

Django Models - CRUD



Prof. Bruno Gomes



@profbrunogomes



Aula de Hoje

2

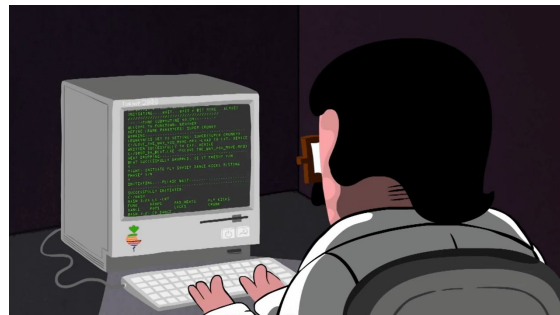
- » **Django**
 - » **ORM**
 - » **SQLite**
 - » **Models**
 - » **Sintaxe**
 - » **Tipos de Campos**
 - » **Restrições de Campo**
 - » **CRUD**



Antes de começar

3

- » Baixar **aula6.zip** da pasta do Drive;
- » Extrair no mesmo diretório do Virtualenv, e iniciar servidor;
- » Testar os endereços:
 - » <http://localhost:8000/cadastro>
 - » <http://localhost:8000/cursos>



1.

Banco de Dados

Documentação

5

» <https://docs.djangoproject.com/en/4.1/topics/db/models/>



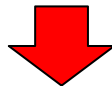
- » *Object-relational mapping* - Mapeamento objeto-relacional
- » Ferramenta que faz um mapa entre as classes do projeto e o banco de dados
- » Vantagens:
 - » independência em relação ao banco de dados SQL
 - » acesso direto a objetos relacionados
 - » implementação fácil e flexível de operações CRUD
 - » validação de campos

- » Na ORM as **tabelas** do banco de dados são representadas através de **classes** e os registros de cada tabela são representados como instâncias das classes correspondentes.
- » Não é necessário escrever código SQL para criar o banco, ou mesmo fazer operações (Inserir, Editar, Remover....).
 - » Somente precisará usar SQL se a consulta que pretende fazer for muito complexa ou específica demais.

Exemplo

8

```
class Usuario(models.Model):  
    nome = models.CharField('Nome', max_length=100)  
    cpf = models.CharField('CPF', max_length=11)
```



```
create table Usuario(  
id integer NOT NULL PRIMARY KEY AUTOINCREMENT,  
Nome varchar(100) NOT NULL,  
CPF varchar(11) NOT NULL  
)
```


2.

SQLite

Banco de Dados - SQLite

10

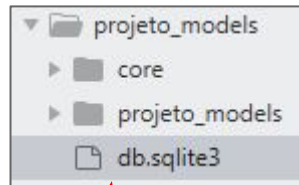
- » Vamos usar inicialmente o banco de dados **SQLite**:
 - » Mais fácil e simples para testes iniciais
 - » Banco de dados SQL embutido
- » É escrito em C, não possui licença alguma, qualquer pessoa pode baixar as fontes no site, compilar, modificar, executar e utilizar
- » Django já vem, por padrão, configurado para usar o SQLite

Configurando o Banco de Dados no Projeto

11

» No arquivo: **settings.py**:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),  
    }  
}
```



projeto_models/settings.py

Configurando o Banco de Dados no Projeto

12

- » No terminal, sincronizar o projeto com o BD através do comando a seguir:

Console

```
python manage.py migrate
```

- » O comando **migrate** deve ser executado sempre a primeira vez que desejar sincronizar/ligar o projeto com o banco de dados.

```
(PrimeiroAmbiente) C:\Users\2729795\Desktop\projetos\projeto_mod
e.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying sessions.0001_initial... OK
```

Dica: SQLite Browser

13

- » Baixar a versão **no installer** do visualizador do SQLite:
 - » <https://sqlitebrowser.org/dl/>

Windows

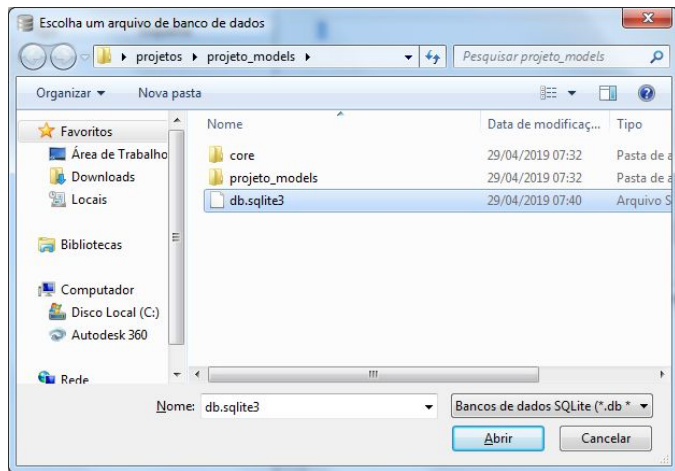
Our latest release (3.11.2) for Windows:

- DB Browser for SQLite - Standard installer for 32-bit Windows & Windows XP
- DB Browser for SQLite - .zip (no installer) for 32-bit Windows & Windows XP
- DB Browser for SQLite - Standard installer for 64-bit Windows
- DB Browser for SQLite - .zip (no installer) for 64-bit Windows
- Note - There's no PortableApp version for 3.11.1 (yet). It'll hopefully be ready in

Abrindo o Banco no SQLite Browser

14

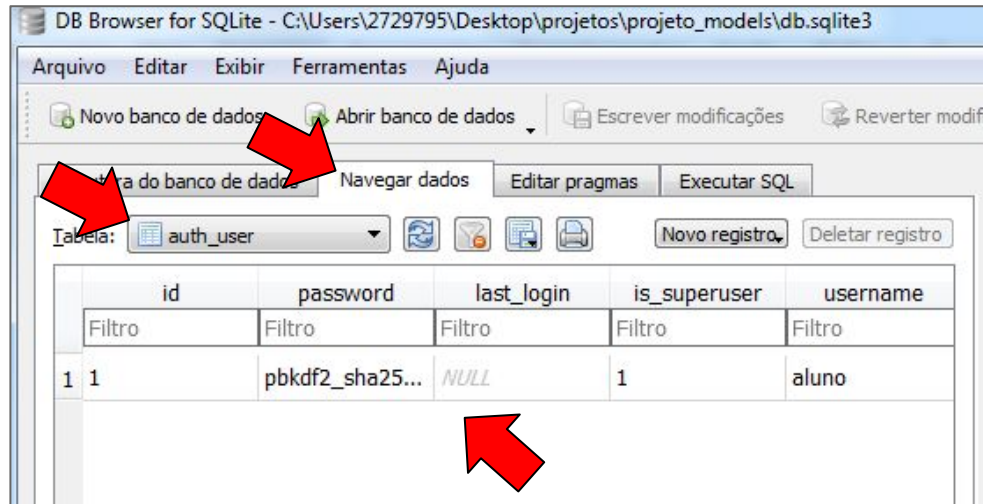
- » Abrir o **SQLite Browser**, depois clicar em **Abrir banco de dados**, e seleccionar o arquivo **db.sqlite3** (dentro da pasta do projeto).



Visualizando os dados de uma tabela

15

- » Clicar em **Navegar dados**, depois selecionar a tabela.



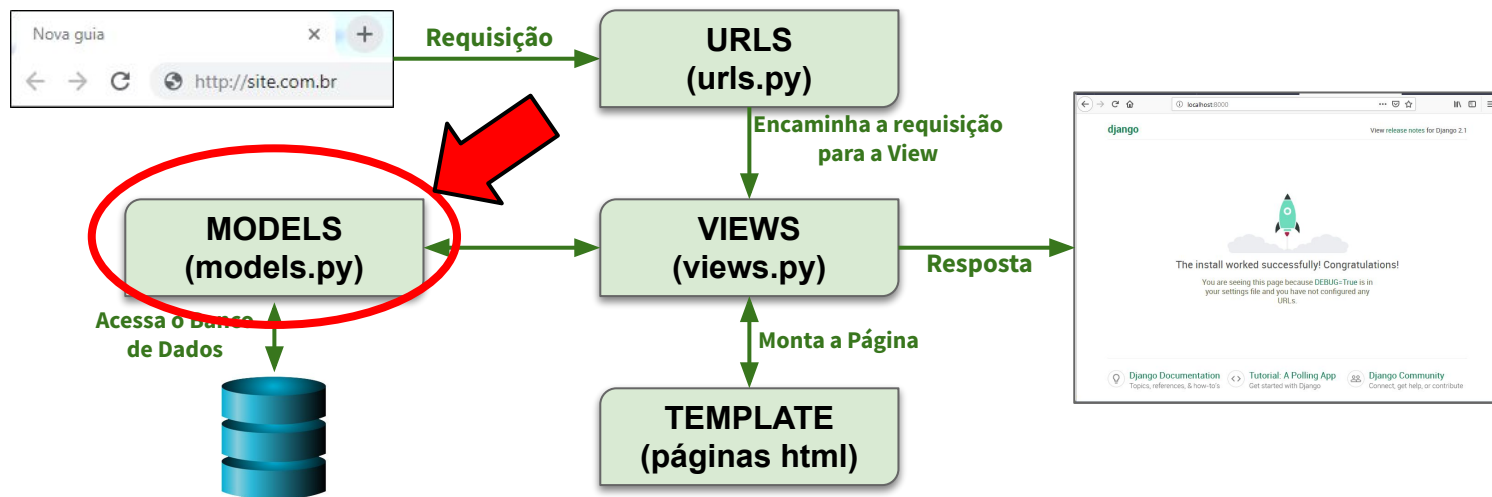
3.

Models

URLs

17

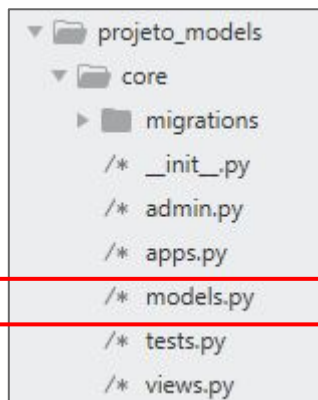
» No Ciclo de Vida do Django:



Models

18

- » O banco de dados é descrito no arquivo **models.py**, que fica no diretório da aplicação:



Trabalhando com Models

19

- » O banco é definido na aplicação (neste caso dentro de core);
- » Para criar uma tabela, deve-se criar uma classe que herda de Model (neste exemplo vamos criar Curso):

```
from django.db import models  
  
class Cursos(models.Model):
```

core/models.py

Trabalhando com Models

20

- » Para criar um campo, basta criar uma variável da classe (de instância), e salvar um objeto de models (a função acessada será o tipo do dado, neste exemplo será de texto - CharField):

```
from django.db import models

class Cursos(models.Model):
    nome = models.CharField('Nome', max_length=100)
```

core/models.py

Atualizando o Banco de Dados

21

- » **Passo 1:** reconhecer as alterações feitas na estrutura do banco através do comando makemigrations:
 - » Alteração: criação de cursos

```
python manage.py makemigrations core
```

Console

```
Migrations for 'core':  
core\migrations\0001_initial.py  
- Create model Curso
```

Atualizando o Banco de Dados

22

- » **Passo 2:** enviar ao banco as alterações através do comando migrate:

```
python manage.py migrate
```

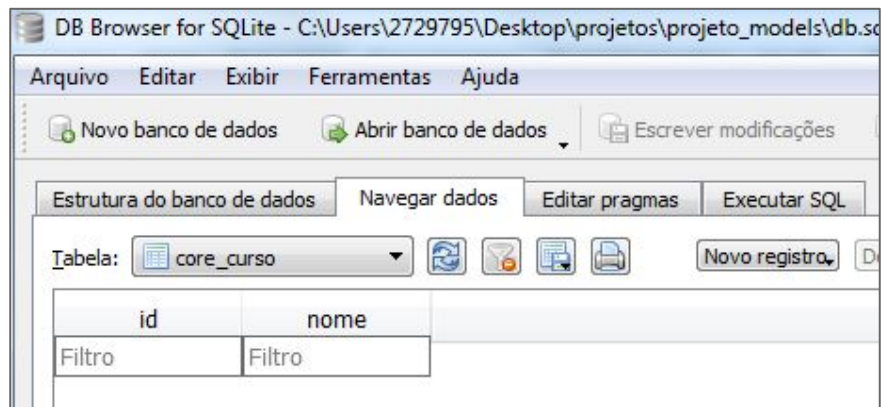
Console

```
Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, core, sessions  
Running migrations:  
  Applying core.0001_initial... OK
```

Visualizar a nova tabela

23

- » Visualizar a tabela curso no DB Browser (pressionar F5 para atualizar):

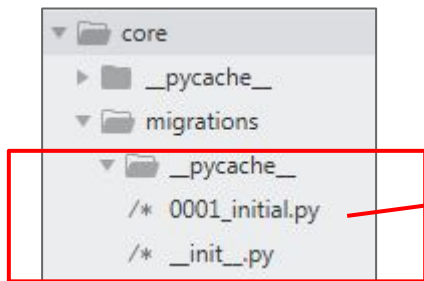


- » Obs.: As tabelas são criadas automaticamente utilizando a sintaxe app_tabela:
 - » Neste exemplo, core_curso.

Observação

24

- » Ao digitar o comando makemigrations, uma pasta chamada migrations foi criada no projeto;
- » Nela é possível visualizar arquivos .py que contém todas as alterações feitas no banco de dados:



```
class Migration(migrations.Migration):  
  
    initial = True  
  
    dependencies = [  
    ]  
  
    operations = [  
        migrations.CreateModel(  
            name='Cursos',  
            fields=[  
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),  
                ('nome', models.CharField(max_length=100, verbose_name='Nome')),  
            ],  
        ),  
    ]
```

Curiosidade: Migrations

25

- » Funciona como um controle de versão de banco de dados;
- » Facilita a modificação e o compartilhamento da estrutura do banco de dados entre equipes;
- » É um mecanismo que mantém um histórico da criação e alterações no banco de dados, e permite reverter alterações.

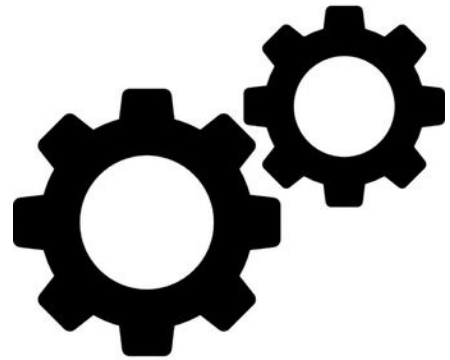
Documentação:

<https://docs.djangoproject.com/en/4.1/topics/migrations/>

Curiosidade: Migrations

26

1. As alterações que um desenvolvedor faz no Banco de Dados vão sendo salvas na pasta migrations;
2. Quando outro desenvolvedor acessar a pasta migrations, visualizará as modificações feitas por outras pessoas, e consegue atualizar o seu banco de dados sem problemas de consistência.



Curiosidade: Migrations

27

- » Antes de executar o comando migrate, o console exibe esta informação:

```
You have 17 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.  
Run 'python manage.py migrate' to apply them.
```

- » Ela informa que tem migrações que não foram aplicadas:
 - » São tabelas e índices que são criadas no banco assim que digita a primeira vez o comando migrate.
- » Após executar o comando migrate, esta mensagem desaparece.

3.1

Sintaxe

Criar uma tabela

29

» Sintaxe:



```
class Nome_Tabela(models.Model):
```

Adicionando atributos à Tabela

30

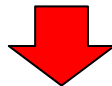
» Sintaxe:

```
class Nome_Tabela(models.Model):  
    nome_campo = models.TipoCampo(propriedades_campo)
```

Exemplo

31

```
class Usuario(models.Model):  
    nome = models.CharField('Nome', max_length=100)  
    cpf = models.CharField('CPF', max_length=11)
```



```
create table Usuario(  
id integer NOT NULL PRIMARY KEY AUTOINCREMENT,  
Nome varchar(100) NOT NULL,  
CPF varchar(11) NOT NULL  
)
```

Observações

32

- » Cada atributo de um modelo representa um campo no banco de dados;
- » Um **campo id** (chave primária) é adicionado automaticamente e não é necessário incluí-lo em `models.py`
 - » O campo gerado é do tipo inteiro autoincremento;
 - » Se desejar alterar a chave primária, é possível sobrescrever.

3.2

Tipos de Campos

Documentação

34

- » <https://docs.djangoproject.com/pt-br/4.1/ref/models/fields/#model-field-types>



Tipos de Campos

35

- » Cada campo no seu modelo deve ser instanciado da classe **Field** apropriada.
- » Tipo da classe utilizada determina:
 - » O tipo da coluna no banco de dados (ex: INTEGER, VARCHAR, TEXT).
 - » O widget HTML padrão para usar quando renderizar o campo de um form (ex. `<input type="text">`, `<select>`).

CharField

36

» Campo de texto para pequenas e grandes Strings.

» Sintaxe:

» **class CharField(max_length=None, **options)**

Obrigatório

Veremos na próxima sessão

» Exemplo:


```
models.CharField(max_length=30)
```

```
models.CharField('Nome', max_length=100)
```

IntegerField

37

- » Campo que aceita números inteiros.
- » Sintaxe:
 - » **class IntegerField(**options)**
- » Exemplo:




```
models.IntegerField()  
  
models.IntegerField('Nome')
```

BinaryField

38

- » Campo que aceita os valores 0 ou 1.
- » Sintaxe:
 - » **class BinaryField(**options)**
- » Exemplo:




```
models.BinaryField()  
models.BinaryField('Nome')
```


BooleanField

39

- » Campo que aceita true ou false.
- » Sintaxe:
 - » **class BooleanField(**options)**
- » Exemplo:



```
models.BooleanField()  
models.BooleanField('Nome')
```

DateField

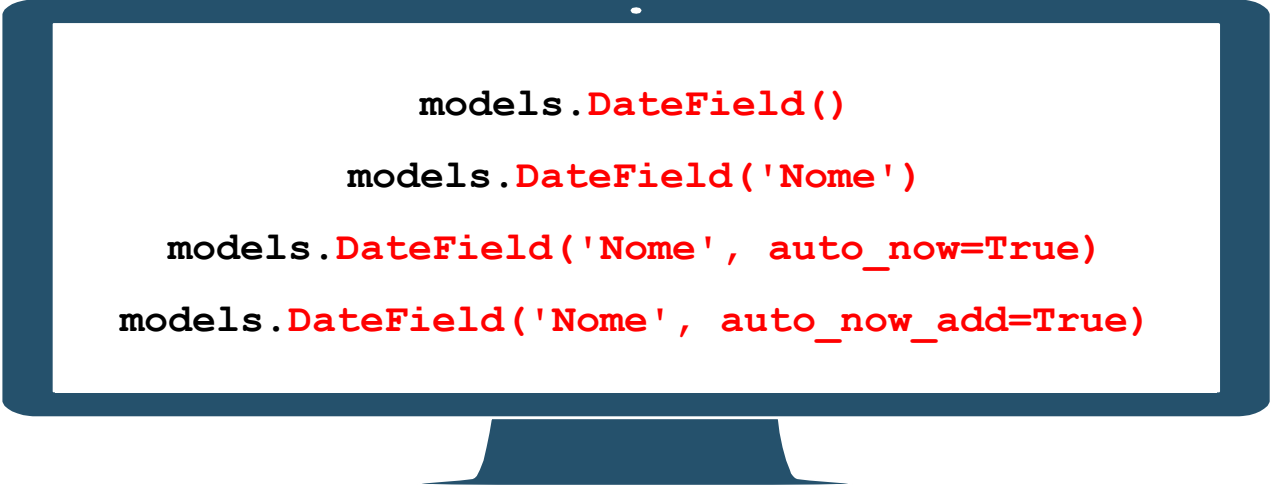
40

- » Campo que aceita apenas datas.
- » Sintaxe:
 - » **class DateField(auto_now=False, auto_now_add=False, **options)**
- » **auto_now:** Atualiza o campo automaticamente toda vez que o objeto é salvo (última atualização);
- » **auto_now_add:** Salva automaticamente a data que o objeto foi salvo a primeira vez.

DateField

41

» Exemplos:



```
models.DateField()  
models.DateField('Nome')  
models.DateField('Nome', auto_now=True)  
models.DateField('Nome', auto_now_add=True)
```

DateTimeField

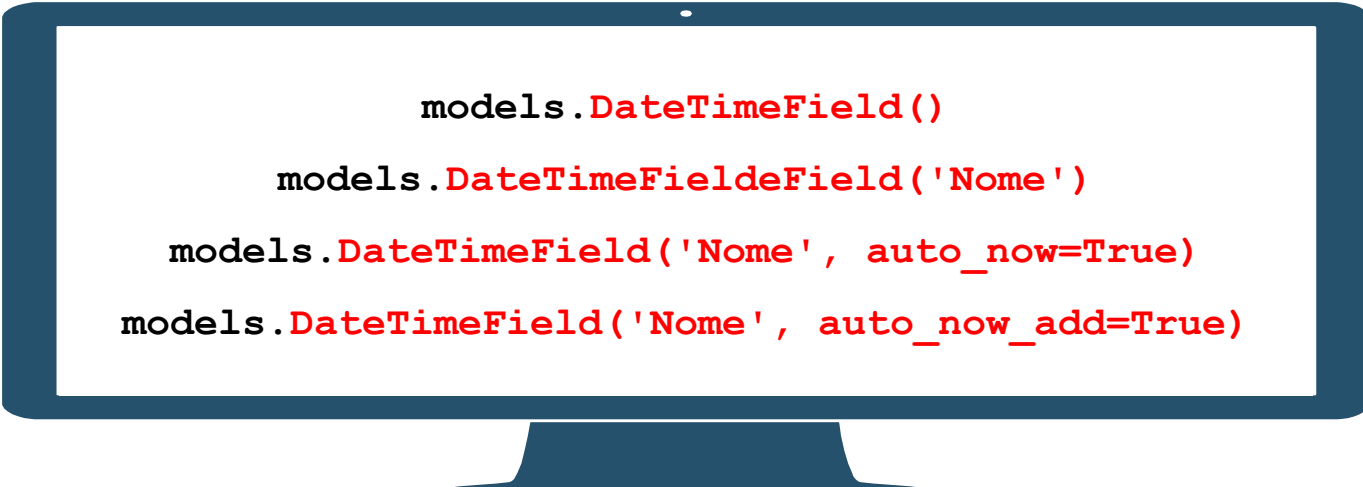
42

- » Campo que aceita data e hora.
- » Sintaxe:
 - » **class DateField(auto_now=False, auto_now_add=False, **options)**
- » **auto_now:** Salva automaticamente a data toda vez que o objeto é atualizado (última atualização);
- » **auto_now_add:** Salva automaticamente a data que o objeto foi salvo a primeira vez (salva somente 1 vez).

DateTimeField

43

» Exemplos:



```
models.DateTimeField()  
models.DateTimeFieldField('Nome')  
models.DateTimeField('Nome', auto_now=True)  
models.DateTimeField('Nome', auto_now_add=True)
```

DecimalField

44

- » Campo que armazena números com casas decimais.
- » Sintaxe:
 - » **class DecimalField(max_digits=None, decimal_places=None, **options)**
- » **max_digits:** O número máximo de dígitos permitido no número.
- » **decimal_places:** O número de casas decimais permitidos

DecimalField

45

» Ejemplos:



```
models.DecimalField('Salario', max_digits=5, decimal_places=2)
```

EmailField

46

- » Campo que armazena e-mails (campo de texto).
- » Sintaxe:
 - » **class EmailField(max_length=254, **options)**
- » Exemplo:



```
models.EmailField(max_length=30)  
models.EmailField('Nome', max_length=100)
```

FileField

47

- » Campo para armazenar arquivos.
- » Sintaxe:
 - » **class FileField(upload_to=None, max_length=100, **options)**
- » **upload_to**: definir o diretório onde será salvo o arquivo
- » Exemplos:

```
models.FileField(upload_to='uploads/')  
models.FileField('Artigo', upload_to='uploads/')
```

ImageField


48

- » Campo para armazenar imagens.
- » Sintaxe:
 - » **class ImageField(upload_to=None, height_field=None, width_field=None, max_length=100, **options)**
- » **height_field**: campo que será armazenado automaticamente a altura da imagem
- » **width_field**: campo que será armazenado automaticamente a largura da imagem

ImageField

49

» Exemplos:



```
models.ImageField(upload_to='uploads/')  
models.ImageField('Artigo', upload_to='uploads/',  
                  max_length=100)
```

3.3

Restrições de Campo

Documentação

51

- » <https://docs.djangoproject.com/en/4.1/ref/models/fields/#field-options>



Opções de Campo

52

- » **max_length:** Quantidade máxima de caracteres que o campo suporta:
- » Exemplo:

```
models.CharField(max_length=30)
```

Opções de Campo

53

- » **null:** Se True, o Django irá usar valores “vazios” como NULL no banco de dados. Padrão é False;
- » Exemplo:

```
models.CharField(max_length=30, null=true)
```

Opções de Campo

54

- » **blank:** Se True, é permitido o campo estar em “branco”. O padrão é False;
- » Exemplo:

```
models.CharField(max_length=30, blank=true)
```


4

CRUD

- » CRUD - *Create, Read, Update and Delete*)
 - » São as quatro operações básicas utilizadas em bases de dados relacionais.
 - » **Criação, consulta, atualização e remoção de dados.**
- » Praticamente todo sistema utiliza estas operações, por isso é importante saber como funcionam.
- » As operações são implementadas na **view**.

Modificando Cursos

58

» Adicionar os campos:

```
class Cursos(models.Model):  
    nome = models.CharField('Nome', max_length=100)  
    data_inicio = models.DateField('Data de Início', null=True)  
    vagas = models.IntegerField('Vagas', null=True)
```

core/models.py

Atualizar Banco

59

- » Executar os comandos:

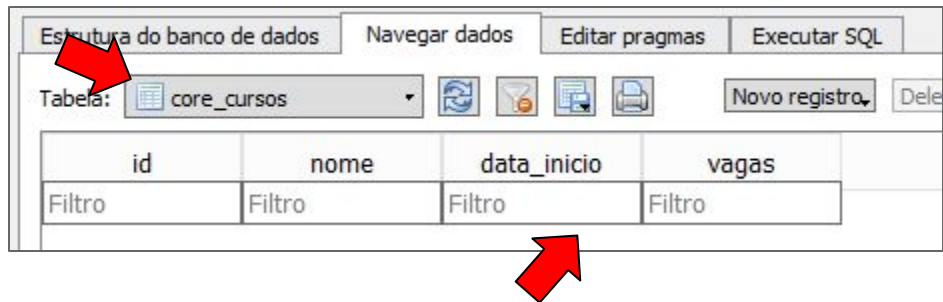
```
python manage.py makemigrations core  
python manage.py migrate
```

Console

Atualizar Banco

60

- » Verificar no SQLite Browser se os campos apareceram:



4.1

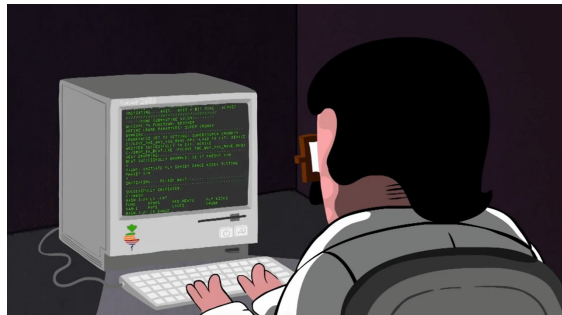
CRUD

Listando

Antes de começar

62

- » Se não tiver o código do projeto até o momento, baixar **aula6_2.zip** da pasta do Drive;
- » Extrair no mesmo diretório do Virtualenv, e digitar os comandos:
 - » `python manage.py makemigrations core`
 - » `python manage.py migrate`
- » Iniciar servidor e testar os endereços:
 - » <http://localhost:8000/cadastro>
 - » <http://localhost:8000/cursos>



SINTAXE: Listando Cursos

63

- » O comando para recuperar todos os objetos de uma tabela do banco de dados é:

```
Model.objects.all()
```

- » **Model** é o nome da tabela (classe no Model) que deseja recuperar;
- » Exemplo: para recuperar Cursos, será:
 - » **Cursos.objects.all()**

Listando Cursos

64

- » Primeiro passo: importar o modelo **Curso** em views.py:


```
from django.shortcuts import render  
from .models import Cursos
```

core/views.py

Listando Cursos

65

- » Segundo passo: na função já existente em views.py, listar os cursos e enviar como contexto ao template cursos.html:



```
def cursos(request):  
    cursos = Cursos.objects.all()  
    contexto = {  
        'lista_cursos': cursos  
    }  
    return render(request, 'cursos.html', contexto)
```

core/views.py

Listando Cursos

66

- » Terceiro passo: exibir os cursos no template cursos.html:

```
<h1>Administração de Cursos</h1>
```

```
<p>Lista de Cursos:</p>
```

```
{% for curso in lista_cursos %}
```

```
    <p>{{curso.nome}}</p>
```

```
{% empty %}
```

```
    <p>Nenhum curso cadastrado</p>
```

```
{% endfor %}
```



core/templates/cursos.html

Listando Cursos

67

- » Terceiro passo: exibir os cursos no template cursos.html:

```
<h1>Administração de Cursos</h1>
```

```
<p>Lista de Cursos:</p>
```

```
{% for curso in lista_cursos %}
```

```
    <p>{{curso.nome}}</p>
```

```
{% empty %}
```

```
    <p>Nenhum curso cadastrado</p>
```

```
{% endfor %}
```

O comando **empty** é executado quando a lista do for é vazia, ou seja, quando não há repetições.

core/templates/cursos.html

Testar no Navegador

68

- » Iniciar o servidor e Acessar:

```
http://localhost:8000/cursos/
```

Administração de Cursos

Lista de Cursos:

Nenhum curso cadastrado

4.2

CRUD

Cadastrar

Cadastrando Cursos

70

- » 1º passo: adicionar um formulário vazio no template **cadastro.html**:

```
<h1>Cadastro de Curso</h1>
```

```
<form method="post">
```

```
    <button type="submit">Salvar</button>
```

```
</form>
```

core/templates/cadastro.html

Observação

Atenção

71

- » Não é necessário adicionar o atributo action e informar o destino, o django configura automaticamente.

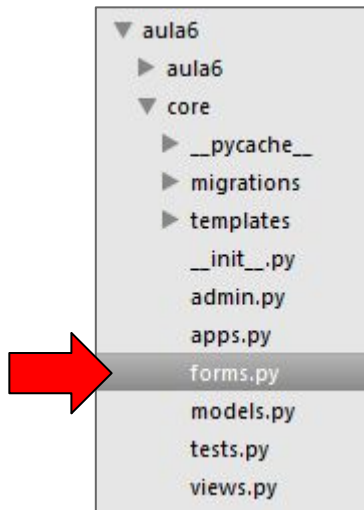
```
<form method="post">
```

- » O django gera os campos do formulário de forma automática:
 - » Baseado na modelagem feita em models.py
- » Isto é possível devido ao **ModelForm**.
- » Documentação:
 - » <https://docs.djangoproject.com/en/4.1/topics/forms/modelforms/#modelform>

Configurando ModelForm

73

- » 2º passo: Criar o arquivo chamado **forms.py** dentro da aplicação core:



Configurando ModelForm

74

- » 3º passo: Adicionar os imports necessários para o funcionamento do ModelForm em **forms.py**:

```
from django.forms import ModelForm
from .models import Cursos
```

core/forms.py

Configurando ModelForm

75

- » 4º passo: Em **forms.py** criar a classe CursosForm (herda de ModelForm) e implementar o nome e os campos que irão aparecer no formulário:

```
class CursosForm(ModelForm):  
    class Meta:  
        model = Cursos  
        fields = ['nome', 'data_inicio', 'vagas']
```

core/forms.py

Exibindo ModelForm

76

- » 5º passo: Importar CursosForm em **views.py**:

```
from django.shortcuts import render
from .models import Cursos
from .forms import CursosForm
```

core/views.py

Exibindo ModelForm

77

- » 6º passo: Na função cadastro em **views.py**, criar CursosForm e enviá-lo via contexto para o template cadastro.html:

```
def cadastro(request):  
    form = CursosForm(request.POST or None)  
    contexto = {  
        'form': form  
    }  
    return render(request, 'cadastro.html', contexto)
```

core/views.py

- » A função cadastro será usada para exibir a página de cadastro, como também para receber os dados informados e salvá-los no banco.
- » Por isso, no momento da criação de **CursosForm**:
 - » **None** significa que o usuário está abrindo a página de cadastro pela primeira vez, então deverá gerar um formulário vazio;
 - » **request.POST** significa que o usuário digitou e enviou os dados, então será criado um objeto CursosForm com os dados informado do formulário.

```
form = CursosForm(request.POST or None)
```

Exibindo ModelForm

79

- » 7º passo: No template **cadastro.html**, adicionar a tag de segurança `csrf_token`, e imprimir a variável `form` (ModelForm):

```
<form method="POST">  
    {% csrf_token %}  
    {{form}}  
<button type="submit">Salvar</button>
```

[core/templates/cadastro.html](#)

- » **{% csrf_token %}**
 - » Irá ser convertido para uma tag de segurança csrf, garantindo que os dados enviados e recebidos são da mesma aplicação.
 - » Exemplo da tag convertida:

```
<input type="hidden" name="csrfmiddlewaretoken"
value="3vHHfc25VK8jy1EkF0S1N8umvyp5S4wMBe3bKk0LtDQGhd93
9hWThsLbWMkQE9Aq">
```

Testar no Navegador

81

» Acessar:

`http://localhost:8000/cadastro`

Navegador

Cadastro de Curso

Nome:

- This field is required.

Data de Início:

- This field is required.

Vagas:

- This field is required.

Salvar

Salvando dados

- » 8º Passo: Para salvar os dados do formulário, é necessário ainda em views.py:
 - » Validar
 - » Salvar

Obs.: Este if será verdadeiro apenas quando o usuário digitar as informações no formulário e clicar para salvar. Caso ele esteja abrindo a página de cadastro a primeira vez, será falso e executará as instruções abaixo.

```
from django.shortcuts import render

def cadastro(request):
    form = CursosForm(request.POST or None)

    if form.is_valid():
        form.save()

    contexto = {
        'form': form
    }
    return render(request, 'cadastro.html',
                  contexto)
```

core/views.py

Salvando dados

- » 9º Passo: Após salvar, não é bom abrir a página de cadastro novamente, e sim voltar para a página de gerenciamento com a lista de cursos. Então, redirecionar o usuário para o template cursos.

Obs.: Não esquecer de importar a biblioteca redirect.

```
from django.shortcuts import render, redirect

def cadastro(request):
    form = CursosForm(request.POST or None)

    if form.is_valid():
        form.save()
        return redirect('cursos')

    contexto = {
        'form': form
    }
    return render(request, 'cadastro.html',
                  contexto)
```

core/views.py

Testar no Navegador

84

- » Acessar e cadastrar um curso:



`http://localhost:8000/cadastro`

Navegador

- » Obs.: A data deve ser no padrão:
 - » **2019-05-20**

Cadastro de Curso

Nome:

- This field is required.

Algoritmos

Data de Início:

- This field is required.

2019-05-20

Vagas:

- This field is required.

30

Salvar

Administração de Cursos

Lista de Cursos:

Algoritmos

Link para Cadastro

85

- » No template **cursos.html**, acrescentar um link para a página de cadastro:

```
<h1>Administração de Cursos</h1>

<a href="{% url 'cadastro' %}">
  <button type="button">Cadastrar</button>
</a>

<p>Lista de Cursos:</p>
```

core/templates/cursos.html

Administração

Cadastrar

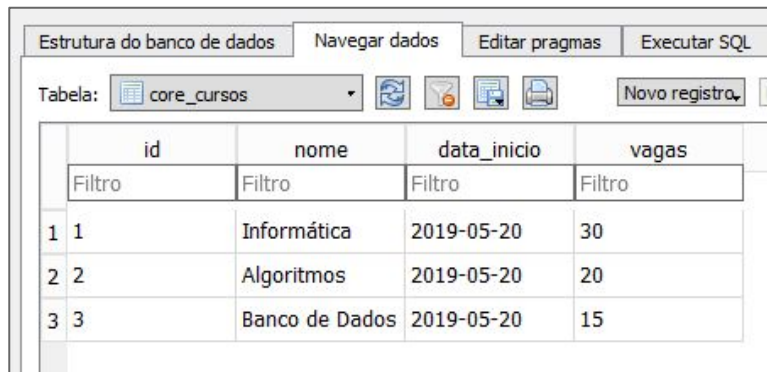
Lista de Cursos:

Algoritmos

Antes de seguir

86

- » Cadastrar mais 2 cursos no banco de dados.
- » Visualizar o resultado no SQLite Browser:



The screenshot shows the SQLite Browser application window. At the top, there are four tabs: 'Estrutura do banco de dados', 'Navegar dados', 'Editar pragmas', and 'Executar SQL'. The 'Navegar dados' tab is active. Below the tabs, there is a section labeled 'Tabela:' with a dropdown menu showing 'core_cursos'. To the right of the dropdown are several icons for database operations and a 'Novo registro' button. Below this section is a table with four columns: 'id', 'nome', 'data_inicio', and 'vagas'. The table contains three rows of data, each with an index on the left.

	id	nome	data_inicio	vagas
	Filtro	Filtro	Filtro	Filtro
1	1	Informática	2019-05-20	30
2	2	Algoritmos	2019-05-20	20
3	3	Banco de Dados	2019-05-20	15

4.3

CRUD

Editar

Entendendo o EDITAR

Atenção

88

- » Para cada elemento listado, deve aparecer uma opção de EDITAR. Ao clicar, a página de cadastro abrirá com os dados do curso no formulário.
- » Para que o projeto saiba qual curso será editado, é necessário enviar alguma informação do curso pelo link:
 - » Usamos a **chave primária**.

Algoritmos [EDITAR](#)

Banco de Dados [EDITAR](#)

PEOO [EDITAR](#)

Autoria Web [EDITAR](#)

Cadastro de Curso

Nome:

- This field is required.

Algoritmos Data de Início:

- This field is required.

2019-05-20 Vagas:

- This field is required.

30

Editando Curso

89

- » 1º passo: para editar, é necessário criar uma rota em **urls.py**. Deverá receber como parâmetro a chave primária:

```
from core.views import cursos, cadastro, atualizar  
path('atualizar/<int:id>/', atualizar, name='atualizar'),
```

aula6/urls.py

Editando Curso

90

- » 2º passo: Criar a função atualizar em views.py, recebendo a chave primária como parâmetro e o retorno sendo o template cadastro.html:

```
def atualizar(request, id):  
    return render(request, 'cadastro.html')
```

core/views.py

Editando Curso

91

- » 3º passo: na função atualizar, buscar o curso no banco de dados pelo id:

```
def atualizar(request, id):  
  
    curso = Cursos.objects.get(pk=id)  
  
    return render(request, 'cadastro.html')
```

core/views.py

- » É possível buscar informações utilizando outros campos:

```
Cursos.objects.get(nome='Algoritmos')  
Cursos.objects.get(vagas=30)
```

Editando Curso

93

- » 4º passo: após consultar curso, criar o objeto CursosForm:

```
def atualizar(request, id):  
    curso = Cursos.objects.get(pk=id)  
  
    form = CursosForm(request.POST or None, instance=curso)  
  
    return render(request, 'cadastro.html')
```

core/views.py

```
form = CursosForm(request.POST or None, instance=curso)
```

- » Agora temos um novo atributo na criação de CursosForm:
 - » **instance=curso** significa que o formulário é baseado na instância de **Curso** que acabou de ser consultada no banco.
 - » Se estiver abrindo a página a primeira vez, irá gerar um formulário vazio (**None**), e depois preenchê-lo com os dados de curso que vieram do banco;
 - » Se o usuário tiver aberto o formulário, realizou alguma alteração e clicou em salvar, os dados virão através do **request.POST**, e o objeto CursoForm será criado com os dados atualizados referente ao objeto Curso do banco.

Editando Curso

95

- » 5º passo: se o formulário estiver sendo aberto a primeira vez, enviar o objeto CursosForm via contexto para o template cadastro.html, caso contrário, deve validar e salvar as alterações:

core/views.py

```
form = CursosForm(request.POST or None, instance=curso)

if form.is_valid():
    form.save()
    return redirect('cursos')

contexto = {
    'form': form
}

return render(request, 'cadastro.html', contexto)
```


Editando Curso

96

- » 6º passo: No template **cursos.html**, adicionar o link para a edição, acessando a URL atualizar e passando o parâmetro id de curso:

```
{% for curso in lista_cursos %}
    <p>{{curso.nome}}
        <a href="{% url 'atualizar' curso.id %}">EDITAR</a>
    </p>
{% empty %}
```

aula6/templates/cursos.html

Testar no Navegador

- » Acessar e atualizar os cursos cadastrados:



`http://localhost:8000/cursos`

- » Obs.: A data deve ser no padrão:
 - » **2019-05-20**

97

Lista de Cursos:

Algoritmos [EDITAR](#)

Banco de Dados [EDITAR](#)

PEOO [EDITAR](#)

Autoria Web [EDITAR](#)

Cadastro de Curso

Nome:

- This field is required.

Data de Início:

- This field is required.

Vagas:

- This field is required.

4.3

CRUD

Deletar

Entendendo o DELETAR

Atenção

99

- » Para cada elemento listado, deve aparecer uma opção de DELETAR. Ao clicar, o curso será automaticamente removido do banco.
- » Para que o projeto saiba qual curso será removido, é necessário enviar alguma informação do curso pelo link:
 - » Usamos a **chave primária**.

Lista de Cursos:

Algoritmos [EDITAR](#) [DELETAR](#)

Banco de Dados [EDITAR](#) [DELETAR](#)

PEOO [EDITAR](#) [DELETAR](#)

Autoria Web [EDITAR](#) [DELETAR](#)

Lista de Cursos:

Algoritmos [EDITAR](#) [DELETAR](#)

Banco de Dados [EDITAR](#) [DELETAR](#)

PEOO [EDITAR](#) [DELETAR](#)

Removendo Curso

100

- » 1º passo: para remover, é necessário criar uma rota em **urls.py**. Deverá receber como parâmetro a chave primária:

```
from core.views import cursos, cadastro, atualizar, deletar  
path('deletar/<int:id>/', deletar, name='deletar'),
```

aula6/urls.py

Removendo Curso

101

- » 2º passo: Criar a função deletar em views.py, recebendo o parâmetro id:

```
def deletar(request, id):
```

core/views.py

Removendo Curso

102

- » 3º passo: na função atualizar, buscar o curso no banco de dados pelo id :

```
def deletar(request, id):  
    curso = Cursos.objects.get(pk=id)
```

core/views.py

Removendo Curso

103

» 4º passo: remover o curso através da função delete():

```
def deletar(request, id):  
    curso = Cursos.objects.get(pk=id)  
    curso.delete()
```

core/views.py

Removendo Curso

104

- » 5º passo: redirecionar o usuário para o template cursos:

```
def deletar(request, id):  
    curso = Cursos.objects.get(pk=id)  
    curso.delete()  
    return redirect('cursos')
```

core/views.py

Removendo Curso

105

- » 6º passo: no template **cursos.html**, adicionar o link para a remoção, acessando a URL deletar e passando o parâmetro id de curso:

```
{% for curso in lista_cursos %}
  <p>{{curso.nome}}
    <a href="{% url 'atualizar' curso.id %}">EDITAR</a>
    <a href="{% url 'deletar' curso.id %}">DELETAR</a>
  </a>
{% empty %}
```

aula6/templates/cursos.html

Testar no Navegador

106

- » Acessar e remover 2 cursos:



Lista de Cursos:

Algoritmos [EDITAR](#) [DELETAR](#)

Banco de Dados [EDITAR](#) [DELETAR](#)

PEOO [EDITAR](#) [DELETAR](#)

Autoria Web [EDITAR](#) [DELETAR](#)

Lista de Cursos:

Algoritmos [EDITAR](#) [DELETAR](#)

Banco de Dados [EDITAR](#) [DELETAR](#)

PEOO [EDITAR](#) [DELETAR](#)

Dúvidas?



- » Criar um projeto Django (nome que você preferir) e app Core:
 - » Criar um **modelo (model)** de sua preferência, que tenha 2 atributos e 1 foto. (ex: curso é o modelo, e os atributos são nome, data_inicio e foto).
 - » Implementar o CRUD deste modelo, utilizando o layout disponível na pasta da disciplina (**aula6_layout.zip**)
 - » Deve passar as páginas para o projeto (templates e estáticos), e implementar os formulários do CRUD exatamente como estão no layout (consultar a aula_07 e vídeos).

Atividade

109

» Páginas:

Nome do Modelo

Cadastrar Modelo

Atributo 1	Atributo 2	Foto	Gerenciar
Valor 1	Valor 2	[[Foto]]	<button>Editar</button> <button>Excluir</button>
Valor 1	Valor 2	[[Foto]]	<button>Editar</button> <button>Excluir</button>
Valor 1	Valor 2	[[Foto]]	<button>Editar</button> <button>Excluir</button>

Nome do Modelo

Cadastro:

Atributo 1:

Atributo 2:

Foto:

No file selected.

Salvar