

# django

## AULA 06

MANIPULAÇÃO DE DADOS E CRUD

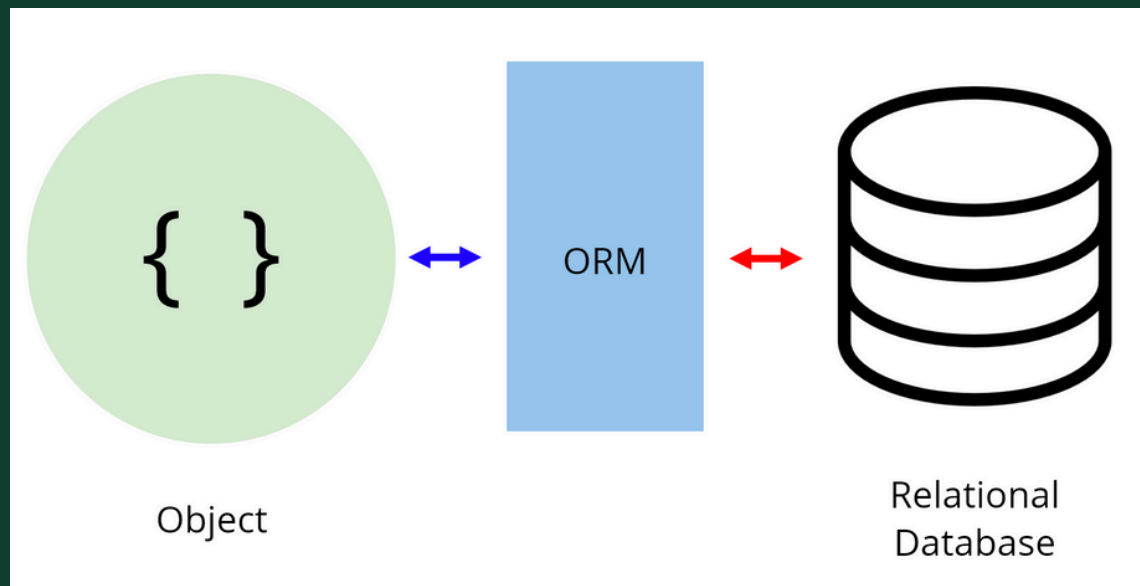
# O QUE VEREMOS HOJE

- 01 ORM
- 02 MÉTODOS ORM NO DJANGO
- 03 MÉTODOS HTTP
- 04 CRUD

# 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.



# MÉTODOS DO ORM NO DJANGO

Os métodos do ORM do Django são abstrações que permitem interagir com o banco de dados usando Python, sem escrever SQL diretamente.

Eles facilitam operações como criar, buscar, atualizar e deletar dados, traduzindo comandos Python em consultas SQL.

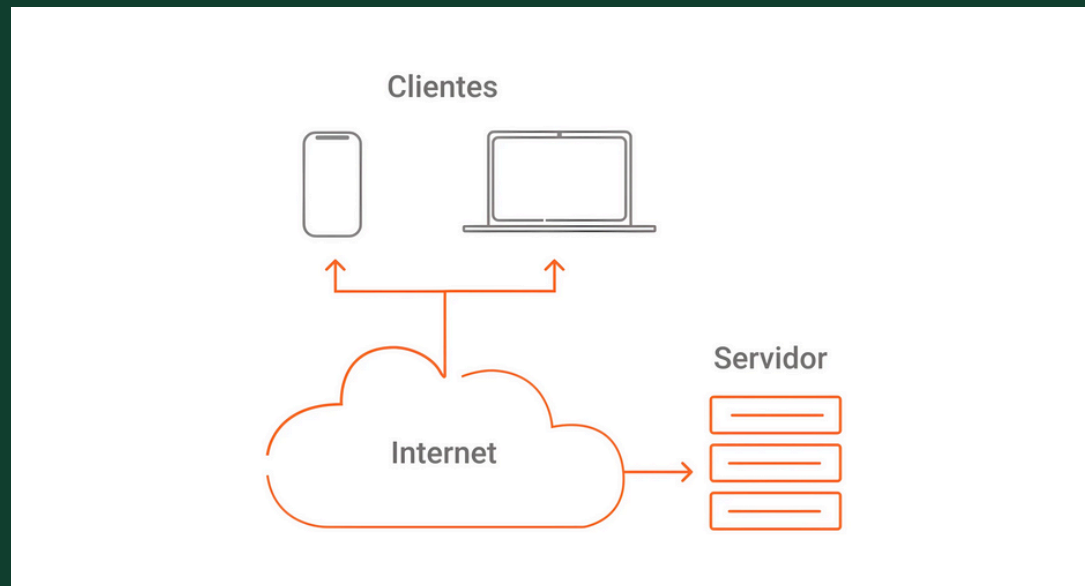
Método do ORM (Django)	Comando SQL
<code>Model.objects.create()/ Model.save()</code>	<code>INSERT INTO tabela (coluna1, coluna2) VALUES (valor1, valor2);</code>
<code>Model.objects.all()</code>	<code>SELECT * FROM tabela;</code>
<code>Model.objects.get(id=1)</code>	<code>SELECT * FROM tabela WHERE id = 1;</code>
<code>Model.objects.filter(id=1).update(ca mpo=x)</code>	<code>UPDATE tabela SET campo = x WHERE id = 1;</code>
<code>Model.objects.filter(id=1).delete()</code>	<code>DELETE FROM tabela WHERE id = 1;</code>

# MÉTODOS HTTP

Os métodos HTTP são verbos utilizados para indicar a ação que o cliente deseja realizar no servidor, como criar, ler, atualizar ou excluir dados.

Os principais métodos HTTP são: GET (recuperar dados), POST (criar novos dados), PUT (atualizar dados) e DELETE (excluir dados).

Esses métodos são fundamentais para o funcionamento do padrão CRUD (Create, Read, Update, Delete) na interação entre o cliente e o servidor.



# CRUD

CRUD é um acrônimo que representa as quatro operações básicas que é possível realizar em dados em um sistema:

- Create (Criar)
- Read (Ler/Consultar)
- Update (Atualizar)
- Delete (Deletar/Excluir)

Essas operações são fundamentais para qualquer aplicação que interaja com dados, como um sistema de gerenciamento de usuários, um sistema de estoque, etc.



# CREATE

O Create representa a funcionalidade de adicionar novos registros ao banco de dados.

No Django, geralmente envolve:

- Modelo (Model): Representa a estrutura da tabela no banco.
- Formulário: Para o usuário fornecer os dados.
- View: Processar os dados enviados e salvá-los no banco.

```
1 from django.shortcuts import render, redirect
2 from .forms import ProdutoForm
3 from .models import Produto
4
5 def criar_produto(request):
6     if request.method == 'POST':
7         form = ProdutoForm(request.POST)
8         if form.is_valid():
9             form.save()
10            return redirect('lista_produtos') # Redireciona para a página de listagem de produtos
11     else:
12         form = ProdutoForm()
13     return render(request, 'criar_produto.html', {'form': form})
```

# CREATE

No template só precisamos colocar o formulário para preenchimento dos dados.

O método HTTP utilizado no formulário precisa ser o POST para envio de dados que serão criados.



```
1 <body>
2     <h1>Cadastrar Produto</h1>
3     <!-- Template com Renderização Básica -->
4     <form method="post">
5         {% csrf_token %}
6         <button type="submit">Enviar</button>
7     </form>
8 </body>
```



# READ

O Read no Django permite buscar e exibir informações do banco de dados, utilizando o modelo (Model) e enviando os dados recuperados para o template através da view.

O método `.all()` retorna todos os registros de um modelo.



```
1 from django.shortcuts import render, redirect
2 from .forms import ProdutoForm
3 from .models import Produto
4
5 def lista_produtos(request):
6     context = {
7         'produtos': Produto.objects.all()
8     }
9     return render(request, 'produtos.html', context)
10
```

# READ

A exibição de todos os itens no template pode ser feito utilizando o for para percorrer a lista.

```
1  <!-- Lista de produtos -->
2  <h1>Lista de Produtos</h1>
3  <table>
4  <thead>
5      <tr>
6          <th>Nome</th>
7          <th>Preço</th>
8          <th>Ações</th>
9      </tr>
10 </thead>
11 <tbody>
12     {% for produto in produtos %}
13         <tr>
14             <td>{{ produto.nome }}</td>
15             <td>R$ {{ produto.preco }}</td>
16         </tr>
17     {% endfor %}
18 </tbody>
19 </table>
```

# READ

O Read também pode é usado quando precisamos buscar um registro específico.

Para esses casos, utilizamos o método `get()`, que retorna apenas um registro que satisfaça a condição passada.


O método `get()` é ideal quando sabemos que apenas um item será retornado, como ao buscar por um identificador único (ex.: `id`).

```
1 from django.shortcuts import render, redirect
2 from django.http import Http404
3 from .forms import ProdutoForm
4 from .models import Produto
5
6
7 def buscar_produto(request, id):
8
9     try:
10         produto = Produto.objects.get(id=id)
11
12     except Produto.DoesNotExist:
13         raise Http404('Produto não encontrado')
14
15     return render(request, 'produto.html', {'produto': produto})
```

# READ

Para casos em que precisamos da informação única do registro, é comum que essa informação seja passada através de parâmetros dinâmicos na URL.

O parâmetro `<int:id>` é usado para capturar dinamicamente um valor da URL. No exemplo, ele captura um número inteiro (o ID) e o passa como argumento para a view correspondente.



```
1 from django.urls import path
2 from .views import criar_produto, lista_produtos, buscar_produto
3
4 urlpatterns = [
5     path('criar/', criar_produto, name='criar_produto'),
6     path('listar/', lista_produtos, name='lista_produtos'),
7     path('buscar/<int:id>/', buscar_produto, name='buscar_produto')
8 ]
```

# ATIVIDADE PRÁTICA

No App de Tarefas faça a operação de READ que exiba tanto a lista de todas as tarefas quanto a exibição de apenas uma determinada tarefa com base no ID.

Vão ser necessárias duas views e duas páginas diferentes para exibir cada uma delas.

# UPDATE

O Update no Django é utilizado para modificar registros existentes no banco de dados.

Nessa operação também é necessário identificar o registro que será atualizado, por isso, também é preciso de um identificador único.

OBS: nesse exemplo está sendo usado o método POST, pois os formulários HTML não suportam o método PUT nativamente.

```
1  from django.shortcuts import render, get_object_or_404, redirect
2  from .models import Produto
3  from .forms import ProdutoForm
4
5  def update_produto(request, id):
6      produto = get_object_or_404(Produto, id=id) # Busca o produto pelo ID
7      if request.method == 'POST':
8          form = ProdutoForm(request.POST, instance=produto) # Associa o produto ao formulário
9          if form.is_valid():
10             form.save() # Atualiza o registro no banco de dados
11             return redirect('lista_produtos')
12      else:
13          form = ProdutoForm(instance=produto) # Preenche o formulário com os dados existentes
14          return render(request, 'update_produto.html', {'form': form})
15
16
```

# UPDATE

Na URL também podemos receber o identificador do registro usando dados dinâmicos que será passado para view quando for feito o request.

O parâmetro `<int:id>` na URL indica que o Django espera um número inteiro como parte da URL. Esse número será capturado e enviado como argumento para a função `atualizar_produto` na view.

```
1 from django.urls import path
2 from .views import criar_produto, lista_produtos, buscar_produto, atualizar_produto
3
4 urlpatterns = [
5     path('criar/', criar_produto, name='criar_produto'),
6     path('listar/', lista_produtos, name='lista_produtos'),
7     path('buscar/<int:id>/', buscar_produto, name='buscar_produto'),
8     path('atualizar/<int:id>/', atualizar_produto, name='atualizar_produto')
9 ]
```

# DELETE

O Delete no Django é utilizado para remover registros existentes do banco de dados.

Assim como no Update, é necessário identificar o registro que será excluído, utilizando um identificador único (ex.: ID).

OBS: No exemplo, usamos o método POST para confirmar a exclusão, pois formulários HTML não suportam o método DELETE nativamente.

```
1  from django.shortcuts import render, get_object_or_404, redirect
2  from .models import Produto
3  from .forms import ProdutoForm
4
5  def delete_produto(request, id):
6      produto = get_object_or_404(Produto, id=id)
7      if request.method == 'POST':
8          produto.delete() # Remove o registro do banco de dados
9          return redirect('lista_produtos')
10     return render(request, 'delete_produto.html', {'produto': produto})
11
```



# DELETE

No template podemos incluir a exclusão de várias formas, uma das formas é na listagem dos dados.

O confirm é um método que permite a exibição de um alerta antes da exclusão, por questões de segurança

```
1 <!-- Lista de produtos -->
2 <h1>Lista de Produtos</h1>
3 <table>
4 <thead>
5 <tr>
6 <th>Nome</th>
7 <th>Preço</th>
8 <th>Ações</th>
9 </tr>
10 </thead>
11 <tbody>
12 {% for produto in produtos %}
13 <tr>
14 <td>{{ produto.nome }}</td>
15 <td>R$ {{ produto.preco }}</td>
16 <td>
17 <!-- Botão para excluir -->
18 <form method="post" action="{% url 'delete_produto' produto.id %}" style="display: inline;">
19 {% csrf_token %}
20 <button type="submit" onclick="return confirm('Tem certeza que deseja excluir o produto {{ produto.nome }}?')">
21 Excluir
22 </button>
23 </form>
24
25 <!-- Link para editar (opcional) -->
26 <a href="{% url 'update_produto' produto.id %}">Editar</a>
27 </td>
28 </tr>
29 {% endfor %}
30 </tbody>
31 </table>
32
```

# ATIVIDADE PRÁTICA

No App de Tarefas, implemente os métodos Update e Delete utilizando o ID para identificar o item que vai ser atualizar ou excluído.