django-generic-aggregation Documentation

Release 0.3.2

charles leifer

January 22, 2016

Contents

annotate() and aggregate() for generically-related data. also a handy function for filtering GFK-model querysets.

Note: Use django's GenericRelation where possible, as this can make the queries generated more efficient by using a JOIN rather than a subquery.

CHAPTER 1

installation

```
# install from pypi
pip install django-generic-aggregation

# or install via git
pip install -e git+git://github.com/coleifer/django-generic-aggregation.git#egg=generic_aggregation
```

CHAPTER 2

examples

The examples below assume the following simple models:

```
class Rating(models.Model):
    rating = models.IntegerField()
    object_id = models.IntegerField()
    content_type = models.ForeignKey(ContentType)
    content_object = GenericForeignKey(ct_field='content_type', fk_field='object_id')

class Food(models.Model):
    name = models.CharField(max_length=50)
    ratings = generic.GenericRelation(Rating) # reverse generic relation
```

You want to figure out which items are highest rated (*generic_annotate()*)

```
from django.db.models import Avg

food_qs = Food.objects.filter(name__startswith='a')
generic_annotate(food_qs, Rating, Avg('ratings__rating'))

# you can mix and match queryset / model
generic_annotate(food_qs, Rating.objects.all(), Avg('ratings__rating'))
```

You want the average rating for all foods that start with 'a' (*generic_aggregate()*)

```
food_qs = Food.objects.filter(name__startswith='a')
generic_aggregate(food_qs, Rating, Avg('ratings__rating'))
```

You want to only display ratings for foods that start with 'a' (*generic_filter()*)

food_qs = Food.objects.filter(name__startswith='a') generic_filter(Rating.objects.all(), food_qs)

CHAPTER 3

important detail

As you may have noted in the above examples (at least those using annotate and aggregate), the aggregate we pass in is prefixed with `ratings__`. The double-underscore prefix refers to the `ratings` attribute of the Food model, which is a `django.contrib.contenttypes.generic.GenericRelation` instance. We are querying *across* that relation to the field on the Ratings model that we are interested in. When possible, use a GenericRelation and construct your queries in this manner.

If you do not have a GenericRelation on the model being queried, it will use a "fallback" method that will return the correct results, though queried in a slightly different manner (a subquery will be used as opposed to a left outer join).

If for some reason the Generic Foreign Key's "object_id" field is of a different type than the Primary Key of the related model – which is probably the case if you're using django.contrib.comments, as it uses a TextField – a `CAST` expression is required by some RDBMS'. Django will not put it there for you, so again, the code will use the "fallback" methods in this case, which add the necessary `CAST`.

View the code for the nitty-gritty details.

CHAPTER 4

api

generic_aggregation.**generic_annotate**(*qs_model*, *generic_qs_model*, *aggregator*[, *gfk_field=None*[, *alias='score'*]])

Find blog entries with the most comments:

```
qs = generic_annotate(Entry.objects.public(), Comment.objects.public(), Count('comments__id'))
for entry in qs:
    print entry.title, entry.score
```

Find the highest rated foods:

```
generic_annotate(Food, Rating, Avg('ratings__rating'), alias='avg')
for food in qs:
    print food.name, '- average rating:', food.avg
```

Note: In both of the above examples it is assumed that a GenericRelation exists on Entry to Comment (named "comments") and also on Food to Rating (named "ratings"). If a GenericRelation does *not* exist, the query will still return correct results but the code path will be different as it will use the fallback method.

Warning: If the underlying column type differs between the qs_model's primary key and the generic_qs_model's foreign key column, it will use the fallback method, which can correctly CASTself.

Parameters

- **qs_model** – A model or a queryset of objects you want to perform annotation on, e.g. blog entries
- **generic_qs_model** – A model or queryset containing a GFK, e.g. comments
- **aggregator** – an aggregation, from django.db.models, e.g. Count('id') or Avg('rating')
- **gfk_field** – explicitly specify the field w/the gfk
- **alias** – attribute name to use for annotation

Return type a queryset containing annotate rows

generic_aggregation.**generic_aggregate**(*qs_model*, *generic_qs_model*, *aggregator*[, *gfk_field=None*])

Find total number of comments on blog entries:

```
generic_aggregate(Entry.objects.public(), Comment.objects.public(), Count('comments__id'))
```

Find the average rating for foods starting with 'a':

```
    a_foods = Food.objects.filter(name__startswith='a')
    generic_aggregate(a_foods, Rating, Avg('ratings__rating'))
```

Note: In both of the above examples it is assumed that a GenericRelation exists on Entry to Comment (named "comments") and also on Food to Rating (named "ratings"). If a GenericRelation does *not* exist, the query will still return correct results but the code path will be different as it will use the fallback method.

Warning: If the underlying column type differs between the qs_model's primary key and the generic_qs_model's foreign key column, it will use the fallback method, which can correctly CASTself.

Parameters

- **qs_model** – A model or a queryset of objects you want to perform annotation on, e.g. blog entries
- **generic_qs_model** – A model or queryset containing a GFK, e.g. comments
- **aggregator** – an aggregation, from django.db.models, e.g. Count('id') or Avg('rating')
- **gfk_field** – explicitly specify the field w/the gfk

Return type a scalar value indicating the result of the aggregation

generic_aggregation.**generic_filter**(*generic_qs_model*, *filter_qs_model*[, *gfk_field=None*])

Only show me ratings made on foods that start with "a":

a_foods = Food.objects.filter(name__startswith='a') generic_filter(Rating.objects.all(), a_foods)

Only show me comments from entries that are marked as public:

generic_filter(Comment.objects.public(), Entry.objects.public())

Parameters

- **generic_qs_model** – A model or queryset containing a GFK, e.g. comments
- **qs_model** – A model or a queryset of objects you want to restrict the generic_qs to
- **gfk_field** – explicitly specify the field w/the gfk

Return type a filtered queryset

4.1 Indices and tables

- genindex
- modindex
- search

g

G