EAV-Django Documentation

Release 1.4.6

Andrey Mikhaylenko

Jun 12, 2017

Contents

1 Priorities **3**

2 Features **5**

3 Examples **7**

4 Data types **9**

5 Documentation **11**

6 Dependencies **13**

7 Alternatives, Forks **15**

8 Author **17**

9 Licensing **19**

10 Details **21**
10.1 API Reference 21
10.2 Contributors 22

11 Indices and tables **25**

Python Module Index **27**

EAV-Django is a reusable Django application which provides an implementation of the Entity-Attribute-Value data model.

> Entity-Attribute-Value model (EAV), also known as object-attribute-value model and open schema which is used in circumstances where the number of attributes (properties, parameters) that can be used to describe a thing (an "entity" or "object") is potentially very vast, but the number that will actually apply to a given entity is relatively modest.

(See the Wikipedia article for more details.)

EAV-Django works fine with traditional RDBMS (tested on SQLite and MySQL).

CHAPTER 1

Priorities

The application grew from an online shop project, so it is pretty practical and not just an academic exercise. The main priorities were:

1. flexibility of data,
2. efficiency of queries, and
3. maximum maintainability without editing the code.

Of course this implies trade-offs, and the goal was to find the least harmful combination for the general case.

CHAPTER 2

Features

All provided models are abstract, i.e. EAV-Django does not store any information in its own tables. Instead, it provides a basis for your own models which will have support for EAV out of the box.

The EAV API includes:

- *Create/update/access:* model instances provide standart API for both "real" fields and EAV attributes. The abstraction, however, does not stand in your way and provides means to deal with the underlying stuff.
- *Query:* BaseEntityManager includes uniform approach in *filter()* and *exclude()* to query "real" and EAV attributes.
- Customizable *schemata for attributes*.
- *Admin:* all dynamic attributes can be represented and modified in the Django admin with no or little effort (using *eav.admin.BaseEntityAdmin*). Schemata can be edited separately, as ordinary Django model objects.
- *Facets:* facet search is an important feature of online shops, catalogues, etc. Basically you will need a form representing a certain subset of model attributes with appropriate widgets and choices so that the user can choose desirable values of some properties, submit the form and get a list of matching items. In general case django-filter would do, but it won't work with EAV, so EAV-Django provides a complete set of tools for that.

CHAPTER 3

Examples

Let's define an EAV-friendly model, create an EAV attribute and see how it behaves. By "EAV attributes" I mean those stored in the database as separate objects but accessed and searched in such a way as if they were columns in the entity's table:

```
from django.db import models
from eav.models import BaseEntity, BaseSchema, BaseAttribute

class Fruit(BaseEntity):
    title = models.CharField(max_length=50)

class Schema(BaseSchema):
    pass

class Attr(BaseAttribute):
    schema = models.ForeignKey(Schema, related_name='attrs')

# in Python shell:

# define attribute named "colour"
>>> colour = Schema.objects.create(
...     title = 'Colour',
...     name = 'colour',               # omit to populate/slugify from title
...     datatype = Schema.TYPE_TEXT
... )

# create an entity
>>> e = Fruit.objects.create(title='Apple', colour='green')

# define "real" and EAV attributes the same way
>>> e.title
'Apple'
>>> e.colour
'green'

>>> e.save()    # deals with EAV attributes automatically
```

```
# list EAV attributes as Attr instances
>>> e.attrs.all()
[<Attr: Apple: Colour "green">]

# search by an EAV attribute as if it was an ordinary field
>>> Fruit.objects.filter(colour='yellow')
[<Fruit: Apple>]

# all compound lookups are supported
>>> Fruit.objects.filter(colour__contains='yell')
[<Fruit: Apple>]
```

Note that we can access, modify and query *colour* as if it was a true Entity field, but at the same time its name, type and even existance are completely defined by a Schema instance. A Schema object can be understood as a class, and related Attr objects are its instances. In other words, Schema objects are like CharField, IntegerField and such, only defined on data level, not hard-coded in Python. And they can be "instantiated" for any Entity (unless you put custom constraints which are outside of EAV-Django's area of responsibility).

The names of attributes are defined in related schemata. This can lead to fears that once a name is changed, the code is going to break. Actually this is not the case as names are only directly used for manual lookups. In all other cases the lookups are constructed without hard-coded names, and the EAV objects are interlinked by primary keys, not by names. The names are present if forms, but the forms are generated depending on current state of metadata, so you can safely rename the schemata. What you *can* break from the admin interface is the types. If you change the data type of a schema, all its attributes will remain the same but will use another column to store their values. When you restore the data type, previously stored values are visible again.

You can find more examples in the source code: see directory "example/" and the tests.

CHAPTER 4

Data types

Metadata-driven structure extends flexibility but implies some trade-offs. One of them is increased number of JOINs (and, therefore, slower queries). Another is fewer data types. Theoretically, we can support all data types available for a storage, but in practice it would mean creating many columns per attribute with just a few being used – exactly what we were trying to avoid by using EAV. This is why EAV-Django only supports some basic types (though you can extend this list if needed):

- Schema.TYPE_TEXT, a TextField;
- Schema.TYPE_FLOAT, a FloatField;
- Schema.TYPE_DATE, a DateField;
- Schema.TYPE_BOOL, a NullBooleanField;
- Schema.TYPE_MANY for multiple choices (i.e. lists of values).

All EAV attributes are stored as records in a table with unique combinations of references to entities and schemata. (Entity is referenced through the contenttypes framework, schema is referenced via foreign key.) In other words, there can be only one attribute with given entity *and* schema. The schema is a definition of attribute. The schema defines name, title, data type and a number of other properties which apply to any attribute of this schema. When we access or search EAV attributes, the EAV machinery always uses schemata as attributes metadata. Why? Because the attribute's name is stored in related schema, and the value is stored in *a* column of the attributes table. We don't know which column it is until we look at metadata.

In the example provided above we've only played with a text attribute. All other types behave exactly the same except for TYPE_MANY. The many-to-many is a special case as it involves an extra model for choices. EAV-Django provides an abstract model but requires you to define a concrete model (e.g. Choice), and point to it from the attribute model (i.e. put foreign key named "choice"). The Choice model will also have to point at Schema. Check the tests for an example.

CHAPTER 5

Documentation

Currently there is no tutorial. Still, the code itself is rather well-documented and the whole logic is pretty straightforward.

Please see:

- tests, as they contain good examples of model definitions and queries;
- the bundled example ("grocery shop", comes with fixtures);
- the discussion group.

CHAPTER 6

Dependencies

In theory, Python 2.5 to 2.7 is supported; however, the library is only tested against **Python 2.6 and 2.7**.

You'll also need **Django 1.1 or newer** and a couple of small libraries: django_autoslug and django_view_shortcuts. This is usually handled automatically by the installer.

CHAPTER 7

Alternatives, Forks

django-eav A fork of eav-django that became a new app. Doesn't seem to be actively developed but is probably better in certain aspects. The original author of eav-django encourages users to give this app a try, too.

CHAPTER 8

Author

This application was initially created by Andrey Mikhaylenko. For complete list of contributors consult the AUTHORS file.

Please feel free to file issues and/or submit patches.

CHAPTER 9

Licensing

EAV-Django is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

EAV-Django is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this program; see the file COPYING.LESSER. If not, see GNU licenses.

CHAPTER 10

Details

API Reference

Admin

class eav.admin.**BaseEntityAdmin**(*model*, *admin_site*)
Base class for entity admin classes.

render_change_form(*request*, *context*, ***kwargs*)
Wrapper for ModelAdmin.render_change_form. Replaces standard static AdminForm with an EAV-friendly one. The point is that our form generates fields dynamically and fieldsets must be inferred from a prepared and validated form instance, not just the form class. Django does not seem to provide hooks for this purpose, so we simply wrap the view and substitute some data.

class eav.admin.**BaseSchemaAdmin**(*model*, *admin_site*)
Base class for schema admin classes.

class eav.admin.**BaseEntityInline**(*parent_model*, *admin_site*)
Inline model admin that works correctly with EAV attributes. You should mix in the standard StackedInline or TabularInline classes in order to define formset representation, e.g.:

```
class ItemInline(BaseEntityInline, StackedInline):
    model = Item
    form = forms.ItemForm
```

formset
alias of BaseEntityInlineFormSet

Fields

class eav.fields.**RangeField**(**args*, ***kwargs*)
A multi-value field which consists of tho float fields.

widget
alias of `RangeWidget`

Forms

class `eav.forms.`**BaseSchemaForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*)
Base class for schema forms.

clean_name()
Avoid name clashes between static and dynamic attributes.

class `eav.forms.`**BaseDynamicEntityForm**(*data=None*, **args*, ***kwargs*)
ModelForm for entity with support for EAV attributes. Form fields are created on the fly depending on Schema defined for given entity instance. If no schema is defined (i.e. the entity instance has not been saved yet), only static fields are used. However, on form validation the schema will be retrieved and EAV fields dynamically added to the form, so when the validation is actually done, all EAV fields are present in it (unless Rubric is not defined).

check_eav_allowed()
Returns True if dynamic attributes can be added to this form. If False is returned, only normal fields will be displayed.

save(*commit=True*)
Saves this `form`'s cleaned_data into model instance `self.instance` and related EAV attributes.

Returns `instance`.

Object Managers

Widgets

class `eav.widgets.`**RangeWidget**(*attrs=None*)
Represents a range of numbers.

Contributors

EAV-Django was originally created by:

- Andrey Mikhaylenko <neithere@gmail.com>.

And here is a probably incomplete list of contributors – people who have submitted ideas, patches, reported bugs, added translations and generally made EAV-Django better:

- Danila Shtan
- Janosch Scharlipp
- Jordi Llonch
- alTus
- Felipe Vieira
- Adrien Lemaire

- Igor Tokarev
- Vladimir Korsun
- Jon Atkinson
- Your Name Here ;)

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

eav, 21
eav.admin, 21
eav.fields, 21
eav.forms, 22
eav.managers, 22
eav.widgets, 22

Index

B

BaseDynamicEntityForm (class in eav.forms), 22
BaseEntityAdmin (class in eav.admin), 21
BaseEntityInline (class in eav.admin), 21
BaseSchemaAdmin (class in eav.admin), 21
BaseSchemaForm (class in eav.forms), 22

C

check_eav_allowed() (eav.forms.BaseDynamicEntityForm method), 22
clean_name() (eav.forms.BaseSchemaForm method), 22

E

eav (module), 21
eav.admin (module), 21
eav.fields (module), 21
eav.forms (module), 22
eav.managers (module), 22
eav.widgets (module), 22

F

formset (eav.admin.BaseEntityInline attribute), 21

R

RangeField (class in eav.fields), 21
RangeWidget (class in eav.widgets), 22
render_change_form() (eav.admin.BaseEntityAdmin method), 21

S

save() (eav.forms.BaseDynamicEntityForm method), 22

W

widget (eav.fields.RangeField attribute), 21