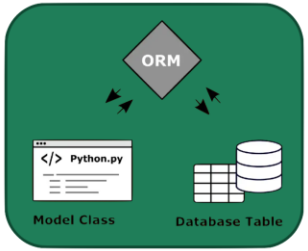
Universidade Federal de Goiás
Instituto de Informática
Prof. Ronaldo Martins da Costa

Django Object-Relational Mapping (ORM)

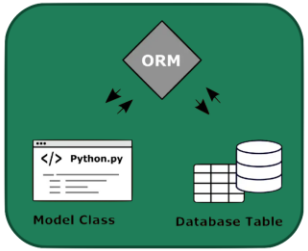


- *Object-Relational Mapping* (ORM), ou em português, mapeamento objeto-relacional, é uma técnica para aproximar o paradigma de desenvolvimento de aplicações orientadas a objetos ao paradigma do banco de dados relacional

- O uso da técnica de mapeamento objeto-relacional é realizado através de um mapeador objeto-relacional que geralmente é a biblioteca ou framework que ajuda no mapeamento e uso do banco de dados

- A documentação do Django para este tópico pode ser encontrada em: https://docs.djangoproject.com/pt-br/4.1/topics/db/queries/

Django Object-Relational Mapping (ORM)

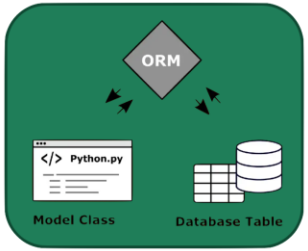


- **QuerySet** - Um QuerySet representa uma coleção de objetos do banco de dados

- Ela pode ter nenhum, um ou muitos filtros (critérios que refinam a coleção baseado nos parâmetros dados)

- Em termos SQL, um QuerySet equipara-se a uma consulta SELECT, e um filtro com uma clausula limitadora como WHERE ou LIMIT

Django Object-Relational Mapping (ORM)



- Executando algumas instruções

O comando a seguir disponibiliza um shell para execução de comandos

```
python manage.py shell
```

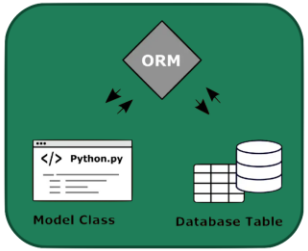
Assim como em uma view, é possível importar modelos

```
from exemplo01.models import pessoa
```

Esa linha vai imprimir todos os registros da tabela pessoa

```
pessoa.objects.all()
```

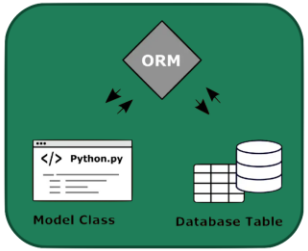
Django Object-Relational Mapping (ORM)



- O Django retorna os registros em uma estrutura de matriz

```
regs = pessoa.objects.all()
print(regs[0].nome)
print(regs[0].email)
print(regs[0].celular)
```

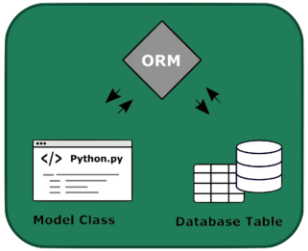
Django Object-Relational Mapping (ORM)



- Retornando colunas específicas de uma tabela

```
regs = pessoa.objects.values('nome', 'email')
print(regs[0]['nome'])
print(regs[0]['email'])
```

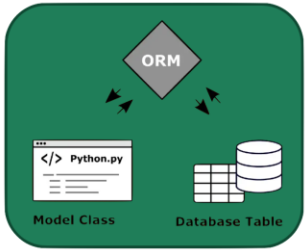
Django Object-Relational Mapping (ORM)



- Retornando colunas específicas de uma tabela com o values_list

```
regs = pessoa.objects.values_list('nome', 'email')
print(regs[0][0])
print(regs[0][1])
```

Django Object-Relational Mapping (ORM)



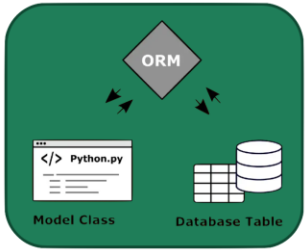
- Filtrando os resultados
  A linha a seguir retornará apenas os registros cuja função seja igual a “Médico”

```
pessoa.objects.filter(funcao="Médico")
```

- SQL equivalente

```
Select * from pessoa where funcao="Médico"
```

Django Object-Relational Mapping (ORM)



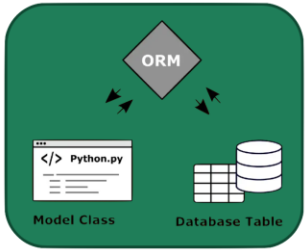
- Filtrando os resultados
  Se desejar filtrar por mais que um campo da tabela, a sintaxe será

```
pessoa.objects.filter(funcao="Médico", ativo=True)
```

- SQL equivalente

```
Select * from pessoa where funcao="Médico” and ativo=True
```

Django Object-Relational Mapping (ORM)



- Filtrando os resultados
  Se desejar filtrar por mais que um campo da tabela, a sintaxe será

```
pessoa.objects.filter(funcao="Médico", ativo=True,
   nascimento="2002-05-06")
```

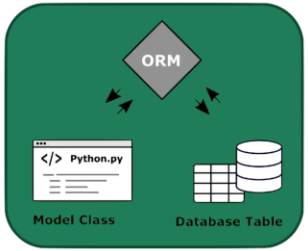
- SQL equivalente

```
Select * from pessoa where funcao="Médico" and ativo=True and
   nascimento="2002-05-05"
```

- A , (vírgula) faz o papel de AND em SQL

- E como utilizar um **OR** em no QuerySet ?

Django Object-Relational Mapping (ORM)



- Filtrando os resultados **OR**
  É necessário importar uma biblioteca chamada **Q**

```
from django.db.models import Q
```

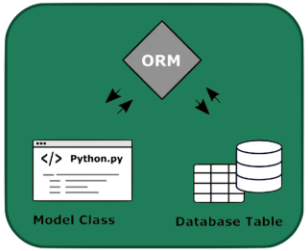
- Vamos construir uma query que retorne todos os registros onde a função for Médico ou Professor

```
query = Q( Q(funcao="Médico") | Q(funcao="Professor") )
pessoa.objects.filter(query)
```

- SQL equivalente

```
Select * from pessoa where funcao="Médico" or função="Professor"
```

Django Object-Relational Mapping (ORM)



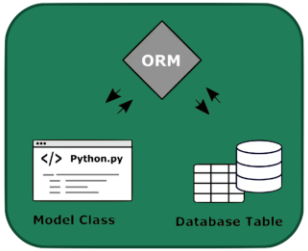
- Utilizando **AND** e **OR** em uma mesma Query
  É necessário importar uma biblioteca chamada **Q**

```
from exemplo01.models import pessoa
from django.db.models import Q
query = Q( Q(funcao="Médico") | Q(funcao="Professor") )
pessoa.objects.filter(query, ativo=True)
```

- SQL equivalente

```
Select * from pessoa where funcao="Médico" or função="Professor”
   and ativo=True
```

Django Object-Relational Mapping (ORM)



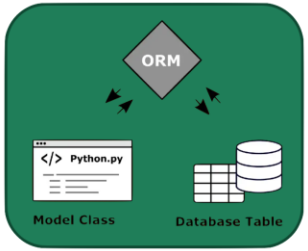
- Utilizando **EXCLUDE**

```
from exemplo01.models import pessoa
pessoa.objects.filter(nascimento="1980-01-
    01").exclude(ativo=False)
```

- SQL equivalente

```
Select * from exemplo01_pessoa where nascimento="1980-01-01" and
    ativo=True
```

Django Object-Relational Mapping (ORM)



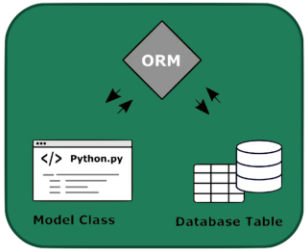
- Para ordenar uma query, utilizamos o **ORDER_BY**

```
from exemplo01.models import pessoa
pessoa.objects.filter(nascimento="1980-01-
   01").exclude(ativo=False).order_by("nome")
```

- SQL equivalente

```
Select * from exemplo01_pessoa where nascimento="1980-01-01" and
   ativo=True order by nome asc
```

Django Object-Relational Mapping (ORM)



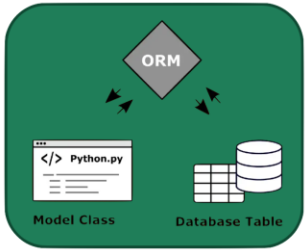
- Para ordenar de forma descendente, utilizamos o **-**

```
from exemplo01.models import pessoa
pessoa.objects.filter(nascimento="1980-01-
   01").exclude(ativo=False).order_by("-nome")
```

- SQL equivalente

```
Select * from exemplo01_pessoa where nascimento="1980-01-01" and
   ativo=True order by nome desc
```

Django Object-Relational Mapping (ORM)



- **GET**

  Deve ser utilizado “para uma busca por uma chave única”

```
pessoa.objects.get(nome='Fulano de Tal')
```

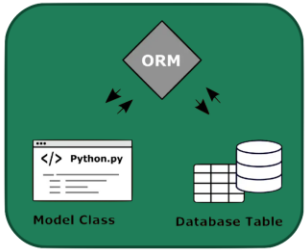
- Se existirem dois registros com a mesma chave o retorno vai ser

```
get() returned more than one pessoa -- it returned 2!
```

- Se não existir um registro com a chave especificada o retorno será:

```
matching query does not exist
```

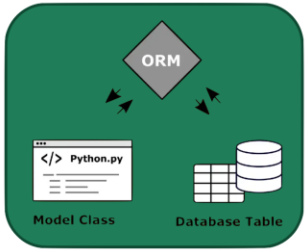
Django Object-Relational Mapping (ORM)



GET

```
regs = pessoa.objects.get(nome='Fulano de Tal')
regs.nome
regs.email
regs.celular
```

Django Object-Relational Mapping (ORM)



- **CREATE** cria um registro na tabela.

  Vamos utilizar uma estrutura semelhante a pagina4 para construir uma função de Create no DB

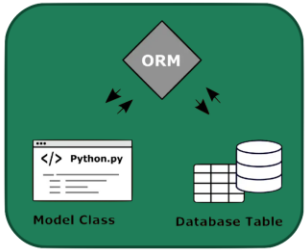
```
Copie a pagina4.html para pagina5.html
```

Django Object-Relational Mapping (ORM)

- Vamos criar uma nova rota em nosso arquivo urls.py

```
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index_alias'),
    path('pagina0', views.pagina0, name='pagina0_alias'),
    path('pagina1', views.pagina1, name='pagina1_alias'),
    path('pagina2', views.pagina2, name='pagina2_alias'),
    path('pagina3', views.pagina3, name='pagina3_alias'),
    path('pagina4', views.pagina4, name='pagina4_alias'),
    path('pagina5', views.pagina5, name='pagina5_alias'),
    path('pessoa_menu', views.pessoa_menu.as_view(), name='pessoa_menu_alias'),
    path("pessoa_list/", views.pessoa_list.as_view(), name='pessoa_list_alias'),
    path("pessoa_create/", views.pessoa_create.as_view(), name='pessoa_create_alias'),
    path("pessoa_update/<int:pk>/", views.pessoa_update.as_view(), name='pessoa_update_alias'),
    path('pessoa_delete/<int:pk>/', views.pessoa_delete.as_view(), name='pessoa_delete_alias'),
]
```

Django Object-Relational Mapping (ORM)

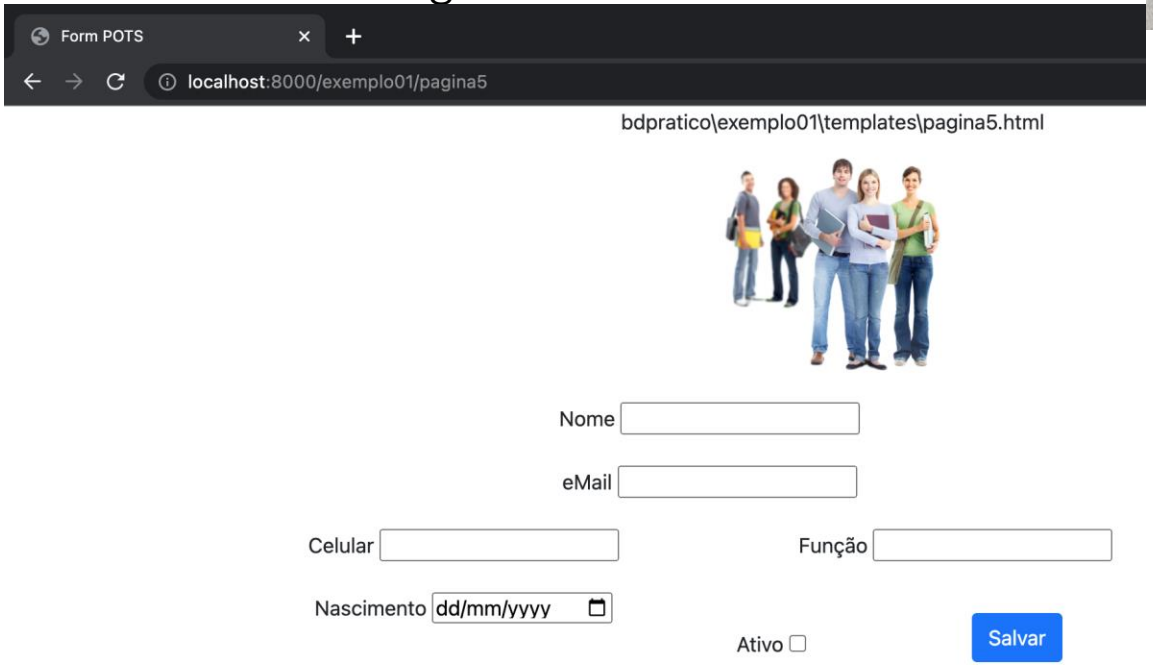


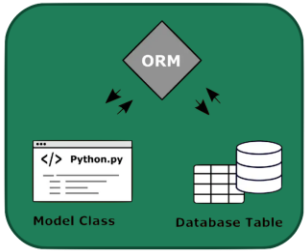
- Finalmente, altere o código contido em views.py

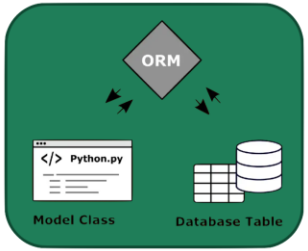
```
def pagina5(request):
    xnome = request.POST.get('nome')
    xemail = request.POST.get('email')
    xcelular = request.POST.get('celular')
    xfuncao = request.POST.get('funcao')
    xnascimento = request.POST.get('nascimento')
    xativo = request.POST.get('ativo')

    print("Nome:", xnome)
    print("eMail:", xemail)
    print("Celular:", xcelular)
    print("Funcao:", xfuncao)
    print("Nascimento:", xnascimento)
    print("ativo:", xativo)

    if (xnome is not None):
        xativo = False
        if (xativo == 'on'):
            xativo = True
        pessoa.objects.create(nome=xnome, email=xemail, celular=xcelular,
                              funcao=xfuncao, nascimento=xnascimento, ativo=xativo)

    return render(request, 'pagina5.html')
```

Django Object-Relational Mapping (ORM)

- Se os ajustes no projeto foram realizados corretamente, você verá uma tela como esta e o registro estará na tabela do DB







UPDATE

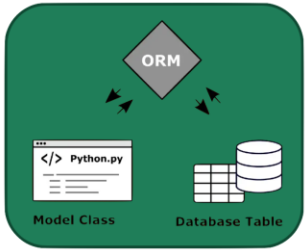
Pode ser utilizado para atualizar um registro, combinando com o *filter*

```
pessoa.objects.filter(id=1).update(nome='Atualizando o nome')
```

Também pode ser utilizado para atualiza todos os registros

```
pessoa.objects.update(ativo=True)
```

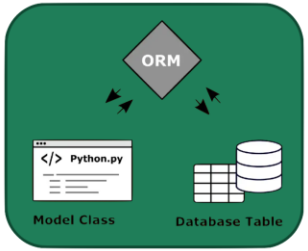
Django Object-Relational Mapping (ORM)



- **EXISTS** Retorna True caso uma *queryset* retorno registros e Falso em caso contrário

```
pessoa.objects.filter(id=1).exists()

pessoa.objects.filter(ativo=True).exists()
```

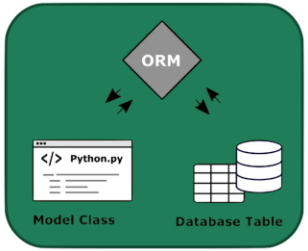
Django Object-Relational Mapping (ORM)



- **COUNT** Retorna um inteiro representando a quantidade de registros retornados pela *queryset*

```
pessoa.objects.filter(ativo=True).count()
```

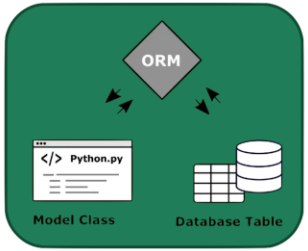
Django Object-Relational Mapping (ORM)



- **FIRST** e **LAST** Retornam respectivamente o primeiro e o último registro de um *queryset*

```
pessoa.objects.filter(ativo=True).first()

pessoa.objects.filter(ativo=True).last()
```

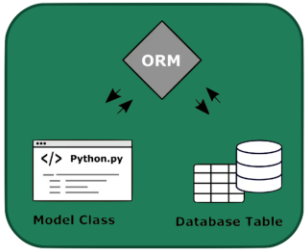
Django Object-Relational Mapping (ORM)



- **IN** Retornam os registros que satisfaze uma lista

```
pessoa.objects.filter(id__in=[1, 2, 3, 4, 5])
```

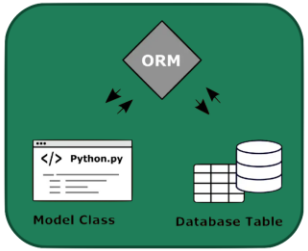
Django Object-Relational Mapping (ORM)



- **GT** → Maior que
- **GTE** → Maior que ou igual
- **LT** → Menor que
- **LTE** → Menor que ou igual

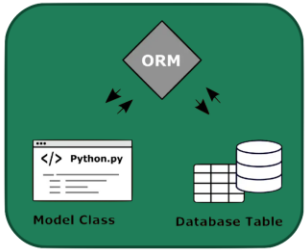
```
pessoa.objects.filter(id__gt=5)
pessoa.objects.filter(id__lt=5)
pessoa.objects.filter(id__gte=5)
pessoa.objects.filter(id__lte=5)
```

Django Object-Relational Mapping (ORM)



- **STARTSWITH** Retorna os registros iniciados com uma *string* (é sensível a letras maiúsculas e minúsculas

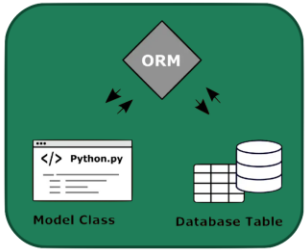
```
pessoa.objects.filter(nome__startswith='Jo')
```



- **CONTAINS** Retorna os registros onde um determinado campo contenha uma *string* (é sensível a letras maiúsculas e minúsculas

```
pessoa.objects.filter(nome__contains='Jo')
```

Django Object-Relational Mapping (ORM)

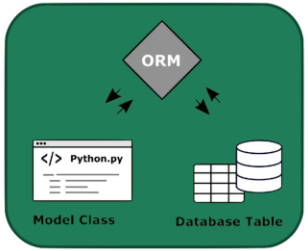


- Retorna entre tabelas

  Vamos adicionar mais 2 tabelas em nossa APP

```
class procedimento(models.Model):
    descricao = models.CharField(max_length=50, null=False, blank=False, verbose_name='Descricao')
    cid = models.CharField(max_length=20, null=False, blank=False, verbose_name='CID')
    valor = models.FloatField(null=True, blank=True, default=None, verbose_name='Valor')

    def __str__(self):
        return self.descricao

    class Meta:
        ordering = ['descricao']
```

Django Object-Relational Mapping (ORM)

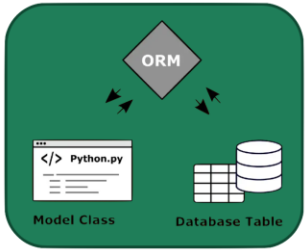


- Retorna entre tabelas

  Vamos adicionar mais 2 tabelas em nossa APP

```
class procedimento_executado(models.Model):
    pessoa = models.ForeignKey(pessoa, on_delete=models.CASCADE)
    procedimento = models.ForeignKey(procedimento, on_delete=models.CASCADE)
    obs = models.CharField(max_length=50, null=False, blank=False, verbose_name='Obs')
    quantidade = models.FloatField(null=True, blank=True, default=None, verbose_name='Quantidade')

    def __str__(self):
        return self.obs

    class Meta:
        ordering = ['pessoa', 'procedimento']
```

Django Object-Relational Mapping (ORM)



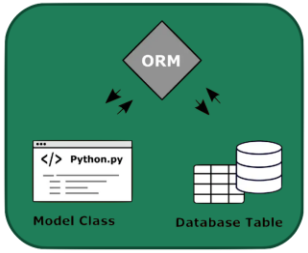
- Após atualizar o arquivo models.py execute

```
python manage.py makemigrations exemplo01

python manage.py migrate
```

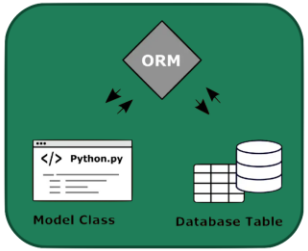
Django Object-Relational Mapping (ORM)

- Se os ajustes no projeto foram realizados corretamente, você verá uma tela como esta e o registro estará na tabela do DB

```
(BigData-env) ronaldocosta@Ronaldos-MacBook-Pro bdpratico % python manage.py makemigrations exemplo01
Migrations for 'exemplo01':
  exemplo01/migrations/0006_procedimento_alter_pessoa_nascimento_and_more.py
    - Create model procedimento
    - Alter field nascimento on pessoa
    - Create model procedimento_executado
(BigData-env) ronaldocosta@Ronaldos-MacBook-Pro bdpratico % python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, exemplo01, sessions
Running migrations:
  Applying exemplo01.0006_procedimento_alter_pessoa_nascimento_and_more... OK
```



Django Object-Relational Mapping (ORM)

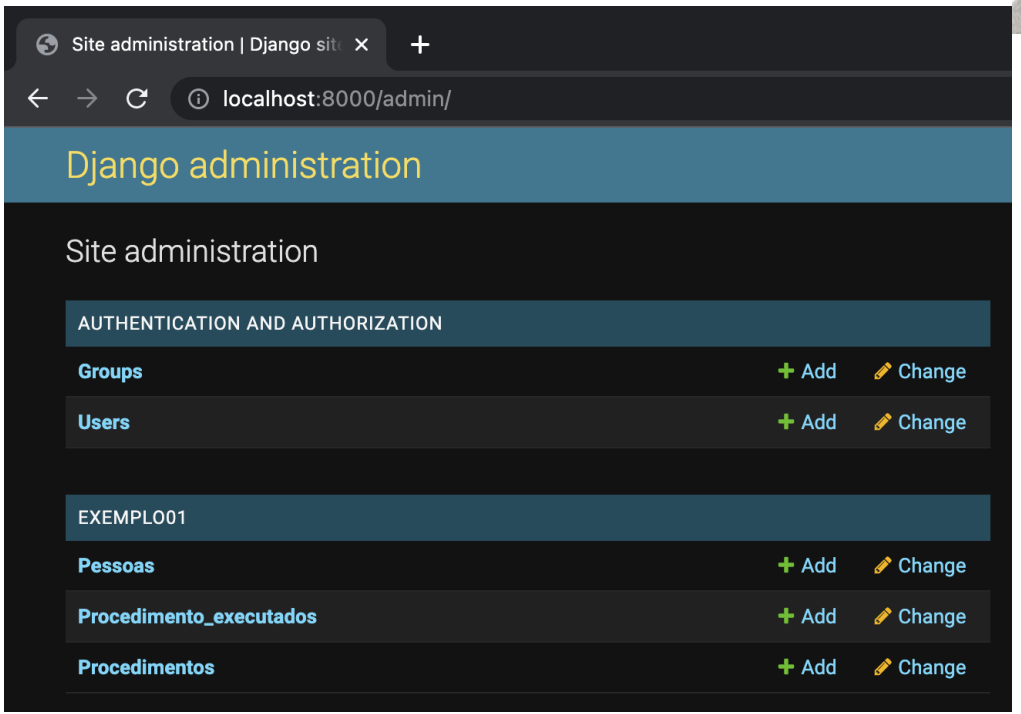


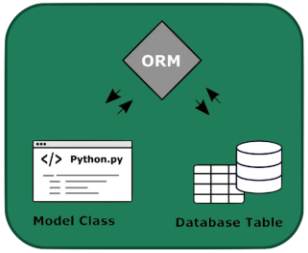
- Para facilitar a manutenção destas tabelas, vamos atualizar o arquivo admin.py

```
from django.contrib import admin
from .models import *
@admin.action(description="Habilitar Registros Selecionados")
def habilitar_pessoas(ModelAdmin, request, queryset):
    for p in queryset:
        p.ativo = True
        p.save()
@admin.action(description="Desabilitar Registros Selecionados")
def desabilitar_pessoas(ModelAdmin, request, queryset):
    queryset.update(ativo=False)
class PessoaCustomizado(admin.ModelAdmin):
    list_display = ('nome', 'email', 'celular', 'funcao', 'nascimento', 'calcula_idade', 'ativo')
    actions = [habilitar_pessoas, desabilitar_pessoas]

    @admin.display(description='Idade')
    def calcula_idade(self, obj):
        from datetime import date
        hoje = date.today()
        idade = hoje.year - obj.nascimento.year
        return idade

admin.site.register(pessoa, PessoaCustomizado)
admin.site.register(procedimento)
admin.site.register(procedimento_executado)
```

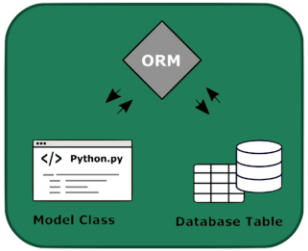
Django Object-Relational Mapping (ORM)

- Se os ajustes no projeto foram realizados corretamente, você verá uma tela como esta





Django Object-Relational Mapping (ORM)



- Executando algumas instruções

Vamos importar as novas tabelas

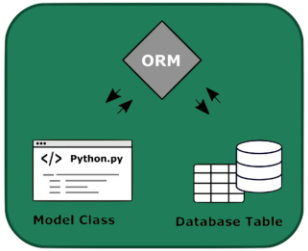
```
from exemplo01.models import pessoa, procedimento, procedimento_executado
```

Criando *queryset* com campos das outras tabelas

```
procedimento_executado.objects.filter(pessoa__ativo=True)

procedimento_executado.objects.filter(pessoa__ativo=True, procedimento__cid=512)
```

Dois sinais “__” (*undersore*) indicam para o Django a tabela que contém o campo desejado

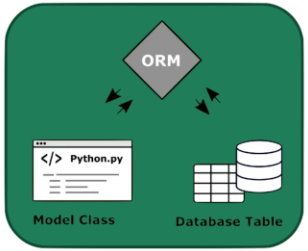
Django Object-Relational Mapping (ORM)



- Visualizando os campos das tabelas através da *queryset*

```
reg=procedimento_executado.objects.filter(pessoa__ativo=True, procedimento__cid=512)

print(reg[0].pessoa.celular)
print(reg[0].procedimento.valor)
```

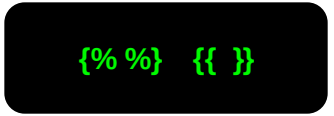
Django Object-Relational Mapping (ORM)



- Verificando a frase SQL que o Django criou

```
reg=procedimento_executado.objects.filter(pessoa__ativo=True, procedimento__cid=512)

print(str(reg.query))
```

Tags de Template do Django



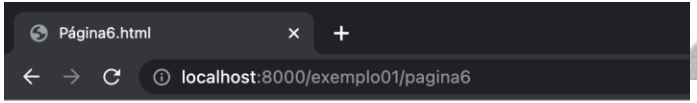
- A linguagem de template do Django permite inserir algumas funcionalidade nos templates. Vamos criar o pagina6.html.

```
{% for regs in pessoas %}
    <p>
        {% if regs.ativo %}
            <strong>
                {{ regs.nome }} - {{ regs.email }} - {{ regs.celular }} - {{ regs.ativo}}
            </strong>
        {% else %}
            {{ regs.nome }} - {{ regs.email }} - {{ regs.celular }} - {{ regs.ativo}}
        {% endif %}
    </p>
{% endfor %}
```

- Neste exemplo, todos os registros que ativo for igual a True ficarão com fonte “<Strong>”

Tags de Template do Django

- A pagina2.html foi alterada e uma nova rota criada, tela a seguir exibe o resultado pagina6.html



Hello World!

bdpratico\exemplo01\templates\pagina6.html

Antes do Almoço - email@antesalmoco.com.br - 1111111111 - False

Fulano de Tal 2 - fulanotal@gmail.com - (98) 76543-2109 - True

Joao da Silva 1 - joaosilva@gmail.com - (12) 34567-8901 - True

Joao da Silva 4 - joaosilva2@gmail.com - (98) 76543-2109 - False

Nova Pessoa 6 - novapessoa@gmail.com - 2238729387 - True

Novo nome - lkjlkjklj - klkljkjlkj - True

Outra Nova 7 - outranova@gmail.com - 7373463748 - True

Ronaldo Martins da Costa 3 - ronaldocosta@ufg.br - (62) 98271-6303 - True

create save - create - save - True

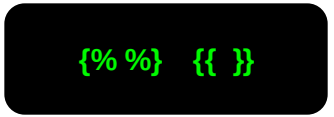
reg 13 - sdlkjsdlkjsdlkj - sdkjhsdkjh - True

reg 17 - lkj - qlkjlkj - True

reg 19 - kjh kjh kjh kjh kjh kjh - kjh kjh kjh - False

reg 21 - gdsfsfsdfsdfsd - None - True

Tags de Template do Django

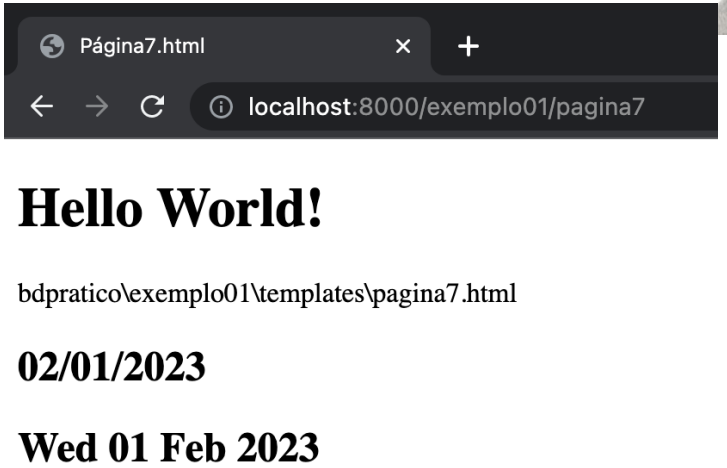


- O exemplo a seguir demonstra algumas funções para trabalhar com datas

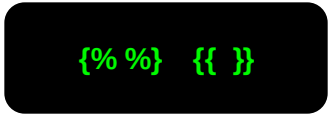
```
<body>
       <h1>Hello World!</h1>
       <p>bdpratico\exemplo01\templates\pagina7.html</p>
       <h2>{{ data|date:"SHORT_DATE_FORMAT" }}</h2>
       <h2>{{ data|date:"D d M Y" }}</h2>
       {% for regs in pessoas %}
           <p>
               {% if regs.ativo %}
                   <strong>
                       {{ regs.nome }} - {{ regs.email }} - {{ regs.celular }} - {{ regs.ativo}}
                   </strong>
               {% else %}
                   {{ regs.nome }} - {{ regs.email }} - {{ regs.celular }} - {{ regs.ativo}}
               {% endif %}
           </p>
       {% endfor %}
   </body>
```

Tags de Template do Django

- A pagina7.html demonstra as funções para manipulação de datas



Tags de Template do Django

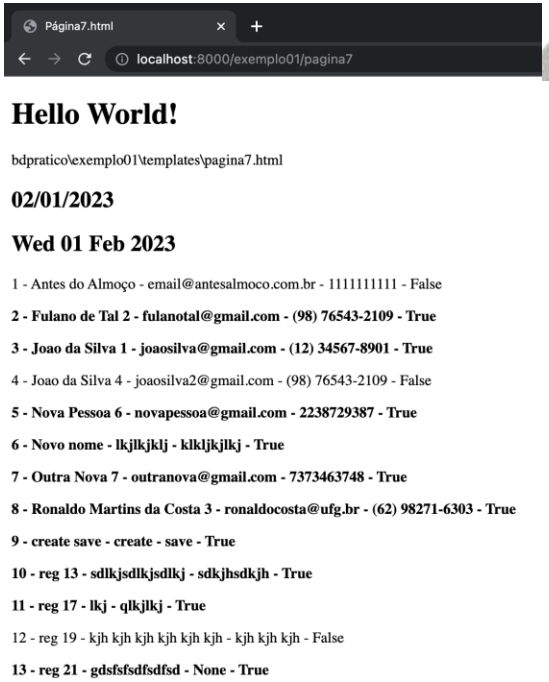


- Existem diversas tags de template que podem inserir elementos úteis nos templates

```
<body>
        <h1>Hello World!</h1>
        <p>bdpratico\exemplo01\templates\pagina7.html</p>
        <h2>{{ data|date:"SHORT_DATE_FORMAT" }}</h2>
        <h2>{{ data|date:"D d M Y" }}</h2>
        {% for regs in pessoas %}
            <p>
                {% if regs.ativo %}
                    <strong>
                        {{forloop.counter}} - {{ regs.nome }} - {{ regs.email }} - {{ regs.celular }} - {{ regs.ativo}}
                    </strong>
                {% else %}
                    {{forloop.counter}} - {{ regs.nome }} - {{ regs.email }} - {{ regs.celular }} - {{ regs.ativo}}
                {% endif %}
            </p>
        {% endfor %}
    </body>
```

Tags de Template do Django

- A pagina7.html também demonstra um contador inserido no template



Tutorial com mais funcionalidades para templates no django:
https://docs.djangoproject.com/en/4.1/ref/templates/builtins/

Tags de Template do Django

- Vamos agora utilizar o recurso de estender (***extends***) um template
- Crie bdpratico\exemplo01\templates\base.html

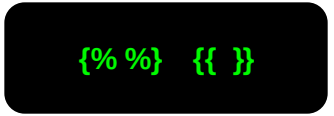
```
<!DOCTYPE html>
{% load static %}
{% load bootstrap5 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
{% bootstrap_messages %}
<html lang="pt-br">
    <head>
        <title>
            {% block titulo %}{% endblock %}
        </title>
    </head>
    <body>
        <center>
            <p>bdpratico\exemplo01\templates\base.html</p>
            <br>
            <h2>Esta é o template base.html que será estendida para os demais templates</h2>
            <br>
            {% block corpo %}{% endblock %}
        </center>
    </body>
```

Tags de Template do Django

- Vamos agora utilizar o recurso de estender (*extends*) um template
- Crie bdpratico\exemplo01\templates\base.html

```
    <center>
        <br>
        <hr>
        <h3>{% block rodape %}{% endblock %}</h3>
        <h4>
            <img src="{% static 'direitos.png' %}" class="img-fluid" alt="Teste" width="5%">
            Direitos autorais do Site
        </h4>
    </center>
</html>
```

Tags de Template do Django



- Agora crie:

bdpratico\exemplo01\templates\exemplo01\listar_pessoas.html

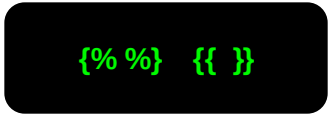
```
{% extends 'base.html' %}

{% block titulo %}Coloque um Título Aqui{% endblock %}

{% block corpo %}
    <table class="table table-striped table-hover">
        <th>Nome</th>
        <th>eMail</th>
        <th>Celular</th>
        {% for regs in pessoas %}
            <tr>
                <td>{{ regs.nome }}</td>
                <td>{{ regs.email }}</td>
                <td>{{ regs.celular }}</td>
            </tr>
        {% endfor %}
    </table>
{% endblock %}

{% block rodape %}Enviamos aqui o que desejamos que apareça escrito na área do rodapé{% endblock %}
```

Tags de Template do Django

- Vamos criar uma nova rota em nosso arquivo urls.py

```
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index_alias'),
    path('pagina0', views.pagina0, name='pagina0_alias'),
    path('pagina1', views.pagina1, name='pagina1_alias'),
    path('pagina2', views.pagina2, name='pagina2_alias'),
    path('pagina3', views.pagina3, name='pagina3_alias'),
    path('pagina4', views.pagina4, name='pagina4_alias'),
    path('pagina5', views.pagina5, name='pagina5_alias'),
    path('pagina6', views.pagina6, name='pagina6_alias'),
    path('pagina7', views.pagina7, name='pagina7_alias'),
    path('pagina8', views.pagina8, name='pagina8_alias'),
    path('pessoa_menu', views.pessoa_menu.as_view(), name='pessoa_menu_alias'),
    path("pessoa_list/", views.pessoa_list.as_view(), name='pessoa_list_alias'),
    path("pessoa_create/", views.pessoa_create.as_view(), name='pessoa_create_alias'),
    path("pessoa_update/<int:pk>/", views.pessoa_update.as_view(), name='pessoa_update_alias'),
    path('pessoa_delete/<int:pk>/', views.pessoa_delete.as_view(), name='pessoa_delete_alias'),
]
```

Tags de Template do Django

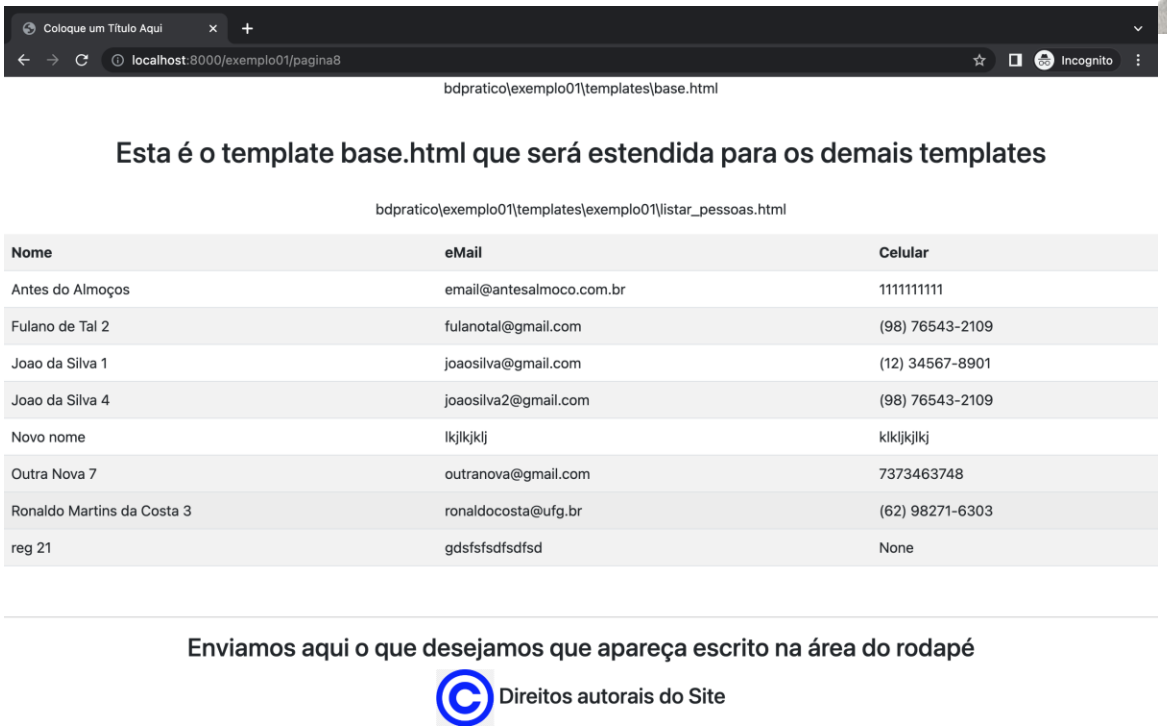


- Agora atualizamos o arquivo views.py

```
def pagina8(request):
    from .models import pessoa
    dicionario = {}
    registros = pessoa.objects.all()
    dicionario['pessoas'] = registros
    return render(request, 'exemplo01/listar_pessoas.html', dicionario)
```

Tags de Template do Django

- Se todas as alterações foram feitas corretamente, esta tela será apresentada



Permissões com django.contrib.auth

- Já vimos como utilizar a APP auth do Django para autenticação de usuários. Agora veremos como utilizar o sistema de permissões desta APP

- O Django trabalha essencialmente com as permissões:
  - can add
  - can change
  - can delete
  - can view

- Estas permissões podem ser atribuídas para usuários ou grupos para cada uma das tabelas do models.py de uma APP

Permissões com django.contrib.auth

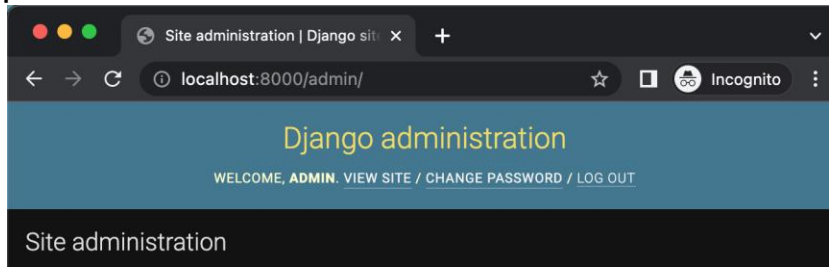
- Altere a view pessoa_list conforme segue

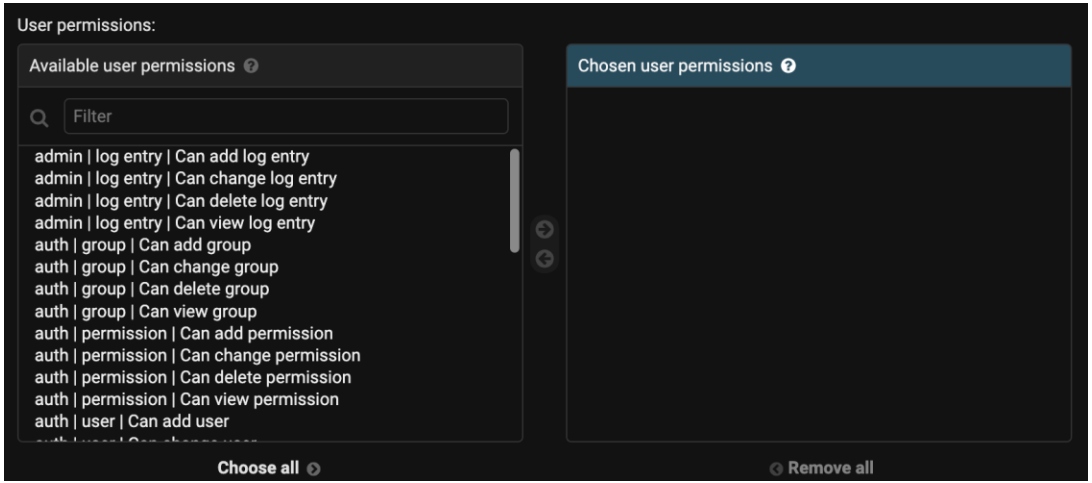
```
from django.views.generic import ListView
class pessoa_list(ListView):
    def dispatch(self, request, *args, **kwargs):
        if request.user.has_perm("exemplo01.view_pessoa"):
            return super().dispatch(request, *args, **kwargs)
        else:
            return HttpResponse("Sem permissão para listar pessoas")
    from .models import pessoa
    model = pessoa
```

- Esta é a maneira de validar as permissões para as ***Class-Based Views***

Permissões com django.contrib.auth

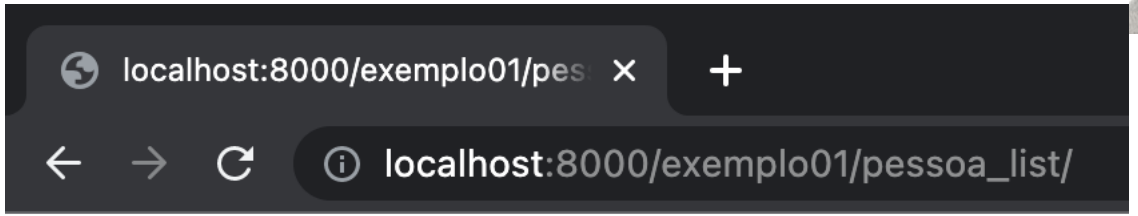
- Utilize a APP de administração do Django para ajustar as permissões





Tags de Template do Django

- Se os ajustes foram feitos corretamente você vai obter uma tela semelhante a esta



Sem permissão para listar pessoas

Permissões com django.contrib.auth

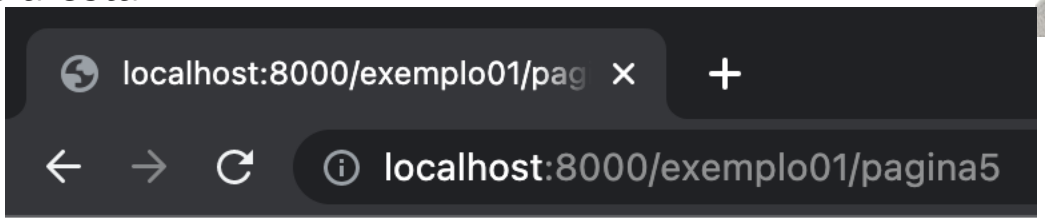
- Altere a view pagina5 conforme segue

```
def pagina5(request):
    if not(request.user.has_perm('exemplo01.add_pessoa')):
        return HttpResponse("Sem permissão para adicionar pessoas")
    xnome = request.POST.get('nome')
    xemail = request.POST.get('email')
    xcelular = request.POST.get('celular')
    xfuncao = request.POST.get('funcao')
    xnascimento = request.POST.get('nascimento')
    xativo = request.POST.get('ativo')
    print("Nome:", xnome)
    print("eMail:", xemail)
    print("Celular:", xcelular)
    print("Funcao:", xfuncao)
    print("Nascimento:", xnascimento)
    print("ativo:", xativo)
    if (xnome is not None):
        xativo = False
        if (xativo == 'on'):
            xativo = True
        pessoa.objects.create(nome=xnome, email=xemail, celular=xcelular,
                funcao=xfuncao, nascimento=xnascimento, ativo=xativo)
    return render(request, 'pagina5.html')
```

- Esta é a maneira de validar as permissões para outras views

Tags de Template do Django

- Se os ajustes foram feitos corretamente você vai obter uma tela semelhante a esta



Sem permissão para adicionar pessoas

Importando e Exportando Dados

- Para desenvolver as funcionalidade de importação e exportação de dados, utilizaremos a biblioteca Pandas

- Pandas é uma biblioteca para uso em Python, open-source e de uso gratuito (sob uma licença BSD), que fornece ferramentas para análise e manipulação de dados

- O pandas permite trabalhar com diferentes tipos de dados, por exemplo:
  - Dados tabulares, como uma planilha Excel ou uma tabela SQL;
  - Dados ordenados de modo temporal ou não;
  - Matrizes;
  - Qualquer outro conjunto de dados, que não necessariamente precisem estar rotulados;

Importando e Exportando Dados

- Instalando o Pandas

```
pip install pandas
```

Importando e Exportando Dados

- Vamos criar uma nova rota em nosso arquivo urls.py

```
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index_alias'),
    path('pagina0', views.pagina0, name='pagina0_alias'),
    path('pagina1', views.pagina1, name='pagina1_alias'),
    path('pagina2', views.pagina2, name='pagina2_alias'),
    path('pagina3', views.pagina3, name='pagina3_alias'),
    path('pagina4', views.pagina4, name='pagina4_alias'),
    path('pagina5', views.pagina5, name='pagina5_alias'),
    path('pagina6', views.pagina6, name='pagina6_alias'),
    path('pagina7', views.pagina7, name='pagina7_alias'),
    path('pagina8', views.pagina8, name='pagina8_alias'),
    path('pagina9', views.pagina9, name='pagina9_alias'),
    path('pessoa_menu', views.pessoa_menu.as_view(), name='pessoa_menu_alias'),
    path("pessoa_list/", views.pessoa_list.as_view(), name='pessoa_list_alias'),
    path("pessoa_create/", views.pessoa_create.as_view(), name='pessoa_create_alias'),
    path("pessoa_update/<int:pk>/", views.pessoa_update.as_view(), name='pessoa_update_alias'),
    path('pessoa_delete/<int:pk>/', views.pessoa_delete.as_view(), name='pessoa_delete_alias'),
]
```

Importando e Exportando Dados

- Criando a view pagina9

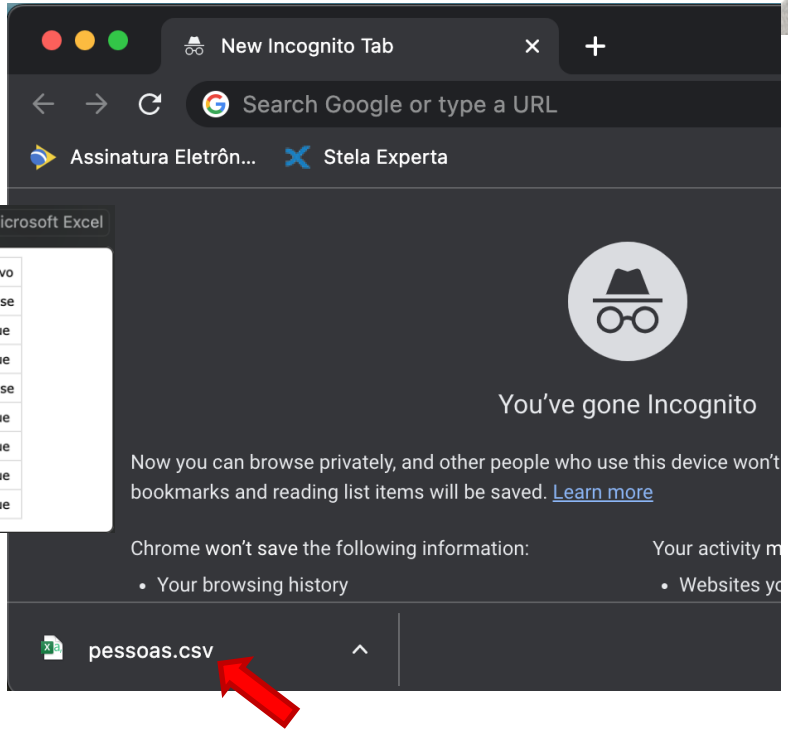
```
def pagina9(request):
    import pandas as pd
    from .models import pessoa
    eixo_y = []
    p = pessoa.objects.all()
    for _regs in p:
        eixo_x = []
        eixo_x.append(_regs.id)
        eixo_x.append(_regs.nome)
        eixo_x.append(_regs.email)
        eixo_x.append(_regs.celular)
        eixo_x.append(_regs.nascimento)
        eixo_x.append(_regs.ativo)
        eixo_y.append(eixo_x)
    _rotulos_colunas = []
    _rotulos_colunas.append("id")
    _rotulos_colunas.append("nome")
    _rotulos_colunas.append("email")
    _rotulos_colunas.append("celular")
    _rotulos_colunas.append("nascimento")
    _rotulos_colunas.append("ativo")
    df = pd.DataFrame(eixo_y, columns=_rotulos_colunas)
    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment; filename=pessoas.csv'
    df.to_csv(path_or_buf=response)
    return response
```

Importando e Exportando Dados

- Se os ajustes foram feitos corretamente você vai obter uma tela semelhante a esta

pessoas.csv — Open with Microsoft Excel

	id	nome	email	celular	nascimento	ativo
0	11	Antes do Almoços	email@antesalmoco.com.br	1111111111	1980-01-01	False
1	2	Fulano de Tal 2	fulanotal@gmail.com	(98) 76543-2109	1998-12-30	True
2	1	Joao da Silva 1	joaosilva@gmail.com	(12) 34567-8901	2002-05-06	True
3	4	Joao da Silva 4	joaosilva2@gmail.com	(98) 76543-2109	1975-07-09	False
4	25	Novo nome	lkjlkjklj	klkljkjlkj	2023-01-04	True
5	7	Outra Nova 7	outranova@gmail.com	7373463748	2023-01-23	True
6	3	Ronaldo Martins da Costa 3	ronaldocosta@ufg.br	(62) 98271-6303	1971-08-27	True
7	21	reg 21	gdsfsfsdfsdfsd		1980-01-01	True



Importando e Exportando Dados

- Vamos criar uma nova rota em nosso arquivo urls.py

```
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index_alias'),
    path('pagina0', views.pagina0, name='pagina0_alias'),
    path('pagina1', views.pagina1, name='pagina1_alias'),
    path('pagina2', views.pagina2, name='pagina2_alias'),
    path('pagina3', views.pagina3, name='pagina3_alias'),
    path('pagina4', views.pagina4, name='pagina4_alias'),
    path('pagina5', views.pagina5, name='pagina5_alias'),
    path('pagina6', views.pagina6, name='pagina6_alias'),
    path('pagina7', views.pagina7, name='pagina7_alias'),
    path('pagina8', views.pagina8, name='pagina8_alias'),
    path('pagina9', views.pagina9, name='pagina9_alias'),
    path('pagina10', views.pagina10, name='pagina10_alias'),
    path('pessoa_menu', views.pessoa_menu.as_view(), name='pessoa_menu_alias'),
    path("pessoa_list/", views.pessoa_list.as_view(), name='pessoa_list_alias'),
    path("pessoa_create/", views.pessoa_create.as_view(), name='pessoa_create_alias'),
    path("pessoa_update/<int:pk>/", views.pessoa_update.as_view(), name='pessoa_update_alias'),
    path('pessoa_delete/<int:pk>/', views.pessoa_delete.as_view(), name='pessoa_delete_alias'),
]
```

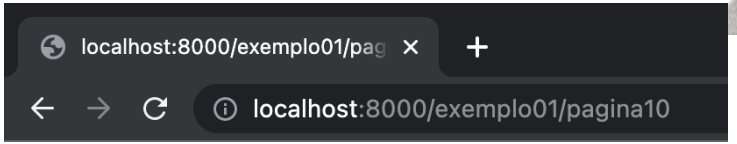
Importando e Exportando Dados

- Criando a view pagina10

```
def pagina10(request):
    import pandas as pd
    df=pd.read_csv('/Users/ronaldocosta/Downloads/pessoas.csv',sep=',')
    for linha, coluna in df.iterrows():
        print(linha, "ID:", coluna['id'])
        print(linha, "Nome:", coluna['nome'])
        print(linha, "eMail:", coluna['email'])
        print(linha, "Celular:", coluna['celular'])
        print(linha, "Nascimento:", coluna['nascimento'])
        print(linha, "Ativo:", coluna['ativo'])
    return HttpResponse("Arquivo Importado")
```

Importando e Exportando Dados

- Se os ajustes foram feitos corretamente você vai obter uma tela semelhante a esta




```
System check identified no issues (0 silenced).
February 01, 2023 - 14:55:03
Django version 4.1.5, using settings 'bdpratico.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
0 ID: 11
0 Nome: Antes do Almoços
0 eMail: email@antesalmoco.com.br
0 Celular: 1111111111
0 Nascimento: 1980-01-01
0 Ativo: False
1 ID: 2
1 Nome: Fulano de Tal 2
1 eMail: fulanotal@gmail.com
1 Celular: (98) 76543-2109
1 Nascimento: 1998-12-30
1 Ativo: True
2 ID: 1
2 Nome: Joao da Silva 1
```

Universidade Federal de Goiás
Instituto de Informática
Prof. Ronaldo Martins da Costa