

# PROGRAMAÇÃO BACK-END

**django**

The web framework for  
perfectionists with deadlines.

# PROGRAMAÇÃO BACK-END

Django  
Models

# PROGRAMAÇÃO BACK-END

Models

# PROGRAMAÇÃO BACK-END

Um **Model** (modelo) em Django é uma **tabela** em seu **banco** de dados.

Até agora, a saída foi vista como dados estáticos de modelos Python ou HTML.

Django permite trabalhar com dados, sem precisar alterar ou fazer upload de arquivos no processo.

No Django, os **dados** são criados em **objetos**, chamados **Modelos**, e na verdade são **tabelas** em um **banco** de dados.

# PROGRAMAÇÃO BACK-END

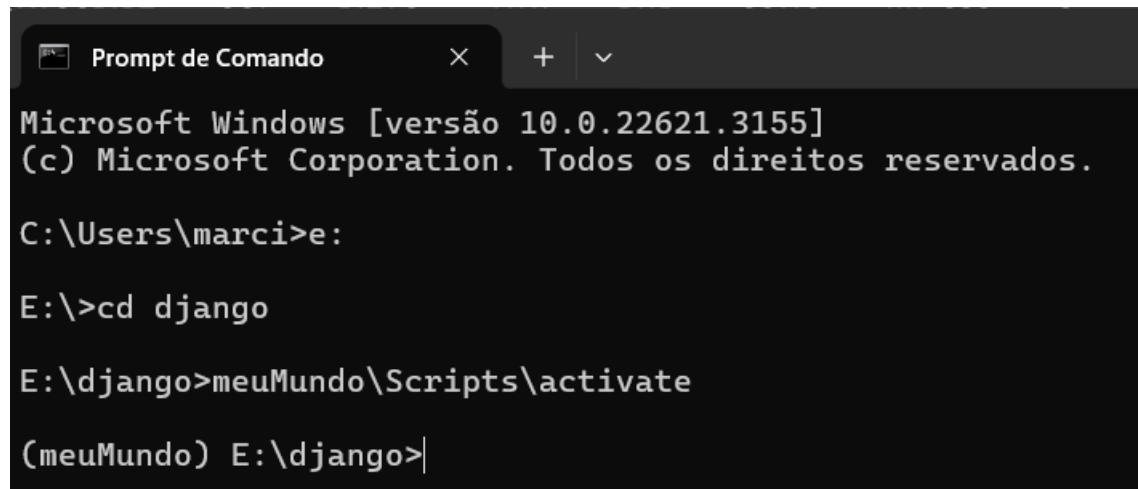
## Criando Tabela (Model)

Para criar um model, navegue até o arquivo **models.py** na pasta **/members/**.

Abra-o e adicione uma tabela **Member** criando um **classe** Member descreva os campos da tabela nele.

# PROGRAMAÇÃO BACK-END

**OBS: Não esqueça de ir na pasta e rodar o script activate.bat**



```
Prompt de Comando + ▾
Microsoft Windows [versão 10.0.22621.3155]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\marci>e:

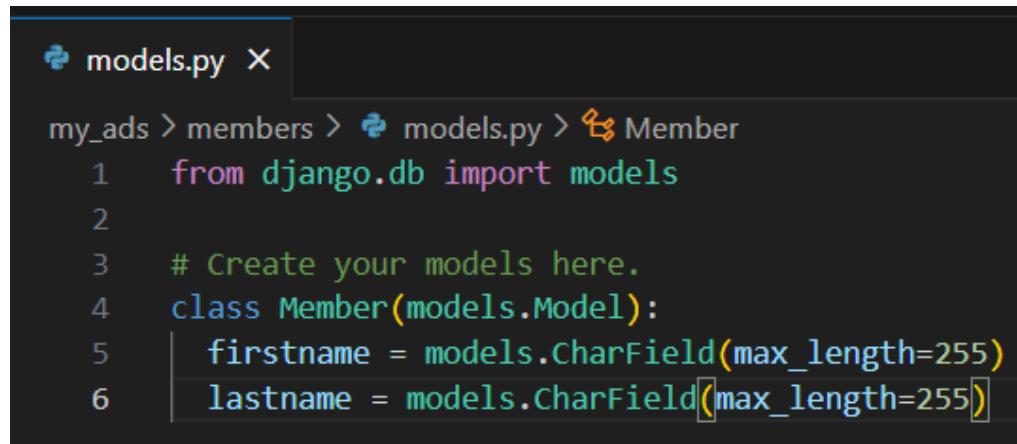
E:\>cd django

E:\django>meuMundo\Scripts\activate

(meuMundo) E:\django>
```

# PROGRAMAÇÃO BACK-END

Abra-o e adicione uma tabela **Member** criando um classe Member descreva os campos da tabela nele.



```
models.py
my_ads > members > models.py > Member
1 from django.db import models
2
3 # Create your models here.
4 class Member(models.Model):
5     firstname = models.CharField(max_length=255)
6     lastname = models.CharField(max_length=255)
```

- ✓ O primeiro campo, **firstname**, é um campo de Texto e conterá o primeiro nome dos membros.
- ✓ O segundo campo, **lastname**, também é um campo de Texto, com o sobrenome do membro.
- ✓ Ambos **firstname** e **lastname** estão configurados para ter no máximo 255 caracteres.

# PROGRAMAÇÃO BACK-END

## Banco de dados SQLite

Quando criamos o projeto Django, é criado um banco de dados **SQLite** vazio.

Ele foi criado na pasta **my\_ads** raiz e possui o nome de arquivo **db.sqlite3**.

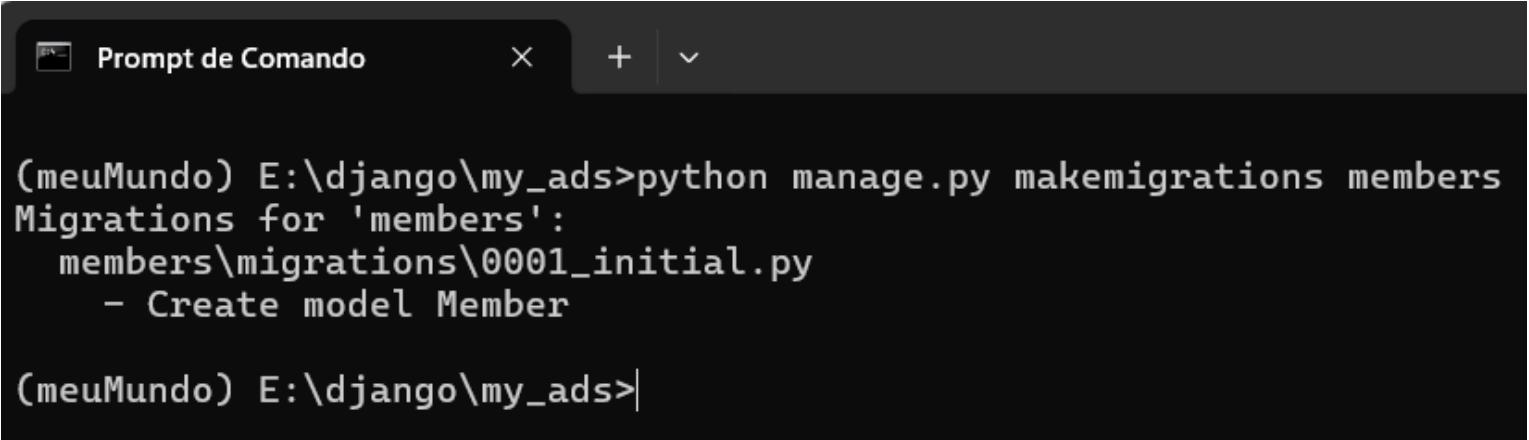
Por padrão, todos os **modelos** (models) criados no projeto Django (arquivo `models.py`) serão criados como **tabelas** neste banco de dados.

# PROGRAMAÇÃO BACK-END

## Migrar

Agora que descrevemos um modelo no **models.py** arquivo, devemos executar um comando para realmente **criar a tabela no banco de dados**. Navegue até a pasta **/my\_ads/** e execute este comando:

**python manage.py makemigrations members**

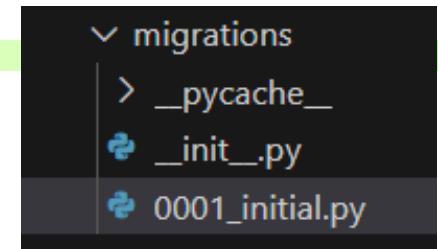


```
Prompt de Comando × + ▾
(meuMundo) E:\django\my_ads>python manage.py makemigrations members
Migrations for 'members':
  members\migrations\0001_initial.py
    - Create model Member

(meuMundo) E:\django\my_ads>
```

# PROGRAMAÇÃO BACK-END

**Django cria um arquivo descrevendo as alterações e armazena o arquivo na pasta /migrations:**



```
models.py  0001_initial.py x
my_ads > members > migrations > 0001_initial.py > ...
1 # Generated by Django 5.0.3 on 2024-03-13 17:41
2
3 from django.db import migrations, models
4
5
6 class Migration(migrations.Migration):
7
8     initial = True
9
10    dependencies = [
11        ]
12
13    operations = [
14        migrations.CreateModel(
15            name='Member',
16            fields=[
17                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
18                ('firstname', models.CharField(max_length=255)),
19                ('lastname', models.CharField(max_length=255)),
20            ],
21        ),
22    ]
23
```

# PROGRAMAÇÃO BACK-END

Observe que o Django insere um campo **id** para suas tabelas, que é um auto **increment number** (o primeiro registro recebe o valor 1, o segundo registro 2 etc.), esse é o comportamento padrão do Django, você pode substituí-lo descrevendo seu próprio campo id.

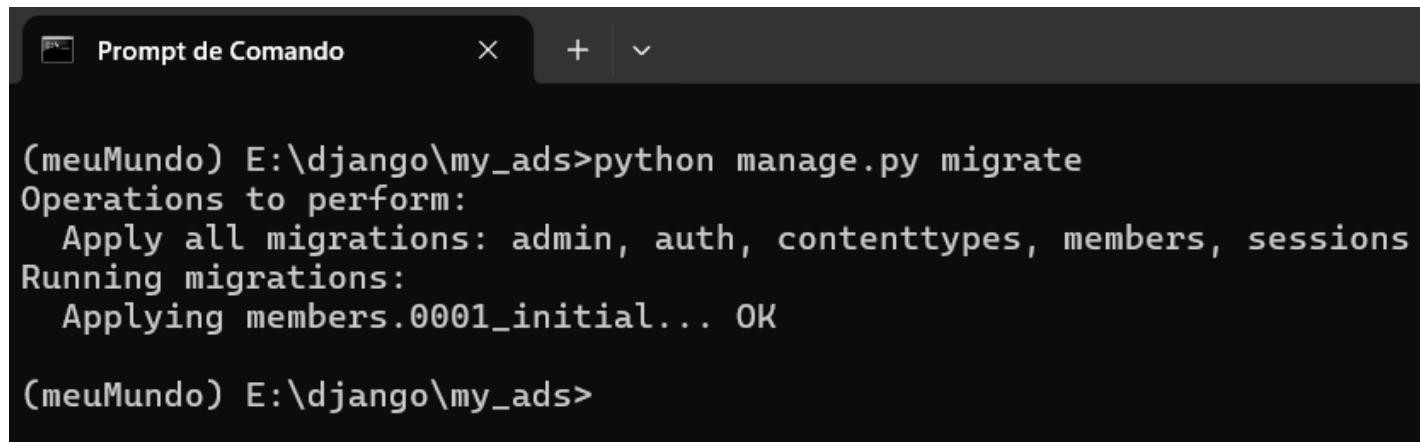
A **tabela ainda não foi criada**, você terá que executar mais um comando, então o Django criará e executará uma **instrução SQL**, baseada no conteúdo do novo arquivo na pasta **/migrations/**.

# PROGRAMAÇÃO BACK-END

**Execute o comando de migração:**

**python manage.py migrate**

**O que resultará nesta saída:**



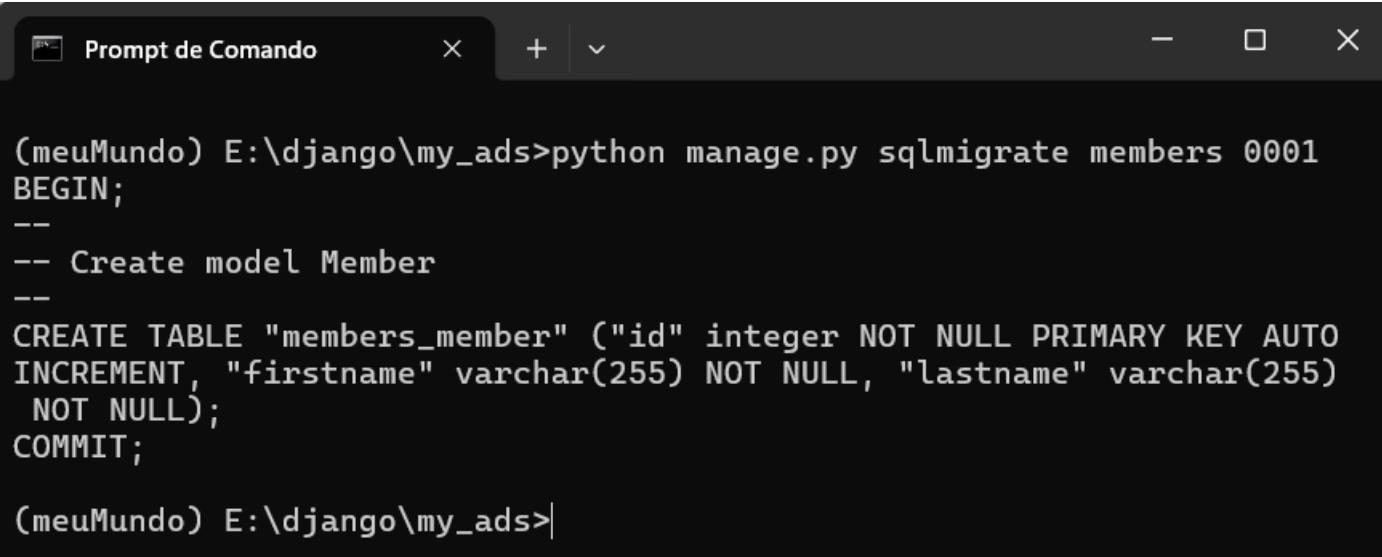
```
(meuMundo) E:\django\my_ads>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, members, sessions
Running migrations:
  Applying members.0001_initial... OK
(meuMundo) E:\django\my_ads>
```

**Agora você tem uma tabela Member em seu banco de dados!**

# PROGRAMAÇÃO BACK-END

## Ver o comando SQL

**Como observação: você pode visualizar a instrução SQL que foi executada na migração aplicada anteriormente. Tudo que você precisa fazer é executar este comando, com o número da migração: `python manage.py sqlmigrate members 0001`**



```
(meuMundo) E:\django\my_ads>python manage.py sqlmigrate members 0001
BEGIN;
--
-- Create model Member
--
CREATE TABLE "members_member" ("id" integer NOT NULL PRIMARY KEY AUTO
INCREMENT, "firstname" varchar(255) NOT NULL, "lastname" varchar(255)
NOT NULL);
COMMIT;

(meuMundo) E:\django\my_ads>
```

# PROGRAMAÇÃO BACK-END

Pode-se utilizar uma outra ferramenta para ver as tabelas, como exemplo DB Browser para SQLite versão 3.12.2.

members_member		
	id	integer "id" integer NOT NULL
	firstname	varchar(255) "firstname" varchar(255) NOT NULL
	lastname	varchar(255) "lastname" varchar(255) NOT NULL

# PROGRAMAÇÃO BACK-END

Inserindo Dados  
(Manualmente)

# PROGRAMAÇÃO BACK-END

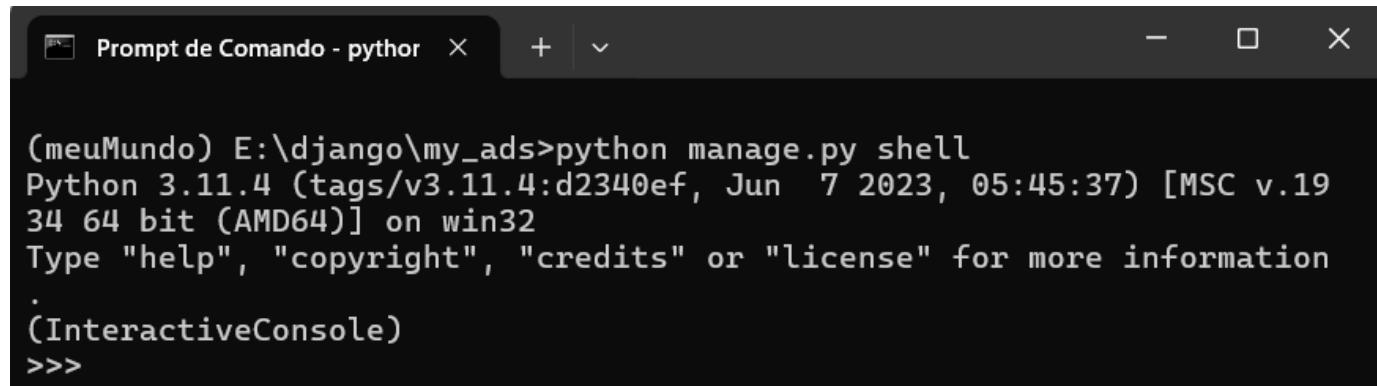
## Adicionar registros

A tabela Members criada anteriormente está vazia.

Usaremos o interpretador Python (Python Shell) para adicionar alguns membros a ele.

Para abrir um shell Python, digite este comando:

**python manage.py shell**



```
(meuMundo) E:\django\my_ads>python manage.py shell
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.19
34 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information
.
(InteractiveConsole)
>>>
```

# PROGRAMAÇÃO BACK-END

**Na parte inferior, após os três, >>> escreva o seguinte:**

**from members.models import Member**

**Pressione [enter] e escreva isto para ver a tabela Member vazia:**

**Member.objects.all()**

**Isso deve fornecer um objeto QuerySet vazio, como este:**

**QuerySet []**

```
.  
(InteractiveConsole)  
>>> from members.models import Member  
>>> Member.objects.all()  
<QuerySet []>  
>>> |
```

# PROGRAMAÇÃO BACK-END

**Adicione um registro à tabela, executando estas duas linhas:**

```
>>> member = Member(firstname='Emil', lastname='Refsnes')
>>> member.save()
```

**Execute este comando para ver se a tabela Member possui um membro:**

```
>>> Member.objects.all().values()
```

```
>>> member = Member(firstname='Emil', lastname='Refsnes')
>>> member.save()
>>> Member.objects.all().values()
<QuerySet [{<!--> 'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'}]>
>>> |
```

# PROGRAMAÇÃO BACK-END

## Adicionar vários registros

Pode-se adicionar vários registros criando uma lista de Member objetos e executar .save() em cada entrada:

```
>>> member1 = Member(firstname='Tobias', lastname='Refsnes')
>>> member2 = Member(firstname='Linus', lastname='Refsnes')
>>> member3 = Member(firstname='Lene', lastname='Refsnes')
>>> member4 = Member(firstname='Stale', lastname='Refsnes')
>>> member5 = Member(firstname='Jane', lastname='Doe')
>>> members_list = [member1, member2, member3, member4,
member5]
>>> for x in members_list:
>>>     x.save()
```

# PROGRAMAÇÃO BACK-END

```
>>> member1 = Member(firstname='Tobias', lastname='Refsnes')
>>> member2 = Member(firstname='Linus', lastname='Refsnes')
>>> member3 = Member(firstname='Lene', lastname='Refsnes')
>>> member4 = Member(firstname='Stale', lastname='Refsnes')
>>> member5 = Member(firstname='Jane', lastname='Doe')
>>> members_list = [member1, member2, member3, member4, member5]
>>> for x in members_list:
...     x.save()
...
```

**Agora existem 6 membros na tabela Member, execute o comando para visualizar:**

**>>> Member.objects.all().values()**

```
>>> Member.objects.all().values()
<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'}, {'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes'}, {'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes'}, {'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes'}, {'id': 5, 'firstname': 'Stale', 'lastname': 'Refsnes'}, {'id': 6, 'firstname': 'Jane', 'lastname': 'Doe'}]>
```

# PROGRAMAÇÃO BACK-END

Atualizando Dados  
(Manualmente)

# PROGRAMAÇÃO BACK-END

## Atualizar registros

Para atualizar registros que já estão no banco de dados, primeiro precisamos obter o registro que queremos atualizar:

```
>>> from members.models import Member  
>>> x = Member.objects.all()[4]
```

x agora representará o membro no índice 4, que é "Stale Refsnes", mas para ter certeza, vamos ver se está correto:

```
>>> x.firstname
```

# PROGRAMAÇÃO BACK-END

Isso deve lhe dar este resultado:

'Stale'

```
>>> from members.models import Member  
>>> x = Member.objects.all()[4]  
>>> x.firstname  
'Stale'  
>>> |
```

Agora podemos alterar os valores deste registro:

```
>>> x.firstname = "Stalikken"  
>>> x.save()
```

Execute este comando para ver se a tabela Member foi atualizada:

```
>>> Member.objects.all().values()
```

# PROGRAMAÇÃO BACK-END

**Resultado: (firstname de id = 5 foi alterado)**

```
>>> x.firstname = "Stalikken"
>>> x.save()
>>> Member.objects.all().values()
<QuerySet [{"id": 1, "firstname": "Emil", "lastname": "Refsnes"}, {"id": 2, "firstname": "Tobias", "lastname": "Refsnes"}, {"id": 3, "firstname": "Linus", "lastname": "Refsnes"}, {"id": 4, "firstname": "Lene", "lastname": "Refsnes"}, {"id": 5, "firstname": "Stalikken", "lastname": "Refsnes"}, {"id": 6, "firstname": "Jane", "lastname": "Doe"}]>
>>> |
```



# PROGRAMAÇÃO BACK-END

Deletando Dados  
(Manualmente)

# PROGRAMAÇÃO BACK-END

## Excluir registros

Para excluir um registro em uma tabela, comece obtendo o registro que deseja excluir:

```
>>> from members.models import Member  
>>> x = Member.objects.all()[5]
```

x agora representará o membro no índice 5, que é "Jane Doe", mas para ter certeza, vamos ver se está correto:

```
>>> x.firstname
```

Isso deve lhe dar este resultado:  
**'Jane'**

# PROGRAMAÇÃO BACK-END

podemos excluir o registro:

```
>>> x.delete()
```

O resultado será:

```
(1, {'members.Member': 1})
```

O que nos diz quantos itens foram excluídos e de qual modelo.

Se olharmos para o Modelo de Membro, podemos ver que 'Jane Doe' foi removido do Modelo:

```
>>> Member.objects.all().values()
```

# PROGRAMAÇÃO BACK-END

## Resultado: (Não tem o id = 6)

```
>>> x = Member.objects.all()[5]
>>> x.firstname
'Jane'
>>> x.delete()
(1, {'members.Member': 1})
>>> Member.objects.all().values()
<QuerySet [{"id": 1, "firstname": "Emil", "lastname": "Refsnes"}, {"id": 2, "firstname": "Tobias", "lastname": "Refsnes"}, {"id": 3, "firstname": "Linus", "lastname": "Refsnes"}, {"id": 4, "firstname": "Lene", "lastname": "Refsnes"}, {"id": 5, "firstname": "Stalikken", "lastname": "Refsnes"}]>
>>>
```

# PROGRAMAÇÃO BACK-END

Adicionar campos ao modelo

# PROGRAMAÇÃO BACK-END

## Adicionar campos no modelo

Para adicionar um campo a uma tabela depois de criada, abra o arquivo **models.py** e faça as alterações:

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

```
class Member(models.Model):  
    firstname = models.CharField(max_length=255)  
    lastname = models.CharField(max_length=255)  
    phone = models.IntegerField(null=True)  
    joined_date = models.DateField(null=True)
```

# PROGRAMAÇÃO BACK-END

O Conteúdo de `models.py` ficará assim:

```
models.py × 0001_initial.py  
my_ads > members > models.py > Member  
1 from django.db import models  
2  
3 # Create your models here.  
4 class Member(models.Model):  
5     firstname = models.CharField(max_length=255)  
6     lastname = models.CharField(max_length=255)  
7     phone = models.IntegerField(null=True)  
8     joined_date = models.DateField(null=True)
```

faça a migração mais uma vez:  
**python manage.py makemigrations members**  
Antes saia do shell do python

```
Use exit() or Ctrl-Z plus Return to exit  
>>> exit()  
(meuMundo) E:\django\my_ads>
```

# PROGRAMAÇÃO BACK-END

## O que resultará nisso:

```
(meuMundo) E:\django\my_ads>python manage.py makemigrations members
Migrations for 'members':
  members\migrations\0002_member_joined_date_member_phone.py
    - Add field joined_date to member
    - Add field phone to member

(meuMundo) E:\django\my_ads>
```

## Execute o comando de migração:

**python manage.py migrate**

## O que resultará nesta saída:

```
(meuMundo) E:\django\my_ads>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, members, sessions
Running migrations:
  Applying members.0002_member_joined_date_member_phone... OK
```

# PROGRAMAÇÃO BACK-END

Podemos inserir dados nos dois novos campos com a mesma abordagem que fizemos no capítulo Atualizar dados :

**Primeiro entramos no Python Shell:**

**python manage.py shell**

Depois digitamos:

```
>>> from members.models import Member
>>> x = Member.objects.all()[0]
>>> x.phone = 5551234
>>> x.joined_date = '2022-01-05'
>>> x.save()
```

# PROGRAMAÇÃO BACK-END

## Resultará

```
(meuMundo) E:\django\my_ads>python manage.py shell
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.19
34 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information
.
(InteractiveConsole)
>>> from members.models import Member
>>> x = Member.objects.all()[0]
>>> x.phone = 5551234
>>> x.joined_date = '2022-01-05'
>>> x.save()
```

# PROGRAMAÇÃO BACK-END

Execute este comando para ver se a tabela Member foi atualizada:

**>>> Member.objects.all().values()**

```
>>> Member.objects.all().values()
<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes', 'phone': 5551234, 'joined_date': datetime.date(2022, 1, 5)}, {'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes', 'phone': None, 'joined_date': None}, {'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes', 'phone': None, 'joined_date': None}, {'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes', 'phone': None, 'joined_date': None}, {'id': 5, 'firstname': 'Stalikken', 'lastname': 'Refsnes', 'phone': None, 'joined_date': None}]>
```

# PROGRAMAÇÃO BACK-END

Conexão no PostgreSQL

# PROGRAMAÇÃO BACK-END

## Conexão a Bancos

Para tornar o Django capaz de se conectar ao seu banco de dados, você deve especificá-lo na tupla **DATABASES** do arquivo **settings.py**.

```
74 # Database
75 # https://docs.djangoproject.com/en/5.0/ref/settings/#databases
76
77 DATABASES = {
78     'default': {
79         'ENGINE': 'django.db.backends.sqlite3',
80         'NAME': BASE_DIR / 'db.sqlite3',
81     }
82 }
```

# PROGRAMAÇÃO BACK-END

Para conexão ao servidor PostgreSQL, altere para:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'dvdrental',  
        'USER': 'postgres',  
        'PASSWORD': '12345678',  
        'HOST': '127.0.0.1',  
        'PORT': '5432'  
    }  
}
```

# PROGRAMAÇÃO BACK-END

Instale o pacote psycopg2:

**python -m pip install psycopg2**

```
(meuMundo) E:\django\my_ads>python -m pip install psycopg2
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.
com
Collecting psycopg2
  Downloading psycopg2-2.9.9-cp311-cp311-win_amd64.whl (1.2 MB)
    1.2/1.2 MB 8.2 MB/s eta 0:00:00
Installing collected packages: psycopg2
Successfully installed psycopg2-2.9.9

[notice] A new release of pip is available: 23.1.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

(meuMundo) E:\django\my_ads>
```

# PROGRAMAÇÃO BACK-END

**Na pasta my\_ads execute o comando migrate:**

**python manage.py migrate**

```
(meuMundo) E:\django\my_ads>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, members, 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 auth.0012_alter_user_first_name_max_length... OK
  Applying members.0001_initial... OK
  Applying members.0002_member_joined_date_member_phone... OK
  Applying sessions.0001_initial... OK
```

```
(meuMundo) E:\django\my_ads>
```