Renato Cruz

Analista de helpdesk

Este slide foi criado a partir do meu artigo:

- Django ORM (Cookbook)

Django ORM (Cookbook)

Este texto não é um tutorial ou algo do gênero, estou apenas centralizando alguns exemplos 😊.

Estou fazendo este texto porque recentemente analisei alguns códigos e reparei que muitas pessoas realizam o calculo e o tratamento de alguns dados dentro das **views**, quando na realidade esses dados já poderiam vir tratados e calculados ao se realizar uma QuerySet bem estruturada.

> Os código apresentados aqui são apenas para exemplo, altere conforme as suas necessidades!

Modelo que utilizei para os exemplos:

```python
from django.core.validators import MaxValueValidator

from django.db import models


class Autor(models.Model):
  nome = models.CharField(max_length=30)
  sobrenome = models.CharField(max_length=30)
  idade = models.PositiveSmallIntegerField(validators=[MaxValueValidator(110)])


class Livro(models.Model):
  titulo = models.CharField(max_length=100)
  autor = models.ForeignKey(Autor, on_delete =models.CASCADE)
```

Para criar as migrações:

```
python manage.py makemigrations
```

Para executar as migrações:

```
python manage.py migrate
```

Para visualizar o código SQL que foi gerado ao se executar o comando **migrate** é utilizado o comando **sqlmigrate**, a sua sintaxe é:

```
python manage.py sqlmigrate NomeDoApp NomeDaMigration
```

No **meu** caso:

```
python manage.py sqlmigrate orm 0001
```

Como resultado temos:

```
BEGIN;
--
-- Create model Autor
--
CREATE TABLE "orm_autor" (
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
    "nome" varchar(30) NOT NULL,
    "sobrenome" varchar(30) NOT NULL,
    "idade" smallint unsigned NOT NULL CHECK ("idade" >= 0),
);
--
-- Create model Livro
--
CREATE TABLE "orm_livro" (
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
    "titulo" varchar(100) NOT NULL,
    "autor_id" integer NOT NULL REFERENCES "orm_autor" ("id") DEFERRABLE INITIALLY DEFERRED,
);
CREATE INDEX "orm_livro_autor_id_2d6a4a38" ON "orm_livro" ("autor_id");
COMMIT;
```

> **OBS**: Observe que o ORM do Django gera o nome da tabela a partir do nome do **app** + **nome da classe**.

> Esse comportamento pode ser alterado utilizando-se uma **classe meta**, contudo dei preferencia por manter o padrão.

Para ativar o shell interativo do Django:

```
python manage.py shell
```

Código

Na maior parte das situações tentei escrever o código SQL e o que seria equivalente no ORM.

Por padrão o Django utiliza SQLite, com isso testei a **maior parte** dos comando no DB Browser for SQLite.

Perceba que **podem haver erros** 😅 e que essas **querys** podem mudar ou serem implementadas de outras formas dependêndo do banco de dados que se está utilizando.

Caso encontre algum erro ou queira adicionar alguma **query** entre em contato 😉.

Inserindo dados

sqlite-sql:

```
INSERT INTO orm_autor (nome, sobrenome, idade) VALUES ('renato', 'cruz', '36');
```

Django ORM:

```
Autor.objects.create(nome='renato', sobrenome='cruz', idade=36)
```

Consultando todos os dados

sqlite-sql:

```
SELECT * FROM orm_autor;
```

Django ORM:

```
Autor.objects.all()
```

O comando acima retorna uma lista então podemos:

```
autores = Autor.objects.all()

for autor in autores:
  print(autor.nome)
  print(autor.sobrenome)
  print(autor.idade)
```

Consultar colunas especificas.

sqlite-sql:

```sql
SELECT nome, sobrenome FROM orm_autor;
```

Django ORM:

```python
Autor.objects.only('nome', 'sobrenome')
```

Consulta ignorando linhas duplicadas.

sqlite-sql:

```
SELECT DISTINCT nome, sobrenome FROM orm_autor;
```

Django ORM:

```
Autor.objects.values('nome', 'sobrenome').distinct()
```

Consulta com limite de resultados.

sqlite-sql:

```sql
SELECT * FROM orm_autor LIMIT 10;
```

Django ORM:

```
Autor.objects.all()[:10]
```

Consulta paginada.

sqlite-sql:

```sql
SELECT * FROM orm_autor LIMIT 5 OFFSET 5;
```

Django ORM:

```
Autor.objects.all()[5:10]
```

Consulta com filtro.

sqlite-sql:

```sql
SELECT * FROM orm_autor WHERE id = 1;
```

Django ORM:

```python
Autor.objects.filter(pk=1)
```

> **OBS**: Particularmente gosto de utilizar **pk** ao invés de **id**, isso porque o Python tem um **método interno** chamado **id()**.

Consulta utilizando comparação.

sqlite-sql:

```
SELECT * FROM orm_autor WHERE idade > 20;
SELECT * FROM orm_autor WHERE idade >= 20;
SELECT * FROM orm_autor WHERE idade < 20;
SELECT * FROM orm_autor WHERE idade <= 20;
SELECT * FROM orm_autor WHERE idade != 20;
```

Django ORM:

```
Autor.objects.filter(idade__gt=18)
Autor.objects.filter(idade__gte=18)
Autor.objects.filter(idade__lt=18)
Autor.objects.filter(idade__lte=18)
Autor.objects.exclude(idade=18)
```

Onde:

- **gt**: Maior que.
- **gte**: Maior ou igual que.
- **lt**: Menor que.
- **lte**: Menor ou igual que.
- **exclude()**: Qualquer resultado diferente do especificado.

Consultar um intervalo (BETWEEN).

sqlite-sql:

```sql
SELECT * FROM orm_autor WHERE idade BETWEEN 36 AND 40;
```

Django ORM:

```
Autor.objects.filter(idade__range=(36, 40))
```

Consulta buscando um padrão.

sqlite-sql:

```
SELECT * FROM orm_autor WHERE nome LIKE 'Renato%';
SELECT * FROM orm_autor WHERE nome ILIKE 'Renato%';
SELECT * FROM orm_autor WHERE nome LIKE '%Renato';
SELECT * FROM orm_autor WHERE nome ILIKE '%Renato';
SELECT * FROM orm_autor WHERE nome LIKE '%Renato%';
SELECT * FROM orm_autor WHERE nome ILIKE '%Renato%';
```

Django ORM:

```
Autor.objects.filter(nome__startswith='Renato')
Autor.objects.filter(nome__istartswith='Renato')
Autor.objects.filter(nome__endswith='Renato')
Autor.objects.filter(nome__iendswith='Renato')
Autor.objects.filter(nome__contains='Renato')
Autor.objects.filter(nome__icontains='Renato')
```

Onde:

- **startswith**: Começa com (Case-sensitive - Maiúscula e Minúsculas fazem diferença).
- **istartswith**: Começa com (Case-insensitive - Maiúscula e Minúsculas **não** fazem diferença).
- **endswith**: Terminal com (Case-sensitive).
- **iendswith**: Terminal com (Case-insensitive).
- **contains**: Contém (Case-sensitive).
- **icontains**: Contém (Case-insensitive).

> **OBS**: SQLite **não suporta** case-sensitive.

Consultar se um resultado está em uma lista.

sqlite-sql:

```
SELECT id FROM orm_autor WHERE id in (1, 2);
```

Django ORM:

```
Autor.objects.filter(id__in=[1, 2])
```

Consulta com AND.

sqlite-sql:

```sql
SELECT * FROM orm_autor WHERE nome='renato' AND idade > 28;
```

Django ORM:

```
Autor.objects.filter(nome='renato', idade__gt=28)
```

Consulta com OR.

sqlite-sql:

```
SELECT * FROM orm_autor WHERE nome='renato' OR idade > 28;
```

Django ORM:

```
from django.db.models import Q

Autor.objects.filter(Q(nome='renato') | Q(idade__gt=28))
```

Consulta com NOT.

sqlite-sql:

```
SELECT * FROM orm_autor WHERE NOT nome='renato';
```

Django ORM:

```
Autor.objects.exclude(nome='renato')
```

Consulta com Null.

sqlite-sql:

```
SELECT * FROM orm_autor WHERE idade is NULL;
```

Django ORM:

```
Autor.objects.filter(idade__isnull=True)
```

Outra forma:

```
Autor.objects.filter(idade=None)
```

Consulta com Not Null.

sqlite-sql:

```
SELECT * FROM orm_autor WHERE idade is NOT NULL;
```

Django ORM:

```
Autor.objects.filter(idade__isnull=False)
```

Outra forma:

```
Autor.objects.exclude(idade=None)
```

Consulta ordenada (Crescente).

sqlite-sql:

```
SELECT * FROM orm_autor order by idade ASC;
```

Django ORM:

```
Autor.objects.order_by('idade')
```

Outra forma:

```
Autor.objects.order_by('idade').asc()
```

Consulta ordenada (Decrescente).

sqlite-sql:

```
SELECT * FROM orm_autor order by idade DESC;
```

Django ORM:

```
Autor.objects.order_by('-idade')
```

Outra forma:

```
Autor.objects.order_by('idade').desc()
```

Atualizando uma linha.

sqlite-sql:

```
UPDATE orm_autor SET idade = 20 WHERE id = 1;
```

Django ORM:

```
autor = Autor.objects.get(pk=1)
autor.idade = 20
autor.save()
```

Outra forma:

```
Autor.objects.filter(pk=1).update(idade=20)
```

Atualizando múltiplas linhas.

sqlite-sql:

```
UPDATE orm_autor SET idade = idade * 1.5;
```

Django ORM:

```
from django.db.models import F

Autor.objects.update(idade=F('idade') * 1.5)
```

Exemplo com `filter()`:

```
for autor in Autor.objects.filter(idade=36):
  autor.idade = 30
  autor.save()
```

Apagando todos os dados.

sqlite-sql:

```
DELETE FROM orm_autor;
```

Django ORM:

```
Autor.objects.all().delete()
```

Apagando linhas especificas

sqlite-sql:

```sql
DELETE FROM orm_autor WHERE idade < 20;
```

Django ORM:

```
Autor.objects.filter(idade__lt=20).delete()
```

Apagando um registro

sqlite-sql:

```sql
DELETE FROM orm_autor WHERE id = 1;
```

Django ORM:

```
Autor.objects.get(pk=1).delete()
```

Funções de agregação.

Funções de agregação são funções SQL que permitem executar uma operação aritmética nos valores de uma coluna.

MIN()

sqlite-sql:

```
SELECT MIN(idade) FROM orm_autor;
```

Django ORM:

```
from django.db.models import Min

Autor.objects.all().aggregate(Min('idade'))
```

MAX().

sqlite-sql:

```
SELECT MAX(idade) FROM orm_autor;
```

Django ORM:

```
from django.db.models import Max

Autor.objects.all().aggregate(Max('idade'))
```

AVG()

sqlite-sql:

```
SELECT AVG(idade) FROM orm_autor;
```

Django ORM:

```
from django.db.models import Avg

Autor.objects.all().aggregate(Avg('idade'))
```

SUM()

sqlite-sql:

```
SELECT SUM(idade) FROM orm_autor;
```

Django ORM:

```
from django.db.models import Sum

Autor.objects.all().aggregate(Sum('idade'))
```

COUNT()

sqlite-sql:

```
SELECT COUNT(*) FROM orm_autor;
```

Django ORM:

```
Autor.objects.count()
```

Outro exemplo:

sqlite-sql:

```sql
SELECT COUNT(*) FROM orm_autor WHERE idade = 60;
```

Django ORM:

```
Autor.objects.filter(idade=60).count()
```

Consultas com GROUP BY.

Exemplo retorna quantas vezes um mesmo nome se repete.

sqlite-sql:

```
SELECT nome, COUNT(*) as count FROM orm_autor GROUP BY nome;
```

Django ORM:

```
from django.db.models import Count

Autor.objects.values('nome').annotate(count=Count('nome'))
```

Consultas com HAVING.

O HAVING determina uma condição de busca para um grupo ou um conjunto de registros.

Serão exibidos os nomes que possuam **mais** de 1 ocorrência.

sqlite-sql:

```
SELECT nome, COUNT('nome') as count FROM orm_autor GROUP BY nome HAVING count > 1;
```

Django ORM:

```
from django.db.models import Count

Autor.objects.values('nome').annotate(count=Count('nome')).filter(count__gt=1)
```

Tabelas com chave estrangeira (Many-to-one)

Criando livro através do objeto autor:

Django ORM:

```
novo_autor = Autor(nome='rafaela', sobrenome='da silva', idade=20)
novo_autor.save()

novo_livro = novo_autor.livro_set.create(titulo='livro da rafaela.')
```

> **OBS**: Poderia ter sido consultado um autor existente ao invés de se criar um novo.

Criando livro através do objeto livro

Django ORM:

```
novo_autor = Autor(nome='gisele', sobrenome='fonseca', idade=28)
novo_autor.save()

novo_livro = Livro(titulo='livro da gisele.', autor=novo_autor)
novo_livro.save()

> **OBS**: Poderia ter sido consultado um autor existente ao invés de se criar um novo.
```

✅ Extra ✅

Outros ORM (Object-Relational Mapping - Mapeamento Objeto-Relacional) para Python:

- Peewee.
- Pony.
- SQLAlchemy.

Algumas ferramentas uteis para gestão de banco de dados:

- DB Browser for SQLite.
- DBeaver.

✅