django-pg-fts Documentation

Release 0.1.1

David Miguel

October 02, 2014

Contents

Implementation PostgeSQL for Full Text Search for django 1.7, taking advantage of new features Migrations and Custom Lookups.

CHAPTER 1

Features:

- FieldLookup's search, isearch, tsquery
- Ranking support with normalization, and weights using annotations
- Migrations classes to help create and remove index's, support for 'gin' or 'gist'
- Migrations classes to help create and remove of trigger's
- Multiple dictionaries support with trigger and FieldLookup's
- Support for python 2.7, 3.3 and 3.4

Contents:

1.1 Installation

Contents

1.1.1 Development version

Clone from GitHub:

```
git clone git://github.com/dvdmgl/django-pg-fts.git django-pg-fts
```

You should run the tests:

```
python runtests.py
```

Or running tox for py27, py33, py34 for django 1.7 and master branch:

```
tox
```

Install using pip from source:

```
pip install git+https://github.com/dvdmgl/django-pg-fts
```

Or using setup.py:

```
python setup.py
```

1.2 Tutorial

1.2.1 Getting started

Create a new project like `fooproject` and `article` app:

```
django-admin startproject fooproject
cd fooproject
django-admin startapp article
```

Add ‘‘pg_fts‘‘ To ‘‘INSTALLED_APPS‘‘

As with most django applications, you should add `pg_fts` to the `INSTALLED_APPS` in your `settings.py` file:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    # ...
    'pg_fts',
    'article'  # as an example app
)
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        # ...
}
```

Caution: this is a PostgreSQL module be sure that your database `ENGINE` in `DATABASES` is `'django.db.backends.postgresql_psycopg2'`

Single dictionary example

Set up your article.models and add `TSVectorField`:

```
from pg_fts.fields import TSVectorField
from django.db import models

class Article(models.Model):
    title = models.CharField(max_length=255)
    article = models.TextField()

    fts_index = TSVectorField(
        (('title', 'A'), 'article'),
        dictionary='portuguese'
    )

    def __str__(self):
        return self.title
```

The `dictionary='portuguese'` indicates that will be used the `pg_catalog.portuguese` dictionary.

For the option `(('title', 'A'), 'article')` the `('title', 'A')` `'title'` refers to the field `title` and `'A'` is the rank given to this field, `'article'` refers to the field `article` and it will be given the default rank value of `'D'`.

Caution: `TSVectorField` option `db_index` cannot be used, as the correct type of index is *gin* or *gist* will be created in `CreateFTSIndexOperation`

Now create migrations for this module:

```
python makemigrations article
```

Migrations create index and trigger

This will create the migration code to apply to your model, but before applying `migrate` lets edit the created migration and import:

```
from pg_fts.migrations import CreateFTSIndexOperation, CreateFTSTriggerOperation
```

At the end of the array `operations` add operation `CreateFTSIndexOperation` to create the `gin` index for `TSVectorField` `fts_index`:

```
CreateFTSIndexOperation(
    name='Article',
    fts_vector='fts_index',
    index='gin'
),
```

Note: `CreateFTSTriggerOperation` only updates vector on future updates/inserts.

For indexing the current data add to operations `UpdateVectorOperation`:

```
UpdateVectorOperation(
    name='Article',
    fts_vector='fts_index',
)
```

And also add `CreateFTSTriggerOperation` to create an automatic trigger for updating the `fts_index`:

```
CreateFTSTriggerOperation(
    name='Article',
    fts_vector='fts_index',
),
```

The complete code in `migrations/0001_initial.py` should be like this:

```
class Migration(migrations.Migration):

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Article',
            fields=[
                ('id', models.AutoField(verbose_name='ID', serialize=False, auto_created=True, prima
                ('title', models.CharField(max_length=255)),
                ('article', models.TextField()),
                ('fts_index', pg_fts.fields.TSVectorField(editable=False, serialize=False, null=True,
            ],
            options={
            },
```

```
            bases=(models.Model,),
        ),
        # create gin index to Article.fts_index
        CreateFTSIndexOperation(
            name='Article',
            fts_vector='fts_index',
            index='gin'
        ),
        # create trigger to Article.fts_index
        CreateFTSTriggerOperation(
            name='Article',
            fts_vector='fts_index'
        ),
    ]
```

To see the migration to be applied to your database, run:

```
python manage.py sqlmigrate article 0001
```

It should display:

```
BEGIN;

CREATE TABLE "article_article" ("id" serial NOT NULL PRIMARY KEY, "title" varchar(255) NOT NULL, "art
CREATE INDEX article_article_fts_index ON article_article USING gin(fts_index);

CREATE FUNCTION article_article_fts_index_update() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        new.fts_index = setweight(to_tsvector('portuguese', COALESCE(NEW.title, '')), 'A') || setweig
    END IF;
    IF TG_OP = 'UPDATE' THEN
        IF NEW.title <> OLD.title OR NEW.article <> OLD.article THEN
            new.fts_index = setweight(to_tsvector('portuguese', COALESCE(NEW.title, '')), 'A') || set
        END IF;
    END IF;
RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
CREATE TRIGGER article_article_fts_index_update BEFORE INSERT OR UPDATE ON article_article
FOR EACH ROW EXECUTE PROCEDURE article_article_fts_index_update();


COMMIT;
```

Now apply the migrations to your database:

```
python manage.py migrate article
```

Using lookups

With `python manage.py shell`:

```
>>> from testapp.models import Article
>>> Article.objects.create(title='PHP', article='what a pain, the worst of c, c++, perl all mixed in
>>> Article.objects.create(title='Python', article='is awesome')
>>> Article.objects.create(title='django', article='is awesome, made in python, multiple databases su
>>> Article.objects.create(title='Wordpress', article="what a pain, made in PHP, it's ok if you just
```

```
>>> Article.objects.create(title='Javascript', article='A functional language, with c syntax. The bra
>>> Article.objects.filter(fts_index__search='django')
[<Article: django>]
>>> Article.objects.filter(fts_index__search='Python')
[<Article: Python>, <Article: django>]
>>> Article.objects.filter(fts_index__search='templates')
[<Article: Wordpress>, <Article: django>]
# postgress & and
search = Article.objects.filter(fts_index__search='templates awesome')
>>> print(search.query)
SELECT "article_article"."id", "article_article"."title", "article_article"."article", "article_artic
print(search)
[<Article: django>] # only django has template language AND is awesome
isearch = Article.objects.filter(fts_index__isearch='templates awesome')
>>> print(isearch.query)
SELECT "article_article"."id", "article_article"."title", "article_article"."article", "article_artic
print(isearch)
[<Article: Python>, <Article: Wordpress>, <Article: django>]
# wordpress oh no and in 2nd position, let's rank the results
```

Ranking results

To rank results 12.3.3. Ranking Search Results let's use django annotate.

For this lets use `FTSRank`, `FTSRankCd`

```
>>> from pg_fts.ranks import FTSRank, FTSRankCd
>>> ranks = Article.objects.annotate(rank=FTSRank(fts_index__isearch='templates awesome')).order_by('
>>> ranks
[<Article: django>, <Article: Python>, <Article: Wordpress>]
# that's better, wordpress has templates, but it's not awesome, but let's check ranks
>>> [(r.title, r.rank) for r in ranks]
[('django', 0.0607927), ('Python', 0.0303964), ('Wordpress', 0.0303964)]
# lucky for python appear before wordpress, let's normalize the results
>>> ranks_cd = Article.objects.annotate(rank=FTSRankCd(fts_index__isearch='awesome templates', normal
>>> [(r.title, r.rank) for r in ranks_cd]
[('Python', 0.047619), ('django', 0.0457674), ('Wordpress', 0.0234196)]
```

Python and django are awesome, check the postgres documentation for more about normalization

Multiple dictionary example

Multiple dictionary support:

```
class ArticleMulti(models.Model):
    title = models.CharField(max_length=255)
    article = models.TextField()
    # dictionary field to be used in query and trigger
    dictionary = models.CharField(
        max_length=15,
        choices=(('english', 'english'), ('portuguese', 'portuguese')),
        default='english',
        db_index=True
    )

    fts_index = TSVectorField(
```

```
        (('title', 'A'), 'article'),
        dictionary='dictionary'  # refers to dictionary field in model
    )

    def __str__(self):
        return self.title
```

Migrations create index and trigger

Like before in Single dictionary example:

```
from pg_fts.migrations import CreateFTSIndexOperation, CreateFTSTriggerOperation
```

At the end of the array `operations`:

```
CreateFTSIndexOperation(
    name='ArticleMulti',
    fts_vector='fts_index',
    index='gin'
),
CreateFTSTriggerOperation(
    name='ArticleMulti',
    fts_vector='fts_index',
),
```

But running `python manage.py sqlmigrate article 0002` generates the appropriate trigger

```
BEGIN;

--- ...

$$ LANGUAGE 'plpgsql';
CREATE TRIGGER article_article_fts_index_update BEFORE INSERT OR UPDATE ON article_article
FOR EACH ROW EXECUTE PROCEDURE article_article_fts_index_update();

CREATE FUNCTION article_articlemulti_fts_index_update() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        new.fts_index = setweight(to_tsvector(NEW.dictionary::regconfig, COALESCE(NEW.title, '')), 'A
    END IF;
    IF TG_OP = 'UPDATE' THEN
        IF NEW.dictionary <> OLD.dictionary OR NEW.title <> OLD.title OR NEW.article <> OLD.article T
            new.fts_index = setweight(to_tsvector(NEW.dictionary::regconfig, COALESCE(NEW.title, ''))
        END IF;
    END IF;
RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
CREATE TRIGGER article_articlemulti_fts_index_update BEFORE INSERT OR UPDATE ON article_articlemulti
FOR EACH ROW EXECUTE PROCEDURE article_articlemulti_fts_index_update();

--- ...

COMMIT;
```

Now the `INSERT` and `UPDATE` uses `NEW.dictionary::regconfig` for getting the language from dictionary

Using lookups

Now the lookup checks the `DictionaryTransform` for dictionary transformations.

For English search:

```
en = ArticleMulti.objects.filter(fts_index__english__search='django')
```

For Portuguese search:

```
pt = ArticleMulti.objects.filter(fts_index__portuguese__search='django')
```

Note: Should be applied the filter for the dictionary field:

```
en.filter(dictionary='english')
pt.filter(dictionary='portuguese')
```

```
>>> ArticleMulti.objects.create(title='PHP', article='what a pain, the worst of c, c++, perl all mixe
>>> ArticleMulti.objects.create(title='Python', article='is awesome', dictionary='english')
>>> ArticleMulti.objects.create(title='django', article='is awesome, made in python', dictionary='eng
>>> ArticleMulti.objects.create(title='Wordpress', article="what a pain, made in PHP, it's ok if you
>>> ArticleMulti.objects.create(title='Javascript', article='A functional dictionary, with c syntax.
## Portuguese
>>> ArticleMulti.objects.create(title='PHP', article='que dor, o pior do c, c++ e perl tudo junto par
>>> ArticleMulti.objects.create(title='Python', article='é Brutal', dictionary='portuguese')
>>> ArticleMulti.objects.create(title='django', article='é Altamente, feito em python', dictionary='p
>>> ArticleMulti.objects.create(title='Wordpress', article="que dor, feito em PHP, não é mau para que
>>> ArticleMulti.objects.create(title='Javascript', article='Uma linguagem funcional, mas tem sintaxe
>>> django_pt = ArticleMulti.objects.filter(fts_index__portuguese__search='django', dictionary='portu
>>> ArticleMulti.objects.filter(fts_index__portuguese__search='pesadelo')
[<ArticleMulti: Javascript>]
>>> django_pt[0].article
'é Altamente, feito em python'
>>> django_en = ArticleMulti.objects.filter(fts_index__english__search='django', dictionary='english'
>>> django_en[0].article
'is awesome, made in python'
```

Ranking results

To rank results in case of multiple dictionaries, use the appropriate `FTSRankDictionary`, `FTSRankCdDictionary`

Works like the Single Dictionary but with Multiple lookups

```
>>> ArticleMulti.objects.filter(dictionary='portuguese').annotate(
    rank=(FTSRankDictionary(
        fts_index__portuguese__search='pesadelo')).order_by('rank')
```

Removing and updating migrations

If you remove, rename, alter one off the fields related to `TSVectorField`

Changing the single dictionary Article to a multiple dictionary Article instead of creating a ArticleMulti

reverse migration to 0001 so does not include ArticleMulti:

```
python manage.py migrate article 0001
```

Delete the 0002 migration, remove ArticleMulti from models.py and add / change Article to:

```
class Article(models.Model):
    title = models.CharField(max_length=255)
    article = models.TextField()

dictionary = models.CharField(
    max_length=15,
    choices=(('english', 'english'), ('portuguese', 'portuguese')),
    default='english'
)

fts_index = TSVectorField(
    (('title', 'A'), 'article'),
    dictionary='dictionary'  # now it refers to the dictionary field
)

def __str__(self):
    return self.title
```

Let django find the model alterations for us:

```
python manage.py makemigrations article
```

But we have to edit the migrations 0002 file before applying and add to operations `DeleteFTSTriggerOperation` and `DeleteFTSIndexOperation` **before** django auto migrations, and **at the end** of operations the `CreateFTSIndexOperation` and `CreateFTSTriggerOperation`.

The migrations 0002 file should be like this:

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.db import models, migrations
import pg_fts.fields
from pg_fts.migrations import (CreateFTSIndexOperation,
                               CreateFTSTriggerOperation,
                               DeleteFTSIndexOperation,
                               DeleteFTSTriggerOperation)


class Migration(migrations.Migration):

    dependencies = [
        ('article', '0001_initial'),
    ]

    operations = [
        # remove previous created CreateFTSTriggerOperation
        DeleteFTSTriggerOperation(
            name='Article',
            fts_vector='fts_index'
        ),
        # remove previous created CreateFTSIndexOperation
        DeleteFTSIndexOperation(
            name='Article',
            fts_vector='fts_index',
```

```
            index='gin'
        ),
        # the django created changes
        migrations.AddField(
            model_name='article',
            name='dictionary',
            field=models.CharField(default='english', choices=[('english', 'english'), ('portuguese',
            preserve_default=True,
        ),
        migrations.AlterField(
            model_name='article',
            name='fts_index',
            field=pg_fts.fields.TSVectorField(dictionary='dictionary', serialize=False, default='', n
        ),
        # add new index
        CreateFTSIndexOperation(
            name='Article',
            fts_vector='fts_index',
            index='gin'
        ),
        # and create new trigger
        CreateFTSTriggerOperation(
            name='Article',
            fts_vector='fts_index'
        ),

    ]
```

Warning: Pay special attention to the order of creation and deleting.
You can only apply `CreateFTSIndexOperation` and `CreateFTSTriggerOperation` after django created operations.
The `DeleteFTSTriggerOperation` and `DeleteFTSIndexOperation` before django removing/altering operations
Not to forget **USE AT YOUR OWN RISK**

1.3 Migrations

Important: If you're not familiar with django Migrations, please check django Migrations Documentation 1st.

Important: The order of Migrations.operations matters, you can only apply a index to a field if a field exists, if you delete a model/field you should remove any operation first before removing model/field.

1.3.1 BaseVectorOperation options

The following arguments are available to all FTSMigration types

name

`CreateFTSTriggerOperation.name`

Name of the model

fts_vector

CreateFTSTriggerOperation.**fts_vector**

Name of the TSVectorField

Creating

1.3.2 Trigger

For creating of trigger is provided CreateFTSTriggerOperation.

CreateFTSTriggerOperation

class CreateFTSTriggerOperation(*options***)

How it works

The trigger only updates the TSVectorField if data is changed in the fields that is indexing, with it's weight (default is 'D') and language.

Example for this model:

```
class Article(models.Model):
    title = models.CharField(max_length=255)
    article = models.TextField()
    fts = TSVectorField(
        (('title', 'A'), 'article'),
        dictionary='portuguese'
    )
```

In Migrations:

```
class Migration(migrations.Migration)
    dependencies = [
        ('article', '00XX_create_fts_field'),
    ]

    operations = [
        CreateFTSTriggerOperation(
            name='Article',
            fts_vector='fts',
        ),
    ]
```

Will create this trigger:

```
CREATE FUNCTION article_article_fts_update() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        new.fts = setweight(to_tsvector('portuguese', COALESCE(NEW.title, '')), 'A') || setweight(to_
    END IF;
    IF TG_OP = 'UPDATE' THEN
        IF NEW.title <> OLD.title OR NEW.article <> OLD.article THEN
            new.fts = setweight(to_tsvector('portuguese', COALESCE(NEW.title, '')), 'A') || setweight
        END IF;
```

```
    END IF;
RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
CREATE TRIGGER article_article_fts_update BEFORE INSERT OR UPDATE ON article_article
FOR EACH ROW EXECUTE PROCEDURE article_article_fts_update();
```

Important: Trigger will only work for future changes, for existing data use `UpdateVectorOperation`.

1.3.3 Index

For creating of indexes is provided `CreateFTSIndexOperation`.

index

`CreateFTSIndexOperation.`**`index`**

Options:

- `gin` GIN (Generalized Inverted Index)-based index. Faster to build, slower to lookup.
- `gist` GiST (Generalized Search Tree)-based index. Faster to lookup, slower to build.

For information about the `gin` and `gist` indexes types consult PostgreSQL documentation 12.9. GiST and GIN Index Types

Example:

```
class Migration(migrations.Migration)
    dependencies = [
        ('article', '0003_fts_create_field'),
    ]

    operations = [
        CreateFTSIndexOperation(
            name='Article',
            fts_vector='fts_index',
            index='gin'
        ),
    ]
```

Migrating from existing application

For existing application with data is provided `UpdateVectorOperation`, this will update the vector.

Changing and Removing

1.3.4 Changing Index

If you have a existing index created by `CreateFTSIndexOperation` of type `gin` to migrate for `gist` you have to 1st remove the existing index with `DeleteFTSIndexOperation` and create a of type `gist` with `CreateFTSIndexOperation`.

Example:

```
class Migration(migrations.Migration)
    dependencies = [
        ('article', '0003_fts_create_index_trigger'),
    ]

    operations = [
        DeleteFTSIndexOperation(
            name='Article',
            fts_vector='fts_index',
            index='gin'
        ),
        CreateFTSIndexOperation(
            name='Article',
            fts_vector='fts_index',
            index='gist'
        ),
    ]
```

1.3.5 Alterations on `TSVectorField`

If you change `TSVectorField` is fields, ranks or dictionary you have to:

1. remove the trigger with `DeleteFTSTriggerOperation` and only after you can create
2. a new trigger with `CreateFTSTriggerOperation`.

For updating `TSVectorField` run `UpdateVectorOperation`.

Hint: If the fields are the same (fields and rank) but you are updating to multiple dictionaries, for efficiency, keep the previous dictionary as default, as the lexemes and weight will be the same in `TSVectorField`. There is no need to run `UpdateVectorOperation`

1.3.6 Removing Index

For removing the index is provided `DeleteFTSIndexOperation`.

`index`

`CreateFTSIndexOperation.`**`index`**

The previous index type, important for regressions.

1.3.7 Removing Trigger

For removing the index is provided `DeleteFTSTriggerOperation`.

1.4 Ranking

1.4.1 RankBase options

The following arguments are available to all FTSRank types

lookup

FTSRank.**lookup**

The `TSVectorField` and its available lookup's `search`, `isearch` and `tsquery`.

Will raise exception if lookup isn't valid.

See also:

For `TSVectorField` lookups check *TSVectorField lookups*

normalization

FTSRank.**normalization**

A list or tuple of integers with the normalization values, can be used a bit mask `[1,2]` will be converted to `1|2` in sql.

Accepted Values = 0, 1, 2, 4, 8, 16, 32

Default is PostgresSQL function default.

Will raise exception if `normalization` isn't valid.

weights

FTSRank.**weights**

A list or tuple of integers/floats [D-weight, C-weight, B-weight, A-weight] with the weight values `[0.1, 0.2, 0.4, 1]` will be converted to `{0.1, 0.2, 0.4, 1.0}` in sql.

Default is PostgresSQL function default.

Will raise exception if `weights` isn't valid.

See also:

More about `normalization` and `weights` check PostgreSQL documentation 12.3.3. Ranking Search Results

1.4.2 FTSRank

class FTSRank (***options*)

Uses PostgreSQL `ts_rank` function, ranks on frequency of matching lexemes.

Examples:

Search and order by rank:

```
Article.objects.annotate(
    rank=(FTSRank(fts__search='once upon a time'))).order_by('-rank')
```

Search and with normalization:

```
Article.objects.annotate(
    normalized=(FTSRank(
        fts__search='once upon a time',
        noramlization=(1,2)
    ))).order_by('-normalized')
```

1.4.3 FTSRankCd

class FTSRankCd (***options*)

Uses PostgreSQL `ts_rank_cd` function, ranks over cover density.

Usage is the same as FTSRank, just with this class.

1.4.4 Multiple dictionaries support

For this case there are two classes `FTSRankDictionay`, `FTSRankCdDictionary`.

The usage is the same as normal `FTSRank` or `FTSRankCd`, was added the special support for lookups with dictionary transformation.

1.5 TSVector

1.5.1 TSVectorField

class TSVectorField (*fields*, *dictionary*)

`TSVectorField.`**fields**

A tuple containing a tuple of fields and rank to be indexed, it can be only the field name the default the rank 'D' will be added

Example:

```
('field_name', ('field_name2', 'A'))
```

Will result in:

```
(('field_name', 'D'), ('field_name2', 'A'))
```

`TSVectorField.`**dictionary**

- Can be string with the dictionary name `pg_catalog.pg_ts_config` consult PostgreSQL documentation 12.6. Dictionaries
- A CharField with choices in case of multiple dictionaries.

> **Caution:** Dictionary(ies) used must be installed in your database, check `pg_catalog.pg_ts_config`

Will raise exception exceptions.FieldError if lookup isn't tsquery, search or isearch or not a valid option dictionary (in case of multiple dictionaries)

> **Caution:** TSVectorField does not support iexact, it will raise an exception

1.5.2 FTSLookups

search

Only return results that match all the lexemes.

isearch

Return results if match any of the lexemes.

tsquery

Raw tsquery, for full control over PostgreSQL to_tsquery. Check PostgreSQL documentation pg_docs12.3.2. Parsing Queries <textsearch-controls.html#TEXTSEARCH-PARSING-QUERIES

1.5.3 Single dictionary examples

```
class Article(models.Model):
    title = models.CharField(max_length=255)
    article = models.TextField()
    fts = TSVectorField(
        (('title', 'A'), 'article'),
        dictionary='portuguese'
    )

    def __str__(self):
        return self.title
```

```
>>> Article.objects.bulk_create([
    Article(title='Python', article='The python programing language')
    Article(title='Monty Python', article='British surreal comedy group')
    Article(title="Monty Python's Flying Circus", article="Monty Python's Flying Circus was a series
])
```

Using `search` for *python programing* and *python comedy*:

```
>>> Article.objects.filter(fts__search='python programing')
[<Article: Python>]
>>> Article.objects.filter(fts__search='comedy group')
[<Article: Monty Python's Flying Circus>, <Article: Monty Python>]
>>> Article.objects.filter(fts__search='PHP')
[]
```

If you use `isearch` will search for any of the lexemes:

```
>>> Article.objects.filter(fts__isearch='python programing')
[<Article: Python>, <Article: Monty Python's Flying Circus>, <Article: Monty Python>]
>>> Article.objects.filter(fts__isearch='PHP')
[]
```

See also:

For ordering results use *Ranking*

1.5.4 `DictionaryTransform`

Preforms a transformation for dictionary for searching with *portuguese* lexemes `fts__portuguese__search`

It checks if the dictionary is in options, only works with multiple dictionaries.

1.5.5 Multiple dictionaries examples

```
class Article(models.Model):
    title = models.CharField(max_length=255)
    article = models.TextField()

    dicts = models.CharField(  # the dictionary field
        max_length=15,
        choices=(('english', 'english'), ('portuguese', 'portuguese')),
        default='english',
        db_index=True
    )

    fts = TSVectorField(
        (('title', 'A'), 'article'),
        dictionary='dicts'  # refers to the dictionary field
    )

    def __str__(self):
        return '%s in %s' % (self.title, self.language)
```

```
>>> Article.objects.bulk_create([
    Article(title='Python', article='The python programing language', language='english')
    Article(title='Monty Python', article='British surreal comedy group', language='english')
    Article(title="Monty Python's Flying Circus", article="Monty Python's Flying Circus was a series
    Article(title='Python', article='A linguagem de programação python', language='portuguese')
    Article(title='Monty Python', article='Um grupo de comédia britânico', language='portuguese')
    Article(title="Os Malucos do Circo", article="Os Malucos do Circo ou Monty Python's Flying Circus
])
```

Use the dictionary transform, also we should filter by the dictionary used:

```
>>> Article.objects.filter(fts__english__search='python programing', dicts='english')
[<Article: Python in english>]
>>> Article.objects.filter(fts__english__search='monty python', dicts='portuguese')
[<Article: Monty Python in portuguese>, <Article: Os Malucos do Circo in portuguese>]
```

Note: Not filtering the results by it's dictionary is possible, but results won't be reliable:

```
>>> Article.objects.filter(fts__english__search='python programing')
[<Article: Python in english>]  # nothing strange
>>> Article.objects.filter(fts__portuguese__search='python programing')
[]  # no results
```

Note: The `DictionaryTransform` only accepts dictionaries that are defined in options.

1.6 pg_fts package

1.6.1 Submodules

1.6.2 pg_fts.fields module

class `pg_fts.fields.`**`TSVectorField`**(*fields*, *dictionary='english'*, ***kwargs*)

Parameters

- **fields** – A tuple containing a tuple of fields and rank to be indexed, it can be only the field name the default the rank 'D' will be added

  Example:

  ```
  ('field_name', ('field_name2', 'A'))
  ```

  Will result in:

  ```
  (('field_name', 'D'), ('field_name2', 'A'))
  ```

- **dictionary** – available options:
  - **Can be string with the dictionary name `'pg_catalog.pg_ts_config'`** consult PostgreSQL documentation 12.6. Dictionaries
  - A TextField name in case of multiple dictionaries

  Caution:
  Dictionary(ies) used must be installed in your database, check
  `pg_catalog.pg_ts_config`

Raises exceptions.FieldError if lookup isn't tsquery, search or isearch or not a valid option dictionary (in case of multiple dictionaries)

Caution: TSVectorField does not support iexact, it will raise an exception

class `pg_fts.fields.`**`TSVectorBaseField`**(*dictionary='english'*, ***kwargs*)

Base vector field

Parameters **dictionary** – Dictionary name as is in PostgreSQL `'pg_catalog.pg_ts_config'` more information in PostgreSQL documentation 12.6. Dictionaries

Raises exceptions.FieldError if lookup isn't tsquery, search or isearch

class `pg_fts.fields.`**`TSVectorTsQueryLookup`**(*lhs*, *rhs*)

TSVectorField Lookup tsquery

Raw to_tsquery lookup, check the documentation for more details 12.3.2. Parsing Queries

Parameters **values** (*query*) – valid PostgreSQL tsquery

Caution: If the query is not a valid PostgreSQL to_tsquery will result in a syntax error

Example:

```
Article.objects.filter(
    tsvector__tsquery="'single-quoted phrases' & prefix:A*B & !not | or | weight:ABC"
)
```

SQL equivalent:

```
"tsvector" @@ to_tsquery('english', '''singlequoted phrases'' & prefix:A*B & !not | or | weight:
```

Note:

The lookup will get dictionary value in `TSVectorField`, this case *english*

In case of multiple dictionaries see `DictionaryTransform`

---

class `pg_fts.fields.`**`TSVectorSearchLookup`**(*lhs*, *rhs*)
TSVectorField Lookup search

An to_tsquery with AND *&* operator

Parameters **values** (*query*) – a string with words

Example:

```
Article.objects.filter(
    tsvector__search="an and query"
)
```

SQL equivalent::

```
.. code-block:: sql
```

"tsvector" @@ to_tsquery('english', 'an & and & query')

Note:

The lookup will get dictionary value in `TSVectorField`, this case *english*

In case of multiple dictionaries see `DictionaryTransform`

---

class `pg_fts.fields.`**`TSVectorISearchLookup`**(*lhs*, *rhs*)
TSVectorField Lookup isearch

An to_tsquery with OR | operator

Parameters **values** (*query*) – a string with words

Example:

```
Article.objects.filter(
    tsvector__isearch="an and query"
)
```

SQL equivalent:

```
"tsvector" @@ to_tsquery('english', 'an | and | query')
```

Note:

The lookup will get dictionary value in `TSVectorField`, this case *english*

In case of multiple dictionaries see `DictionaryTransform`

---

class `pg_fts.fields.`**`DictionaryTransform`**(*dictionary*, **args*, ***kwargs*)
TSVectorField dictionary transform

Parameters **values** (*query*) – a valid dictionary in `TSVectorField` field dictionary.options

Example:

```
# in models.py

class Article(models.Model):
    title = models.CharField(max_length=255)
    article = models.TextField()
    dictionary = models.CharField(
        max_length=15,
        choices=(
            ('english', 'english'),
            ('portuguese', 'portuguese')
        ),
        default='english',
        db_index=True
    )
    fts_index = TSVectorField(
        (('title', 'A'), 'article'),
        dictionary='dictionary'
    )

>>> Article.objects.filter(
        tsvector__english__isearch="an and query"
    )
# will create a ts_query with english dictionary
# SQL -> to_tsquery('english', 'an | and | query')

>>> Article.objects.filter(
        tsvector__portuguese__isearch="an and query"
    )
# will create a ts_query with portuguese dictionary
# SQL -> to_tsquery('portuguese', 'an | and | query')

>>> Article.objects.filter(
        tsvector__isearch="an and query"
    )
# will create a ts_query with english dictionary, it's the default in
# dictionary, but if there was no default it wold get the 1st option in
# options
# SQL -> to_tsquery('english', 'an | and | query')
# but if dictionary is not in options will raise FieldError
>>> Article.objects.filter(
        tsvector__japonese__isearch="an and query"  # will raise an error
    )
FieldError: The 'japonese' is not in article.Article.dictionary choices
```

1.6.3 pg_fts.aggregates module

Note: Deprecated use `pg_fts.ranks` instead.

1.6.4 pg_fts.ranks module

Classes to represent the default SQL aggregate functions

class `pg_fts.ranks.`**`FTSRankCd`**`(`***extra*`)`
Interface for PostgreSQL ts_rank_cd

Provides a interface for 12.3.3. Ranking Search Results *ts_rank_cd*

Parameters

- **fieldlookup** – required
- **normalization** – list or tuple of integers PostgreSQL normalization values

Returns rank_cd

Raises exceptions.FieldError if lookup isn't valid

Example:

```
Article.objects.annotate(
    rank=FTSRank(fts_index__search='Hello world',
                 normalization=[1,2]))
```

SQL equivalent:

```
SELECT
    ...,
    ts_rank_cd("article"."fts_index" @@ to_tsquery('english', 'Hello & world'), 1|2) AS "rank"
WHERE
    "article"."fts_index" @@ to_tsquery('english', 'Hello & world')
```

class pg_fts.ranks.**FTSRank**(**extra*)

Interface for PostgreSQL ts_rank

Provides a interface for 12.3.3. Ranking Search Results *ts_rank*

Example:

```
Article.objects.annotate(rank=FTSRank(fts_index__search='Hello world',
                         normalization=[1,2]))
```

SQL equivalent:

```
SELECT
    ...,
    ts_rank("article"."fts_index" @@ to_tsquery('english', 'Hello & world'), 1|2) AS "rank"
WHERE
    "article"."fts_index" @@ to_tsquery('english', 'Hello & world')
```

Parameters

- **fieldlookup** – required
- **normalization** – iterable integer
- **weights** – iterable float

Returns rank

Raises exceptions.FieldError if lookup isn't valid

class pg_fts.ranks.**FTSRankDictionay**(**extra*)

Interface for PostgreSQL ts_rank with **language lookup**

Provides a interface for 12.3.3. Ranking Search Results *rank*

Parameters

- **fieldlookup** – required
- **normalization** – list or tuple of integers PostgreSQL normalization values

Returns rank

Raises exceptions.FieldError if lookup isn't valid

Example:

```
Article.objects.annotate(
    rank=FTSRankDictionay(fts_index__portuguese__search='Hello world',
                          normalization=[1,2]))
```

SQL equivalent:

```
SELECT
    ...,
    ts_rank("article"."fts_index" @@ to_tsquery('portuguese', 'Hello & world'), 1|2) AS "rank"
WHERE
    "article"."fts_index" @@ to_tsquery('portuguese', 'Hello & world')
```

class pg_fts.ranks.**FTSRankCdDictionary**(***extra*)

Interface for PostgreSQL ts_rank_cd with **language lookup**

Provides a interface for 12.3.3. Ranking Search Results *rank_cd* with **language lookup**

Parameters

- **fieldlookup** – required
- **normalization** – list or tuple of integers PostgreSQL normalization values

Returns rank_cd

Raises exceptions.FieldError if lookup isn't valid

Example:

```
Article.objects.annotate(
    rank=FTSRankCdDictionary(fts_index__portuguese__search='Hello world',
                             normalization=[1,2]))
```

SQL equivalent:

```
SELECT
    ...,
    ts_rank("article"."fts_index" @@ to_tsquery('portuguese', 'Hello & world'), 1|2) AS "rank"
WHERE
    "article"."fts_index" @@ to_tsquery('portuguese', 'Hello & world')
```

1.6.5 pg_fts.introspection module

class pg_fts.introspection.**PgFTSIntrospection**

Helper class the introspect the database

get_dictionay_list(*cursor*)

introspects pg_catalog.pg_ts_config

Returns A list of dictionaries names installed in postgres

get_functions_list(*cursor*)
introspects the database for functions

Returns A list of functions

get_trigger_list(*cursor*)
introspects the database for triggers

Returns A list of triggers

1.6.6 pg_fts.migrations module

class pg_fts.migrations.**CreateFTSIndexOperation**(*name*, *fts_vector*, *index*)
Creates a index for TSVectorField

Parameters

- **name** – The Model name
- **fts_vector** – The TSVectorField field name
- **index** – The type of index 'gin' or 'gist' for more information go to PostgreSQL documentation 12.9. GiST and GIN Index Types

class pg_fts.migrations.**CreateFTSTriggerOperation**(*name*, *fts_vector*)
Creates a custom trigger for updating the TSVectorField with rank values

Parameters

- **name** – The Model name
- **fts_vector** – The TSVectorField field name

class pg_fts.migrations.**DeleteFTSIndexOperation**(*name*, *fts_vector*, *index*)
Removes index created by CreateFTSIndexOperation

Parameters

- **name** – The Model name
- **fts_vector** – The TSVectorField field name
- **index** – The type of index 'gin' or 'gist' for more information go to

class pg_fts.migrations.**DeleteFTSTriggerOperation**(*name*, *fts_vector*)
Deletes trigger generated by CreateFTSTriggerOperation

Parameters

- **name** – The Model name
- **fts_vector** – The TSVectorField field name

class pg_fts.migrations.**UpdateVectorOperation**(*name*, *fts_vector*)
Updates changes to TSVectorField for existing models

Parameters

- **name** – The Model name
- **fts_vector** – The TSVectorField field name

1.6.7 pg_fts.utils module

class pg_fts.utils.**TranslationDictionary**(*dictionaries=None*, *default=None*)
TranslationDictionary

1.6.8 Module contents

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

p

C

D

F

G

I

L

N

P

T

U

W