Schemaless Postgres con Django

django-hstore

Chi sono?

Federico Capoano

Web Developer

@nemesisdesign on twitter and github

Wireless Community Network enthusiast (Ninux.org)

Vi presento HStore

Estensione PotsgreSQL

Key / Value store

Vantaggi HStore

Schemaless DB

ma...

senza rinunciare alla robustezza di SQL

Limiti HStore

Solo stringhe o NULL

1 solo livello

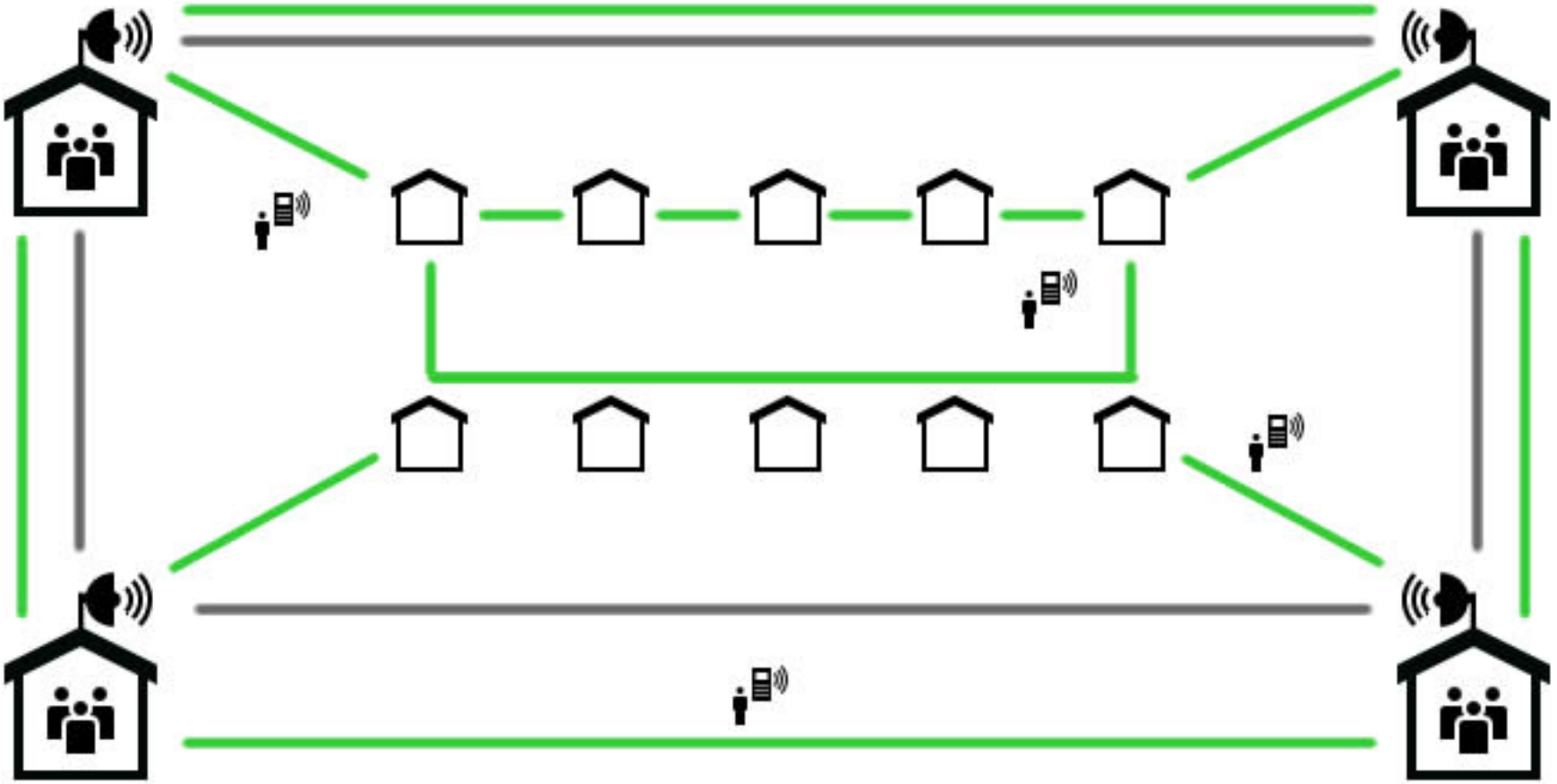
Un pò di storia

Come ho iniziato a contribuire

a django-hstore?

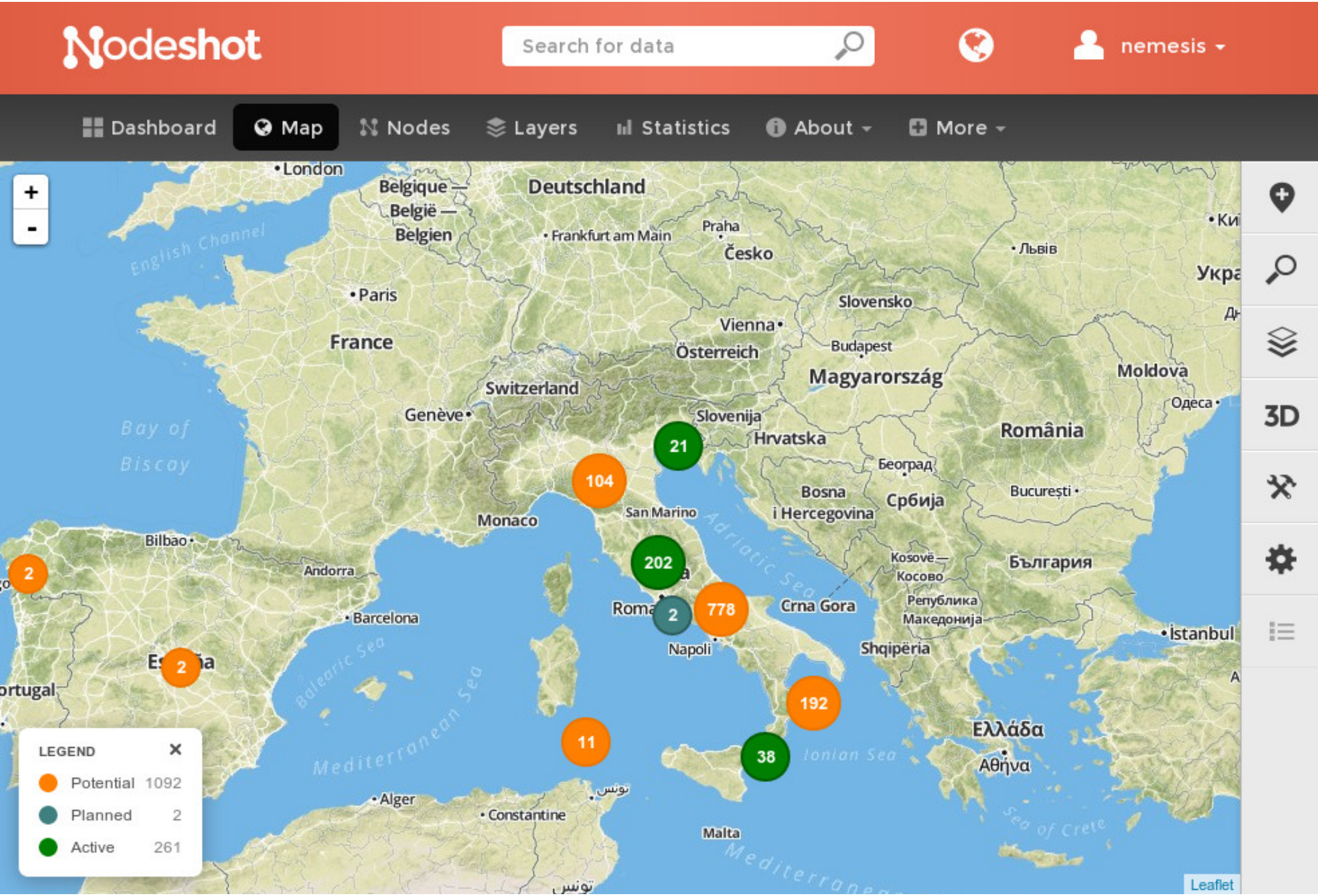
ninux.org

Community Network Cittadina



5 GHz / 17 GHz

2.4 GHz


Nodeshot
Search for data
nemesis
Dashboard
Map
Nodes
Layers
Statistics
About
More
London
Belgique
België
Belgien
Deutschland
Frankfurt am Main
Praha
Česko
Paris
France
Slovensko
Vienna
Österreich
Budapest
Magyarország
Moldova
Switzerland
Genève
Slovenija
Hrvatska
România
Београд
Bosna
i Hercegovina
Србија
București
Monaco
San Marino
Bilbao
Andorra
Barcelona
Kosovë
Косово
Република
Македонија
Crna Gora
България
Roma
Napoli
Shqipëria
İstanbul
Ελλάδα
Αθήνα
Alger
Constantine
Malta
Bay of Biscay
English Channel
Mediterranean Sea
Balearic Sea
Ionian Sea
Sea of Crete
Adriatic Sea
21
104
202
2
778
2
2
192
11
38
LEGEND
Potential 1092
Planned 2
Active 261
3D
Leaflet

Nodeshot

Repo:

https://github.com/ninuxorg/nodeshot

Docs:

http://nodeshot.readthedocs.org/en/latest/

Mailing list:

http://ml.ninux.org/mailman/listinfo/nodeshot

django-hstore

Primi commit sul mio fork

progetto interessante...

...ma...

Non funziona proprio tutto

django-hstore 1.1.1

Repository originale abbandonato

Data release: Agosto 2012

Tanti fork non comunicanti tra loro

django-hstore 1.1.1

E ora che si fa???

Questa è la volta buona che do il mio contributo

broadcast

Jordan McCoy (autore originale)

Andrei Antoukh (djorm-ext-hstore)

Autori dei vari fork

djangonauts

Nuova github org generica: **Djangonauts**

Creazione **Mailing list**

Fusione **dj-ext-orm** e **django-hstore**

Repository **unificato**, approccio **community**

Nuove release

1.2.0: Gennaio 2014

1.2.1: Gennaio 2014

1.2.2: Marzo 2014

1.2.3: Aprile 2014

Features

Quali sono le principali features di django-hstore?

DictionaryField

Model field che implementa HSTORE nei modelli django

ReferenceField

Una sorta di GenericMany2Many

Relazioni con qualsiasi altro oggetto del DB

No custom DB backend

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'name',
        'USER': 'user',
        'PASSWORD': 'pass',
        'HOST': 'localhost',
        'PORT': '',
    }
}
```

PostGIS compatibility

```
DATABASES = {
    'default': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'name',
        'USER': 'user',
        'PASSWORD': 'pass',
        'HOST': 'localhost',
        'PORT': '',
    }
}
```

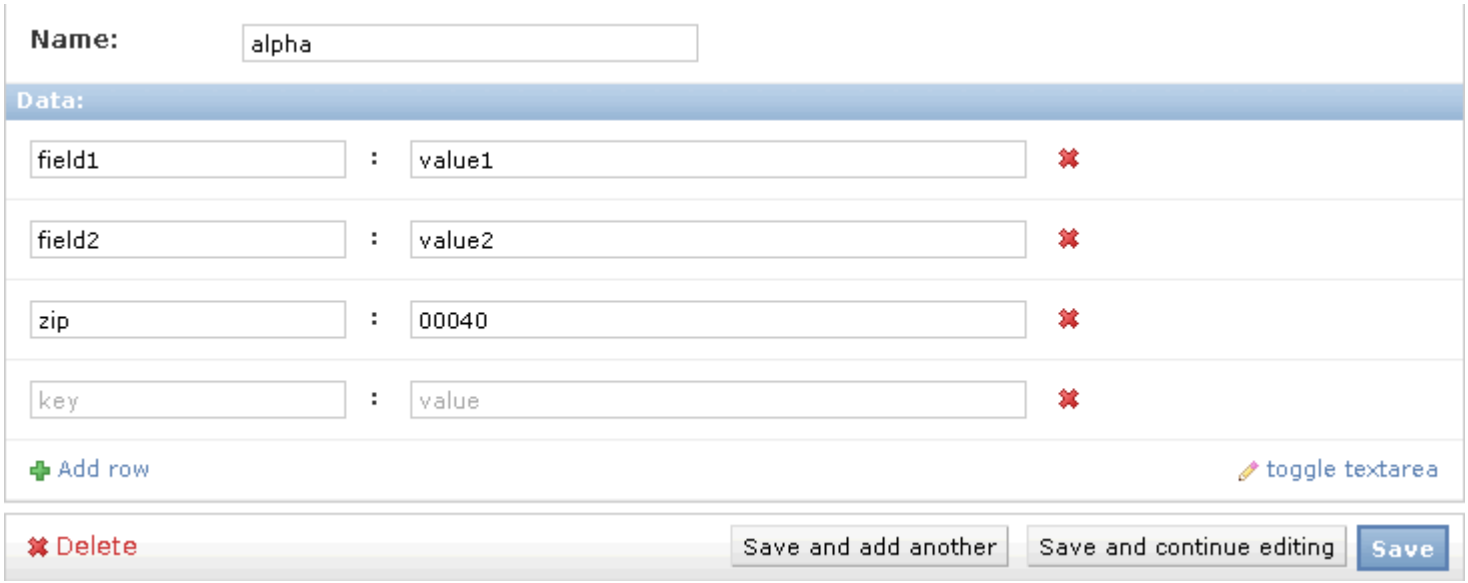
HStoreManager

Esegue SQL specifico per HSTORE

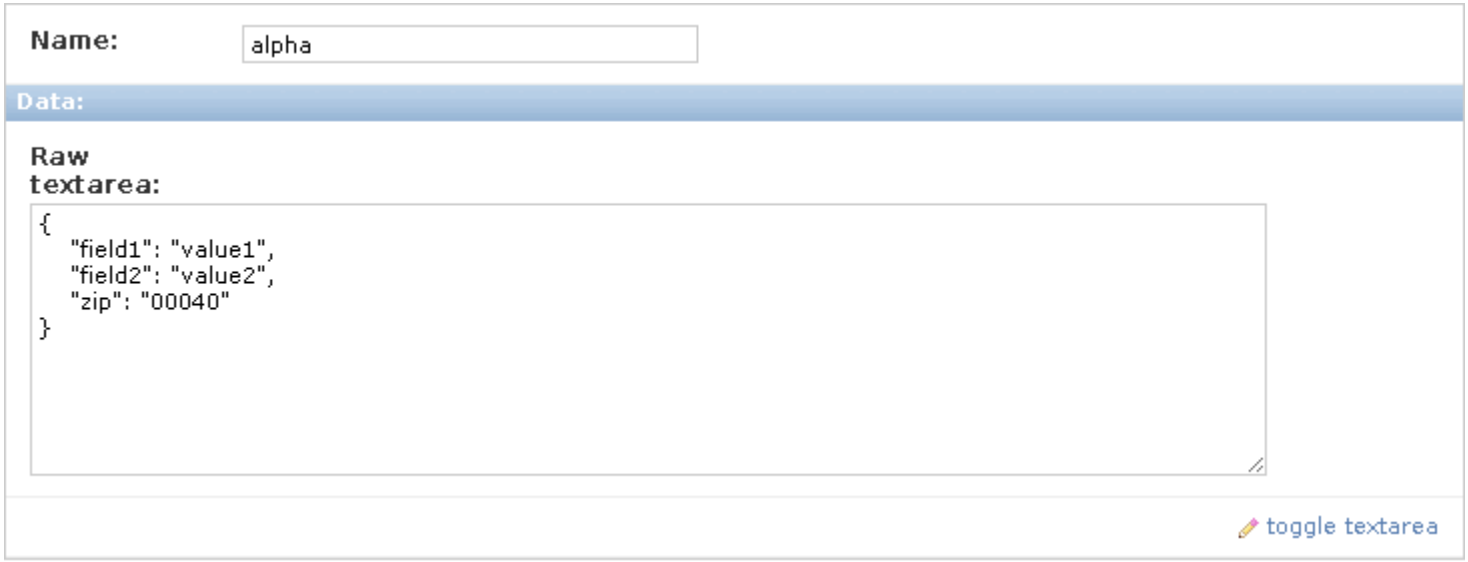
HStoreGeoManager

Unisce HstoreManager e GeoManager (geodjango)

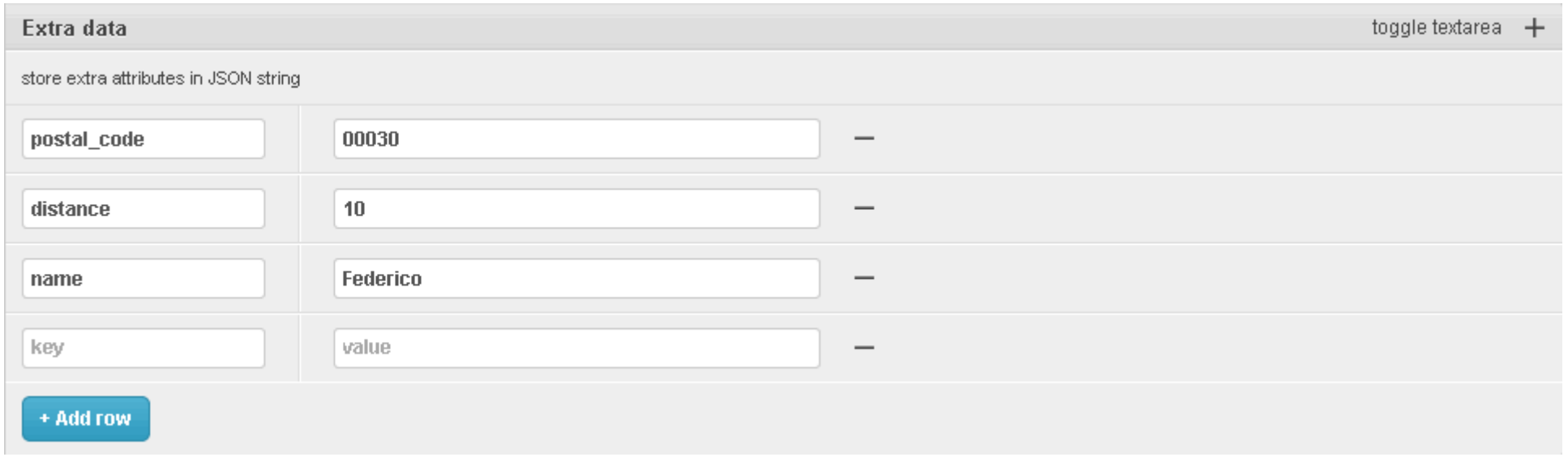
Django admin widget



Django admin widget



Grappelli admin widget



Grappelli admin widget

Python 3 ready!

Thanks to Andrei Antoukh **@niwibe**

Come si fa?

Passiamo all'azione!

Installazione versione stabile

```
$> pip install django-hstore
```

Installazione versione dev

```
$> pip install -e
git+git://github.com/djangonauts/django-hs
tore#egg=django-hstore
```

Setup

```
INSTALLED_APPS = (
    ...
    "django_hstore",
    ...
)
```

File statici per admin widget

```
python manage.py collectstatic
```

Model Setup (DictionaryField)

```
from django.db import models
from django_hstore import hstore


class Something(models.Model):
    name = models.CharField(max_length=32)
    data = hstore.DictionaryField()

    objects = hstore.HStoreManager()
    # IF YOU ARE USING POSTGIS:
    # objects = hstore.HStoreGeoManager()
```

Model Setup (ReferenceField)

```
from django.db import models
from django_hstore import hstore


class Something(models.Model):
    name = models.CharField(max_length=32)
    refs = hstore.ReferencesField()

    objects = hstore.HStoreManager()
    # IF YOU ARE USING POSTGIS:
    # objects = hstore.HStoreGeoManager()
```

Python API

Vediamo le operazioni principali dell'ORM

Create

```
instance = Something.objects.create(
    name='something',
    data={'a': '1', 'b': '2'}
)
```

Conversione automatica

```
instance = Something.objects.create(
    name='something',
    data={
        'int': 1,
        'bool': True
    }
)

instance.data['int'] == '1'
instance.data['bool'] == 'true'
```

Filter: equivalence

```
Something.objects.filter(data={'a': '1', 'b': '2'})
```

Trova gli oggetti che hanno un dizionario così come quello indicato

Filter: comparison

```
Something.objects.filter(data__gt={'a': '1'})
Something.objects.filter(data__gte={'a': '1'})
Something.objects.filter(data__lt={'a': '2'})
Something.objects.filter(data__lte={'a': '2'})
```

Greater than, less than, ecc.

Filter: contains key & value

```
Something.objects.filter(
    data__contains={'a': ['1', '2']}
)
```

Trova gli elementi che contengono la chiave 'a'

Con un valore che può essere 1 e 2

Filter: contains keys

```
Something.objects.filter(data__contains=['a', 'b'])
```

Trova gli elementi che contengono le chiavi 'a' e 'b'

Filter: classic text lookup

```
Something.objects.filter(data__contains='value')
Something.objects.filter(data__icontains='my_key')
```

Trova gli elementi che contengono il testo indicato

(cerca sia nelle chiavi che nei valori)

HstoreManager

Metodi ORM che si traducono in SQL specifico per Postgres HSTORE

HKeys

```
>>> Something.objects.hkeys(id=2, attr='data')
['a', 'b']
```

Trova le chiavi utilizzate da un elemento che usa HSTORE

HPeek

```
>>> Something.objects.hpeek(
>>>     id=1, attr='data', key='a'
>>> )
'value'
```

Recupera il valore di una chiave per l'oggetto specificato

HRemove

```
>>> Something.objects.all().hremove('data', 'b')
```

Elimina la chiave 'b' da tutti gli elementi

ReferenceField

Vediamo come si può utilizzare il ReferenceField

Salvare riferimenti ad altri oggetti

```
# recuperiamo l'oggetto 'a'
a = AnotherModel.objects.get(slug='another_object')
# creiamo un nuovo oggetto “ReferenceContainer”
r = ReferenceContainer(name='test')
# linkiamo 'a' nel nostro “ReferenceContainer”
r.refs['another_object'] = a
r.save()
```

Recuperiamo i riferimenti

```
r = ReferenceContainer.objects.get(name='test')
r.refs['another_object']
# query al DB
'<AnotherModel: AnotherModel object>'

r.refs['another_object']
# recupera dalla cache
'<AnotherModel: AnotherModel object>'
```

Rappresentazione relazioni

```
refs = {
    'another_object': 'app.models.AnotherObject:1'
}
```

"<path_to_model>:<id>"

No ForeignKeys

Non ci sono vincoli di integrità referenziale

Devono essere gestiti a livello applicativo

Impostazione del progetto

Indicazioni generali su come viene gestito il progetto

Deprecation policy

Tre versioni minori di django supportate:

Eg: django 1.6 attualmente in uso

1.6, 1.5 e 1.4 supportati

Test coverage

Attualmente 93%

Contribution requirement:

È richiesto che non scenda al di sotto del 90%

Discussioni

Mailing list

Issues su Github

Contributi

Pull request con documentazione e test

In alcuni casi diamo accesso in scrittura temporaneo per lavorare su un branch (es: django 1.7 branch)

Links

Repo:
https://github.com/djangonauts/django-hstore

Docs:
http://djangonauts.github.io/django-hstore/

Mailing list:
https://groups.google.com/forum/#!forum/django-hstore

Rringrazio tutti i contributor di django-hstore

284+ repo followers
86+ forks
23+ contributors

E' un esperienza fantastica

Grazie per aver seguito il talk!

Domande?