django-composite-foreignkey Documentation

Release 1.1.0

Darius BERNARD

Oct 04, 2018

Contents

Contents:

CHAPTER 1

Installation

1. Install using pip:

   ```
   pip install django-composite-foreignkey
   ```

2. Alternatively, you can install download or clone this repo and call

   ```
   pip install -e .
   ```

After installation, the *Quickstart* will get you on your way to using `django-bootstrap3`.

CHAPTER 2

Quickstart

After *Installation*, you can use `django-composite-foreignkey` in your models.

the django-composite-foreignkey give you two fields : CompositeForeignKey and CompositeOneToOneField. each one has the same behavior: it don't create a field on the database, but use a/some existings ones.

2.1 Example simple composite ForeignKey models

2.1.1 CompositeForeignKey

2.1.2 CompositeOneToOneField

these 2 models is linked by either a CompositeForeignKey or a CompositeOneToOneField on Contact.customer. there is no *customer_id* field, but it use instead the shared fields company and customer_id. CompositeForeignKey support a advanced mapping which allow the fields from both models to no beeing named identicaly.

in the prevous exemple, the folowing fields is linked :

Contact	Customer
company_code	company
customer_code	company_code

where a «normal» ForeignKey shoud be :

Contact	Customer
customer_id	pk

Note: you can provide the to_fields attribute as a set instead of a dict if ALL fields is a simple linke to the related model and no special value is required.

```
to_fields={"company", "customer_id"}
```

is equivalent to

```
to_fields={"company": "company", "customer_id": "customer_id"}
```

Extra	Customer
company	
customer_id	

2.2 Example advanced composite ForeignKey models

in this exemple, the Address Model can be used by either Supplier OR Customer. the linked fields is for Customer :

Customer	Address
company	company
customer_id	customer_id
RawFieldValue("C")	type_tiers

The model Address have a field named «type_tiers» that allow to dinstinguish if the «tiers_id» is for a Supplier or a Customer. si the Customer model will always have an address with «S» in the «type_tiers» field. so be it via the *RawFieldValue* which tel exactly that : don't search on the table, the value is always «C».

for convenience, a oposit version of *RawFieldValue* exists and mean «search on the table field X». it is *LocalFieldValue("X")*.

so the class Supplier could be wrote:

We also can refer by CompositeForeignKey in more flexible way using FunctionBasedFieldValue instead of RawFieldValue:

```
from django.conf import global_settings
from django.utils import translation

from compositefk.fields import CompositeForeignKey

class Supplier(models.Model):
    company = models.IntegerField()
    supplier_id = models.IntegerField()


class SupplierTranslations(models.Model):
    master = models.ForeignKey(
        Supplier,
        on_delete=CASCADE,
        related_name='translations',
        null=True,
    )
    language_code = models.CharField(max_length=255, choices=global_settings.
↪LANGUAGES)
    name = models.CharField(max_length=255)
    title = models.CharField(max_length=255)
```

(continues on next page)

(continued from previous page)

```
    class Meta:
        unique_together = ('language_code', 'master')


active_translations = CompositeForeignKey(
    SupplierTranslations,
    on_delete=DO_NOTHING,
    to_fields={
        'master_id': 'id',
        'language_code': FunctionBasedFieldValue(translation.get_language)
    })


active_translations.contribute_to_class(Supplier, 'active_translations')
```

in this example, the Supplier Model joins with SupplierTranslations in current active language and supplier_instance.active_translations.name will return different names depend on which language was activated by translation.activate(..):

```
translation.activate('en')
print Supplier.objects.get(id=1).active_translations.name
translation.activate('your_language_code')
print Supplier.objects.get(id=1).active_translations.name
```

output should be:

- 'en_language_name'
- 'your_language_name'

2.3 Treate specific values as None

sometimes, some database is broken and some values should be treated as None to make sur no query will be made. ie if company code is «-1» instead of None, the query shall not seach for related model with company = -1 since this is an old aplicative exception.

you just have one thing to do that : null_if_equal

in this exemple, if company is -1, OR customer_id is -1 too, no query will be made and custome.address will be equal to None. it is the same behavior as if a normal foreignkey address had address_id = None.

Note: you must allow null value to permit that (which will not have any impact on database).

Note: these cases should not be possible on database that use ForeignKey constraint. but with some legacy database that won't, this feathure is mandatory to bypass the headarch comming with broken logic on special values.

2.4 Set Specific attribute to None

Sometimes, all fields used in the composite relation is not only used for this one. in our Contact class, the company can be used in other fields. you can use the arguments *nullable_fields* to give the list of fields to set to null in case you wante to remove the link. since if one of the composite field is resolved to None, the field will return None.

so Contact.customer = None is equal to Contact.customer_code = None if nullable_fields=[“customer_code”]

nullable_fields can be a dict, which provide the value to put instead of None of each updated fields, which can synergize well with *null_if_equal*

2.5 Test application

The test application provides a number of useful examples.

https://github.com/onysos/django-composite-foreignkey/tree/master/testapp/

CHAPTER 3

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

3.1 Types of Contributions

3.1.1 Report Bugs

Report bugs at https://github.com/onysos/django-composite-foreignkey/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.
- maybe some fixture to reproduce the bug

3.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

3.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

3.1.4 Write Documentation

django-composite-foreignkey could always use more documentation, whether as part of the official django-composite-foreignkey docs, in docstrings, or even on the web in blog posts, articles, and such.

3.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/onysos/django-composite-foreignkey/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.2 Get Started!

Ready to contribute? Here's how to set up *django-composite-foreignkey* for local development.

1. Fork the *django-composite-foreignkey* repo on GitHub.
2. Clone your fork locally:

```
$ git clone https://github.com/your_username_here/django-composite-foreignkey.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-composite-foreignkey
$ cd django-composite-foreignkey/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 compositefk tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/onysos/django-composite-foreignkey/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 4

known issues

since this fields use multiple fields as identifier, we don't have a field like *fieldname_id* like normal ForeignKey. some libs assert this sort of field exists and sometimes, we can't tel them otherwise. so you mill need to hack a little some part of your code to make sur your fields is well treated.

4.1 Django Rest Framework

4.1.1 version

tested on django 1.8 and rest_framework 3.2.4

4.1.2 error

TypeError: <MyModel: XXXXXXX> is not JSON serializable

4.1.3 explication

the serializer will try to get the pk of the CompositeForeignKey. for a normal FK, it will git the *fieldname_id*, but for us, it is impossible.

4.1.4 fix

the best way of fixing this is to override the models *serializable_value*

from :

```
def serializable_value(self, field_name):
    """
    Returns the value of the field name for this instance. If the field is
    a foreign key, returns the id value, instead of the object. If there's
    no Field object with this name on the model, the model attribute's
    value is returned directly.

    Used to serialize a field's value (in the serializer, or form output,
    for example). Normally, you would just access the attribute directly
    and not use this method.
    """
    try:
        field = self._meta.get_field(field_name)
    except FieldDoesNotExist:
        return getattr(self, field_name)
    return getattr(self, field.attname)
```

to :

```
def serializable_value(self, field_name):
    try:
        field = self._meta.get_field(field_name)
    except FieldDoesNotExist:
        return getattr(self, field_name)
    if isinstance(field, CompositeForeignKey):
        return getattr(self, field.attname).pk
    return getattr(self, field.attname)
```

this will just, in case of a CompositeForeignKey, get the related model pk instead of falsly returning the original model.

CHAPTER 5

Full README

5.1 django-composite-foreignkey

allow to create a django foreignkey that don't link with pk of other model, but with multi column matching local model columns or fixed values. some databases have a composite Primary Key, leading to impossiblity for a django foreign key to be used.

today, Django don't support Composite Primary Key see ticket and ForeignKey don't support multicolumn. but fortunaly, the base class of ForeignKey support it well, so this lib just add a little wrapper around ForeignObject to make it more usefull. the real add of this implementation is that is support the customisation of the link with Raw values.

this implementation of CompositeForeignKey skip the complexity of Composite Primary Key by forcing the providing of the corresponding column of the other model, not forcefully a PrimaryKey.

5.1.1 Installation

1. Install using pip:

   ```
   pip install django-composite-foreignkey
   ```

2. Alternatively, you can install download or clone this repo and call

   ```
   pip install -e .
   ```

5.1.2 Example

you have this model

```
class Customer(models.Model):

    company = models.IntegerField()
    customer_id = models.IntegerField()
```

(continues on next page)

(continued from previous page)

```
    name = models.CharField(max_length=255)
    address = CompositeForeignKey(Address, on_delete=CASCADE, to_fields={
        "tiers_id": "customer_id",
        "company": LocalFieldValue("company"),
        "type_tiers": RawFieldValue("C")
    })

    class Meta(object):
        unique_together = [
            ("company", "customer_id"),
        ]


class Contact(models.Model):
    company_code = models.IntegerField()
    customer_code = models.IntegerField()
    surname = models.CharField(max_length=255)
    # virtual field
    customer = CompositeForeignKey(Customer, on_delete=CASCADE, related_name='contacts
↪', to_fields={
        "customer_id": "customer_code",
        "company": "company_code"
    })
```

you can use Contact.customer like any ForeignKey, but behinde the scene, it will query the Customer Table using company and customer id's.

5.1.3 Documentation

The full documentation is at http://django-composite-foreignkey.readthedocs.org/en/latest/.

5.1.4 Requirements

- Python 2.7, 3.4, 3.5, 3.6, 3.7
- Django 1.11, 2.0, 2.1

Contributions and pull requests for other Django and Python versions are welcome.

5.1.5 Bugs and requests

If you have found a bug or if you have a request for additional functionality, please use the issue tracker on GitHub.

https://github.com/onysos/django-composite-foreignkey/issues

5.1.6 License

You can use this under GPLv3.

5.1.7 Author

Original author & Development lead: Darius BERNARD.

5.1.8 Thanks

Thanks to django for this amazing framework. And thanks to django-bootstrap3 to the structure of the apps.