Django

Documentação

Os argumentos a seguir estão disponíveis para todos os tipos de campo. Todos são opcionais.

`default`

`Field.default`

O valor padrão para o campo. Isso pode ser um valor ou um objeto que pode ser chamado. Se chamável, será chamado toda vez que um novo objeto for criado.

O padrão não pode ser um objeto mutável (instância de modelo, **`list`**, **`set`**, etc.), pois uma referência à mesma instância desse objeto seria usada como valor padrão em todas as novas instâncias de modelo. Em vez disso, envolva o padrão desejado em um callable. Por exemplo, se você deseja especificar um padrão **`dict`** para **`JSONField`**, use uma função:

```
def contact_default():
    return {"email": "to1@example.com"}

contact_info = JSONField("ContactInfo", default=contact_default)
```

`lambda`s não podem ser usados para opções de campo como **`default`** porque não podem ser serializados por migrations . Veja essa documentação para outras ressalvas.

Para campos como **`ForeignKey`** esse mapear para instâncias de modelo, os padrões devem ser o valor do campo que eles referenciam (**`pk`** a menos que **`to_field`** seja definido) em vez de instâncias de modelo.

O valor padrão é usado quando novas instâncias de modelo são criadas e um valor não é fornecido para o campo. Quando o campo é uma chave primária, o padrão também é usado quando o campo é definido como **`None`**.

`primary_key`

`Field.primary_key`

Se **`True`**, este campo é a chave primária para o modelo.

Se você não especificar **`primary_key=True`** nenhum campo em seu modelo, o Django adicionará automaticamente um campo para manter a chave primária, então você não precisa definir **`primary_key=True`** nenhum de seus campos, a menos que queira substituir o comportamento padrão da chave primária. O tipo de campos de chave primária criados automaticamente pode ser

especificado por aplicativo **`AppConfig.default_auto_field`** ou globalmente na **`DEFAULT_AUTO_FIELD`** configuração. Para saber mais, consulte Campos de chave primária automáticos .

`primary_key=True` implica **`null=False`** e **`unique=True`**. Apenas uma chave primária é permitida em um objeto.

O campo de chave primária é somente leitura. Se você alterar o valor da chave primária em um objeto existente e salvá-lo, um novo objeto será criado ao lado do antigo.

`unique`

`Field.unique`

Se **`True`**, este campo deve ser único em toda a tabela.

Isso é aplicado no nível do banco de dados e pela validação do modelo. Se você tentar salvar um modelo com um valor duplicado em um **`unique`** campo, um **`django.db.IntegrityError`** será gerado pelo **`save()`** método do modelo.

Esta opção é válida em todos os tipos de campo, exceto **`ManyToManyField`** e **`OneToOneField`**.

Observe que quando **`unique`** é **`True`**, não é necessário especificar **`db_index`**, pois **`unique`** implica na criação de um índice.

`verbose_name`

`Field.verbose_name`

Um nome legível por humanos para o campo. Se o nome detalhado não for fornecido, o Django irá criá-lo automaticamente usando o nome do atributo do campo, convertendo sublinhados em espaços. Consulte Nomes de campo detalhados .

Tipos de campo

`AutoField`

`classe` `AutoField(` *`** opções`*`)`

Um **`IntegerField`** que incrementa automaticamente de acordo com os IDs disponíveis. Normalmente, você não precisará usar isso diretamente; um campo de chave primária será adicionado automaticamente ao seu modelo se você não especificar o contrário. Consulte Campos de chave primária automáticos .

`CharField`

`class` `CharField(` *`max_length = None, ** opções`*`)`

Um campo de string, para strings de tamanho pequeno a grande.

Para grandes quantidades de texto, use **`TextField`**.

O widget de formulário padrão para este campo é um arquivo **`TextInput`**.

`IntegerField`

`classe` **`IntegerField`**`(` ***`** opções`*** `)`

Um número inteiro. Valores de **`-2147483648`** a **`2147483647`** são seguros em todos os bancos de dados suportados pelo Django.

Ele usa **`MinValueValidator`** e **`MaxValueValidator`** para validar a entrada com base nos valores que o banco de dados padrão suporta.

O widget de formulário padrão para este campo é **`NumberInput`** quando **`localize`** é **`False`** ou **`TextInput`** não.

`TextField`

`classe` **`TextField`**`(`***`**opções`***`)`

Um grande campo de texto. O widget de formulário padrão para este campo é um arquivo **`Textarea`**.

Se você especificar um **`max_length`** atributo, ele será refletido no **`Textarea`** widget do campo de formulário gerado automaticamente. No entanto, não é aplicado no nível do modelo ou do banco de dados. Use um **`CharField`** para isso.

Campos de relacionamento

Django também define um conjunto de campos que representam relações.

`ForeignKey`

`classe` **`ForeignKey`**`(to,` ***`on_delete`***`,` ***`**opções`***`)`

Uma relação muitos-para-um. Requer dois argumentos posicionais: a classe à qual o modelo está relacionado e a **`on_delete`** opção.

Para criar um relacionamento recursivo – um objeto que tem um relacionamento muitos-para-um consigo mesmo – use .**`models.ForeignKey('self',on_delete=models.CASCADE)`**

Se você precisar criar um relacionamento em um modelo que ainda não foi definido, poderá usar o nome do modelo, em vez do próprio objeto do modelo:

```
from django.db import models

class Car(models.Model):
    manufacturer = models.ForeignKey(
        'Manufacturer',
        on_delete=models.CASCADE,
    )
    # ...

class Manufacturer(models.Model):
    # ...
    pass
```