# Visualizando query SQL a partir do ORM Django

**Beatriz Uezu**

# Hello!
# Beatriz Uezu

Formada em Análise e Desenvolvimento de Sistemas pela FATEC SP, coorganizadora do PyLadies São Paulo e Django Girls São Paulo e desenvolvedora Python no luizalabs

pyladies São Paulo

django girls

luizalabs

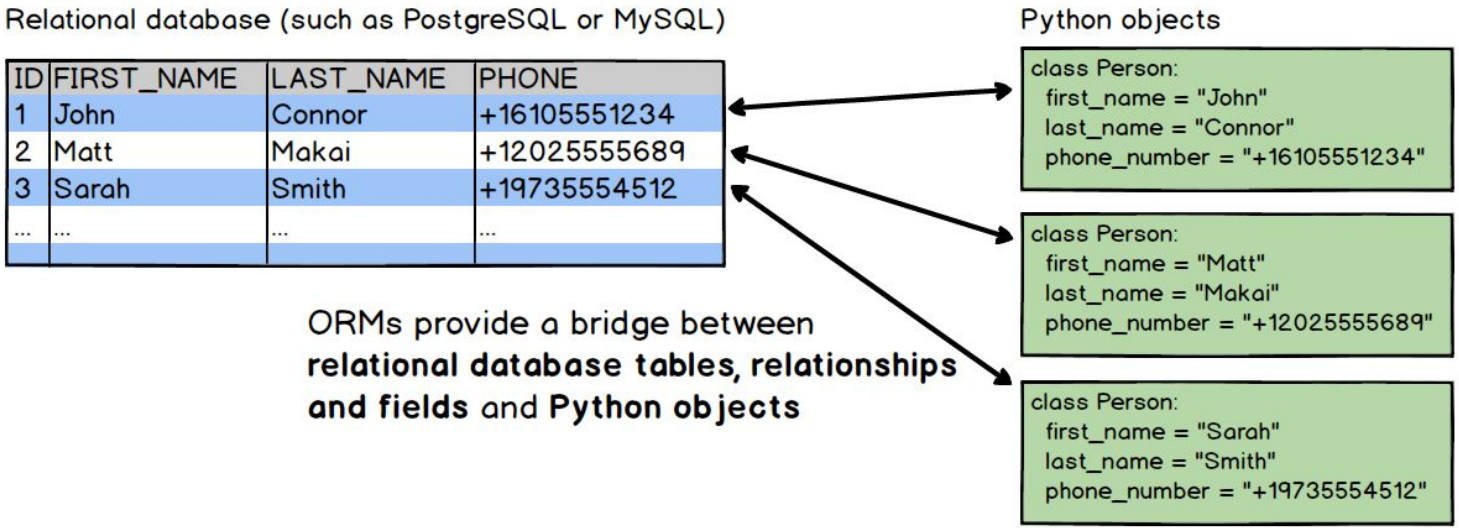
# Motivo

- Trabalhei com VB.net e SQL e migrei para Python/Django

- Tive dificuldade em entender como a ORM do Django funcionava e em como montar

# ORM O QUE?

- ORM (Object-Relational Mapper) é uma biblioteca que automatiza a transferência de dados do banco de dados relacional entre objetos do model.




https://www.fullstackpython.com/object-relational-mappers-orms.html

# Model vs Tabela

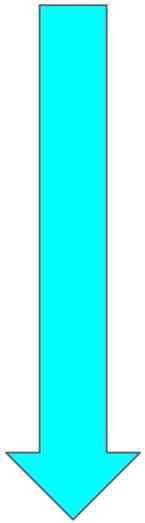
```python
# categoria/models.py
class Categoria(models.Model):
    nome = models.CharField('Nome', max_length=128)

# produto/models.py

class Produto(models.Model):
    nome = models.CharField(
        'Nome',
        max_length=128
    )
    valor = models.DecimalField(
        'Valor',
        max_digits=10,
        decimal_places=2,
        blank=True,
        null=True
    )
    categoria = models.ForeignKey(Categoria)
```

```
mysql> desc categoria_categoria;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| nome  | varchar(128)| NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

```
mysql> desc produto_produto;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| id          | int(11)      | NO   | PRI | NULL    | auto_increment |
| nome        | varchar(128) | NO   |     | NULL    |                |
| valor       | decimal(10,2)| YES  |     | NULL    |                |
| categoria_id| int(11)      | NO   | MUL | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
```

# Como funciona?

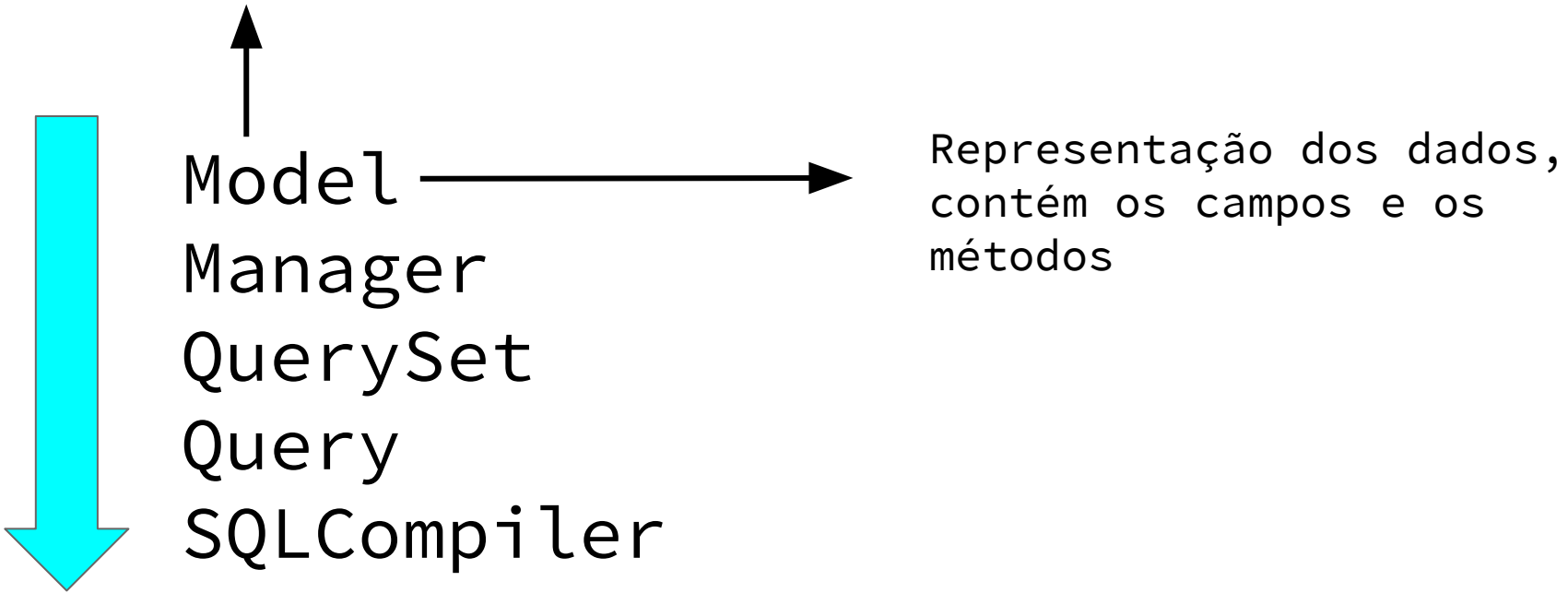
```
>> Produto.objects.all()
```



Model
Manager
QuerySet
Query
SQLCompiler

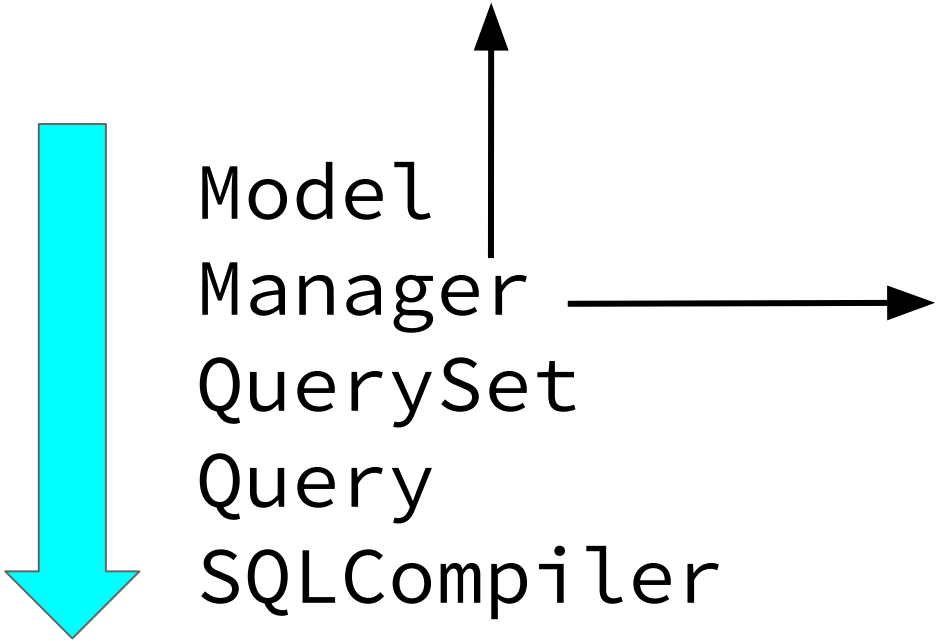
# Como funciona?

```
>> Produto.objects.all()
```



# Como funciona?

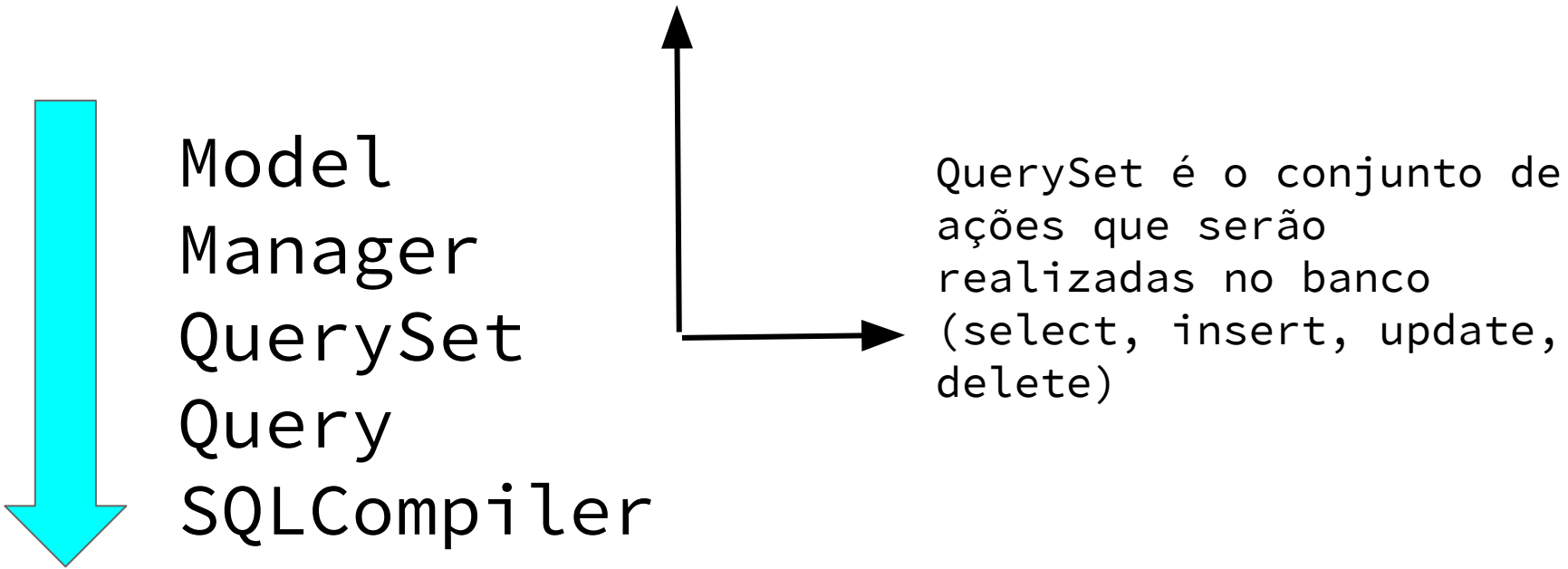
```
>> Produto.objects.all()
```




Está acoplado ao um Model, para acessar qualquer objetos salvo no banco é preciso acessar o Manager, isto é o objects presente em todo Model

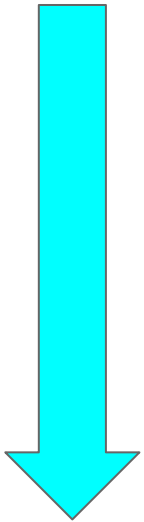
# Como funciona?

```
>> Produto.objects.all()
```



# Como funciona?

```
>> Produto.objects.all()
```



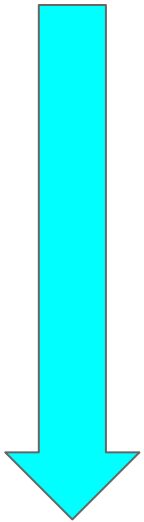
Model
Manager
QuerySet
Query
SQLCompiler

Cria uma estrutura de dados complexa com todos os elementos presentes em uma consulta.

Gera uma representação SQL de um QuerySet

# Como funciona?

```
>> Produto.objects.all()
```



Model
Manager
QuerySet
Query
SQLCompiler ⟶

Recebe o SQL da Query e executa de acordo com as regras específicas do backend escolhido

# .all()

```
>> Produto.objects.all()
<QuerySet [<Produto: Produto 1 da Categoria 1>,
<Produto: Produto 2 da Categoria 1>, <Produto:
Produto 3 da Categoria 1>, <Produto: Produto 4 da
Categoria 1>, <Produto: Produto 5 da Categoria 1>,
<Produto: Produto 1 da Categoria 2>, <Produto:
Produto 2 da Categoria 2>, <Produto: Produto 3 da
Categoria 2>, <Produto: Produto 4 da Categoria 2>,
<Produto: Produto 5 da Categoria 2>, <Produto:
Produto 1 da Categoria 3>, <Produto: Produto 2 da
Categoria 3>, <Produto: Produto 3 da Categoria 3>,
<Produto: Produto 4 da Categoria 3>, <Produto:
Produto 5 da Categoria 3>, <Produto: Produto 1 da
Categoria 4>, <Produto: Produto 2 da Categoria 4>,
<Produto: Produto 3 da Categoria 4>, <Produto:
Produto 4 da Categoria 4>, <Produto: Produto 5 da
Categoria 4>, '...(remaining elements
truncated)...']>
```

```
>> orm = Produto.objects.all()

>> print(orm.query)
SELECT `produto_produto`.`id`,
`produto_produto`.`nome`,
`produto_produto`.`valor`,
`produto_produto`.`categoria_id`
FROM `produto_produto`
```

# .filter()

```
>> Produto.objects.filter(id=1)
<QuerySet [<Produto: Produto 1 da
Categoria 1>]>
```

```
>> orm = Produto.objects.filter(id=1)

>> print(orm.query)
SELECT `produto_produto`.`id`,
`produto_produto`.`nome`,
`produto_produto`.`valor`,
`produto_produto`.`categoria_id` FROM
`produto_produto` WHERE
`produto_produto`.`id` = 1
```

# .filter(campo_lookup)

- contains
- in
- gt
- gte
- lt
- lte
- range
- date

...

- year
- month
- day
- time
- hour
- minute
- hour
- isnull
-

```
>> Produto.objects.filter(id__in=[1,3,5])

>>> orm =
Produto.objects.filter(id__in=[1, 3 ,5])

>>> print(orm.query)
SELECT `produto_produto`.`id`,
`produto_produto`.`nome`,
`produto_produto`.`valor`,
`produto_produto`.`categoria_id` FROM
`produto_produto` WHERE
`produto_produto`.`id` IN (1, 3, 5)
```

https://docs.djangoproject.com/en/1.11/ref/models/querysets/#field-lookups

# .filter(fk__campo)

```
>> Produto.objects.filter(
categoria__nome='categoria_1'
)
```

```
<QuerySet [<Produto: Produto 1 da
Categoria 1>, <Produto: Produto 2 da
Categoria 1>, <Produto: Produto 3 da
Categoria 1>, <Produto: Produto 4 da
Categoria 1>, <Produto: Produto 5 da
Categoria 1>]>
```

```
>> orm =
Produto.objects.filter(categoria__nome=
'categoria_1')

>> print(orm.query)
SELECT `produto_produto`.`id`,
`produto_produto`.`nome`,
`produto_produto`.`valor`,
`produto_produto`.`categoria_id` FROM
`produto_produto` INNER JOIN
`categoria_categoria` ON
(`produto_produto`.`categoria_id` =
`categoria_categoria`.`id`) WHERE
`categoria_categoria`.`nome` = categoria_1
```

# .get()

```
>> Produto.objects.get(id=1)
<Produto: Produto 1 da
Categoria 1>
```

```
>> script =
Produto.objects.get(id=1)

>> script.query
AttributeError: 'Produto' object
has no attribute 'query'
```

# .GET()

```
>> Produto.objects.get(id=1)
<Produto: Produto 1 da
Categoria 1>
```

```
>> from django.dd import connection

>> connection.queries

[{'sql': 'SELECT
`produto_produto`.`id`,
`produto_produto`.`nome`,
`produto_produto`.`valor`,
`produto_produto`.`categoria_id`
FROM `produto_produto` WHERE
`produto_produto`.`id` = 1',
'time': '0.000'}]
```

# .filter() vs .get()

```
>> script = Produto.objects.filter(id=1)

>> type(script)
```

```
django.db.models.query.QuerySet
```

```
>>  script = Produto.objects.get(id=1)

>> type(script)
```

```
core.produto.models.Produto
```

# .exclude()

```
>> Produto.objects.exclude(id__gte=4)

 <QuerySet [<Produto: Produto 1 da
Categoria 1>, <Produto: Produto 2 da
Categoria 1>, <Produto: Produto 3 da
Categoria 1>]>
```

```
>> orm =
Produto.objects.exclude(id__gte=4)

>> print(orm.query)
SELECT `produto_produto`.`id`,
`produto_produto`.`nome`,
`produto_produto`.`valor`,
`produto_produto`.`categoria_id`
FROM `produto_produto` WHERE NOT
(`produto_produto`.`id` >= 4)
```

# .values()

```
>> Produto.objects.values('nome', 'valor')

<QuerySet [{'valor': Decimal('2.00'), 'nome':
'Produto 1 da Categoria 1'}, {'valor':
Decimal('5.00'), 'nome': 'Produto 2 da
Categoria 1'}, {'valor': Decimal('20.00'),
'nome': 'Produto 3 da Categoria 1'},
{'valor': Decimal('15.00'), 'nome': 'Produto
4 da Categoria 1'}, {'valor':
Decimal('10.00'), 'nome': 'Produto 5 da
Categoria 1'}, {'valor': Decimal('5.00'),
'nome': 'Produto 1 da Categoria 2'},
{'valor': Decimal('7.00'), 'nome': 'Produto 2
da Categoria 2'}, {'valor': Decimal('20.00'),
'nome': 'Produto 3 da Categoria 2'},
'...(remaining elements truncated)...']>
```

```
>> orm =
Produto.objects.values('nome',
'valor')

>> print(orm.query)
SELECT `produto_produto`.`nome`,
`produto_produto`.`valor` FROM
`produto_produto`
```

# .values_list()

```
>> Produto.objects.values_list('nome')

<QuerySet [('Produto 1 da Categoria
1',), ('Produto 2 da Categoria 1',),
('Produto 3 da Categoria 1',),
('Produto 4 da Categoria 1',),
('Produto 5 da Categoria 1',),
('Produto 1 da Categoria 2',),
('Produto 2 da Categoria 2',),
('Produto 3 da Categoria 2',),
('Produto 4 da Categoria
2',),'...(remaining elements
truncated)...']>
```

```
>>  orm =
Produto.objects.values_list('nome')

>> print(orm.query)

SELECT `produto_produto`.`nome`
FROM `produto_produto`
```

# .values_list(field, flat=True)

```
>> Produto.objects.values_list('nome', flat=True)

<QuerySet ['Produto 1 da Categoria 1', 'Produto 2 da Categoria 1', 'Produto 3 da Categoria 1', 'Produto 4 da Categoria 1', 'Produto 5 da Categoria 1', 'Produto 1 da Categoria 2', 'Produto 2 da Categoria 2', 'Produto 3 da Categoria 2', 'Produto 4 da Categoria 2', 'Produto 5 da Categoria 2', 'Produto 1 da Categoria 3', 'Produto 2 da Categoria 3', '...(remaining elements truncated)...']>
```

```
>> orm = Produto.objects.values_list('nome', flat=True)

>> print(orm.query)

SELECT `produto_produto`.`nome` FROM `produto_produto`
```

# .annotate()

```
>> from django.db.models import Sum

>>Produto.objects.values('categoria
').annotate(Sum('valor'))

<QuerySet [{'categoria': 1,
'valor__sum': Decimal('52.00')},
{'categoria': 2, 'valor__sum':
Decimal('66.00')}, {'categoria': 3,
'valor__sum': Decimal('147.60')},
{'categoria': 4, 'valor__sum':
Decimal('209.30')}, {'categoria':
5, 'valor__sum':
Decimal('87.20')}]>
```

```
>> orm =
Produto.objects.values('categoria')
.annotate(Sum('valor'))

 >> print(orm.query)
SELECT
`produto_produto`.`categoria_id`,
SUM(`produto_produto`.`valor`) AS
`valor__sum` FROM `produto_produto`
GROUP BY
`produto_produto`.`categoria_id`
ORDER BY NULL
```

# .aggregate()

```
>> from django.db.models import
Avg, Max, Min

>> Produto.objects.aggregate(
valor_maximo=Max('valor'),
valor_minimo=Min('valor'),
valor_medio=Avg('valor')
)

{'valor_maximo': Decimal('100.00'),
'valor_medio': 22.484,
'valor_minimo': Decimal('2.00')}
```

```
>> orm =
Produto.objects.aggregate(valor_max
imo=Max('valor'),
valor_minimo=Min('valor'),
valor_medio=Avg('valor'))

>> print(orm.query)
AttributeError: 'dict' object has
no attribute 'query'
```

# .aggregate()

```
>> from django.db.models import
Avg, Max, Min

>> Produto.objects.aggregate(
valor_maximo=Max('valor'),
valor_minimo=Min('valor'),
valor_medio=Avg('valor')
)

{'valor_maximo': Decimal('100.00'),
'valor_medio': 22.484,
'valor_minimo': Decimal('2.00')}
```

```
>> from django.db import connection

>> connection.queries

[{'sql': 'SELECT
MIN(`produto_produto`.`valor`) AS
`valor_minimo`,
MAX(`produto_produto`.`valor`) AS
`valor_maximo`,
AVG(`produto_produto`.`valor`) AS
`valor_medio` FROM `produto_produto`',
'time': '0.000'}]
```

https://docs.djangoproject.com/pt-br/1.11/faq/models/#faq-databases-and-models

# .annotate() vs .aggregate()

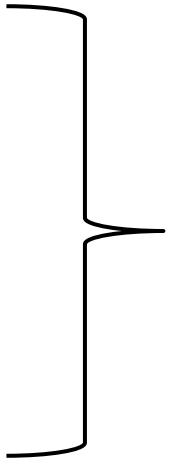
```
>>> script =
Produto.objects.values('categoria')
.annotate(Sum('valor'))


>> type(script)

django.db.models.query.QuerySet
```

```
>> script =
Produto.objects.aggregate(
valor_maximo=Max('valor'),
valor_minimo=Min('valor'),
valor_medio=Avg('valor')
)

>> type(script)

dict
```

https://docs.djangoproject.com/pt-br/1.11/faq/models/#faq-databases-and-models

# Queryset são lazy

- `.all()`
- `.filter()`
- `.annotate()`
- `.exclude()`
- `.values()`



`retornam um Queryset`

https://docs.djangoproject.com/en/1.11/ref/models/querysets/#methods-that-return-new-querysets

# Queryset são lazy

```
>> produtos = Produto.objects.all()

>> categoria_2 = produtos.filter(categoria=2)

>> lista_produto = produtos.values_list('nome', flat=True)

>> print(lista_produto)
```

# Queryset são lazy

QUANTAS CONSULTAS FORAM REALIZADAS NO BANCO DE DADOS?

## APENAS UMA!

# Queryset são lazy

```
>> produtos = Produto.objects.all()

>> categoria_2 = produtos.filter(categoria=2)

>> lista_produto = produtos.values_list(‘nome’, flat=True)

>> print(len(connection.queries))
    0
>> print(lista_produto)

>> print(len(connection.queries))
    1
```

# Queryset são lazy

Significa que as consultas são realizadas no banco de dados quando pedimos!

MAS QUANDO PEDIMOS?

# Queryset são lazy

Então podemos para serem executadas nas seguintes formas:

```
#quando solicitamos somente um resultado
>> Produto.objects.all()[0]

#quando fazemos slicing passando um ‘step’
>> Produto.objects.all()[::2]

#quando iteramos
>> categoria for categoria in Categoria.objects.all()
```

# Queryset são lazy

```
#quando chamamos o len()
>> len(Produto.objects.all())
```

```
#quando chamamos o list()
>> list(Produto.objects.all())
```

```
#quando chamamos o bool()
>> bool(Produto.objects.all())
```

```
#quando chamamos o repr()
>> repr(Produto.objects.all())
```

# Dica

Se precisar limpar a lista de query do connection:

```
>> from django.db import reset_queries

>> reset_queries()
```

# Referências


- http://pythonclub.com.br/django-introducao-queries.html
- https://www.fullstackpython.com/object-relational-mappers-orms.html
- http://www.gilenofilho.com.br/como-funciona-o-orm-do-django/
- https://docs.djangoproject.com/en/1.11/
- https://docs.djangoproject.com/en/1.11/topics/db/queries/
- https://docs.djangoproject.com/en/1.11/faq/models/
- https://docs.djangoproject.com/en/1.11/ref/models/querysets/#when-querysets-are-evaluated

THANKS


A MILLION

github.com/beatrizuezu/de-sql-para-orm-django

## CONTATO

@beatrizuezu

beatriz.uezu@gmail.com

## HABEMUS VAGAS

https://99jobs.com/luizalabs