aiomongodel Documentation

Release 0.2.0

ilex

Sep 11, 2018

Contents

Contents:

CHAPTER 1

API documentation

1.1 Define models

class aiomongodel.**Document**(*, *_empty=False*, ***kwargs*)

Base class for documents.

Each document class should be defined by inheriting from this class and specifying fields and optionally meta options using internal Meta class.

Fields are inherited from base classes and can be overwritten.

Meta options are NOT inherited.

Possible meta options for `class Meta`:

- `collection`: Name of the document's db collection.
- `indexes`: List of `pymongo.IndexModel` for collection.
- `queryset`: Query set class to query documents.
- `default_query`: Each query in query set will be extended using this query through `$and` operator.
- `default_sort`: Default sort expression to order documents in `find`.
- `codec_options`: Collection's codec options.
- `read_preference`: Collection's read preference.
- `write_concern`: Collection's write concern.
- `read_concern`: Collection's read concern.

Note: Indexes are not created automatically. Use `MotorQuerySet.create_indexes` method to create document's indexes.

Example:

```
from pymongo import IndexModel, ASCENDING, DESCENDING

class User(Document):
    name = StrField(regex=r'^[a-zA-Z]{6,20}$')
    is_active = BoolField(default=True)
    created = DateTimeField(default=lambda: datetime.utcnow())

    class Meta:
        # define a collection name
        collection = 'users'
        # define collection indexes. Use
        # await User.q(db).create_indexes()
        # to create them on application startup.
        indexes = [
            IndexModel([('name', ASCENDING)], unique=True),
            IndexModel([('created', DESCENDING)])]
        # order by `created` field by default
        default_sort = [('created', DESCENDING)]

class ActiveUser(User):
    is_active = BoolField(default=True, choices=[True])

    class Meta:
        collection = 'users'
        # specify a default query to work ONLY with
        # active users. So for example
        # await ActiveUser.q(db).count({})
        # will count ONLY active users.
        default_query = {'is_active': True}
```

Initialize document.

Parameters

- **_empty** (*bool*) – If True return an empty document without setting any field.
- ****kwargs** – Fields values to set. Each key should be a field name not a mongo name of the field.

classmethod coll(*db*)

Return raw collection object.

Parameters **db** – Motor database object.

Returns Raw Motor collection object.

Return type MotorCollection

classmethod create(*db*, *session=None*, ***kwargs*)

Create document in mongodb.

Parameters

- **db** – Database instance.
- **session** – Motor session object.
- ****kwargs** – Document's fields values.

Returns Created document instance.

Raises `ValidationError` – If some fields are not valid.

classmethod create_collection(*db*, *session=None*)
Create collection for documents.

Parameters **db** – Database object.

Returns Collection object.

delete(*db*, *session=None*)
Delete current object from db.

classmethod from_data(*data*)
Create document from user provided data.

Parameters **data** (*dict*) – Data dict in form {field_name => value}.

Returns Document isinstance.

classmethod from_mongo(*data*)
Create document from mongo data.

Parameters **data** (*dict*) – SON data loaded from mongodb.

Returns Document instance.

populate_with_data(*data*)
Populate document object with data.

Parameters **data** (*dict*) – Document data in form {field_name => value}.

Returns Self instance.

Raises AttributeError – On wrong field name.

classmethod q(*db*, *session=None*)
Return queryset object.

Parameters

- **db** – Motor database object.
- **session** – Motor client session object.

Returns Queryset object.

Return type *MotorQuerySet*

query_id

reload(*db*, *session=None*)
Reload current object from mongodb.

save(*db*, *do_insert=False*, *session=None*)
Save document in mongodb.

Parameters

- **db** – Database instance.
- **do_insert** (*bool*) – If True always perform insert_one, else perform replace_one with upsert=True.
- **session** – Motor session object.

to_data()
Return internal data of the document.

Note: Internal data can contain embedded document objects, lists etc.

Returns Data of the document.

Return type OrderedDict

to_mongo()
Convert document to mongo format.

update(*db*, *update_document*, *session=None*)
Update current object using query.

Usage:

```
class User(Document):
    name = StrField()
    value = IntField(default=0)

async def go(db):
    u = await User(name='xxx').save(db)
    await u.update(db,
                   {'$set': {User.name.s: 'yyy'},
                    '$inc': {User.value.s: 1}})
```

validate()
Validate data.

Returns Self instance.

Raises `ValidationError` – If document's data is not valid.

classmethod validate_document(*document*)
Validate given document.

Parameters **document** – Document instance to validate.

Raises `ValidationError` – If document's data is not valid.

class aiomongodel.**EmbeddedDocument**(*, *_empty=False*, ***kwargs*)
Base class for embedded documents.

Initialize document.

Parameters

- **_empty** (*bool*) – If True return an empty document without setting any field.
- ****kwargs** – Fields values to set. Each key should be a field name not a mongo name of the field.

classmethod from_data(*data*)
Create document from user provided data.

Parameters **data** (*dict*) – Data dict in form {field_name => value}.

Returns Document isinstance.

classmethod from_mongo(*data*)
Create document from mongo data.

Parameters **data** (*dict*) – SON data loaded from mongodb.

Returns Document instance.

populate_with_data(*data*)
Populate document object with data.

Parameters **data** (*dict*) – Document data in form {field_name => value}.

Returns Self instance.

Raises AttributeError – On wrong field name.

to_data()
Return internal data of the document.

Note: Internal data can contain embedded document objects, lists etc.

Returns Data of the document.

Return type OrderedDict

to_mongo()
Convert document to mongo format.

validate()
Validate data.

Returns Self instance.

Raises ValidationError – If document's data is not valid.

classmethod validate_document(*document*)
Validate given document.

Parameters **document** – Document instance to validate.

Raises ValidationError – If document's data is not valid.

class aiomongodel.document.**Meta**(***kwargs*)
Storage for Document meta info.

collection_name
Name of the document's db collection (note that it should be specified as collection Meta class attribute).

indexes
List of pymongo.IndexModel for collection.

queryset
Query set class to query documents.

default_query
Each query in query set will be extended using this query through $and operator.

default_sort
Default sort expression to order documents in find.

fields
OrderedDict of document fields as {field_name => field}.

fields_synonyms
Dict of synonyms for field as {field_name => synonym_name}.

codec_options
Collection's codec options.

read_preference
Collection's read preference.

write_concern
Collection's write concern.

read_concern
Collection's read concern.

OPTIONS = {'codec_options', 'default_query', 'read_concern', 'fields', 'write_concern'

collection(*db*)
Get collection for documents.

Parameters **db** – Database object.

Returns Collection object.

1.2 Fields

class aiomongodel.fields.**Field**(*, *required=True*, *default=<object object>*, *mongo_name=None*, *name=None*, *allow_none=False*, *choices=None*, *field_type=None*)
Base class for all fields.

name
str – Name of the field.

mongo_name
str – Name of the field in mongodb.

required
bool – Is field required.

allow_none
bool – Can field be assigned with `None`.

default
Default value for field.

choices
dict, set – Dict or set of choices for a field. If it is a `dict` keys are used as choices.

Create field.

Parameters

- **required** (*bool*) – Is field required. Defaults to `True`.
- **default** – Default value for a field. When document has no value for field in `__init__` it try to use default value (if it is not `_Empty`). Defaults to `_Empty`.

Note: Default value is ignored if field is not required.

Note: Default can be a value or a callable with no arguments.

- **mongo_name** (*str*) – Name of the field in MongoDB. Defaults to `None`.

  Note: If `mongo_name` is None it is set to `name` of the field.

- **name** (*str*) – Name of the field. Should not be used explicitly as it is set by metaclass. Defaults to `None`.
- **allow_none** (*bool*) – Can field be assign with `None`. Defaults to `False`.
- **choices** (*dict, set*) – Possible values for field. If it is a `dict`, keys should be possible values. To preserve values order use `collections.OrderedDict`. Defaults to `None`.

Note: If `choices` are given then other constraints are ignored.

from_data(*value*)
Convert value from user provided data to field type.

Parameters **value** – Value provided by user.

Returns Converted value or value as is if error occured. If value is `None` return `None`.

from_mongo(*value*)
Convert value from mongo format to python field format.

s
Return mongodb name of the field.

This property can be used wherever mongodb field's name is required.

Example:

```
User.q(db).find({User.name.s: 'Francesco', User.is_admin.s: True},
                {User.posts.s: 1, User._id.s: 0})
```

Note: Field's `name` and `mongo_name` could be different so `User.is_admin.s` could be for example `'isadm'`.

to_mongo(*value*)
Convert value to mongo format.

Document fields.

class aiomongodel.fields.**AnyField**(**args*, ***kwargs*)
Any type field.

Can store any type of value. Store a value as is. It's up to developer if a value can be stored in mongodb.

from_data(*value*)
Convert value from user provided data to field type.

Parameters **value** – Value provided by user.

Returns Converted value or value as is if error occured. If value is `None` return `None`.

class aiomongodel.fields.**StrField**(*, *regex=None*, *allow_blank=False*, *min_length=None*, *max_length=None*, ***kwargs*)
String field.

Create string field.

Parameters

- **regex** (*str*) – Regular expression for field's values. Defaults to `None`.
- **allow_blank** (*bool*) – Can field be assigned with blank string. Defaults to `False`.
- **min_length** (*int*) – Minimum length of field's values. Defaults to `None`.
- **max_length** (*int*) – Maximum length of field's values. Defaults to `None`.
- ****kwargs** – Other arguments from `Field`.

class aiomongodel.fields.**EmailField**(*, *regex=re.compile('^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$')*, ***kwargs*)

Email field.

Create Email field.

Parameters

- **regex** (*str, re.regex*) – Pattern for email address.
- ****kwargs** – Other arguments from `Field` and `StrField`.

class aiomongodel.fields.**IntField**(***kwargs*)

Integer field.

Create int field.

class aiomongodel.fields.**FloatField**(***kwargs*)

Float field.

Create float field.

class aiomongodel.fields.**DecimalField**(***kwargs*)

Decimal number field.

This field can be used only with MongoDB 3.4+.

Create Decimal field.

from_mongo(*value*)

Convert value from mongo format to python field format.

to_mongo(*value*)

Convert value to mongo format.

class aiomongodel.fields.**DateTimeField**(***kwargs*)

Date and time field based on datetime.datetime.

from_data(*value*)

Convert value from user provided data to field type.

Parameters **value** – Value provided by user.

Returns Converted value or value as is if error occured. If value is `None` return `None`.

class aiomongodel.fields.**EmbDocField**(*document_class*, ***kwargs*)

Embedded Document Field.

Create Embedded Document field.

Parameters

- **document_class** – A subclass of the `aiomongodel.EmbeddedDocument` class or string with absolute path to such class.
- ****kwargs** – Other arguments from `Field`.

from_data(*value*)
Convert value from user provided data to field type.

Parameters **value** – Value provided by user.

Returns Converted value or value as is if error occured. If value is `None` return `None`.

from_mongo(*value*)
Convert value from mongo format to python field format.

to_mongo(*value*)
Convert value to mongo format.

class aiomongodel.fields.**ListField**(*item_field*, *, *min_length=None*, *max_length=None*, ***kwargs*)
List field.

Create List field.

Parameters

- **item_field** (`Field`) – Instance of the field to reflect list items' type.
- **min_length** (*int*) – Minimum length of the list. Defaults to `None`.
- **max_length** (*int*) – Maximum length of the list. Defaults to `None`.
- ****kwargs** – Other arguments from `Field`.

Raises `TypeError` – If item_field is not instance of the `Field` subclass.

from_data(*value*)
Convert value from user provided data to field type.

Parameters **value** – Value provided by user.

Returns Converted value or value as is if error occured. If value is `None` return `None`.

from_mongo(*value*)
Convert value from mongo format to python field format.

to_mongo(*value*)
Convert value to mongo format.

class aiomongodel.fields.**RefField**(*document_class*, ***kwargs*)
Reference field.

Create Reference field.

Parameters

- **document_class** – A subclass of the `aiomongodel.Document` class or string with absolute path to such class.
- ****kwargs** – Other arguments from `Field`.

from_data(*value*)
Convert value from user provided data to field type.

Parameters **value** – Value provided by user.

Returns Converted value or value as is if error occured. If value is `None` return `None`.

from_mongo(*value*)
Convert value from mongo format to python field format.

to_mongo(*value*)
Convert value to mongo format.

class aiomongodel.fields.**SynonymField**(*original_field*)
Create synonym name for real field.

Create synonym for real document's field.

Parameters **original_field** – Field instance or string name of field.

Example:

```
class Doc(Document):
    _id = StrField()
    name = SynonymField(_id)

class OtherDoc(Document):
    # _id field will be added automaticly.
    obj_id = SynonymField('_id')
```

class aiomongodel.fields.**ObjectIdField**(***kwargs*)
ObjectId field.

from_data(*value*)
Convert value to ObjectId.

Parameters **value** (*ObjectId, str*) – ObjectId value or 24-character hex string.

Returns None or ObjectId value. If value is not ObjectId and can't be converted return as is.

1.3 QuerySet

QuerySet classes.

class aiomongodel.queryset.**MotorQuerySet**(*doc_class*, *db*, *session=None*)
QuerySet based on Motor query syntax.

aggregate(*pipeline*, ***kwargs*)
Return Aggregation cursor.

clone()
Return a copy of queryset.

count(*query={}*, ***kwargs*)
Count documents in collection.

count_documents(*query={}*, ***kwargs*)
Count documents in collection.

create(***kwargs*)
Create document.

Parameters ****kwargs** – fields of the document.

Returns Document instance.

create_indexes()
Create document's indexes defined in Meta class.

delete_many(*query*, ***kwargs*)
Delete many documents.

delete_one(*query*, ***kwargs*)
Delete one document.

find(*query={}*, **args*, *sort=None*, ***kwargs*)
Find documents by query.

Returns Cursor to get actual data.

Return type *MotorQuerySetCursor*

find_one(*query={}*, **args*, ***kwargs*)
Find one document.

Arguments are the same as for `motor.Collection.find_one`. This method does not returns `None` if there is no documents for given query but raises `aiomongodel.errors.DocumentNotFoundError`.

Returns Document model instance.

Raises *aiomongodel.errors.DocumentNotFoundError* – If there is no documents for given query.

get(*_id*, **args*, ***kwargs*)
Get document by its _id.

insert_many(**args*, ***kwargs*)
Insert many documents.

Returns List of inserted `_id` values.

Return type list

insert_one(**args*, ***kwargs*)
Insert one document.

Returns Inserted `_id` value.

Raises *aiomongodel.errors.DuplicateKeyError* – On duplicate key error.

replace_one(*query*, **args*, ***kwargs*)
Replace one document.

Returns Number of modified documents.

Return type int

Raises *aiomongodel.errors.DuplicateKeyError* – On duplicate key error.

update_many(*query*, **args*, ***kwargs*)
Update many documents.

Returns Number of modified documents.

Return type int

Raises *aiomongodel.errors.DuplicateKeyError* – On duplicate key error.

update_one(*query*, **args*, ***kwargs*)
Update one document.

Returns Number of modified documents.

Return type int

Raises *aiomongodel.errors.DuplicateKeyError* – On duplicate key error.

with_options(***kwargs*)
Change collection options.

class aiomongodel.queryset.**MotorQuerySetCursor**(*doc_class*, *cursor*)
Cursor based on motor cursor.

clone()
Get copy of this cursor.

to_list(*length*)
Return list of documents.

Parameters **length** – Number of items to return.

Returns List of document model instances.

Return type list

1.4 Errors

Aiomongodel errors and exceptions.

exception aiomongodel.errors.**AioMongodelException**
Base AioMongodel Exception class.

exception aiomongodel.errors.**DocumentNotFoundError**
Raised when document is not found in db.

exception aiomongodel.errors.**DuplicateKeyError**(*message*)
Raised on unique key constraint error.

Create error.

Parameters **message** (*str*) – String representation of pymongo.errors.DuplicateKeyError.

index_name
Name of the unique index which raised error.

exception aiomongodel.errors.**Error**
Base AioMongodel Error class.

exception aiomongodel.errors.**StopValidation**
Raised when validation of the field should be stopped.

exception aiomongodel.errors.**ValidationError**(*error=None*, *constraint=<object object>*)
Raised on model validation error.

Template for translation of error messages:

```
translation = {
    "field is required": "",
    "none value is not allowed": "",
    "blank value is not allowed": "",
    "invalid value type": "",
    "value does not match any variant": "",
    "value does not match pattern {constraint}": "",
    "length is less than {constraint}": "",
    "length is greater than {constraint}": "",
    "value is less than {constraint}": "",
    "value is greater than {constraint}": "",
    "value should be greater than {constraint}": "",
    "value should be less than {constraint}": "",
```

(continues on next page)

(continued from previous page)

```
    "list length is less than {constraint}": "",
    "list length is greater than {constraint}": "",
    "value is not a valid email address": "",
}
```

error
Can contain a simple error string or dict of nested validation errors.

constraint
A constraint value for validation error.

Create validation error.

Parameters

- **error** – Can be string or dict of {key => ValidationError}
- **constraint** – A constraint value for the error. If it's not empty it is used in error message formatting as `{constraint}`.

as_dict(*translation=None*)
Extract all errors from `self.error` attribute.

Parameters **translation** (*dict*) – A dict of translation for default validation error messages.

Returns If `self.error` is a string return as string. If `self.error` is a dict return dict of {key => ValidationError.as_dict()}

1.5 Utils

Common utils.

aiomongodel.utils.**snake_case**(*camel_case*)
Turn CamelCase string to snake_case.

aiomongodel.utils.**import_class**(*absolute_import_path*)
Import class by its path.

Class is stored in cache after importing.

Example:

```
Document = import_class('aiomongodel.document.Document')
```

Parameters **absolute_import_path** (*str*) – Absolute path for class to import.

Returns Class object.

Return type type

Raises `ImportError` – If class can't be imported by its path.

CHAPTER 2

aiomongodel

An asynchronous ODM similar to PyMODM on top of Motor an asynchronous Python MongoDB driver. Works on `Python 3.5` and up. Some features such as asynchronous comprehensions require at least `Python 3.6`. `aiomongodel` can be used with asyncio as well as with Tornado.

Usage of `session` requires at least MongoDB version 4.0.

2.1 Install

Install *aiomongodel* using *pip*:

```
pip install aiomongodel
```

2.2 Documentation

Read the docs.

2.3 Getting Start

2.3.1 Modeling

To create a model just create a new model class, inherit it from `aiomongodel.Document` class, list all the model fields and place a `Meta` class with model meta options. To create a subdocument, create a class with fields and inherit it from `aiomongodel.EmbeddedDocument`.

```
# models.py

from datetime import datetime
```

(continues on next page)

(continued from previous page)

```
from pymongo import IndexModel, DESCENDING

from aiomongodel import Document, EmbeddedDocument
from aiomongodel.fields import (
    StrField, BoolField, ListField, EmbDocField, RefField, SynonymField,
    IntField, FloatField, DateTimeField, ObjectIdField)

class User(Document):
    _id = StrField(regex=r'[a-zA-Z0-9_]{3, 20}')
    is_active = BoolField(default=True)
    posts = ListField(RefField('models.Post'), default=lambda: list())
    quote = StrField(required=False)

    # create a synonym field
    name = SynonymField(_id)

    class Meta:
        collection = 'users'

class Post(Document):
    # _id field will be added automatically as
    # _id = ObjectIdField(defalut=lambda: ObjectId())
    title = StrField(allow_blank=False, max_length=50)
    body = StrField()
    created = DateTimeField(default=lambda: datetime.utcnow())
    views = IntField(default=0)
    rate = FloatField(default=0.0)
    author = RefField(User, mongo_name='user')
    comments = ListField(EmbDocField('models.Comment'), default=lambda: list())

    class Meta:
        collection = 'posts'
        indexes = [IndexModel([('created', DESCENDING)])]
        default_sort = [('created', DESCENDING)]

class Comment(EmbeddedDocument):
    _id = ObjectIdField(default=lambda: ObjectId())
    author = RefField(User)
    body = StrField()

# `s` property of the fields can be used to get a mongodb string name
# to use in queries
assert User._id.s == '_id'
assert User.name.s == '_id'  # name is synonym
assert Post.title.s == 'title'
assert Post.author.s == 'user'  # field has mongo_name
assert Post.comments.body.s == 'comments.body'  # compound name
```

2.3.2 CRUD

```
from motor.motor_asyncio import AsyncIOMotorClient

async def go(db):
    # create model's indexes
```

(continues on next page)

(continued from previous page)

```
    await User.q(db).create_indexes()

    # CREATE
    # create using save
    # Note: if do_insert=False (default) save performs a replace
    # with upsert=True, so it does not raise if _id already exists
    # in db but replace document with that _id.
    u = await User(name='Alexandro').save(db, do_insert=True)
    assert u.name == 'Alexandro'
    assert u._id == 'Alexandro'
    assert u.is_active is True
    assert u.posts == []
    assert u.quote is None
    # using query
    u = await User.q(db).create(name='Ihor', is_active=False)

    # READ
    # get by id
    u = await User.q(db).get('Alexandro')
    assert u.name == 'Alexandro'
    # find
    users = await User.q(db).find({User.is_active.s: True}).to_list(10)
    assert len(users) == 2
    # using for loop
    users = []
    async for user in User.q(db).find({User.is_active.s: False}):
        users.append(user)
    assert len(users) == 1
    # in Python 3.6 an up use async comprehensions
    users = [user async for user in User.q(db).find({})]
    assert len(users) == 3

    # UPDATE
    u = await User.q(db).get('Ihor')
    u.is_active = True
    await u.save(db)
    assert (await User.q(db).get('Ihor')).is_active is True
    # using update (without data validation)
    # object is reloaded from db after update.
    await u.update(db, {'$push': {User.posts.s: ObjectId()}})

    # DELETE
    u = await User.q(db).get('Ihor')
    await u.delete(db)


loop = asyncio.get_event_loop()
client = AsyncIOMotorClient(io_loop=loop)
db = client.aiomongodel_test
loop.run_until_complete(go(db))
```

2.3.3 Validation

Use model's `validate` method to validate model's data. If there are any invalid data an `aiomongodel.errors.ValidationError` will raise.

Note: Creating model object or assigning it with invalid data does not raise errors! Be careful while saving model without validation.

```
class Model(Document):
    name = StrField(max_length=7)
    value = IntField(gt=5, lte=13)
    data = FloatField()

def go():
    m = Model(name='xxx', value=10, data=1.6)
    # validate data
    # should not raise any error
    m.validate()

    # invalid data
    # note that there are no errors while creating
    # model with invalid data
    invalid = Model(name='too long string', value=0)
    try:
        invalid.validate()
    except aiomongodel.errors.ValidationError as e:
        assert e.as_dict() == {
            'name': 'length is greater than 7',
            'value': 'value should be greater than 5',
            'data': 'field is required'
        }

        # using translation - you can translate messages
        # to your language or modify them
        translation = {
            "field is required": "This field is required",
            "length is greater than {constraint}": ("Length of the field "
                                                    "is greater than "
                                                    "{constraint} characters"),
            # see all error messages in ValidationError docs
            # for missed messages default messages will be used
        }
        assert e.as_dict(translation=translation) == {
            'name': 'Length of the field is greater than 7 characters',
            'value': 'value should be greater than 5',
            'data': 'This field is required'
        }
```

2.3.4 Querying

```
async def go(db):
    # find returns a cursor
    cursor = User.q(db).find({}, {'_id': 1}).skip(1).limit(2)
    async for user in cursor:
        print(user.name)
        assert user.is_active is None  # we used projection

    # find one
    user = await User.q(db).find_one({User.name.s: 'Alexandro'})
```

(continues on next page)

(continued from previous page)

```
    assert user.name == 'Alexandro'

    # update
    await User.q(db).update_many(
        {User.is_active.s: True},
        {'$set': {User.is_active.s: False}})

    # delete
    await User.q(db).delete_many({})
```

2.3.5 Models Inheritance

A hierarchy of models can be built by inheriting one model from another. A `aiomongodel.Document` class should be somewhere in hierarchy for model adn `aiomongodel.EmbeddedDocument` for subdocuments. Note that fields are inherited but meta options are not.

```
class Mixin:
    value = IntField()

class Parent(Document):
    name = StrField()

class Child(Mixin, Parent):
    # also has value and name fields
    rate = FloatField()

class OtherChild(Child):
    # also has rate and name fields
    value = FloatField() # overwrite value field from Mixin

class SubDoc(Mixin, EmbeddedDocument):
    # has value field
    pass
```

2.3.6 Models Inheritance With Same Collection

```
class Mixin:
    is_active = BoolField(default=True)

class User(Mixin, Document):
    _id = StrField()
    role = StrField()
    name = SynonymField(_id)

    class Meta:
        collection = 'users'

    @classmethod
    def from_mongo(cls, data):
        # create appropriate model when loading from db
        if data['role'] == 'customer':
            return super(User, Customer).from_mongo(data)
        if data['role'] == 'admin':
```

(continues on next page)

(continued from previous page)

```
            return super(User, Admin).from_mongo(data)

class Customer(User):
    role = StrField(default='customer', choices=['customer'])  # overwrite role field
    address = StrField()

    class Meta:
        collection = 'users'
        default_query = {User.role.s: 'customer'}

class Admin(User):
    role = StrField(default='admin', choices=['admin'])  # overwrite role field
    rights = ListField(StrField(), default=lambda: list())

    class Meta:
        collection = 'users'
        default_query = {User.role.s: 'admin'}
```

2.3.7 Transaction

```
from motor.motor_asyncio import AsyncIOMotorClient

async def go(db):
    # create collection before using transaction
    await User.create_collection(db)

    async with await db.client.start_session() as session:
        try:
            async with s.start_transaction():
                # all statements that use session inside this block
                # will be executed in one transaction

                # pass session to QuerySet
                await User.q(db, session=session).create(name='user')  # note session␣
↪param
                # pass session to QuerySet method
                await User.q(db).update_one(
                    {User.name.s: 'user'},
                    {'$set': {User.is_active.s: False}},
                    session=session)  # note session usage
                assert await User.q(db, session).count_documents({User.name.s: 'user'}
↪) == 1

                # session could be used in document crud methods
                u = await User(name='user2').save(db, session=session)
                await u.delete(db, session=session)

                raise Exception()  # simulate error in transaction block
         except Exception:
            # transaction was not committed
            assert await User.q(db).count_documents({User.name.s: 'user'}) == 0


loop = asyncio.get_event_loop()
client = AsyncIOMotorClient(io_loop=loop)
```

(continues on next page)

(continued from previous page)

```
db = client.aiomongodel_test
loop.run_until_complete(go(db))
```

2.4 License

The library is licensed under MIT License.

CHAPTER 3

Changelog

3.1 0.2.0 (2018-09-12)

Move requirements to motor>=2.0.

Remove `count` method from `MotorQuerySetCursor`.

Add session support to `MotorQuerySet` and `Document`.

Add `create_collection` method to `Document`.

Fix `__aiter__` of `MotorQuerySetCursor` for python 3.7.

Deprecate `count` method of `MotorQuerySet`.

Deprecate `create` method of `Document`.

3.2 0.1.0 (2017-05-19)

The first `aiomongodel` release.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

G

I

L

M

N

O

P

Q

R

S

T

U

V

W