Nefertari Documentation

Release

Brandicted

May 17, 2016

Contents

CHAPTER 1

Table of Contents

1.1 Base Classes

class nefertari_sqla.documents.**BaseMixin**

Represents mixin class for models.

Attributes:

_auth_fields: String names of fields meant to be displayed to authenticated users.

_public_fields: String names of fields meant to be displayed to non-authenticated users.

_hidden_fields: String names of fields meant to be hidden but editable. _nested_relationships: String names of relationship fields

that should be included in JSON data of an object as full included documents. If relationship field is not present in this list, this field's value in JSON will be an object's ID or list of IDs.

_nesting_depth: Depth of relationship field nesting in JSON. Defaults to 1(one) which makes only one level of relationship nested.

__weakref__

list of weak references to the object (if defined)

classmethod _clean_queryset(*queryset*)

Clean :queryset: from explicit limit, offset, etc.

New queryset is created by querying collection by IDs from passed queryset.

classmethod _delete_many(*items*, *request=None*, *synchronize_session=False*)

Delete :items: queryset or objects list.

When queryset passed, Query.delete() is used to delete it but first queryset is re-queried to clean it from explicit limit/offset/etc.

If some of the methods listed above were called, or :items: is not a Query instance, one-by-one items update is performed.

on_bulk_delete function is called to delete objects from index and to reindex relationships. This is done explicitly because it is impossible to get access to deleted objects in signal handler for 'after_bulk_delete' ORM event.

_is_modified()

Determine if instance is modified.

For instance to be marked as 'modified', it should:

- Have state marked as modified
- Have state marked as persistent
- Any of modified fields have new value

classmethod _pop_iterables(*params*)
Pop iterable fields' parameters from :params: and generate SQLA expressions to query the database.

Iterable values are found by checking which keys from :params: correspond to names of List fields on model. If ListField uses the *postgresql.ARRAY* type, the value is wrapped in a list.

classmethod _update_many(*items*, *params*, *request=None*, *synchronize_session='fetch'*)
Update :items: queryset or objects list.

When queryset passed, Query.update() is used to update it but first queryset is re-queried to clean it from explicit limit/offset/etc.

If some of the methods listed above were called, or :items: is not a Query instance, one-by-one items update is performed.

classmethod add_field_names(*query_set*, *requested_fields*)
Convert list of tuples to dict with proper field keys.

classmethod apply_fields(*query_set*, *_fields*)
Apply fields' restrictions to *query_set*.

First, fields are split to fields that should only be included and fields that should be excluded. Then excluded fields are removed from included fields.

classmethod autogenerate_for(*model*, *set_to*)
Setup *after_insert* event handler.

Event handler is registered for class :model: and creates a new instance of :cls: with a field :set_to: set to an instance on which the event occured.

classmethod check_fields_allowed(*fields*)
Check if *fields* are allowed to be used on this model.

classmethod filter_fields(*params*)
Filter out fields with invalid names.

classmethod filter_objects(*objects*, *first=False*, ***params*)
Perform query with :params: on instances sequence :objects:

Parameters

- **object** – Sequence of :cls: instances on which query should be run.
- **params** – Query parameters to filter :objects:.

classmethod get_collection(***params*)
Query collection and return results.

Notes: * Before validating that only model fields are present in params,

reserved params, query params and all params starting with double underscore are dropped.

•Params which have value "_all" are dropped.

•When `_count` param is used, objects count is returned before applying offset and limit.

Parameters

- **_strict** (*bool*) – If True `params` are validated to contain only fields defined on model, exception is raised if invalid fields are present. When False - invalid fields are dropped. Defaults to `True`.
- **_item_request** (*bool*) – Indicates whether it is a single item request or not. When True and DataError happens on DB request, JHTTPNotFound is raised. JHTTPBadRequest is raised when False. Defaults to `False`.
- **_sort** (*list*) – Field names to sort results by. If field name is prefixed with "-" it is used for "descending" sorting. Otherwise "ascending" sorting is performed by that field. Defaults to an empty list in which case sorting is not performed.
- **_fields** (*list*) – Names of fields which should be included or excluded from results. Fields to excluded should be prefixed with "-". Defaults to an empty list in which case all fields are returned.
- **_limit** (*int*) – Number of results per page. Defaults to None in which case all results are returned.
- **_page** (*int*) – Number of page. In conjunction with `_limit` is used to calculate results offset. Defaults to None in which case it is ignored. Params `_page` and ‘‘_start‘ are mutually exclusive.
- **_start** (*int*) – Results offset. If provided `_limit` and `_page` params are ignored when calculating offset. Defaults to None. Params `_page` and `_start` are mutually exclusive. If not offset-related params are provided, offset equals to 0.
- **query_set** (*Query*) – Existing queryset. If provided, all queries are applied to it instead of creating new queryset. Defaults to None.
- **_count** – When provided, only results number is returned as integer.
- **_explain** – When provided, query performed(SQL) is returned as a string instead of query results.
- **_raise_on_empty** (*bool*) – When True JHTTPNotFound is raised if query returned no results. Defaults to False in which case error is just logged and empty query results are returned.

Returns Query results as `sqlalchemy.orm.query.Query` instance. May be sorted, offset, limited.

Returns Dict of {'field_name': fieldval}, when `_fields` param is provided.

Returns Number of query results as an int when `_count` param is provided.

Returns String representing query ran when `_explain` param is provided.

Raises

- **JHTTPNotFound** – When `_raise_on_empty=True` and no results found.
- **JHTTPNotFound** – When `_item_request=True` and `sqlalchemy.exc.DataError` exception is raised during DB query. Latter exception is raised when querying DB with an identifier of a wrong type. E.g. when querying Int field with a string.
- **JHTTPBadRequest** – When `_item_request=False` and `sqlalchemy.exc.DataError` exception is raised during DB query.
- **JHTTPBadRequest** – When `sqlalchemy.exc.InvalidRequestError` or `sqlalchemy.exc.IntegrityError` errors happen during DB query.

classmethod get_es_mapping(*_depth=None*, *types_map=None*)
Generate ES mapping from model schema.

classmethod get_item(***params*)
Get single item and raise exception if not found.

Exception raising when item is not found can be disabled by passing `_raise_on_empty=False` in params.

Returns Single collection item as an instance of `cls`.

classmethod get_null_values()
Get null values of :cls: fields.

get_related_documents(*nested_only=False*)
Return pairs of (Model, istances) of relationship fields.

Pair contains of two elements:

Model Model class object(s) contained in field.

instances Model class instance(s) contained in field

Parameters **nested_only** – Boolean, defaults to False. When True, return results only contain data for models on which current model and field are nested.

classmethod pk_field()
Get a primary key field name.

class nefertari_sqla.documents.**BaseDocument**(***kwargs*)
Base class for SQLA models.

Subclasses of this class that do not define a model schema should be abstract as well (__abstract__ = True).

classmethod get_field_params(*field_name*)
Get init params of column named :field_name:.

class nefertari_sqla.documents.**ESBaseDocument**(***kwargs*)
Base class for SQLA models that use Elasticsearch.

Subclasses of this class that do not define a model schema should be abstract as well (__abstract__ = True).

1.2 Serializers

class nefertari_sqla.serializers.**JSONEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)
JSON encoder class to be used in views to encode response.

class nefertari_sqla.serializers.**ESJSONSerializer**
JSON encoder class used to serialize data before indexing to ES.

1.3 Fields

class nefertari_sqla.fields.**IntegerField**(**args*, ***kwargs*)

_sqla_type_cls
alias of LimitedInteger

class nefertari_sqla.fields.**BigIntegerField**(**args*, ***kwargs*)

_sqla_type_cls
alias of LimitedBigInteger

class nefertari_sqla.fields.**SmallIntegerField**(**args*, ***kwargs*)

_sqla_type_cls
alias of LimitedSmallInteger

class nefertari_sqla.fields.**BooleanField**(**args*, ***kwargs*)

process_type_args(*kwargs*)

Changed: constraint_name -> name

class nefertari_sqla.fields.**DateField**(**args*, ***kwargs*)

class nefertari_sqla.fields.**DateTimeField**(**args*, ***kwargs*)

class nefertari_sqla.fields.**FloatField**(**args*, ***kwargs*)

_sqla_type_cls
alias of LimitedFloat

class nefertari_sqla.fields.**StringField**(**args*, ***kwargs*)

_sqla_type_cls
alias of LimitedString

process_type_args(*kwargs*)

Changed: max_length -> length

class nefertari_sqla.fields.**TextField**(**args*, ***kwargs*)

_sqla_type_cls
alias of LimitedText

class nefertari_sqla.fields.**UnicodeField**(**args*, ***kwargs*)

_sqla_type_cls
alias of LimitedUnicode

class nefertari_sqla.fields.**UnicodeTextField**(**args*, ***kwargs*)

_sqla_type_cls
alias of LimitedUnicodeText

class nefertari_sqla.fields.**ChoiceField**(**args*, ***kwargs*)

_sqla_type_cls
alias of Choice

class nefertari_sqla.fields.**BinaryField**(**args*, ***kwargs*)

class nefertari_sqla.fields.**DecimalField**(**args*, ***kwargs*)

_sqla_type_cls
alias of LimitedNumeric

class nefertari_sqla.fields.**TimeField**(**args*, ***kwargs*)

class nefertari_sqla.fields.**PickleField**(**args*, ***kwargs*)

class nefertari_sqla.fields.**IntervalField**(**args*, ***kwargs*)

class nefertari_sqla.fields.**IdField**(**args*, ***kwargs*)
Just a subclass of IntegerField that must be used for fields that represent database-specific 'id' field.

class nefertari_sqla.fields.**ForeignKeyField**(**args*, ***kwargs*)
Integer ForeignKey field.

This is the place where *ondelete* rules kwargs should be passed. If you switched from the mongodb engine, copy the same *ondelete* rules you passed to mongo's *Relationship* constructor.

ondelete kwargs may be kept in both fields with no side-effects when switching between the sqla and mongo engines.

Developers are not encouraged to change the value of this field on model to add/update relationship. Use *Relationship* constructor with backreference settings instead.

__init__(**args*, ***kwargs*)
Override to determine *self._sqla_type_cls*.

Type is determined using 'ref_column_type' value from :kwargs:. Its value must be a *Field class of a field that is being referenced by FK field or a *_sqla_type_cls* of that *Field cls.

_generate_schema_item(*cleaned_kw*)
Override default implementation to generate 'ondelete' and 'onupdate' arguments.

_get_referential_action(*kwargs*, *key*)
Determine/translate generic rule name to SQLA-specific rule.

Output rule name is a valid SQL Referential action name. If *ondelete* kwarg is not provided, no referential action will be created.

Valid kwargs for *ondelete* kwarg are: CASCADE Translates to SQL as *CASCADE* RESTRICT Translates to SQL as *RESTRICT* NULLIFY Translates to SQL as 'SET NULL

Not supported SQL referential actions: *NO ACTION*, *SET DEFAULT*

_schema_class
alias of ForeignKey

class nefertari_sqla.fields.**Relationship**
Thin wrapper around relationship.

The goal of this wrapper is to allow passing both relationship and backref arguments to a single function. Backref arguments should be prefixed with '**backref_**'. This function splits relationship-specific and backref-specific arguments and makes a call like:

relationship(..., ..., backref=backref(...))

Lazy setting is set to 'immediate' on the 'One' side of One2One or

One2Many relationships. This is done both for relationship itself and backref so ORM 'after_update' events are fired when relationship is updated. For backref 'uselist' is assumed to be False by default.

From SQLAlchemy docs: immediate - items should be loaded as the parents are loaded, using a separate SELECT statement, or identity map fetch for simple many-to-one references.

class nefertari_sqla.fields.**DictField**(**args*, ***kwargs*)

_sqla_type_cls
alias of JSONType

class nefertari_sqla.fields.**ListField**(**args*, ***kwargs*)

_sqla_type_cls
alias of ChoiceArray

process_type_args(*kwargs*)
Covert field class to its *_sqla_type_cls*.

StringField & UnicodeField are replaced with corresponding Text fields because when String* fields are used, SQLA creates db column of postgresql type 'varying[]'. But when querying that column with text, requested text if submited as 'text[]'.

Changed: item_type field class -> item_type field type

1.4 Changelog

- #90: Deprecated '_version' field
- : Cosmetic name changes in preparation of engine refactoring
- : Added '_nesting_depth' property in models to control the level of nesting, default is 1
- : Nested relationships are now indexed in bulk in ElasticSearch
- : Fixed ES double indexation bug
- : Fixed a bug when using reserved query params with GET tunneling
- : Fixed a bug with BaseMixin.filter_objects() not correctly applying additional filters passed to it
- : Fixed a bug with _update_many() and _delete_many() not working with querysets returned by get_collection()
- : Fixed a bug whereby objects could not be deleted from within processors
- : Disabled Elasticsearch indexing of DictField to allow storing arbitrary JSON data
- : Removed 'updated_at' field from engine
- : Fixed bug with Elasticsearch re-indexing of nested relationships
- : Added python3 support
- : Forward compatibility with nefertari releases
- : Fixed race condition in Elasticsearch indexing
- : Fixed bug with Elasticsearch indexing of nested relationships
- : Fixed password minimum length support by adding before and after validation processors
- : Fixed a bug whereby Relationship could not be created without a backref

- : Fixed ES mapping error when values of field were all null
- : Fixed multiple foreign keys to same model
- : Fixed posting to singular resources e.g. /api/users/<username>/profile
- : Fixed login issue
- : Fixed slow queries to backrefs
- : Relationship indexing

Symbols

A

B

C

D

E