PostgreSQL lookups and functions for Django apps

Release 0.1.3

Feb 13, 2020

Contents:

How often have you had the impression that Django was not providing all the lookup expressions and functions for your queries? Probably not that often, but now here is a **small** collection that I consider quite useful.

You will get started quite quickly!

CHAPTER 1

Quick Start

1.1 Installation

Just use:

```
pip install django-postgres-tweaks
```

As the title says it already, these tools are designed to be used in Django projects/apps. So make sure to add `postgres_utils` or `postgres_utils.apps.PostgresUtilsConfig` to the `INSTALLED_APPS` list in your project's `settings.py`!

That's it.

1.2 Usage

1.2.1 Lookups

The lookups provided by this package/app are automatically loaded when the app is installed. You can go ahead and just use them like Django's built-in lookups, e.g.:

```
Pizza.objects.filter(name__noregex="[ ]+")
```

Assume you have a model called `Pizza` with a `name` field.

1.2.2 Functions

Like the DB functions provided by Django, e.g. in `django.db.models.functions`, you need to need to import them prior to usage. An example query looks like this:

```
Topping.objects\
   .filter(name__contains="Onion")\
   .annotate(onion_color=RegexpReplace("name", " *Onion$", ""))\
   .values("name", "onion_color")\
   .order_by("name")
```

CHAPTER 2

What's provided?

2.1 LookUps

Django offers a series of built-in lookups that can be used in queries, but sometimes one needs additional lookups to take full advantage of an underlying database. Therefore Django allows developers to define custom lookups.

This app provides the following lookups:

class postgres_utils.lookups.**INotRegex**(*lhs*, *rhs*)
__inoregex matches rows that do not match a case-insensitive RegExp

class postgres_utils.lookups.**NotIn**(*lhs*, *rhs*)
__notin matches rows with values not in the list

Use as follows:

```
qs = Model.objects.filter(somefield__notin=[value1, value2, ...])
```

class postgres_utils.lookups.**NotRegex**(*lhs*, *rhs*)
__noregex matches rows that do not match a RegExp

Use as follows:

```
qs = Model.objects.filter(charfield__noregex="pattern")
```

See also PostgreSQL: POSIX Regular Expressions

2.2 Database Functions

Django provides a way for users to use functions provided by the underlying database as annotations, aggregations, or filters. Functions are also expressions, so they can be used and combined with other expressions like aggregate functions.

However the list of PostgreSQL specific functions in Django is very limited.

class postgres_utils.functions.**ArraySubquery**(*queryset*, *output_field=None*, ***extra*)
Convert sub-query results to array

While Django's original `django.db.models.Subquery` is allowed to return only one match, this subquery is converted into an array and can return all matches, e.g.:

```
sub_q = Topping.objects.filter(pizza=OuterRef("id"), vegan=True).values("name")
qs = Pizza.objects.annotate(vegan_toppings=ArraySubquery(sub_q))
```

Parameters **queryset** – The queryset to be executed as subquery.

class postgres_utils.functions.**RegexpReplace**(*expression*, *pattern*, *replacement*, ***extra*)
Use regular expression to replace value in field

Note: This might become available in Django in the future: https://code.djangoproject.com/ticket/28805

```
Topping.objects \
    .filter(name__contains="Onion") \
    .annotate(onion_color=RegexpReplace("name", " *Onion$", ""))
```

Parameters

- **expression** – The expression/field to work on
- **pattern** – The regular expression pattern to match
- **replacement** – The replacement string for matches

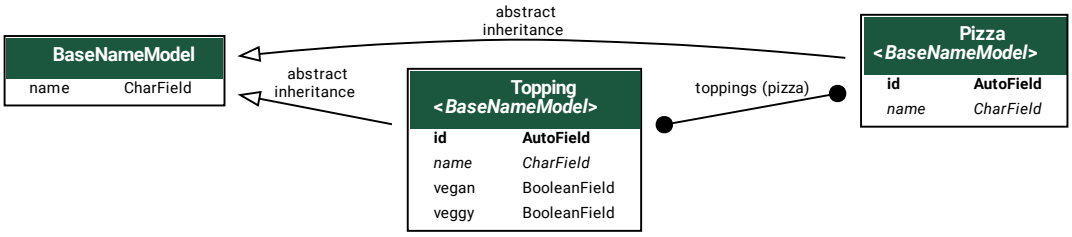
class postgres_utils.functions.**Substring**(*expression*, *pattern*)
Use regular expression to extract a substring from field

```
Topping.objects \
    .filter(name__contains="Sauce") \
    .annotate(souce_type=Substring("name", "[A-Za-z]+(?= Sauce)")) \
    .values("name", "souce_type") \
    .order_by("name")
```

Parameters

- **expression** – The expression/field to work on
- **pattern** – The regular expression pattern to match

In this documentation I will assume the following data model to illustrate code examples:


abstract
inheritance
BaseNameModel
name CharField
abstract
inheritance
Topping
<BaseNameModel>
id AutoField
name CharField
vegan BooleanField
veggy BooleanField
toppings (pizza)
Pizza
<BaseNameModel>
id AutoField
name CharField

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index