O comando **[migrate](https://docs.djangoproject.com/pt-br/3.2/ref/django-admin/" \l "django-admin-migrate)** verifica a configuração em [**INSTALLED\_APPS**](https://docs.djangoproject.com/pt-br/3.2/ref/settings/#std:setting-INSTALLED_APPS) e cria qualquer tabela do banco de dados necessária de acordo com as configurações do banco de dados no seu arquivo **mysite/settings.py** e as migrações que venham com a aplicação

Ao executar **makemigrations**, você está dizendo ao Django que você fez algumas mudanças em seus modelos (neste caso, você fez novas modificações) e que você gostaria que as alterações sejam armazenadas como uma migração.

As migrações são como o Django armazena as alterações nos seus modelos (e, portanto, seu esquema de banco de dados) - eles são apenas arquivos no disco.

Migrações são muito poderosas e permitem que você altere seus modelos ao longo do tempo, à medida que desenvolve o seu projeto, sem a necessidade de remover o seu banco de dados ou tabelas e criar de novo - ele é especializado em atualizar seu banco de dados ao vivo, sem perda de dados. Nós vamos cobri-los com mais profundidade em uma parte posterior do tutorial, mas para agora, lembre-se deste guia de três passos para fazer mudanças nos seus modelos:

* Mude seus modelos (em **models.py**).
* Rode **[python manage.py makemigrations](https://docs.djangoproject.com/pt-br/3.2/ref/django-admin/" \l "django-admin-makemigrations)** para criar migrações para suas modificações
* Rode **[python manage.py migrate](https://docs.djangoproject.com/pt-br/3.2/ref/django-admin/" \l "django-admin-migrate)** para aplicar suas modificações no banco de dados.

Uma view é um “tipo” de página Web em sua aplicação Django que em geral serve a uma função específica e tem um template específico.

Cada view é responsável por fazer uma das duas coisas: devolver um objeto **[HttpResponse](https://docs.djangoproject.com/pt-br/3.2/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** contendo o conteúdo para a página requisitada, ou levantar uma exceção como [**Http404**](https://docs.djangoproject.com/pt-br/3.2/topics/http/views/#django.http.Http404).

Sua view pode ler registros do banco de dados, ou não. Ela pode usar um sistema de templates como o do Django - ou outro sistema de templates Python de terceiros - ou não. Ele pode gerar um arquivo PDF, saída em um XML, criar um arquivo ZIP sob demanda, qualquer coisa que você quiser,usando qualquer biblioteca Python você quiser.

Tudo que o Django espera é que a view devolva um **[HttpResponse](https://docs.djangoproject.com/pt-br/3.2/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)**. Ou uma exceção.

Porque é conveniente, vamos usar a própria API de banco de dados do Django, a qual cobrimos em [Tutorial 2](https://docs.djangoproject.com/pt-br/3.2/intro/tutorial02/). Aqui uma nova tentativa de view **index()**, a qual mostra as últimas 5 “poll questions” do sistema, separada por vírgulas, de acordo com sua data de publicação:

[Parte 1: Requisições e respostas](https://docs.djangoproject.com/pt-br/3.2/intro/tutorial01/) | ok

[Parte 2: Modelos e o site Admin](https://docs.djangoproject.com/pt-br/3.2/intro/tutorial02/) | ok

[Parte 3: Views e templates](https://docs.djangoproject.com/pt-br/3.2/intro/tutorial03/) | ok

[Parte 4: Forms e views genéricas](https://docs.djangoproject.com/pt-br/3.2/intro/tutorial04/) | ok

[Parte 5: Testando](https://docs.djangoproject.com/pt-br/3.2/intro/tutorial05/) | ok

[Parte 6: Arquivos estáticos](https://docs.djangoproject.com/pt-br/3.2/intro/tutorial06/) |

[Parte 7: Personalizando o site admin](https://docs.djangoproject.com/pt-br/3.2/intro/tutorial07/)