# Tutorial Django - Parte 2
# Django ORM and Fixtures

Régis da Silva
about.me/rg3915

github.com/grupy-sp/encontros

24 de Outubro de 2015

# Livraria

**Tema**: Modelagem de banco de dados de uma **livraria**.

Começando...

```
$ git clone https://github.com/rg3915/
      django-orm.git
$ virtualenv -p python3 django-orm
$ cd django-orm
$ source bin/activate
$ make initial
$ make fixtures
$ ./manage.py runserver
```

# Ementa

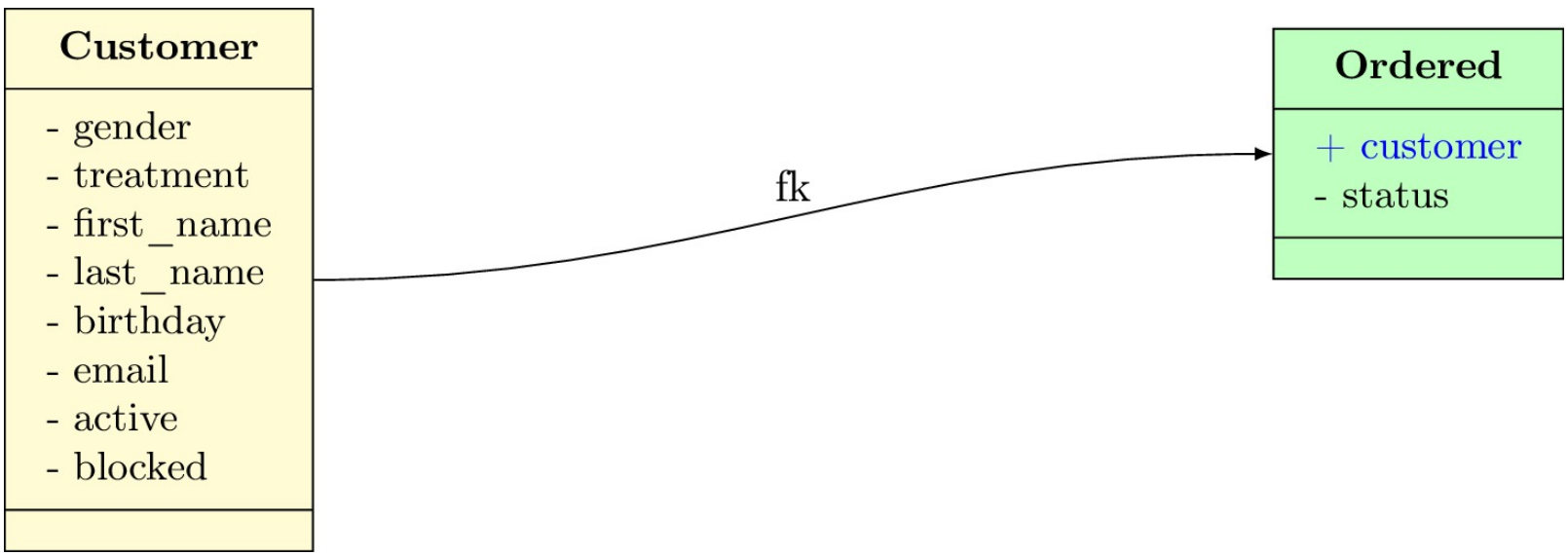
- Modelagem
  - OneToMany
  - OneToOne
  - ManyToMany
  - Abstract Inheritance
  - Multi-table Inheritance
- Fixtures
  - random values
  - csv
  - shell do Django
- Conclusão

# Objetivo

- Criar vários modelos de dados
- Popular o banco de dados

# OneToMany (um para muitos)

É o relacionamento onde usamos **chave estrangeira**, conhecido como **ForeignKey**.

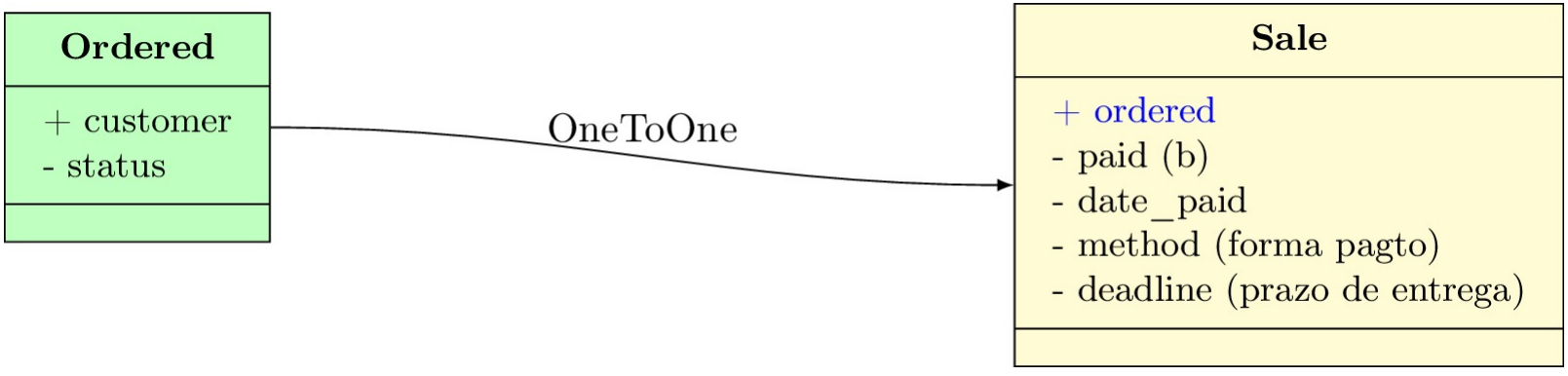



Um **cliente** pode fazer vários **pedidos**, então para reproduzir o esquema acima, usamos o seguinte código:

```
1  class Customer(models.Model):
2      gender = models.CharField(_(u'gênero'), max_ler
3      treatment = models.CharField(
4          _('tratamento'), max_length=4, choices=trea
5      first_name = models.CharField(_('nome'), max_le
6      last_name = models.CharField(_('sobrenome'), ma
7      birthday = models.DateTimeField(_('nascimento')
8      email = models.EmailField(_('e-mail'), blank=Tr
9      active = models.BooleanField(_('ativo'), defaul
10     blocked = models.BooleanField(_('bloqueado'), c
11
12
13 class Ordered(TimeStampedModel):
14     customer = models.ForeignKey(
15         'Customer', verbose_name=_('cliente'), rela
16     status = models.CharField(
17         _('status'), max_length=2, choices=status_l
```

# OneToOne (um para um)

Neste tipo de relacionamento também usamos **chave estrangeira**, só que um registro de uma tabela se relaciona apenas com um registro da outra tabela.

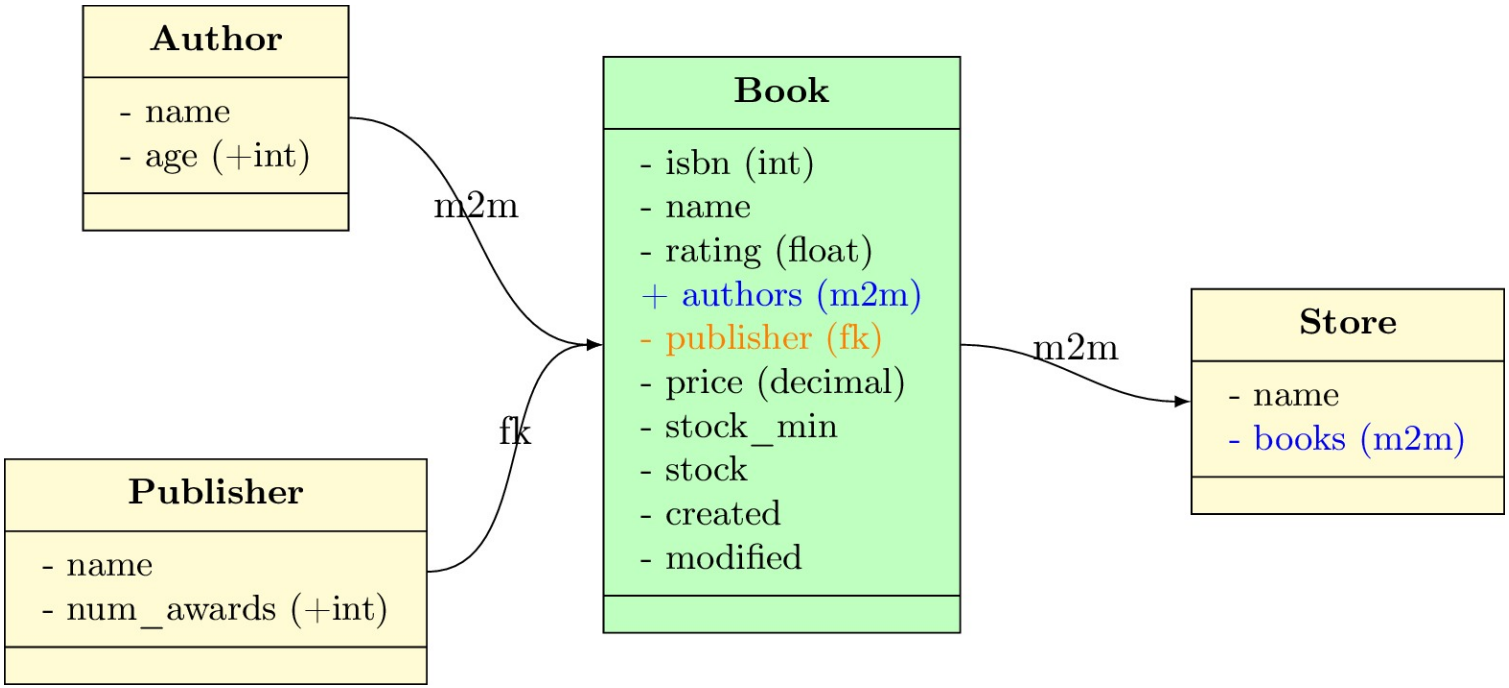



Uma **venda** pode ser feita a partir de apenas um **pedido**, então para reproduzir o esquema acima, usamos o seguinte código:

```
1  class Ordered(TimeStampedModel):
2      customer = models.ForeignKey(
3          'Customer', verbose_name=_('cliente'), rela
4      status = models.CharField(
5          _('status'), max_length=2, choices=status_l
6
7
8  class Sale(models.Model):
9      ordered = models.OneToOneField('Ordered',
10                                    verbose_name=_('pedido'))
11
12     paid = models.BooleanField(_('pago'), default=F
13     date_paid = models.DateTimeField(_('pago em'),
14     method = models.CharField(_('forma de pagto'),
15     deadline = models.CharField(
16         _('prazo de entrega'), max_length=50, blank
```

# ManyToMany (muitos para muitos)

Este relacionamento permite que vários registros de uma tabela se relacione com vários registros da outra tabela.




Um **autor** pode ter vários **livros** e cada **livro** pode ter vários **autores**, então para reproduzir o esquema acima, usamos o seguinte código:

```
1  class Author(models.Model):
2      name = models.CharField(_('nome'), max_length=5
3      age = models.PositiveIntegerField(_('idade'))
4
5
6  class Book(TimeStampedModel):
7      isbn = models.IntegerField()
8      name = models.CharField(_('nome'), max_length=5
9      rating = models.FloatField(_(u'classificação'))
10
11     authors = models.ManyToManyField('Author',
12                                      verbose_name='autores')
13
14     publisher = models.ForeignKey('Publisher', verb
15     price = models.DecimalField(_(u'preço'), max_di
16     stock_min = models.PositiveIntegerField(_(u'Est
17     stock = models.IntegerField(_('Estoque atual'))
```

E o mesmo para **lojas**.

```
1 class Store(models.Model):
2     name = models.CharField(_('nome'), max_length=5
3     books = models.ManyToManyField('Book', verbose_
```

Por baixo dos panos o Django cria uma terceira tabela (escondida).

```
(django-orm):/django-orm$ sqlite3 db.sqlite3
SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
auth_group                  core_pf
auth_group_permissions      core_pj
auth_permission             core_publisher
auth_user                   core_sale
auth_user_groups            core_seller
auth_user_user_permissions  core_store
core_author                 core_store_books
core_book                   django_admin_log
core_book_authors           django_content_type
core_customer               django_migrations
core_ordered                django_session
sqlite> .header on
sqlite> select * from core_book_authors;
id|book_id|author_id
1|1|1
2|1|2
3|2|3
4|2|4
5|3|5
6|4|5
7|5|6
8|6|7
9|7|8
10|8|9
```

Neste caso, temos dois livros com dois autores cada.

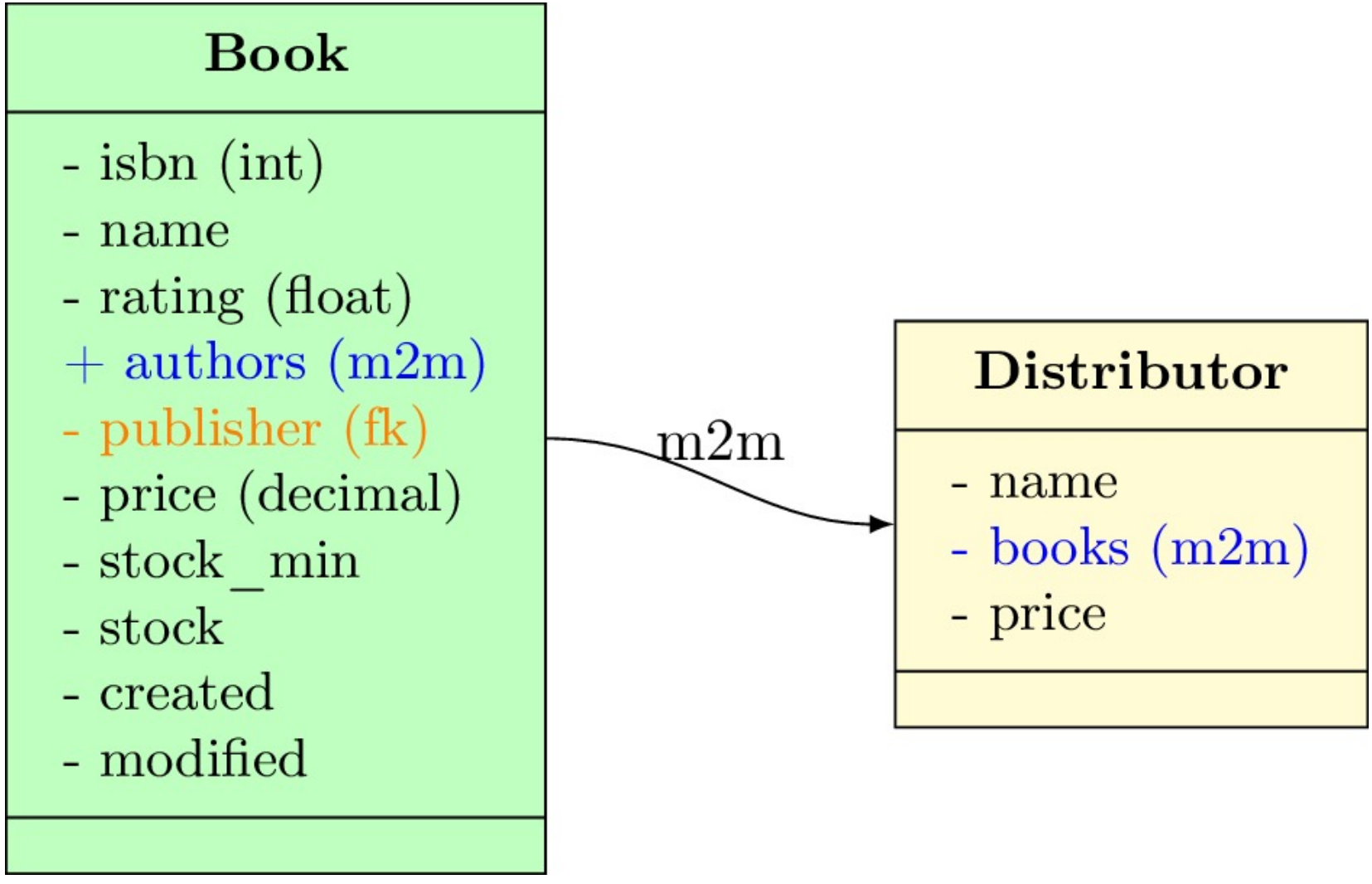
```
id|book_id|author_id
1 |1      |1
2 |1      |2
3 |2      |3
4 |2      |4
```

E ainda, na sequência temos dois livros diferentes do mesmo autor.

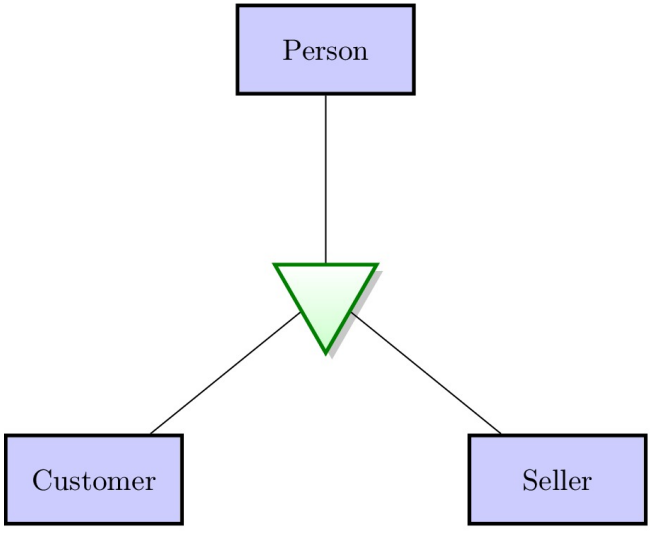
```
id|book_id|author_id
5 |3      |5
6 |4      |5
```

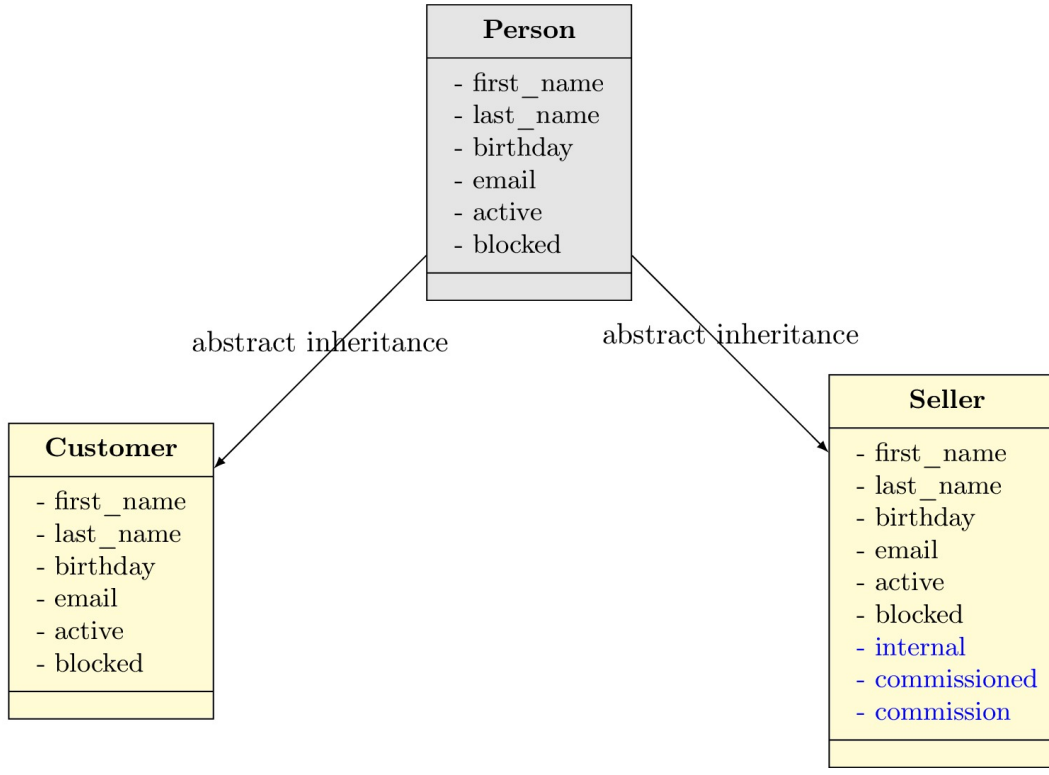
# Mais um exemplo

Um outro exemplo legal é o caso onde vários **livros** podem ser entregues por vários **fornecedores**.

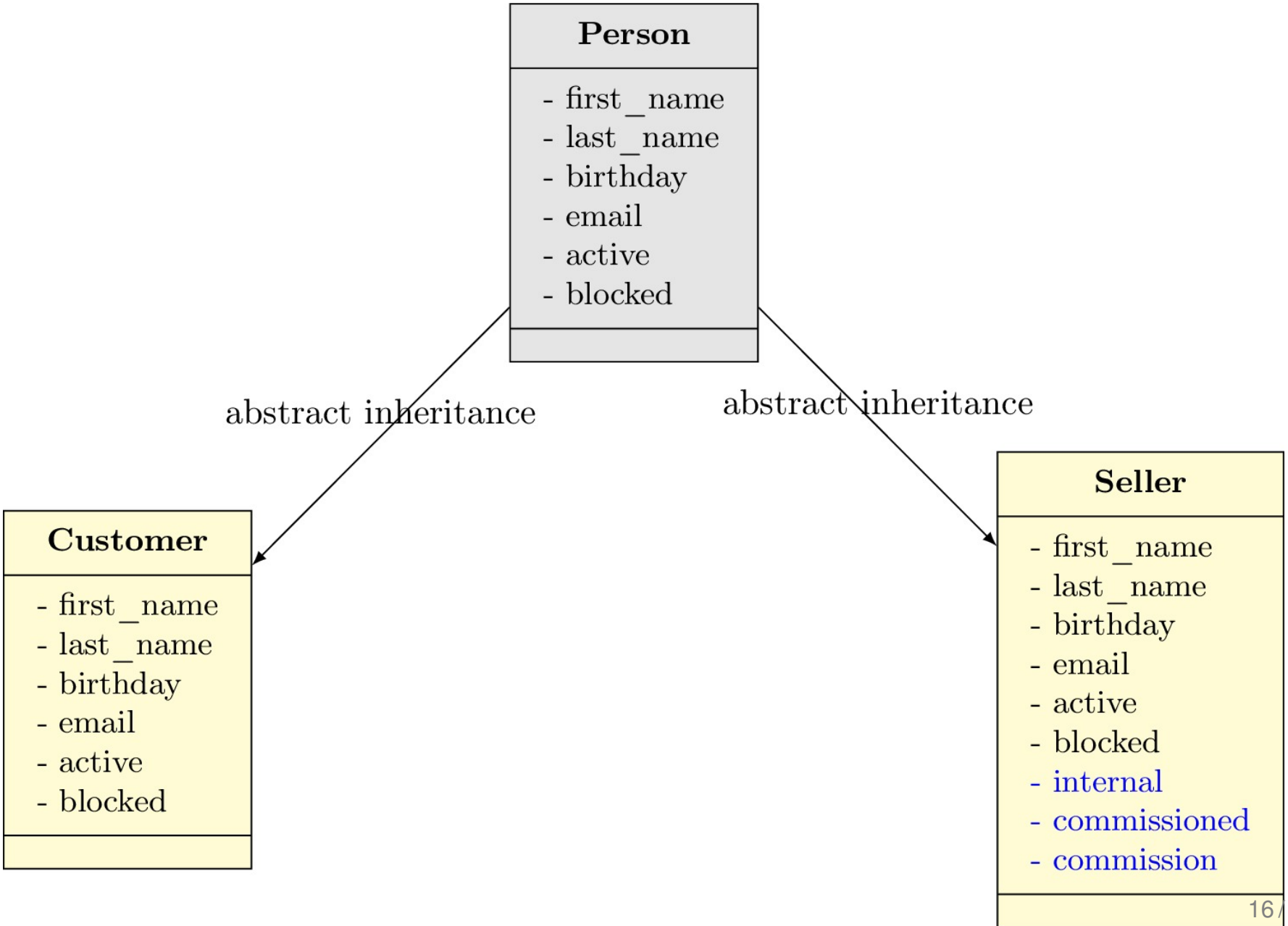


# Abstract Inheritance (Herança Abstrata)



# Abstract Inheritance (Herança Abstrata)



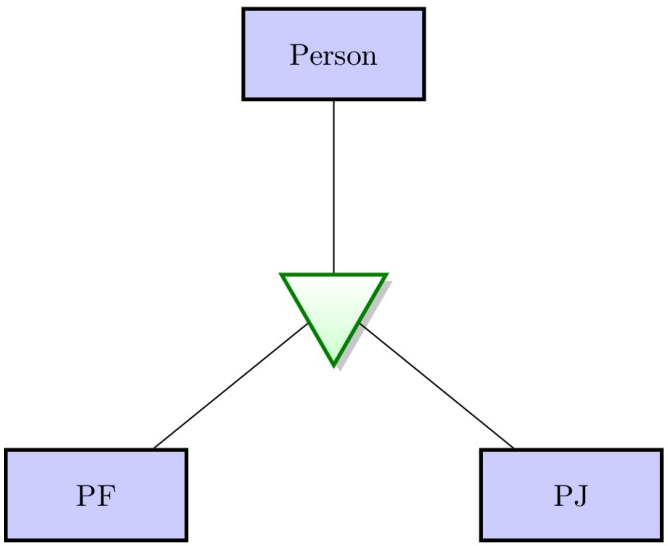
```
1  class Person(models.Model):
2      gender = models.CharField(_(u'gênero'), max_len
3      treatment = models.CharField(
4          _('tratamento'), max_length=4, choices=trea
5      first_name = models.CharField(_('nome'), max_le
6      last_name = models.CharField(_('sobrenome'), ma
7      birthday = models.DateTimeField(_('nascimento')
8      email = models.EmailField(_('e-mail'), blank=Tr
9      active = models.BooleanField(_('ativo'), defaul
10     blocked = models.BooleanField(_('bloqueado'), d
11
12 class Meta:
13     abstract = True
14
15
16 class Customer(Person):
17     pass
```

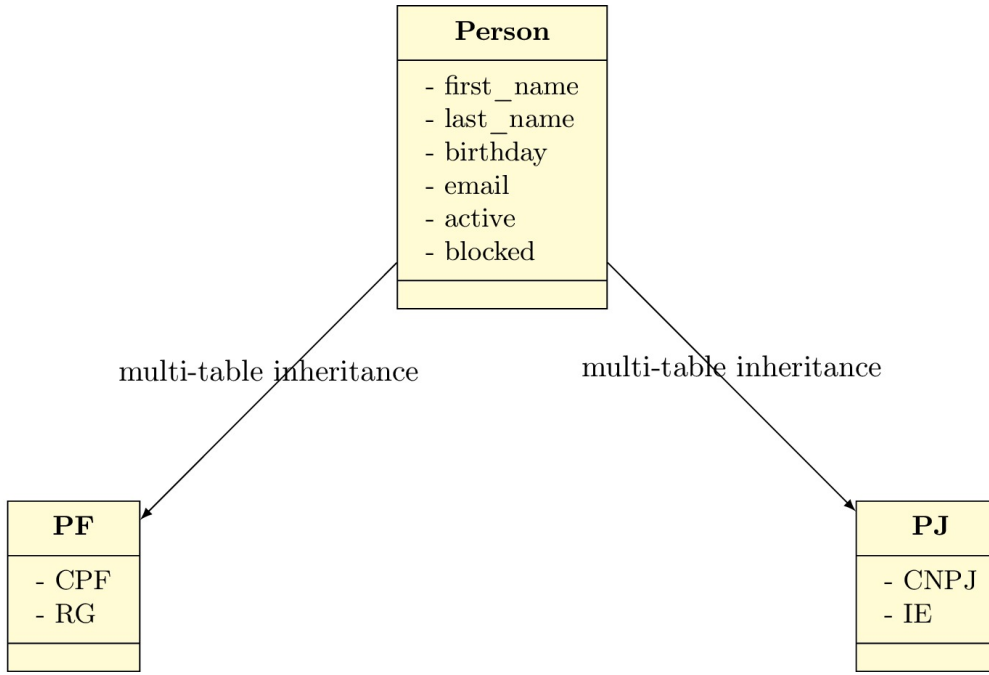
```
1 class Seller(Person):
2     internal = models.BooleanField(_('interno'), de
3     commissioned = models.BooleanField(_('comission
4     commission = models.DecimalField(
5         _(u'comissão'), max_digits=6, decimal_place
```

Note que a tabela **Customer** é uma cópia de **Person**, e **Seller** também é uma cópia, mas com campos adicionais.

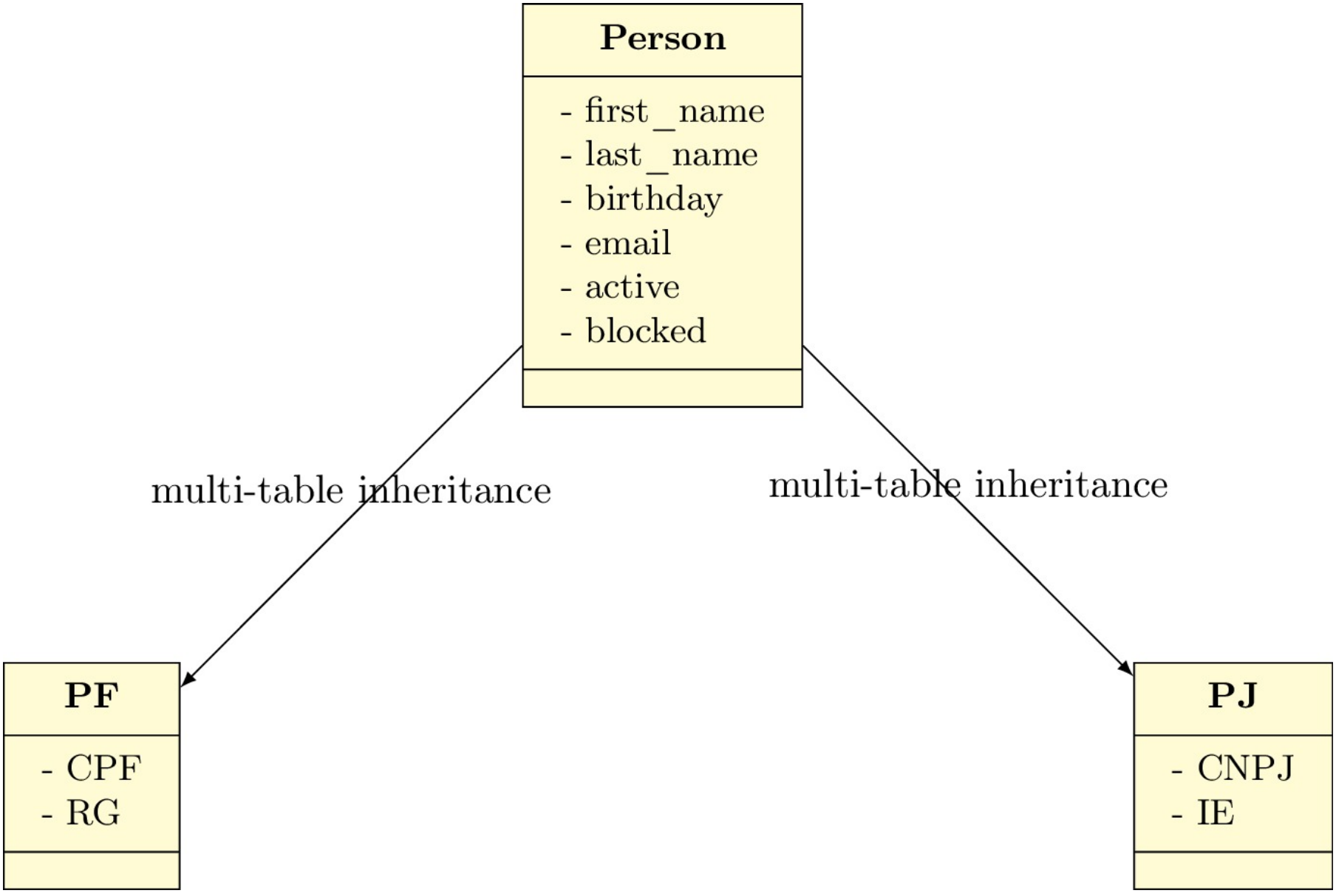
# Multi-table Inheritance (Herança Multi-tabela)

Na herança múltipla o Django cria um relacionamento **um pra um (OneToOne)** automaticamente entre as tabelas.



# Multi-table Inheritance (Herança Multi-tabela)



Entrando no banco de dados vemos que a tabela `core_pf` possui um campo chamado `customer_ptr_id`....

```
(django-orm):/django-orm$ sqlite3 db.sqlite3
SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
auth_group                  core_pf
auth_group_permissions      core_pj
auth_permission             core_publisher
auth_user                   core_sale
auth_user_groups            core_seller
auth_user_user_permissions  core_store
core_author                 core_store_books
core_book                   django_admin_log
core_book_authors           django_content_type
core_customer               django_migrations
core_ordered                django_session
```

```
sqlite> .header on
sqlite> select * from core_pf;
customer_ptr_id|cpf|rg
21|73672676940|8711063350
22|00658098048|2530034077
23|16883989134|6599812885
24|46059878947|1547810704
25|78421481737|8988704462
26|20425523197|9261816529
27|31826335337|4607982483
28|29609540974|9493016489
29|91730182091|5520358350
30|66280845193|5160507105
31|88322580003|3902497230
32|45352190718|4369012653
33|09718147300|2817845095
34|31518519176|8318418262
35|57355240057|4426106553
36|17912744940|3352263910
37|13934177441|7707126560
38|50693983564|7023885413
39|70921573457|7472980426
40|17133023651|7997212582
sqlite>
```

... e que os ids vão de 21 a 40, neste exemplo.

Note que são os mesmos ids na tabela `customer`.

```
sqlite> select * from core_customer where id >= 18 and id <= 42;
id|gender|treatment|first_name|last_name|birthday|email|active|blocked
18|F|d|Stacy|Stangl|1949-12-05 13:30:55.595746|s.stangl@example.com|1|0
19|F|pa|Miranda|Hanson|1981-11-07 01:13:40.670914|m.hanson@example.com|0|0
20|M|e|Jack|Medina|1991-11-15 11:23:49.722211|j.medina@example.com|0|0
21|M|e|Gerald|Peterson|1981-12-10 17:12:47.898895|g.peterson@example.com|0|0
22|F|aa|Esther|Bramlett|1992-11-20 20:20:39.854553|e.bramlett@example.com|1|0
23|F|srta|Krista|Jackson|1991-11-18 15:15:45.769478|k.jackson@example.com|0|1
24|M|e|Gerardo|Hubbard|1928-11-24 04:09:28.600169|g.hubbard@example.com|0|1
25|M|dr|Craig|Hill|1994-11-27 09:53:58.441432|c.hill@example.com|1|1
26|F|sra|Harriet|Tawwater|1928-11-27 15:50:24.922173|h.tawwater@example.com|1|0
27|M|p|Mitchell|Mossien|1946-12-26 03:56:43.911240|m.mossien@example.com|0|1
28|F|d|Monica|Lawrence|1979-12-12 14:12:24.901546|m.lawrence@example.com|0|0
29|M|e|Jerry|Zimmerman|1992-11-24 08:20:04.290605|j.zimmerman@example.com|0|1
30|F|aa|Judith|Capps|1927-11-11 05:11:26.738969|j.capps@example.com|0|1
31|F|pa|Patricia|Gonzalez|1931-12-03 20:08:33.829481|p.gonzalez@example.com|0|1
32|M|e|Isaiah|Norman|1956-11-24 08:01:13.940520|i.norman@example.com|0|1
33|M|e|Albert|Meier|1961-11-21 19:11:28.506320|a.meier@example.com|0|0
34|M|e|Manuel|Costa|1927-12-03 14:39:07.762194|m.costa@example.com|1|0
35|M|a|Henry|Lucas|1945-11-25 11:38:55.132624|h.lucas@example.com|1|1
36|F|aa|Virginia|Thomas|1968-11-09 03:05:39.023918|v.thomas@example.com|1|0
37|F|srta|Rachel|Bonine|1972-12-15 05:05:30.772307|r.bonine@example.com|1|0
38|F|sra|Angela|Conrad|1923-11-22 08:53:49.621286|a.conrad@example.com|0|0
39|F|sra|Inez|Coleman|1917-12-02 16:06:25.840966|i.coleman@example.com|1|1
40|F|pa|Holly|Goldman|1984-11-19 23:34:07.085878|h.goldman@example.com|1|0
41|M|dr|Sam|Shaffer|1966-11-23 06:03:32.513383|s.shaffer@example.com|0|1
42|F|srta|Dollie|Wallace|1947-11-22 12:27:44.980943|d.wallace@example.com|0|0
sqlite>
```

E se você digitar...

```
sqlite> .schema core_pf
CREATE TABLE "core_pf" ("customer_ptr_id"
integer NOT NULL PRIMARY KEY REFERENCES
"core_customer" ("id"),
"cpf" varchar(11) NOT NULL,
"rg" varchar(10) NOT NULL);
```

... você verá nitidamente que existe um relacionamento um pra um entre eles.

# Fixtures

Vamos criar nossas próprias fixtures usando

- Python
- csv
- shell do Django

# random values

Vamos precisar do `rstr`.

```
$ pip install rstr
```

https://pypi.python.org/pypi/rstr/2.1.3

```
$ python
>>> import rstr
>>> rstr.rstr('abcde',10)
'ddcbeedacb'
```

Apenas uma amostra do poder do Python.

```
1  # gen_random_values.py
2  import random
3  import rstr
4  import datetime
5  from decimal import Decimal
6
7
8  def gen_age(min_age=15, max_age=99):
9      # gera numeros inteiros entre 15 e 99
10     return random.randint(min_age, max_age)
```

```
1  def gen_doc(doc='cpf'):
2      if doc == 'cpf':
3          return rstr.rstr('1234567890', 11)
4      elif doc == 'cnpj':
5          return rstr.rstr('1234567890', 14)
6      elif doc == 'rg':
7          return rstr.rstr('1234567890', 10)
```

```
1  def gen_phone():
2      # gera um telefone no formato (xx) xxxx-xxxx
3      return '({0}) {1}-{2}'.format(
4          rstr.rstr('1234567890', 2),
5          rstr.rstr('1234567890', 4),
6          rstr.rstr('1234567890', 4))
```

```python
def gen_timestamp(min_year=1915, max_year=1996):
    # gera um datetime no formato
    # yyyy-mm-dd hh:mm:ss.000000
    year = random.randint(min_year, max_year)
    month = random.randint(11, 12)
    day = random.randint(1, 28)
    hour = random.randint(1, 23)
    minute = random.randint(1, 59)
    second = random.randint(1, 59)
    microsecond = random.randint(1, 999999)
    date = datetime.datetime(
        year, month, day, hour, minute,
        second, microsecond)
        .isoformat(" ")
    return date
```

```
1  def gen_decimal(max_digits, decimal_places):
2    num_as_str = lambda x: ''.join(
3      [str(random.randint(0, 9)) for i in range(x)])
4    return Decimal("%s.%s" % (num_as_str(max_digits
5                                - decimal_places),
6                              num_as_str(decimal_plac
7  gen_decimal.required = ['max_digits', 'decimal_plac
```

# names

Agora vamos precisar do `names`.

```
$ pip install names
```

https://pypi.python.org/pypi/names/

```
$ python
>>> import names
>>> names.get_first_name(gender='male')
'Jean'
>>> names.get_first_name(gender='female')
'Emily'
>>> names.get_last_name()
'Oconnor'
```

E vejamos como gerar nomes aleatórios.

```
# gen_names.py
import random
import names
""" List of values for use in choices in models. "
treatment_male_list = ('a','dr','e','p','sr',)
treatment_female_list = ('aa','d','ea','pa','sra','

def gen_male_first_name():
    treatment = random.choice(treatment_male_list)
    first_name = names.get_first_name(gender='male'
    c = {'treatment': treatment, 'first_name': firs
    return c

def gen_female_first_name():
    treatment = random.choice(treatment_female_list
    first_name = names.get_first_name(gender='femal
    c = {'treatment': treatment, 'first_name': firs
    return c

```

# CSV

Para ler um `csv` façamos o seguinte:

```
1  import csv
2
3  book_list = []
4
5  ''' Lendo os dados de books_.csv '''
6  with open('fixtures/csv/books_.csv', 'r') as f:
7      r = csv.DictReader(f)
8      for dct in r:
9          book_list.append(dct)
10     f.close()
```

Com isso nós temos uma **lista** onde os valores são **dicionários**.

```
[{'name': 'O diário de Anne Frank', 'publisher'   'R
  'authors': 'Mirjam Pressler'},
 {'name': 'O diário de Anne Frank', 'publisher'   'R
   'authors': 'Otto H. Frank'},
 {'name': 'Deixados Para Trás', 'publisher': 'Unite
   'authors': 'Jerry B. Jenkins'},
 {'name': 'Deixados Para Trás', 'publisher': 'Unite
   'authors': 'Tim LaHaye'},
 {'name': 'Jardim secreto', 'publisher': 'Sextante'
   'authors': 'Johanna Basford'},
 {'name': 'Floresta encantada', 'publisher': 'Sexta
   'authors': 'Johanna Basford'},
  ...
]
```

# shell do Django

```
$ python manage.py shell
```

# O que você precisa saber?

```
from core.models import Book, Author, Publisher

''' Criando uma instância do objeto Publisher '''
publisher_obj = Publisher(name='Editora 34',
                          num_awards=8)

''' Salvando o objeto '''
publisher_obj.save()

''' Criando um Author direto com o comando create '
Author.objects.create(name='Dante Alighieri',
                      age=56)
''' Pegando o id de Author '''
author = Author.objects.get(name='Dante Alighieri')

''' Pegando o id de Publisher '''
publisher = Publisher.objects.get(
    pk=publisher_obj.id)
...
```

```
...
''' Criando um livro '''
book_obj = Book(
    name='A Divina Comédia',
    publisher=publisher,
    price=29.20,
)
book_obj.save()

''' Inserindo os autores nos livros '''
book = Book.objects.get(pk=book_obj.id)

''' Como o campo authors é ManyToMany devemos usar
book.authors.add(author)
```

Você pode salvar um arquivo `shell_book.py` e digitar

```
$ ./manage.py shell < fixtures/shell_book.py
```

# Vantagens

- Criando o seu próprio código você sabe o que está fazendo
- Você vai treinar muito Python
- Vai aprender a usar o shell do Django
- Fácil de inserir seus próprios dados

# Desvantagens

- Pode demorar um pouco para criar o código
- Difícil manutenção
- Se fizer uma migração no banco terá que refatorar o código

# Conclusão

- Ninguém recomenda
- Recomendam o mixer ou model-mommy
- Mas para inserir seus próprios dados é uma boa solução

Leia o `Makefile` para ver como foi executado cada comando.

Veja a pasta `fixtures` para ver os códigos Python que geram os valores.

# Obrigado!

Dúvidas?

# Tutorial Django - Parte 2
# Django ORM and Fixtures

Régis da Silva
about.me/rg3915

github.com/grupy-sp/encontros

24 de Outubro de 2015