

django

AULA 04

BANCO DE DADOS, MODELS E DJANGO ADMIN

O QUE VEREMOS HOJE

- 01 APPS NO DJANGO
- 02 BANCO DE DADOS E SQL
- 03 ORM
- 04 MODELS
- 05 DJANGO ADMIN
- 06 VIEWS E BANCO DE DADOS

APPS NO DJANGO

No Django, o conceito de app foi criado para organizar o código em partes independentes e reutilizáveis.

Cada app é uma unidade funcional que encapsula um pedaço específico da lógica da aplicação. Portanto, cada app contém seus próprios models, views, templates e outras partes do código que lidam com uma funcionalidade específica.

Quando trabalhamos com banco de dados no Django, os models e migrations precisam estar em apps, pois é onde o Django busca essas informações.



APPS NO DJANGO

Para criar um App no Django, precisamos rodar no terminal o comando:

django-admin startapp nome_do_app

Com esse comando, o Django já cria toda estrutura de App.

Além disso, precisamos **adicionar o App no INSTALLED_APPS no arquivo settings.py** do projeto.

Por fim, para que as urls do App sejam acessíveis no projeto, precisamos criar um arquivo urls.py no App e incluir no urls.py do projeto.

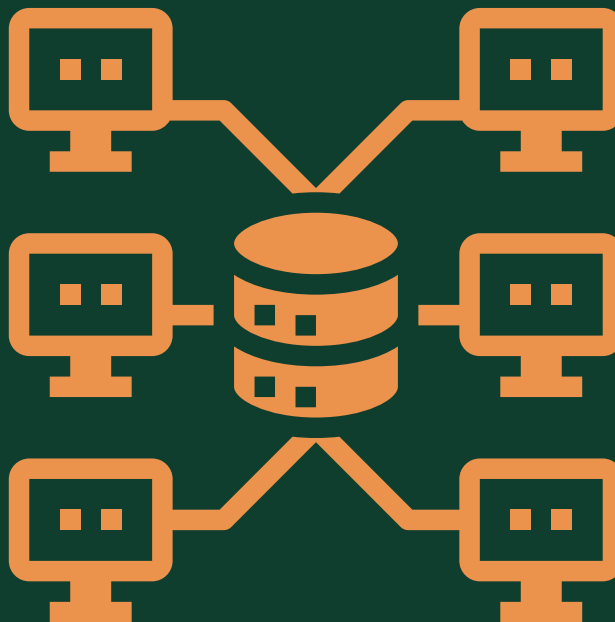


```
1  from django.contrib import admin
2  from django.urls import path, include
3
4  urlpatterns = [
5      path('admin/', admin.site.urls),
6      path('products/', include('products.urls')),
7  ]
8
```

BANCO DE DADOS

Um banco de dados é um sistema organizado de armazenamento e gerenciamento de dados, projetado para fornecer acesso eficiente e seguro a informações.

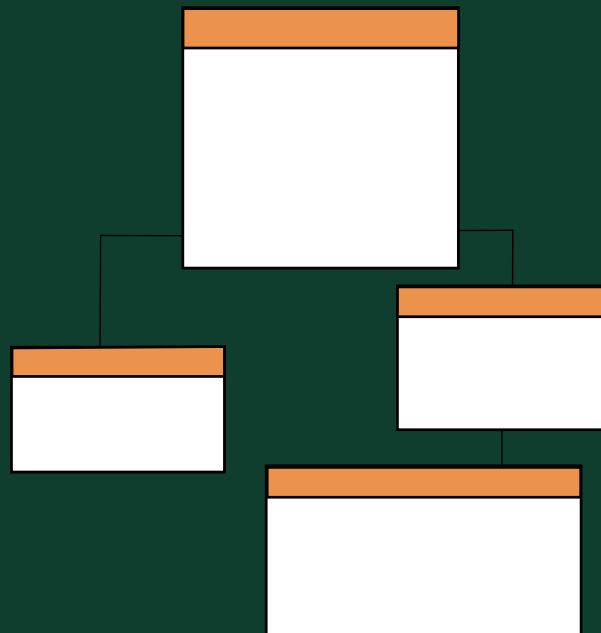
Ele permite que aplicações armazenem, recuperem, manipulem e gerenciem diversos tipos de dados de forma estruturada e padronizada.



BANCO DE DADOS RELACIONAL

Um banco de dados relacional é um tipo de sistema de gerenciamento de banco de dados (SGBD) que organiza os dados em tabelas com linhas e colunas. Cada tabela representa uma entidade, como clientes, produtos ou faturas, e as colunas representam os diferentes atributos dessa entidade.

Bancos de dados relacionais utilizam a linguagem SQL (Structured Query Language) para criar, modificar e consultar os dados de forma estruturada e padronizada.



SQL (STRUCTURED QUERY LANGUAGE)

SQL é uma linguagem de programação específica para trabalhar com bancos de dados relacionais. Ela permite que você crie, manipule e gerencie dados de forma estruturada e eficiente.

Com o SQL, você pode realizar diversas operações, como selecionar, inserir, atualizar e excluir dados, além de criar tabelas, índices e visões no banco de dados.



CONFIGURAÇÕES DO BANCO NO DJANGO

O Django já vem com o SQLite configurado, no settings.py é possível visualizar essa configuração.

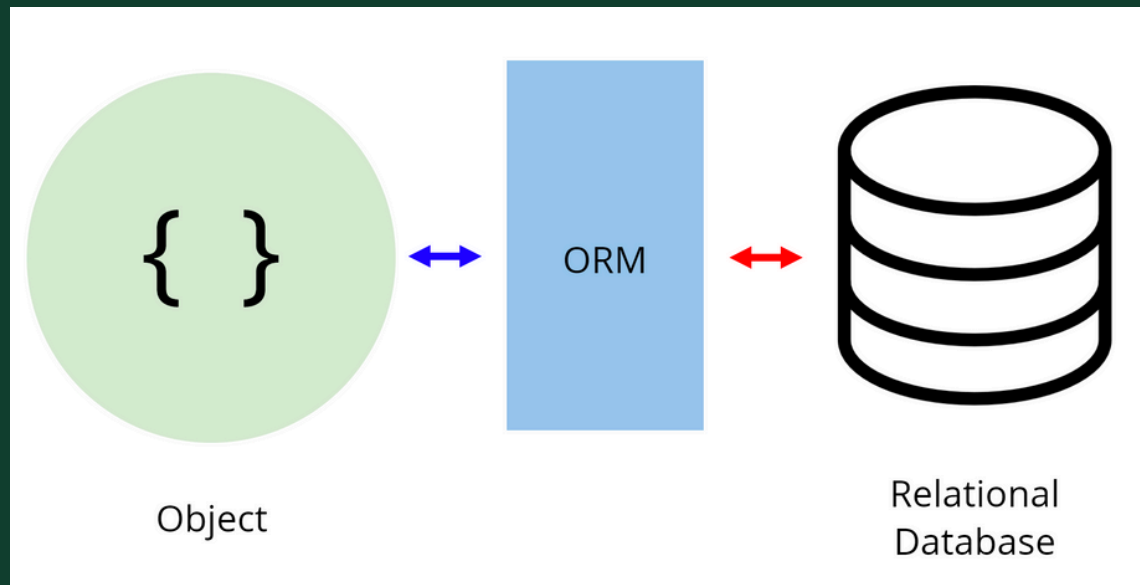
A **engine** é o tipo de banco de dados usado e o **name** especifica o local que o banco de dados será salvo.

```
1
2 # Database
3 # https://docs.djangoproject.com/en/5.1/ref/settings/#databases
4
5 DATABASES = {
6     'default': {
7         'ENGINE': 'django.db.backends.sqlite3',
8         'NAME': BASE_DIR / 'db.sqlite3',
9     }
10 }
11
```


ORM (OBJECT-RELATIONAL MAPPING)

ORM é uma técnica de mapear objetos de uma linguagem de programação orientada a objetos (como Python) diretamente para as tabelas de um banco de dados relacional (como MySQL, PostgreSQL, etc.).

Em vez de escrever consultas SQL diretamente, o ORM permite que você utilize classes e métodos em Python para representar tabelas e colunas do banco de dados.



MODELS NO DJANGO

No Django, os models são a representação de dados em um banco de dados relacional. Eles definem as tabelas e seus campos, assim como as relações entre elas, criando uma estrutura que reflete o domínio da aplicação.

O **field** é a definição de um atributo de um model. Cada field representa uma coluna na tabela do banco de dados.

O Django possui vários tipos de fields, como CharField, IntegerField, DateField, entre outros, que facilitam a definição do tipo de dado.

```
1  from django.db import models
2
3  # Create your models here.
4  class Product(models.Model):
5      name = models.CharField(max_length=255)
6      price = models.FloatField()
7      description = models.TextField(blank=True, null=True)
8      stock = models.IntegerField()
9      created_at = models.DateTimeField(auto_now_add=True)
10
```

MIGRAÇÕES NO DJANGO

Migrações são uma maneira do Django gerenciar alterações na estrutura do banco de dados. Elas transformam os models que definimos em tabelas no banco de dados.

Cada vez que fazemos mudanças nos models (como adicionar um novo campo), precisamos refletir essas mudanças no banco de dados.

Toda vez que criarmos um model ou fizermos uma alteração em um model já existente, temos que rodar no terminal o comando:

```
python manage.py makemigrations nome_do_app
```

Esse comando cria os arquivos de migrations indicando exatamente que mudança está sendo feita.

Para aplicar as mudanças no banco de dados precisamos rodar o comando:

```
python manage.py migrate
```

DJANGO ADMIN

O Django Admin é uma funcionalidade poderosa do framework Django que permite criar uma interface administrativa para gerenciar o seu banco de dados e as funcionalidades da sua aplicação de forma rápida e eficiente.

Com o Django Admin, é possível facilmente visualizar, criar, editar e excluir registros no seu banco de dados.

Por padrão, o Django Admin já vem configurado e conseguimos acessar usando a rota **admin/**.



DJANGO ADMIN

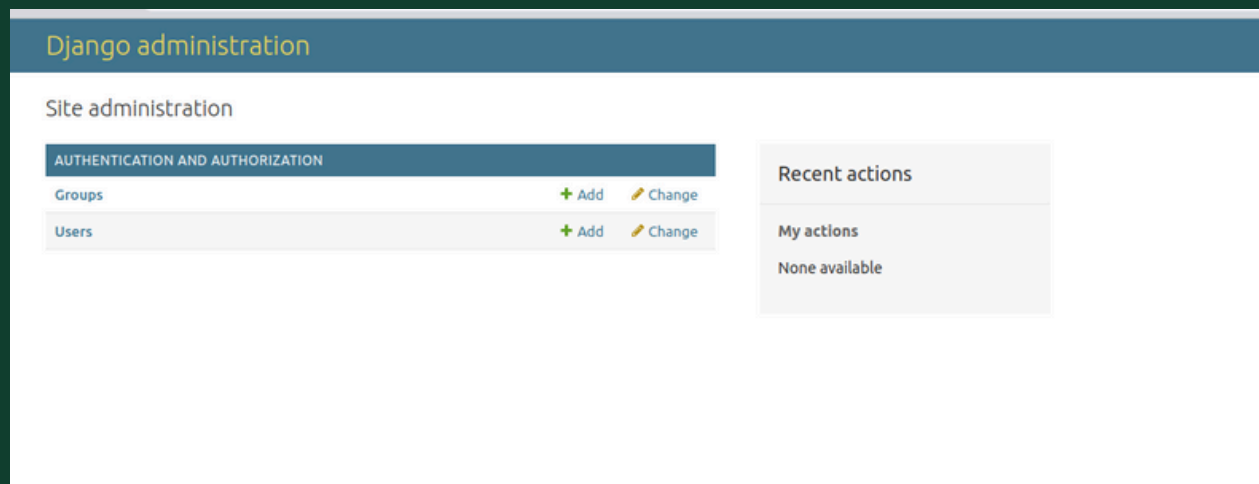
Para acessar o Django Admin, precisamos criar um super usuário que vai ter poderes para administrar os dados pelo Admin.

Para criar um super usuário temos que rodar o comando:

`python manage.py createsuperuser`

Esse comando vai solicitar no terminal o username, email e senha do usuário.


Uma vez feito isso, é só acessar o admin usando esse usuário e senha.



DJANGO ADMIN

Para garantir que um model seja visível no Admin, precisamos fazer o registro dele no `admin.py` do App.

Registrar o model permite que o Django Admin exiba uma interface gráfica para criar e visualizar os dados desse model, como se fosse uma planilha de dados.



```
1  from django.contrib import admin
2  from .models import Product
3  # Register your models here.
4
5  admin.site.register(Product)
```

ATIVIDADE PRÁTICA

Crie um app de gerenciamento de tarefas no Django. Para isso, deve ser:

- Criado um App para tarefas
- Criado o model de tarefas que terá os atributos:
 - nome: CharField()
 - descrição (opcional): TextField()
 - data de criação: DateTimeField()
 - concluido BooleanField()
- Migrado para o banco de dados
- Registrado o model no admin

Depois de registrar no Admin, adicione algumas tarefas pelo Admin.

USANDO DADOS NA VIEW

Podemos buscar os dados do banco diretamente na View, com isso, podemos enviar para um template HTML, por exemplo.

Usando o ORM, podemos fazer a consulta usando Python, ao invés do SQL.

Os métodos do ORM permitem buscar registros, filtrar dados e organizar as informações que serão enviadas pela View.



```
1 from django.shortcuts import render
2 from .models import Product
3
4 def get_products(request):
5     products = Product.objects.all()
6     context = {'products': products}
7     return render(request, 'products.html', context)
8
```


USANDO DADOS NA VIEW

O template vai exibir os dados da mesma forma que é feita quando passamos um dicionário.



```
1  <body>
2      {% for product in products %}
3          <h1>{{ product.name }}</h1>
4          <p>{{ product.description }}</p>
5          <p>{{ product.price }}</p>
6          <p>{{ product.stock }}</p>
7      {% endfor %}
8  </body>
```

ATIVIDADE PRÁTICA

No App de Gerenciamento de Tarefas, exiba as tarefas em template.

Vai ser necessário:

- Criar a view
- Usar a view em urls.py do App
- Criar um template HTML para exibir os dados

OBS: Verifique se a urls.py do Projeto está incluindo as urls do App.