twentytab

Full-Text Search in Django with PostgreSQL

#pycon8 - PyDatabase - Firenze, 2017-04-08

© Paolo Melchiorre @pauloxnet

PYCON OTTO

twentytab

Paolo Melchiorre |

- **Computer Science Engineer**
- **Backend Python Developer** *since* **2006**
- **Django Developer** *since* **2011**
- **Senior Software Engineer @ TwentyTab**
- **Remote Worker**
- **PostgreSQL user, not a DBA**

PYCON OTTO

twentytab

Goal

To show how we have used Django Full Text Search and PostgreSQL in a real project.

PYCON OTTO

twentytab

Motivation

To implement Full-Text Search using only Django and PostgreSQL functionalities, without resorting to external tools.

PYCON OTTO

twentytab

Agenda

- Full-Text Search
- Existing Solutions
- PostgreSQL FTS
- Django Support
- Concerti@Roma
- What’s next
- Conclusions
- Questions

PYCON OTTO

twentytab

Full-Text Search |

*"… **Full-Text Search*** refers to techniques for **searching** a single computer-stored **document** or a **collection** in a **full text database** …"*

-- Wikipedia

* **FTS** = **F**ull-**T**ext **S**earch

PYCON OTTO

twentytab

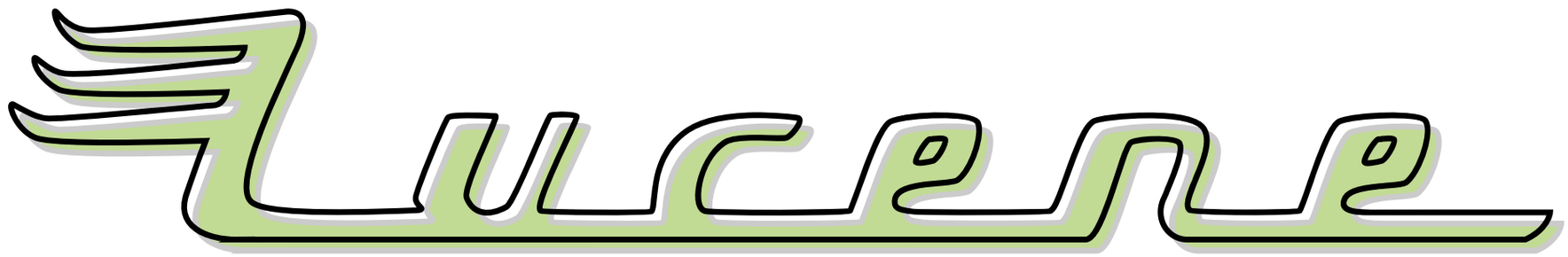
Features of a FTS

- Stemming
- Ranking
- Stopwords
- Multiple languages support
- Accent support
- Indexing
- Phrase search

PYCON OTTO

twentytab

Tested Solutions



elastic

Solr

PYCON OTTO

twentytab

Elasticsearch

Project: Snap Market *(~500,000 mobile users)*

Issues:

- **Management problems**
- **Patching a Java plugin**

```
@@ -52,7 +52,8 @@ public class DecompoundTokenFilter … {
-               posIncAtt.setPositionIncrement(0);
+               if (!subwordsonly)
+                   posIncAtt.setPositionIncrement(0);
                return true;
            }
```

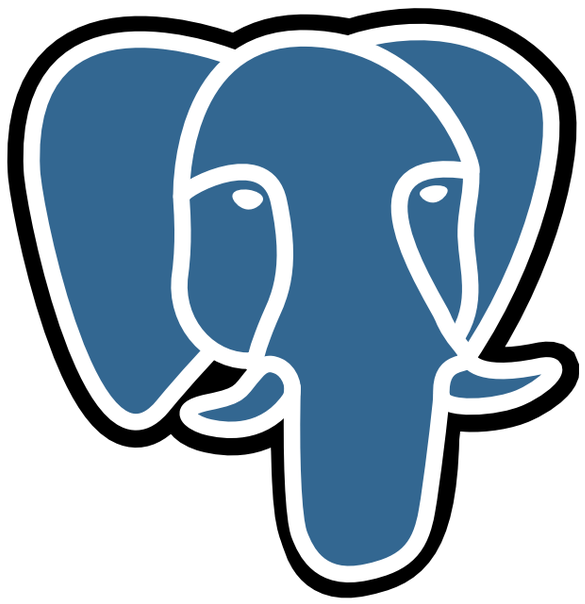
twentytab

Apache Solr

Project: GoalScout *(~25,000 videos)*

Issues:

- **Synchronization problems**
- **All *writes* to PostgreSQL and *reads* from Solr**



twentytab

Existing Solutions

PROS

- **Full featured solutions**
- **Resources** *(documentations, articles, …)*

CONS

- **Synchronization**
- **Mandatory use of driver** *(haystack, bungiesearch…)*
- **Ops oriented: focus on system integrations**

PYCON OTTO

twentytab

FTS in PostgreSQL

- **FTS Support *since* version 8.3**
- **TSVECTOR to represent text data**
- **TSQUERY to represent search predicates**
- **Special Indexes (GIN, GIST)**
- **Phrase Search *since* version 9.6**

PYCON OTTO

twentytab

What are documents |

*"… a **document** is the **unit** of searching in a **full-text** **search** system; for example, a magazine **article** or email **message** …"*

-- PostgreSQL documentation

PYCON OTTO

twentytab

Language Support |

Built-in text search for many languages

```
$ cat /usr/share/postgresql/9.6/tsearch_data/italian.stop
```

PYCON OTTO 8

twentytab

Django Support

- **Module `django.contrib.postgres`**
- **FTS Support *since* version 1.10**
- **BRIN and GIN indexes *since* version 1.11**
- **Dev oriented: focus on programming**

PYCON OTTO

twentytab

Making queries

```
class Blog(models.Model):
    name = models.CharField(max_length=100)
    tagline = models.TextField()

class Author(models.Model):
    name = models.CharField(max_length=200)
    email = models.EmailField()

class Entry(models.Model):
    blog = models.ForeignKey(Blog)
    headline = models.CharField(max_length=255)
    body_text = models.TextField()
    pub_date = models.DateField()
    authors = models.ManyToManyField(Author)
```

twentytab

Standard queries

```
>>> Author.objects.filter(name__contains='Terry')
[<Author: Terry Gilliam>, <Author: Terry Jones>]

>>> Author.objects.filter(name__icontains='Erry')
[<Author: Terry Gilliam>, <Author: Terry Jones>,
<Author: Jerry Lewis>]
```

Unaccented query |

```
>>> Author.objects.filter(name__unaccent__icontains='Helen')
[<Author: Helen Mirren>, <Author: Helena Bonham Carter>,
<Author: Hélène Joy>]
```

Trigram similar

```
>>> Author.objects.filter(
    name__unaccent__lower__trigram_similar='Hélène')
[<Author: Helen Mirren>, <Author: Helena Bonham Carter>, <Author:
Hélène Joy>]
```

twentytab

The search lookup |

```
>>> Entry.objects.filter(body_text__search='Cheese')
[<Entry: Cheese on Toast recipes>, <Entry: Pizza Recipes>]
```

twentytab

SearchVector

```
>>> from django.contrib.postgres.search import SearchVector
>>> Entry.objects.annotate(
...     search=SearchVector('body_text', 'blog__tagline'),
... ).filter(search='Cheese')
[<Entry: Cheese on Toast recipes>, <Entry: Pizza Recipes>]
```

PYCON OTTO

twentytab

SearchQuery

```
>>> from django.contrib.postgres.search import SearchQuery
>>> SearchQuery('potato') & SearchQuery('ireland')
    # potato AND ireland
>>> SearchQuery('potato') | SearchQuery('penguin')
    # potato OR penguin
>>> ~SearchQuery('sausage')
    # NOT sausage
```

twentytab

SearchRank

```
>>> from django.contrib.postgres.search import (
    SearchQuery, SearchRank, SearchVector)
>>> vector = SearchVector('body_text')
>>> query = SearchQuery('cheese')
>>> Entry.objects.annotate(
    rank=SearchRank(vector, query)).order_by('-rank')
[<Entry: Cheese on Toast recipes>, <Entry: Pizza recipes>]
```

PYCON OTTO

twentytab

Search configuration |

```
>>> from django.contrib.postgres.search import (
    SearchQuery, SearchVector)
>>> Entry.objects.annotate(
...     search=SearchVector('body_text', config='french'),
... ).filter(search=SearchQuery('œuf', config='french'))
[<Entry: Pain perdu>]

>>> from django.db.models import F
>>> Entry.objects.annotate(
...     search=SearchVector('body_text', config=F('blog__lang')),
... ).filter(search=SearchQuery('œuf', config=F('blog__lang')))
[<Entry: Pain perdu>]
```

twentytab

Weighting queries

```
>>> from django.contrib.postgres.search import (
        SearchQuery, SearchRank, SearchVector)
>>> vector = SearchVector('body_text', weight='A') +
             SearchVector('blog__tagline', weight='B')
>>> query = SearchQuery('cheese')
>>> Entry.objects.annotate(rank=SearchRank(
       vector, query)).filter(rank__gte=0.3).order_by('rank')
```

SearchVectorField

```
>>> Entry.objects.update(search_vector=SearchVector('body_text'))
>>> Entry.objects.filter(search_vector='cheese')
[<Entry: Cheese on Toast recipes>, <Entry: Pizza recipes>]
```

twentytab

Concerti@Roma

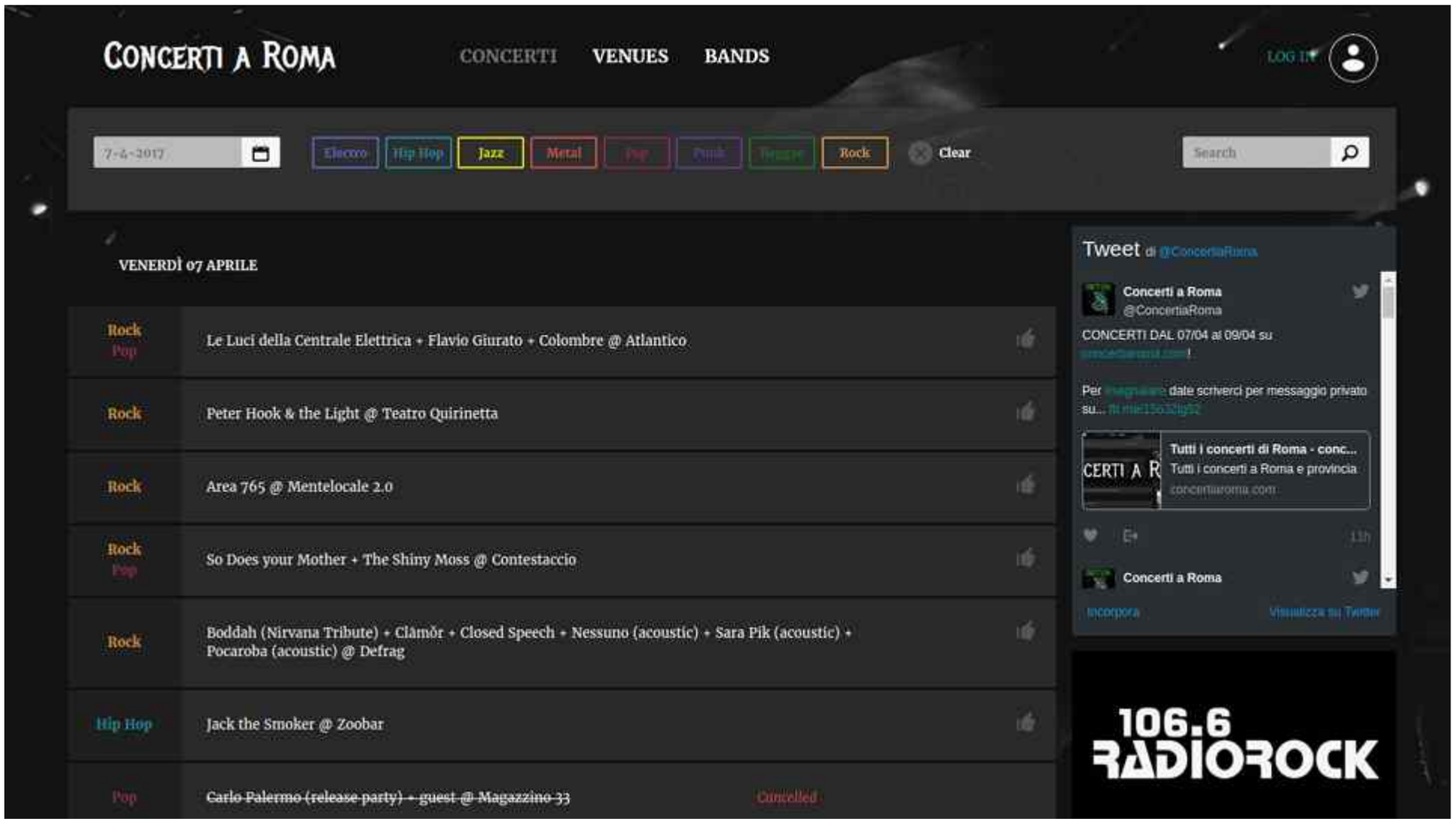
The numbers of the project:

- > 11,000 bands
- ~ 1,000 venues
- ~ 1,500 shows
- > 100 festivals
- ~ 15,000 user/month

PYCON OTTO

twentytab

Concerti@Roma 2

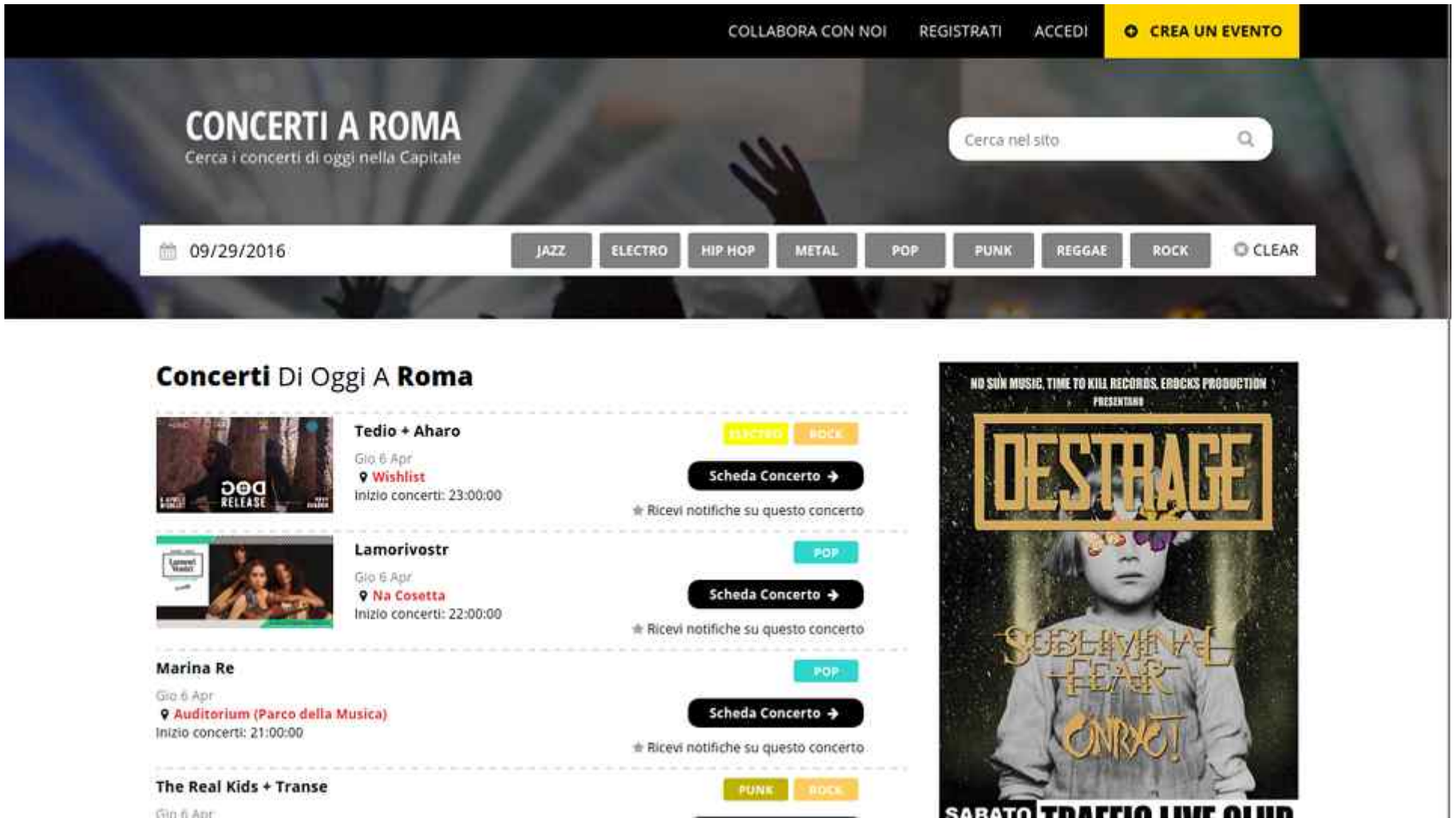


- Python2.7 / Django 1.7 / PostgreSQL 9.1 / `Like`

PYCON OTTO 8

twentytab

Concerti@Roma 3



- Python3.6 / Django1.11 / PostgreSQL 9.6 / FTS

PYCON OTTO

twentytab

C@R Manager

```
class BandManager(models.Manager):
  def search(self, text):
    vector = (
      SearchVector('nickname', weight='A', config=LANG) +
      SearchVector('genres__name', weight='B', config=LANG)+
      SearchVector('description', weight='D', config=LANG)
)
query = SearchQuery(text, config=LANG)
rate = SearchRank(vector, query)
return self.get_queryset().annotate(rate=rate).filter(
  search=query).annotate(search=vector).distinct(
  'id', 'rate').order_by('-rate', 'id')
```

C@R Test Setup

```
class BandTest(TestCase):
    def setUp(self):
        metal, _ = Genre.objects.get_or_create(name='Metal')
        doom, _ = Genre.objects.get_or_create(name='Doom')
        doomraiser, _ = Contact.objects.get_or_create(
            nickname='Doom raiser', description='Lorem…')
        doomraiser.genres.add(doom)
        forgotten_tomb, _ = Contact.objects.get_or_create(
            nickname='Forgotten Tomb', description='Lorem…')
        forgotten_tomb.genres.add(doom)
        ....
```

twentytab

C@R Test Method

```
class BandTest(TestCase):
    def setUp(self):
        ...

    def test_band_search(self):
        band_queryset = Band.objects.search(
            'doom').values_list('nickname', 'rate')
        band_list = [
            ('Doom raiser', 0.675475),
            ('The Foreshadowin', 0.258369),
            ('Forgotten Tomb', 0.243171)]
        self.assertSequenceEqual(list(
            OrderedDict(band_queryset).items()), band_list)
```

PYCON OTTO

twentytab

What's next

- Multiple language ranking
- Search suggestions
- SearchVectorField with triggers
- JSON/JSONB Full-Text Search
- RUM indexing

PYCON OTTO

twentytab

Conclusions

Conditions to implement this solution:

- **No extra dependencies**
- **Not too complex searches**
- **Easy management**
- **No need to synchronize data**
- **PostgreSQL already in your stack**
- **Python-only enviroment**

PYCON OTTO

twentytab

Resources

- https://docs.djangoproject.com/en/1.11/ref/contrib/postgres/search/
- https://www.postgresql.org/docs/9.6/static/textsearch.html
- https://github.com/damoti/django-tsvector-field
- https://en.wikipedia.org/wiki/Full-text_search

PYCON OTTO

twentytab

Acknowledgements

twentytab

- 20tab
- Marc Tamlyn `django.contrib.postgres`

PYCON OTTO

twentytab

Thank you

BY - SA (Attribution-ShareAlike)

- https://creativecommons.org/licenses/by-sa/4.0/

Slides

- https://speakerdeck.com/pauloxnet

PYCON OTTO

twentytab

Social Time

PYCON OTTO

twentytab

Questions ?

PYCON OTTO

twentytab

Contacts

https://twitter.com/pauloxnet

https://linkedin.com/in/paolomelchiorre

https://github.com/pauloxnet

PYCON OTTO