOne({'y': 1}), DeleteOne({'x': 1}),
...             ReplaceOne({'w': 1}, {'z': 1}, upsert=True)]
>>> result = db.test.bulk_write(requests)
>>> result.inserted_count
1
>>> result.deleted_count
1
>>> result.modified_count
0
>>> result.upserted_ids
{2: ObjectId('54f62ee28891e756a6e1abd5')}
>>> for doc in db.test.find({}):
...     print(doc)
...
{'x': 1, '_id': ObjectId('54f62e60fba5226811f634f0')}
{'y': 1, '_id': ObjectId('54f62ee2fba5226811f634f1')}
{'z': 1, '_id': ObjectId('54f62ee28891e756a6e1abd5')}
```

PARAMETERS:

- **requests** (*Sequence[_WriteOp[_DocumentType]]*) – A list of write operations (see examples above).
- **ordered** (*bool*) – If `True` (the default) requests will be performed on the server serially, in the order provided. If an error occurs all remaining operations are aborted. If `False` requests will be performed on the server in arbitrary order, possibly in parallel, and all operations will be attempted.
- **bypass_document_validation** (*bool*) – (optional) If `True`, allows the write to opt-out of document level validation. Default is `False`.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **let** (*Optional[Mapping]*) – Map of parameter names and values. Values must be constant or closed expressions that do not reference document fields. Parameters can then be accessed as variables in an aggregate expression context (e.g. “$$var”).

RETURNS:

An instance of `BulkWriteResult`.

RETURN TYPE:

BulkWriteResult

See also

Why does PyMongo add an _id field to all of my documents?

Note

bypass_document_validation requires server version **>= 3.2**

Changed in version 4.1: Added `comment` parameter. Added `let` parameter.

Changed in version 3.6: Added `session` parameter.

Changed in version 3.2: Added bypass_document_validation support

Added in version 3.0.

insert_one(document, bypass_document_validation=False, session=None, comment=None)

Insert a single document.

```
>>> db.test.count_documents({'x': 1})
0
>>> result = db.test.insert_one({'x': 1})
>>> result.inserted_id
ObjectId('54f112defba522406c9cc208')
>>> db.test.find_one({'x': 1})
{'x': 1, '_id': ObjectId('54f112defba522406c9cc208')}
```

PARAMETERS:

- **document** (*Union[_DocumentType, RawBSONDocument]*) – The document to insert. Must be a mutable mapping type. If the document does not have an _id field one will be added automatically.
- **bypass_document_validation** (*bool*) – (optional) If `True`, allows the write to opt-out of document level validation. Default is `False`.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.

RETURNS:

- An instance of `InsertOneResult`.

RETURN TYPE:

InsertOneResult

See also

Why does PyMongo add an _id field to all of my documents?

Note

bypass_document_validation requires server version **>= 3.2**

Changed in version 4.1: Added `comment` parameter.

Changed in version 3.6: Added `session` parameter.

Changed in version 3.2: Added bypass_document_validation support

Added in version 3.0.

insert_many(documents, ordered=True, bypass_document_validation=False, session=None, comment=None)

Insert an iterable of documents.

```
>>> db.test.count_documents({})
0
>>> result = db.test.insert_many([{'x': i} for i in range(2)])
>>> result.inserted_ids
[ObjectId('54f113fffba522406c9cc20e'), ObjectId('54f113fffba522406c9cc20f')]
>>> db.test.count_documents({})
2
```

PARAMETERS:

- **documents** (*Iterable[Union[_DocumentType, RawBSONDocument]]*) – A iterable of documents to insert.
- **ordered** (*bool*) – If `True` (the default) documents will be inserted on the server serially, in the order provided. If an error occurs all remaining inserts are aborted. If `False`, documents will be inserted on the server in arbitrary order, possibly in parallel, and all document inserts will be attempted.
- **bypass_document_validation** (*bool*) – (optional) If `True`, allows the write to opt-out of document level validation. Default is `False`.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.

RETURNS:

An instance of `InsertManyResult`.

RETURN TYPE:

InsertManyResult

See also

Why does PyMongo add an _id field to all of my documents?

Note

bypass_document_validation requires server version **>= 3.2**

Changed in version 4.1: Added `comment` parameter.

Changed in version 3.6: Added `session` parameter.

Changed in version 3.2: Added bypass_document_validation support

Added in version 3.0.

replace_one(filter, replacement, upsert=False, bypass_document_validation=False, collation=None, hint=None, session=None, let=None, comment=None)

Replace a single document matching the filter.

```
>>> for doc in db.test.find({}):
...     print(doc)
...
{'x': 1, '_id': ObjectId('54f4c5befba5220aa4d6dee7')}
>>> result = db.test.replace_one({'x': 1}, {'y': 1})
>>> result.matched_count
1
>>> result.modified_count
1
>>> for doc in db.test.find({}):
...     print(doc)
...
{'y': 1, '_id': ObjectId('54f4c5befba5220aa4d6dee7')}
```

The *upsert* option can be used to insert a new document if a matching document does not exist.

```
>>> result = db.test.replace_one({'x': 1}, {'x': 1}, True)
>>> result.matched_count
0
>>> result.modified_count
0
>>> result.upserted_id
ObjectId('54f11e5c8891e756a6e1abd4')
>>> db.test.find_one({'x': 1})
{'x': 1, '_id': ObjectId('54f11e5c8891e756a6e1abd4')}
```

PARAMETERS:

- **filter** (*Mapping[str, Any]*) – A query that matches the document to replace.
- **replacement** (*Mapping[str, Any]*) – The new document.
- **upsert** (*bool*) – If `True`, perform an insert if no documents match the filter.
- **bypass_document_validation** (*bool*) – (optional) If `True`, allows the write to opt-out of document level validation. Default is `False`.
- **collation** (*Optional[_CollationIn]*) – An instance of `Collation`.
- **hint** (*Optional[_IndexKeyHint]*) – An index to use to support the query predicate specified either by its string name, or in the same format as passed to `create_index()` (e.g. `[('field', ASCENDING)]`). This option is only supported on MongoDB 4.2 and above.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **let** (*Optional[Mapping[str, Any]]*) – Map of parameter names and values. Values must be constant or closed expressions that do not reference document fields. Parameters can then be accessed as variables in an aggregate expression context (e.g. "$$var").
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.

RETURNS:

- An instance of `UpdateResult`.

RETURN TYPE:

UpdateResult

Changed in version 4.1: Added `let` parameter. Added `comment` parameter.

Changed in version 3.11: Added `hint` parameter.

Changed in version 3.6: Added `session` parameter.

Changed in version 3.4: Added the *collation* option.

Changed in version 3.2: Added bypass_document_validation support.

Added in version 3.0.

update_one(filter, update, upsert=False, bypass_document_validation=False, collation=None, array_filters=None, hint=None, session=None, let=None, comment=None)

Update a single document matching the filter.

```
>>> for doc in db.test.find():
...     print(doc)
...
{'x': 1, '_id': 0}
{'x': 1, '_id': 1}
{'x': 1, '_id': 2}
>>> result = db.test.update_one({'x': 1}, {'$inc': {'x': 3}})
>>> result.matched_count
1
>>> result.modified_count
1
>>> for doc in db.test.find():
...     print(doc)
...
{'x': 4, '_id': 0}
{'x': 1, '_id': 1}
{'x': 1, '_id': 2}
```

If `upsert=True` and no documents match the filter, create a new document based on the filter criteria and update modifications.

```
>>> result = db.test.update_one({'x': -10}, {'$inc': {'x': 3}}, upsert=True)
>>> result.matched_count
0
>>> result.modified_count
0
>>> result.upserted_id
ObjectId('626a678eeaa80587d4bb3fb7')
>>> db.test.find_one(result.upserted_id)
{'_id': ObjectId('626a678eeaa80587d4bb3fb7'), 'x': -7}
```

PARAMETERS:

- **filter** (*Mapping[str, Any]*) – A query that matches the document to update.
- **update** (*Union[Mapping[str, Any], _Pipeline]*) – The modifications to apply.
- **upsert** (*bool*) – If `True`, perform an insert if no documents match the filter.
- **bypass_document_validation** (*bool*) – (optional) If `True`, allows the write to opt-out of document level validation. Default is `False`.
- **collation** (*Optional[_CollationIn]*) – An instance of `Collation`.
- **array_filters** (*Optional[Sequence[Mapping[str, Any]]]*) – A list of filters specifying which array elements an update should apply.
- **hint** (*Optional[_IndexKeyHint]*) – An index to use to support the query predicate specified either by its string name, or in the same format as passed to `create_index()` (e.g. `[('field', ASCENDING)]`). This option is only supported on MongoDB 4.2 and above.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **let** (*Optional[Mapping[str, Any]]*) – Map of parameter names and values. Values must be constant or closed expressions that do not reference document fields. Parameters can then be accessed as variables in an aggregate expression context (e.g. "$$var").
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.

RETURNS:

- An instance of `UpdateResult`.

RETURN TYPE:

UpdateResult

Changed in version 4.1: Added `let` parameter. Added `comment` parameter.

Changed in version 3.11: Added `hint` parameter.

Changed in version 3.9: Added the ability to accept a pipeline as the `update`.

Changed in version 3.6: Added the `array_filters` and `session` parameters.

Changed in version 3.4: Added the `collation` option.

Changed in version 3.2: Added `bypass_document_validation` support.

Added in version 3.0.

```
update_many(filter, update, upsert=False, array_filters=None,
    bypass_document_validation=None, collation=None, hint=None,
    session=None, let=None, comment=None)
```

Update one or more documents that match the filter.

```
>>> for doc in db.test.find():
...     print(doc)
...
{'x': 1, '_id': 0}
{'x': 1, '_id': 1}
{'x': 1, '_id': 2}
>>> result = db.test.update_many({'x': 1}, {'$inc': {'x': 3}})
>>> result.matched_count
3
>>> result.modified_count
3
>>> for doc in db.test.find():
...     print(doc)
...
{'x': 4, '_id': 0}
{'x': 4, '_id': 1}
{'x': 4, '_id': 2}
```

PARAMETERS:

- **filter** (*Mapping[str, Any]*) – A query that matches the documents to update.
- **update** (*Union[Mapping[str, Any], _Pipeline]*) – The modifications to apply.
- **upsert** (*bool*) – If `True`, perform an insert if no documents match the filter.
- **bypass_document_validation** (*Optional[bool]*) – If `True`, allows the write to opt-out of document level validation. Default is `False`.
- **collation** (*Optional[_CollationIn]*) – An instance of `Collation`.
- **array_filters** (*Optional[Sequence[Mapping[str, Any]]]*) – A list of filters specifying which array elements an update should apply.
- **hint** (*Optional[_IndexKeyHint]*) – An index to use to support the query predicate specified either by its string name, or in the same format as passed to `create_index()` (e.g. `[('field', ASCENDING)]`). This option is only supported on MongoDB 4.2 and above.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **let** (*Optional[Mapping[str, Any]]*) – Map of parameter names and values. Values must be constant or closed expressions that do not reference document fields. Parameters can then be accessed as variables in an aggregate expression context (e.g. "$$var").
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.

RETURNS:

- An instance of `UpdateResult`.

RETURN TYPE:

UpdateResult

Changed in version 4.1: Added `let` parameter. Added `comment` parameter.

Changed in version 3.11: Added `hint` parameter.

Changed in version 3.9: Added the ability to accept a pipeline as the *update*.

Changed in version 3.6: Added `array_filters` and `session` parameters.

Changed in version 3.4: Added the *collation* option.

Changed in version 3.2: Added bypass_document_validation support.

Added in version 3.0.

`delete_one(filter, collation=None, hint=None, session=None, let=None, comment=None)`

Delete a single document matching the filter.

```
>>> db.test.count_documents({'x': 1})
3
>>> result = db.test.delete_one({'x': 1})
>>> result.deleted_count
1
>>> db.test.count_documents({'x': 1})
2
```

PARAMETERS:

- **filter** (*Mapping[str, Any]*) – A query that matches the document to delete.
- **collation** (*Optional[_CollationIn]*) – An instance of `Collation`.
- **hint** (*Optional[_IndexKeyHint]*) – An index to use to support the query predicate specified either by its string name, or in the same format as passed to `create_index()` (e.g. `[('field', ASCENDING)]`). This option is only supported on MongoDB 4.4 and above.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **let** (*Optional[Mapping[str, Any]]*) – Map of parameter names and values. Values must be constant or closed expressions that do not reference document fields. Parameters can then be accessed as variables in an aggregate expression context (e.g. "$$var").
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.

RETURNS:

- An instance of `DeleteResult`.

RETURN TYPE:

DeleteResult

Changed in version 4.1: Added `let` parameter. Added `comment` parameter.

Changed in version 3.11: Added `hint` parameter.

Changed in version 3.6: Added `session` parameter.

Changed in version 3.4: Added the *collation* option.

Added in version 3.0.

delete_many(filter, collation=None, hint=None, session=None, let=None, comment=None)

Delete one or more documents matching the filter.

```
>>> db.test.count_documents({'x': 1})
3
>>> result = db.test.delete_many({'x': 1})
>>> result.deleted_count
3
>>> db.test.count_documents({'x': 1})
0
```

PARAMETERS:

- **filter** (*Mapping[str, Any]*) – A query that matches the documents to delete.
- **collation** (*Optional[_CollationIn]*) – An instance of `Collation`.
- **hint** (*Optional[_IndexKeyHint]*) – An index to use to support the query predicate specified either by its string name, or in the same format as passed to `create_index()` (e.g. `[('field', ASCENDING)]`). This option is only supported on MongoDB 4.4 and above.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **let** (*Optional[Mapping[str, Any]]*) – Map of parameter names and values. Values must be constant or closed expressions that do not reference document fields. Parameters can then be accessed as variables in an aggregate expression context (e.g. "$$var").
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.

RETURNS:

- An instance of `DeleteResult`.

RETURN TYPE:

DeleteResult

Changed in version 4.1: Added `let` parameter. Added `comment` parameter.

Changed in version 3.11: Added `hint` parameter.

Changed in version 3.6: Added `session` parameter.

Changed in version 3.4: Added the *collation* option.

Added in version 3.0.

aggregate(pipeline, session=None, let=None, comment=None, **kwargs)

Perform an aggregation using the aggregation framework on this collection.

The `aggregate()` method obeys the `read_preference` of this `Collection`, except when `$out` or `$merge` are used on MongoDB <5.0, in which case `PRIMARY` is used.

> Note
>
> This method does not support the 'explain' option. Please use PyMongoExplain instead. An example is included in the Aggregation Framework documentation.

> Note
>
> The `write_concern` of this collection is automatically applied to this operation.

PARAMETERS:

- **pipeline** (*_Pipeline*) – a list of aggregation pipeline stages
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **let** (*Optional[Mapping[str, Any]]*) – A dict of parameter names and values. Values must be constant or closed expressions that do not reference document fields. Parameters can then be accessed as variables in an aggregate expression context (e.g. `"$$var"`). This option is only supported on MongoDB >= 5.0.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – extra aggregate command parameters.

RETURN TYPE:

CommandCursor[_DocumentType]

All optional aggregate command parameters should be passed as keyword arguments to this method. Valid options include, but are not limited to:

> - *allowDiskUse* (bool): Enables writing to temporary files. When set to True, aggregation stages can write data to the _tmp subdirectory of the –dbpath directory. The default is False.
> - *maxTimeMS* (int): The maximum amount of time to allow the operation to run in milliseconds.
> - *batchSize* (int): The maximum number of documents to return per batch. Ignored if the connected mongod or mongos does not support returning aggregate results using a cursor.
> - *collation* (optional): An instance of `Collation`.

RETURNS:

A `CommandCursor` over the result set.

PARAMETERS:

- **pipeline** (*_Pipeline*)
- **session** (*Optional[ClientSession]*)
- **let** (*Optional[Mapping[str, Any]]*)
- **comment** (*Optional[Any]*)
- **kwargs** (*Any*)

RETURN TYPE:

CommandCursor[_DocumentType]

Changed in version 4.1: Added `comment` parameter. Added `let` parameter. Support $merge and $out executing on secondaries according to the collection's `read_preference`.

Changed in version 4.0: Removed the `useCursor` option.

Changed in version 3.9: Apply this collection's read concern to pipelines containing the *$out* stage when connected to MongoDB >= 4.2. Added support for the `$merge` pipeline stage. Aggregations that write always use read preference `PRIMARY`.

Changed in version 3.6: Added the *session* parameter. Added the *maxAwaitTimeMS* option. Deprecated the *useCursor* option.

Changed in version 3.4: Apply this collection's write concern automatically to this operation when connected to MongoDB >= 3.4. Support the *collation* option.

Changed in version 3.0: The `aggregate()` method always returns a CommandCursor. The pipeline argument must be a list.

See also

Aggregation Examples

aggregate_raw_batches(pipeline, session=None, comment=None, **kwargs)

Perform an aggregation and retrieve batches of raw BSON.

Similar to the `aggregate()` method but returns a `RawBatchCursor`.

This example demonstrates how to work with raw batches, but in practice raw batches should be passed to an external library that can decode BSON into another data type, rather than used with PyMongo's `bson` module.

```
>>> import bson
>>> cursor = db.test.aggregate_raw_batches([
...     {'$project': {'x': {'$multiply': [2, '$x']}}}])
>>> for batch in cursor:
...     print(bson.decode_all(batch))
```

Note

aggregate_raw_batches does not support auto encryption.

Changed in version 3.12: Added session support.

Added in version 3.6.

PARAMETERS:

- **pipeline** (*_Pipeline*)
- **session** (*Optional[ClientSession]*)
- **comment** (*Optional[Any]*)
- **kwargs** (*Any*)

RETURN TYPE:

RawBatchCursor[_DocumentType]

watch(pipeline=None, full_document=None, resume_after=None, max_await_time_ms=None, batch_size=None, collation=None, start_at_operation_time=None, session=None, start_after=None, comment=None, full_document_before_change=None, show_expanded_events=None)

Watch changes on this collection.

Performs an aggregation with an implicit initial `$changeStream` stage and returns a `CollectionChangeStream` cursor which iterates over changes on this collection.

```
with db.collection.watch() as stream:
    for change in stream:
        print(change)
```

The `CollectionChangeStream` iterable blocks until the next change document is returned or an error is raised. If the `next()` method encounters a network error when retrieving a batch from the server, it will automatically attempt to recreate the cursor such that no change events are missed. Any error encountered during the resume attempt indicates there may be an outage and will be raised.

```
try:
    with db.collection.watch([{"$match": {"operationType": "insert"}}]) as str
        for insert_change in stream:
            print(insert_change)
except pymongo.errors.PyMongoError:
    # The ChangeStream encountered an unrecoverable error or the
    # resume attempt failed to recreate the cursor.
    logging.error("...")
```

For a precise description of the resume process see the change streams specification.

Note

Using this helper method is preferred to directly calling `aggregate()` with a `$changeStream` stage, for the purpose of supporting resumability.

⚠ Warning

This Collection's `read_concern` must be `ReadConcern("majority")` in order to use the `$changeStream` stage.

PARAMETERS:

- **pipeline** (*Optional[_Pipeline]*) – A list of aggregation pipeline stages to append to an initial `$changeStream` stage. Not all pipeline stages are valid after a `$changeStream` stage, see the MongoDB documentation on change streams for the supported stages.
- **full_document** (*Optional[str]*) – The fullDocument to pass as an option to the `$changeStream` stage. Allowed values: 'updateLookup', 'whenAvailable', 'required'. When set to 'updateLookup', the change notification for partial updates will include both a delta describing the changes to the document, as well as a copy of the entire document that was changed from some time after the change occurred.
- **full_document_before_change** (*Optional[str]*) – Allowed values: 'whenAvailable' and 'required'. Change events may now result in a 'fullDocumentBeforeChange' response field.
- **resume_after** (*Optional[Mapping[str, Any]]*) – A resume token. If provided, the change stream will start returning changes that occur directly after the operation specified in the resume token. A resume token is the _id value of a change document.
- **max_await_time_ms** (*Optional[int]*) – The maximum time in milliseconds for the server to wait for changes before responding to a getMore operation.
- **batch_size** (*Optional[int]*) – The maximum number of documents to return per batch.
- **collation** (*Optional[_CollationIn]*) – The `Collation` to use for the aggregation.
- **start_at_operation_time** (*Optional[Timestamp]*) – If provided, the resulting change stream will only return changes that occurred at or after the specified `Timestamp`. Requires MongoDB >= 4.0.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **start_after** (*Optional[Mapping[str, Any]]*) – The same as *resume_after* except that *start_after* can resume notifications after an invalidate event. This option and *resume_after* are mutually exclusive.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **show_expanded_events** (*Optional[bool]*) – Include expanded events such as DDL events like *dropIndexes*.

RETURNS:

A `CollectionChangeStream` cursor.

RETURN TYPE:

CollectionChangeStream[_DocumentType]

Changed in version 4.3: Added *show_expanded_events* parameter.

Changed in version 4.2: Added `full_document_before_change` parameter.

Changed in version 4.1: Added `comment` parameter.

Changed in version 3.9: Added the `start_after` parameter.

Changed in version 3.7: Added the `start_at_operation_time` parameter.

Added in version 3.6.

See also

The MongoDB documentation on changeStreams.

find(filter=None, projection=None, skip=0, limit=0, no_cursor_timeout=False, cursor_type=CursorType.NON_TAILABLE, sort=None, allow_partial_results=False, oplog_replay=False, batch_size=0, collation=None, hint=None, max_scan=None, max_time_ms=None, max=None, min=None, return_key=False, show_record_id=False, snapshot=False, comment=None, session=None, allow_disk_use=None)

Query the database.

The *filter* argument is a query document that all results must match. For example:

```
>>> db.test.find({"hello": "world"})
```

only matches documents that have a key “hello” with value “world”. Matches can have other keys *in addition* to “hello”. The *projection* argument is used to specify a subset of fields that should be included in the result documents. By limiting results to a certain subset of fields you can cut down on network traffic and decoding time.

Raises `TypeError` if any of the arguments are of improper type. Returns an instance of `Cursor` corresponding to this query.

The `find()` method obeys the `read_preference` of this `Collection`.

PARAMETERS:

- **filter** – A query document that selects which documents to include in the result set. Can be an empty document to include all documents.
- **projection** – a list of field names that should be returned in the result set or a dict specifying the fields to include or exclude. If *projection* is a list "_id" will always be returned. Use a dict to exclude fields from the result (e.g. projection={'_id': False}).
- **session** – a `ClientSession`.
- **skip** – the number of documents to omit (from the start of the result set) when returning the results
- **limit** – the maximum number of results to return. A limit of 0 (the default) is equivalent to setting no limit.
- **no_cursor_timeout** – if False (the default), any returned cursor is closed by the server after 10 minutes of inactivity. If set to True, the returned cursor will never time out on the server. Care should be taken to ensure that cursors with no_cursor_timeout turned on are properly closed.
- **cursor_type** –

  the type of cursor to return. The valid options are defined by `CursorType`:

  - `NON_TAILABLE` - the result of this find call will return a standard cursor over the result set.
  - `TAILABLE` - the result of this find call will be a tailable cursor - tailable cursors are only for use with capped collections. They are not closed when the last data is retrieved but are kept open and the cursor location marks the final document position. If more data is received iteration of the cursor will continue from the last document received. For details, see the tailable cursor documentation.
  - `TAILABLE_AWAIT` - the result of this find call will be a tailable cursor with the await flag set. The server will wait for a few seconds after returning the full result set so that it can capture and return additional data added during the query.
  - `EXHAUST` - the result of this find call will be an exhaust cursor. MongoDB will stream batched results to the client without waiting for the client to request each batch, reducing latency. See notes on compatibility below.

- **sort** – a list of (key, direction) pairs specifying the sort order for this query. See `sort()` for details.
- **allow_partial_results** – if True, mongos will return partial results if some shards are down instead of returning an error.
- **oplog_replay** – **DEPRECATED** - if True, set the oplogReplay query flag. Default: False.

- **batch_size** – Limits the number of documents returned in a single batch.
- **collation** – An instance of `Collation`.
- **return_key** – If True, return only the index keys in each document.
- **show_record_id** – If True, adds a field `$recordId` in each document with the storage engine's internal record identifier.
- **snapshot** – **DEPRECATED** - If True, prevents the cursor from returning a document more than once because of an intervening write operation.
- **hint** – An index, in the same format as passed to `create_index()` (e.g. `[('field', ASCENDING)]`). Pass this as an alternative to calling `hint()` on the cursor to tell Mongo the proper index to use for the query.
- **max_time_ms** – Specifies a time limit for a query operation. If the specified time is exceeded, the operation will be aborted and `ExecutionTimeout` is raised. Pass this as an alternative to calling `max_time_ms()` on the cursor.
- **max_scan** – **DEPRECATED** - The maximum number of documents to scan. Pass this as an alternative to calling `max_scan()` on the cursor.
- **min** – A list of field, limit pairs specifying the inclusive lower bound for all keys of a specific index in order. Pass this as an alternative to calling `min()` on the cursor. `hint` must also be passed to ensure the query utilizes the correct index.
- **max** – A list of field, limit pairs specifying the exclusive upper bound for all keys of a specific index in order. Pass this as an alternative to calling `max()` on the cursor. `hint` must also be passed to ensure the query utilizes the correct index.
- **comment** – A string to attach to the query to help interpret and trace the operation in the server logs and in profile data. Pass this as an alternative to calling `comment()` on the cursor.
- **allow_disk_use** – if True, MongoDB may use temporary disk files to store data exceeding the system memory limit while processing a blocking sort operation. The option has no effect if MongoDB can satisfy the specified sort using an index, or if the blocking sort requires less memory than the 100 MiB limit. This option is only supported on MongoDB 4.4 and above.
- **args** (*Any*)
- **kwargs** (*Any*)

RETURN TYPE:

Cursor[*_DocumentType*]

Note

There are a number of caveats to using `EXHAUST` as cursor_type:

- The *limit* option can not be used with an exhaust cursor.
- Exhaust cursors are not supported by mongos and can not be used with a sharded cluster.
- A `Cursor` instance created with the `EXHAUST` cursor_type requires an exclusive `socket` connection to MongoDB. If the `Cursor` is discarded without being completely iterated the underlying `socket` connection will be closed and discarded without being returned to the connection pool.

Changed in version 4.0: Removed the `modifiers` option. Empty projections (eg {} or []) are passed to the server as-is, rather than the previous behavior which substituted in a projection of `{"_id": 1}`. This means that an empty projection will now return the entire document, not just the `"_id"` field.

Changed in version 3.11: Added the `allow_disk_use` option. Deprecated the `oplog_replay` option. Support for this option is deprecated in MongoDB 4.4. The query engine now automatically optimizes queries against the oplog without requiring this option to be set.

Changed in version 3.7: Deprecated the `snapshot` option, which is deprecated in MongoDB 3.6 and removed in MongoDB 4.0. Deprecated the `max_scan` option. Support for this option is deprecated in MongoDB 4.0. Use `max_time_ms` instead to limit server-side execution time.

Changed in version 3.6: Added `session` parameter.

Changed in version 3.5: Added the options `return_key`, `show_record_id`, `snapshot`, `hint`, `max_time_ms`, `max_scan`, `min`, `max`, and `comment`. Deprecated the `modifiers` option.

Changed in version 3.4: Added support for the `collation` option.

Changed in version 3.0: Changed the parameter names `spec`, `fields`, `timeout`, and `partial` to `filter`, `projection`, `no_cursor_timeout`, and `allow_partial_results` respectively. Added the `cursor_type`, `oplog_replay`, and `modifiers` options. Removed the `network_timeout`, `read_preference`, `tag_sets`, `secondary_acceptable_latency_ms`, `max_scan`, `snapshot`, `tailable`, `await_data`, `exhaust`, `as_class`, and slave_okay parameters. Removed `compile_re` option: PyMongo now always represents BSON regular expressions as `Regex` objects. Use `try_compile()` to attempt to convert from a BSON regular expression to a Python regular expression object. Soft deprecated the `manipulate` option.

See also

The MongoDB documentation on find.

find_raw_batches(filter=None, projection=None, skip=0, limit=0, no_cursor_timeout=False, cursor_type=CursorType.NON_TAILABLE, sort=None, allow_partial_results=False, oplog_replay=False, batch_size=0, collation=None, hint=None, max_scan=None, max_time_ms=None, max=None, `min=None, return_key=False, show_record_id=False, snapshot=False, comment=None, session=None, allow_disk_use=None)`

Query the database and retrieve batches of raw BSON.

Similar to the `find()` method but returns a `RawBatchCursor`.

This example demonstrates how to work with raw batches, but in practice raw batches should be passed to an external library that can decode BSON into another data type, rather than used with PyMongo's `bson` module.

```
>>> import bson
>>> cursor = db.test.find_raw_batches()
>>> for batch in cursor:
...     print(bson.decode_all(batch))
```

Note

find_raw_batches does not support auto encryption.

Changed in version 3.12: Instead of ignoring the user-specified read concern, this method now sends it to the server when connected to MongoDB 3.6+.

Added session support.

Added in version 3.6.

PARAMETERS:

- **args** (*Any*)
- **kwargs** (*Any*)

RETURN TYPE:

RawBatchCursor[*_DocumentType*]

find_one`(filter=None, *args, **kwargs)`

Get a single document from the database.

All arguments to `find()` are also valid arguments for `find_one()`, although any *limit* argument will be ignored. Returns a single document, or `None` if no matching document is found.

The `find_one()` method obeys the `read_preference` of this `Collection`.

PARAMETERS:

- **filter** (*Any* | *None*) – a dictionary specifying the query to be performed OR any other type to be used as the value for a query for `"_id"`.
- **args** (*Any*) – any additional positional arguments are the same as the arguments to `find()`.
- **kwargs** (*Any*) –

  any additional keyword arguments are the same as the arguments to `find()`.

  :: code-block: python

  ```
  >>> collection.find_one(max_time_ms=100)
  ```

RETURN TYPE:

_DocumentType | None

find_one_and_delete(filter, projection=None, sort=None, hint=None, session=None, let=None, comment=None, **kwargs)

Finds a single document and deletes it, returning the document.

```
>>> db.test.count_documents({'x': 1})
2
>>> db.test.find_one_and_delete({'x': 1})
{'x': 1, '_id': ObjectId('54f4e12bfba5220aa4d6dee8')}
>>> db.test.count_documents({'x': 1})
1
```

If multiple documents match *filter*, a *sort* can be applied.

```
>>> for doc in db.test.find({'x': 1}):
...     print(doc)
...
{'x': 1, '_id': 0}
{'x': 1, '_id': 1}
{'x': 1, '_id': 2}
>>> db.test.find_one_and_delete(
...     {'x': 1}, sort=[('_id', pymongo.DESCENDING)])
{'x': 1, '_id': 2}
```

The *projection* option can be used to limit the fields returned.

```
>>> db.test.find_one_and_delete({'x': 1}, projection={'_id': False})
{'x': 1}
```

PARAMETERS:

- **filter** (*Mapping[str, Any]*) – A query that matches the document to delete.
- **projection** (*Optional[Union[Mapping[str, Any], Iterable[str]]]*) – a list of field names that should be returned in the result document or a mapping specifying the fields to include or exclude. If *projection* is a list "_id" will always be returned. Use a mapping to exclude fields from the result (e.g. projection={'_id': False}).
- **sort** (*Optional[_IndexList]*) – a list of (key, direction) pairs specifying the sort order for the query. If multiple documents match the query, they are sorted and the first is deleted.
- **hint** (*Optional[_IndexKeyHint]*) – An index to use to support the query predicate specified either by its string name, or in the same format as passed to `create_index()` (e.g. `[('field', ASCENDING)]`). This option is only supported on MongoDB 4.4 and above.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **let** (*Optional[Mapping[str, Any]]*) – Map of parameter names and values. Values must be constant or closed expressions that do not reference document fields. Parameters can then be accessed as variables in an aggregate expression context (e.g. "$$var").
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – additional command arguments can be passed as keyword arguments (for example maxTimeMS can be used with recent server versions).

RETURN TYPE:

_DocumentType

Changed in version 4.1: Added `let` parameter.

Changed in version 3.11: Added `hint` parameter.

Changed in version 3.6: Added `session` parameter.

Changed in version 3.2: Respects write concern.

Warning

Starting in PyMongo 3.2, this command uses the `WriteConcern` of this `Collection` when connected to MongoDB >= 3.2. Note that using an elevated write concern with this command may be slower compared to using the default write concern.

Changed in version 3.4: Added the *collation* option.

Added in version 3.0.

find_one_and_replace(filter, replacement, projection=None, sort=None,

```
return_document=ReturnDocument.BEFORE, hint=None, session=None,
**kwargs)
```

Finds a single document and replaces it, returning either the original or the replaced document.

The `find_one_and_replace()` method differs from `find_one_and_update()` by replacing the document matched by *filter*, rather than modifying the existing document.

```
>>> for doc in db.test.find({}):
...     print(doc)
...
{'x': 1, '_id': 0}
{'x': 1, '_id': 1}
{'x': 1, '_id': 2}
>>> db.test.find_one_and_replace({'x': 1}, {'y': 1})
{'x': 1, '_id': 0}
>>> for doc in db.test.find({}):
...     print(doc)
...
{'y': 1, '_id': 0}
{'x': 1, '_id': 1}
{'x': 1, '_id': 2}
```

PARAMETERS:

- **filter** (*Mapping[str, Any]*) – A query that matches the document to replace.
- **replacement** (*Mapping[str, Any]*) – The replacement document.
- **projection** (*Optional[Union[Mapping[str, Any], Iterable[str]]]*) – A list of field names that should be returned in the result document or a mapping specifying the fields to include or exclude. If *projection* is a list "_id" will always be returned. Use a mapping to exclude fields from the result (e.g. projection={'_id': False}).
- **sort** (*Optional[_IndexList]*) – a list of (key, direction) pairs specifying the sort order for the query. If multiple documents match the query, they are sorted and the first is replaced.
- **upsert** (*bool*) – When `True`, inserts a new document if no document matches the query. Defaults to `False`.
- **return_document** (*bool*) – If `ReturnDocument.BEFORE` (the default), returns the original document before it was replaced, or `None` if no document matches. If `ReturnDocument.AFTER`, returns the replaced or inserted document.
- **hint** (*Optional[_IndexKeyHint]*) – An index to use to support the query predicate specified either by its string name, or in the same format as passed to `create_index()` (e.g. `[('field', ASCENDING)]`). This option is only supported on MongoDB 4.4 and above.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **let** (*Optional[Mapping[str, Any]]*) – Map of parameter names and values. Values must be constant or closed expressions that do not reference document fields. Parameters can then be accessed as variables in an aggregate expression context (e.g. "$$var").
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – additional command arguments can be passed as keyword arguments (for example maxTimeMS can be used with recent server versions).

RETURN TYPE:

_DocumentType

Changed in version 4.1: Added `let` parameter.

Changed in version 3.11: Added the `hint` option.

Changed in version 3.6: Added `session` parameter.

Changed in version 3.4: Added the `collation` option.

Changed in version 3.2: Respects write concern.

> ⚠ Warning
>
> Starting in PyMongo 3.2, this command uses the `WriteConcern` of this `Collection` when connected to MongoDB >= 3.2. Note that using an elevated write concern with this command may be slower compared to using the default write concern.

Added in version 3.0.

find_one_and_update(filter, update, projection=None, sort=None, return_document=ReturnDocument.BEFORE, array_filters=None, hint=None, session=None, **kwargs)

Finds a single document and updates it, returning either the original or the updated document.

```
>>> db.test.find_one_and_update(
...    {'_id': 665}, {'$inc': {'count': 1}, '$set': {'done': True}})
{'_id': 665, 'done': False, 'count': 25}}
```

Returns `None` if no document matches the filter.

```
>>> db.test.find_one_and_update(
...    {'_exists': False}, {'$inc': {'count': 1}})
```

When the filter matches, by default `find_one_and_update()` returns the original version of the document before the update was applied. To return the updated (or inserted in the case of *upsert*) version of the document instead, use the *return_document* option.

```
>>> from pymongo import ReturnDocument
>>> db.example.find_one_and_update(
...     {'_id': 'userid'},
...     {'$inc': {'seq': 1}},
...     return_document=ReturnDocument.AFTER)
{'_id': 'userid', 'seq': 1}
```

You can limit the fields returned with the *projection* option.

```
>>> db.example.find_one_and_update(
...     {'_id': 'userid'},
...     {'$inc': {'seq': 1}},
...     projection={'seq': True, '_id': False},
...     return_document=ReturnDocument.AFTER)
{'seq': 2}
```

The *upsert* option can be used to create the document if it doesn't already exist.

```
>>> db.example.delete_many({}).deleted_count
1
>>> db.example.find_one_and_update(
...     {'_id': 'userid'},
...     {'$inc': {'seq': 1}},
...     projection={'seq': True, '_id': False},
...     upsert=True,
...     return_document=ReturnDocument.AFTER)
{'seq': 1}
```

If multiple documents match *filter*, a *sort* can be applied.

```
>>> for doc in db.test.find({'done': True}):
...     print(doc)
...
{'_id': 665, 'done': True, 'result': {'count': 26}}
{'_id': 701, 'done': True, 'result': {'count': 17}}
>>> db.test.find_one_and_update(
...     {'done': True},
...     {'$set': {'final': True}},
...     sort=[('_id', pymongo.DESCENDING)])
{'_id': 701, 'done': True, 'result': {'count': 17}}
```

PARAMETERS:

- **filter** (*Mapping[str, Any]*) – A query that matches the document to update.
- **update** (*Union[Mapping[str, Any], _Pipeline]*) – The update operations to apply.
- **projection** (*Optional[Union[Mapping[str, Any], Iterable[str]]]*) – A list of field names that should be returned in the result document or a mapping specifying the fields to include or exclude. If *projection* is a list "_id" will always be returned. Use a dict to exclude fields from the result (e.g. projection={'_id': False}).
- **sort** (*Optional[_IndexList]*) – a list of (key, direction) pairs specifying the sort order for the query. If multiple documents match the query, they are sorted and the first is updated.
- **upsert** (*bool*) – When `True`, inserts a new document if no document matches the query. Defaults to `False`.
- **return_document** (*bool*) – If `ReturnDocument.BEFORE` (the default), returns the original document before it was updated. If `ReturnDocument.AFTER`, returns the updated or inserted document.
- **array_filters** (*Optional[Sequence[Mapping[str, Any]]]*) – A list of filters specifying which array elements an update should apply.
- **hint** (*Optional[_IndexKeyHint]*) – An index to use to support the query predicate specified either by its string name, or in the same format as passed to `create_index()` (e.g. `[('field', ASCENDING)]`). This option is only supported on MongoDB 4.4 and above.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **let** (*Optional[Mapping[str, Any]]*) – Map of parameter names and values. Values must be constant or closed expressions that do not reference document fields. Parameters can then be accessed as variables in an aggregate expression context (e.g. "$$var").
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – additional command arguments can be passed as keyword arguments (for example maxTimeMS can be used with recent server versions).

RETURN TYPE:

_DocumentType

Changed in version 3.11: Added the `hint` option.

Changed in version 3.9: Added the ability to accept a pipeline as the `update`.

Changed in version 3.6: Added the `array_filters` and `session` options.

Changed in version 3.4: Added the `collation` option.

Changed in version 3.2: Respects write concern.

> **Warning**
>
> Starting in PyMongo 3.2, this command uses the `WriteConcern` of this `Collection` when connected to MongoDB >= 3.2. Note that using an elevated write concern with this command may be slower compared to using the default write concern.

Added in version 3.0.

count_documents(filter, session=None, comment=None, **kwargs)

Count the number of documents in this collection.

> **Note**
>
> For a fast count of the total documents in a collection see `estimated_document_count()`.

The `count_documents()` method is supported in a transaction.

All optional parameters should be passed as keyword arguments to this method. Valid options include:

> - *skip* (int): The number of matching documents to skip before returning results.
> - *limit* (int): The maximum number of documents to count. Must be a positive integer. If not provided, no limit is imposed.
> - *maxTimeMS* (int): The maximum amount of time to allow this operation to run, in milliseconds.
> - *collation* (optional): An instance of `Collation`.
> - *hint* (string or list of tuples): The index to use. Specify either the index name as a string or the index specification as a list of tuples (e.g. [(‘a’, pymongo.ASCENDING), (‘b’, pymongo.ASCENDING)]).

The `count_documents()` method obeys the `read_preference` of this `Collection`.

Note

When migrating from `count()` to `count_documents()` the following query operators must be replaced:

Operator	Replacement
$where	$expr
$near	$geoWithin with $center
$nearSphere	$geoWithin with $centerSphere

PARAMETERS:

- **filter** (*Mapping[str, Any]*) – A query document that selects which documents to count in the collection. Can be an empty document to count all documents.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – See list of options above.

RETURN TYPE:

int

Added in version 3.7.

`estimated_document_count(comment=None, **kwargs)`

Get an estimate of the number of documents in this collection using collection metadata.

The `estimated_document_count()` method is **not** supported in a transaction.

All optional parameters should be passed as keyword arguments to this method. Valid options include:

> - *maxTimeMS* (int): The maximum amount of time to allow this operation to run, in milliseconds.

PARAMETERS:

- **comment** (*Any | None*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – See list of options above.

RETURN TYPE:

int

Changed in version 4.2: This method now always uses the count command. Due to an oversight in versions 5.0.0-5.0.8 of MongoDB, the count command was not included in V1 of the MongoDB Stable API. Users of the Stable API with estimated_document_count are recommended to upgrade their server version to 5.0.9+ or set `pymongo.server_api.ServerApi.strict` to `False` to avoid encountering errors.

Added in version 3.7.

`distinct(key, filter=None, session=None, comment=None, **kwargs)`

Get a list of distinct values for *key* among all documents in this collection.

Raises `TypeError` if *key* is not an instance of `str`.

All optional distinct parameters should be passed as keyword arguments to this method. Valid options include:

> - *maxTimeMS* (int): The maximum amount of time to allow the count command to run, in milliseconds.
> - *collation* (optional): An instance of `Collation`.

The `distinct()` method obeys the `read_preference` of this `Collection`.

PARAMETERS:

- **key** (*str*) – name of the field for which we want to get the distinct values
- **filter** (*Optional[Mapping[str, Any]]*) – A query document that specifies the documents from which to retrieve the distinct values.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – See list of options above.

RETURN TYPE:

list

Changed in version 3.6: Added `session` parameter.

Changed in version 3.4: Support the *collation* option.

`create_index(keys, session=None, comment=None, **kwargs)`

Creates an index on this collection.

Takes either a single key or a list containing (key, direction) pairs or keys. If no direction is given, `ASCENDING` will be assumed. The key(s) must be an instance of `str` and the direction(s) must be one of (`ASCENDING`, `DESCENDING`, `GEO2D`, `GEOSPHERE`, `HASHED`, `TEXT`).

To create a single key ascending index on the key `'mike'` we just use a string argument:

```
>>> my_collection.create_index("mike")
```

For a compound index on `'mike'` descending and `'eliot'` ascending we need to use a list of tuples:

```
>>> my_collection.create_index([("mike", pymongo.DESCENDING),
...                                 "eliot"])
```

All optional index creation parameters should be passed as keyword arguments to this method. For example:

```
>>> my_collection.create_index([("mike", pymongo.DESCENDING)],
...                                background=True)
```

Valid options include, but are not limited to:

- *name*: custom name to use for this index - if none is given, a name will be generated.
- *unique*: if `True`, creates a uniqueness constraint on the index.
- *background*: if `True`, this index should be created in the background.
- *sparse*: if `True`, omit from the index any documents that lack the indexed field.
- *bucketSize*: for use with geoHaystack indexes. Number of documents to group together within a certain proximity to a given longitude and latitude.
- *min*: minimum value for keys in a `GEO2D` index.
- *max*: maximum value for keys in a `GEO2D` index.
- *expireAfterSeconds*: <int> Used to create an expiring (TTL) collection. MongoDB will automatically delete documents from this collection after <int> seconds. The indexed field must be a UTC datetime or the data will not expire.
- *partialFilterExpression*: A document that specifies a filter for a partial index.
- *collation* (optional): An instance of `Collation`.
- *wildcardProjection*: Allows users to include or exclude specific field paths from a wildcard index using the {"$**" : 1} key pattern. Requires MongoDB >= 4.2.
- *hidden*: if `True`, this index will be hidden from the query planner and will not be evaluated as part of query plan selection. Requires MongoDB >= 4.4.

See the MongoDB documentation for a full list of supported options by server version.

Warning

dropDups is not supported by MongoDB 3.0 or newer. The option is silently ignored by the server and unique index builds using the option will fail if a duplicate value is detected.

Note

The `write_concern` of this collection is automatically applied to this operation.

PARAMETERS:

- **keys** (*_IndexKeyHint*) – a single key or a list of (key, direction) pairs specifying the index to create
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – any additional index creation options (see the above list) should be passed as keyword arguments.

RETURN TYPE:

str

Changed in version 4.4: Allow passing a list containing (key, direction) pairs or keys for the `keys` parameter.
Changed in version 4.1: Added `comment` parameter.
Changed in version 3.11: Added the `hidden` option.
Changed in version 3.6: Added `session` parameter. Added support for passing maxTimeMS in kwargs.
Changed in version 3.4: Apply this collection's write concern automatically to this operation when connected to MongoDB >= 3.4. Support the *collation* option.
Changed in version 3.2: Added partialFilterExpression to support partial indexes.
Changed in version 3.0: Renamed *key_or_list* to *keys*. Removed the *cache_for* option. `create_index()` no longer caches index names. Removed support for the drop_dups and bucket_size aliases.

See also

The MongoDB documentation on indexes.

`create_indexes(indexes, session=None, comment=None, **kwargs)`

Create one or more indexes on this collection.

```
>>> from pymongo import IndexModel, ASCENDING, DESCENDING
>>> index1 = IndexModel([("hello", DESCENDING),
...                     ("world", ASCENDING)], name="hello_world")
>>> index2 = IndexModel([("goodbye", DESCENDING)])
>>> db.test.create_indexes([index1, index2])
["hello_world", "goodbye_-1"]
```

PARAMETERS:

- **indexes** (*Sequence[IndexModel]*) – A list of `IndexModel` instances.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – optional arguments to the createIndexes command (like maxTimeMS) can be passed as keyword arguments.

RETURN TYPE:

list[str]

Note

The `write_concern` of this collection is automatically applied to this operation.

Changed in version 3.6: Added `session` parameter. Added support for arbitrary keyword arguments.

Changed in version 3.4: Apply this collection's write concern automatically to this operation when connected to MongoDB >= 3.4.

Added in version 3.0.

drop_index(index_or_name, session=None, comment=None, **kwargs)

Drops the specified index on this collection.

Can be used on non-existent collections or collections with no indexes. Raises OperationFailure on an error (e.g. trying to drop an index that does not exist). *index_or_name* can be either an index name (as returned by *create_index*), or an index specifier (as passed to *create_index*). An index specifier should be a list of (key, direction) pairs. Raises TypeError if index is not an instance of (str, unicode, list).

Warning

if a custom name was used on index creation (by passing the *name* parameter to `create_index()`) the index **must** be dropped by name.

PARAMETERS:

- **index_or_name** (*_IndexKeyHint*) – index (or name of index) to drop
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – optional arguments to the createIndexes command (like maxTimeMS) can be passed as keyword arguments.

RETURN TYPE:

None

Note

The `write_concern` of this collection is automatically applied to this operation.

Changed in version 3.6: Added `session` parameter. Added support for arbitrary keyword arguments.

Changed in version 3.4: Apply this collection's write concern automatically to this operation when connected to MongoDB >= 3.4.

`drop_indexes(session=None, comment=None, **kwargs)`

Drops all indexes on this collection.

Can be used on non-existent collections or collections with no indexes. Raises OperationFailure on an error.

PARAMETERS:

- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – optional arguments to the createIndexes command (like maxTimeMS) can be passed as keyword arguments.

RETURN TYPE:

None

Note

The `write_concern` of this collection is automatically applied to this operation.

Changed in version 3.6: Added `session` parameter. Added support for arbitrary keyword arguments.

Changed in version 3.4: Apply this collection's write concern automatically to this operation when connected to MongoDB >= 3.4.

list_indexes(session=None, comment=None)

Get a cursor over the index documents for this collection.

```
>>> for index in db.test.list_indexes():
...     print(index)
...
SON([('v', 2), ('key', SON([('_id', 1)])), ('name', '_id_')])
```

PARAMETERS:

- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.

RETURNS:

An instance of `CommandCursor`.

RETURN TYPE:

CommandCursor[MutableMapping[str, Any]]

Changed in version 4.1: Added `comment` parameter.

Changed in version 3.6: Added `session` parameter.

Added in version 3.0.

index_information(session=None, comment=None)

Get information on this collection's indexes.

Returns a dictionary where the keys are index names (as returned by create_index()) and the values are dictionaries containing information about each index. The dictionary is guaranteed to contain at least a single key, `"key"` which is a list of (key, direction) pairs specifying the index (as passed to create_index()). It will also contain any other metadata about the indexes, except for the `"ns"` and `"name"` keys, which are cleaned. Example output might look like this:

```
>>> db.test.create_index("x", unique=True)
'x_1'
>>> db.test.index_information()
{'_id_': {'key': [('_id', 1)]},
 'x_1': {'unique': True, 'key': [('x', 1)]}}
```

PARAMETERS:

- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.

RETURN TYPE:

MutableMapping[str, Any]

Changed in version 4.1: Added `comment` parameter.

Changed in version 3.6: Added `session` parameter.

`create_search_index(model, session=None, comment=None, **kwargs)`

Create a single search index for the current collection.

PARAMETERS:

- **model** (*Union[Mapping[str, Any], SearchIndexModel]*) – The model for the new search index. It can be given as a `SearchIndexModel` instance or a dictionary with a model "definition" and optional "name".
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Any*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – optional arguments to the createSearchIndexes command (like maxTimeMS) can be passed as keyword arguments.

RETURNS:

The name of the new search index.

RETURN TYPE:

str

Note

requires a MongoDB server version 7.0+ Atlas cluster.

Added in version 4.5.

`create_search_indexes(models, session=None, comment=None, **kwargs)`

Create multiple search indexes for the current collection.

PARAMETERS:

- **models** (*list[SearchIndexModel]*) – A list of `SearchIndexModel` instances.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – optional arguments to the createSearchIndexes command (like maxTimeMS) can be passed as keyword arguments.

RETURNS:

A list of the newly created search index names.

RETURN TYPE:

list[str]

> Note
>
> requires a MongoDB server version 7.0+ Atlas cluster.

Added in version 4.5.

`drop_search_index(name, session=None, comment=None, **kwargs)`

Delete a search index by index name.

PARAMETERS:

- **name** (*str*) – The name of the search index to be deleted.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – optional arguments to the dropSearchIndexes command (like maxTimeMS) can be passed as keyword arguments.

RETURN TYPE:

None

> Note
>
> requires a MongoDB server version 7.0+ Atlas cluster.

Added in version 4.5.

`list_search_indexes(name=None, session=None, comment=None, **kwargs)`

Return a cursor over search indexes for the current collection.

PARAMETERS:

- **name** (*Optional[str]*) – If given, the name of the index to search for. Only indexes with matching index names will be returned. If not given, all search indexes for the current collection will be returned.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*)

RETURNS:

A `CommandCursor` over the result set.

RETURN TYPE:

CommandCursor[Mapping[str, Any]]

> Note
>
> requires a MongoDB server version 7.0+ Atlas cluster.

Added in version 4.5.

`update_search_index(name, definition, session=None, comment=None, **kwargs)`

Update a search index by replacing the existing index definition with the provided definition.

PARAMETERS:

- **name** (*str*) – The name of the search index to be updated.
- **definition** (*Mapping[str, Any]*) – The new search index definition.
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – optional arguments to the updateSearchIndexes command (like maxTimeMS) can be passed as keyword arguments.

RETURN TYPE:

None

> Note
>
> requires a MongoDB server version 7.0+ Atlas cluster.

Added in version 4.5.

`drop(session=None, comment=None, encrypted_fields=None)`

Alias for `drop_collection()`.

PARAMETERS:

- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **encrypted_fields** (*Optional[Mapping[str, Any]]*) – **(BETA)** Document that describes the encrypted fields for Queryable Encryption.

RETURN TYPE:

None

The following two calls are equivalent:

```
>>> db.foo.drop()
>>> db.drop_collection("foo")
```

Changed in version 4.2: Added `encrypted_fields` parameter.

Changed in version 4.1: Added `comment` parameter.

Changed in version 3.7: `drop()` now respects this `Collection`'s `write_concern`.

Changed in version 3.6: Added `session` parameter.

rename(new_name, session=None, comment=None, **kwargs)

Rename this collection.

If operating in auth mode, client must be authorized as an admin to perform this operation. Raises `TypeError` if *new_name* is not an instance of `str`. Raises `InvalidName` if *new_name* is not a valid collection name.

PARAMETERS:

- **new_name** (*str*) – new name for this collection
- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.
- **kwargs** (*Any*) – additional arguments to the rename command may be passed as keyword arguments to this helper method (i.e. `dropTarget=True`)

RETURN TYPE:

MutableMapping[str, Any]

Note

The `write_concern` of this collection is automatically applied to this operation.

Changed in version 3.6: Added `session` parameter.

Changed in version 3.4: Apply this collection's write concern automatically to this operation when connected to MongoDB >= 3.4.

`options(session=None, comment=None)`

Get the options set on this collection.

Returns a dictionary of options and their values - see `create_collection()` for more information on the possible options. Returns an empty dictionary if the collection has not been created yet.

PARAMETERS:

- **session** (*Optional[ClientSession]*) – a `ClientSession`.
- **comment** (*Optional[Any]*) – A user-provided comment to attach to this command.

RETURN TYPE:

MutableMapping[str, Any]

Changed in version 3.6: Added `session` parameter.

`__getitem__(name)`

PARAMETERS:

name (*str*)

RETURN TYPE:

Collection[*_DocumentType*]

`__getattr__(name)`

Get a sub-collection of this collection by name.

Raises InvalidName if an invalid collection name is used.

PARAMETERS:

name (*str*) – the name of the collection to get

RETURN TYPE:

Collection[*_DocumentType*]

Copyright © MongoDB, Inc. 2008-present. MongoDB, Mongo, and the leaf logo are registered trademarks of MongoDB, Inc
Made with Furo