Python | Relational fields in Django models

Last Updated : 07 Jul, 2019

Prerequisite: Django models

Django models represent real-world entities, and it is rarely the case that real-world entities are entirely independent of each other. Hence Django supports relational databases and allows us to establish relations between different models. There are three types of relational fields which Django supports: many-to-one, many-to-many and one-to-one.

Many-to-one fields:

This is used when one record of a model A is related to multiple records of another model B. For example – a model **Song** has many-to-one relationship with a model **Album**, i.e. an album can have many songs, but one song cannot be part of multiple albums. Many-to-one relations are defined using `ForeignKey` field of `django.db.models`.

Below is an example to demonstrate the same.

```
from django.db import models

class Album(models.Model):
    title = models.CharField(max_length = 100)
    artist = models.CharField(max_length = 100)

class Song(models.Model):
    title = models.CharField(max_length = 100)
    album = models.ForeignKey(Album, on_delete = models.CASCADE)
```

It is a good practice to name the many-to-one field with the same name as the related model, lowercase.

Many-to-many fields:

This is used when one record of a model A is related to multiple records of another model B and vice versa. For example – a model **Book** has many-to-many relationship with a model **Author**, i.e. an book can be written by multiple authors and an author can write multiple books. Many-to-many relations are defined using `ManyToManyField` field of `django.db.models`.

Below is an example to demonstrate the same.

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length = 100)
    desc = models.TextField(max_length = 300)

class Book(models.Model):
    title = models.CharField(max_length = 100)
    desc = models.TextField(max_length = 300)
    authors = models.ManyToManyField(Author)
```

It is a good practice to name the many-to-many field with the plural version of the related model, lowercase. It doesn't matter which of the two models contain the many-to-many field, but it shouldn't be put in both the models.

One-to-one fields:

This is used when one record of a model A is related to exactly one record of another model B. This field can be useful as a primary key of an object if that object extends another object in some way. For example – a model **Car** has one-to-one relationship with a model **Vehicle**, i.e. a car is a vehicle. One-to-one relations are defined using `OneToOneField` field of `django.db.models`.

Below is an example to demonstrate the same.

```
from django.db import models

class Vehicle(models.Model):
    reg_no = models.IntegerField()
    owner = models.CharField(max_length = 100)

class Car(models.Model):
    vehicle = models.OneToOneField(Vehicle,
          on_delete = models.CASCADE, primary_key = True)
    car_model = models.CharField(max_length = 100)
```

It is a good practice to name the one-to-one field with the same name as that of the related model, lowercase.

Data integrity options:

Since we are creating models which depend on other models, we need to define the behavior of a record in one model when the corresponding record in the other is deleted. This is achieved by adding an optional `on_delete` parameter in the relational field, which can take the following values:

- `on_delete = models.CASCADE` – This is the default value. It automatically deletes all the related records when a record is deleted.(e.g. when an Album record is deleted all the Song records related to it will be deleted)
- `on_delete = models.PROTECT` – It blocks the deletion of a record having relation with other records.(e.g. any attempt to delete an Album record will be blocked)
- `on_delete = models.SET_NULL` – It assigns NULL to the relational field when a record is deleted, provided `null = True` is set.
- `on_delete = models.SET_DEFAULT` – It assigns default values to the relational field when a record is deleted, a default value has to be provided.
- `on_delete = models.SET()` – It can either take a default value as parameter, or a callable, the return value of which will be assigned to the field.
- `on_delete = models.DO_NOTHING` – Takes no action. Its a bad practice to use this value.

A Abhis...

Previous Article

unique=True - Django Built-in Field Validation

Next Article

Similar Reads

How to Add created_at and updated_at Fields to All Django Models Using...

In Django applications, tracking when records are created and last updated in the database is common. Two of the most frequently used fields for this purpose are...

4 min read

Difference between Relational Algebra and Relational Calculus

Relational Algebra and Relational Calculus are two preliminarily important topics in the study of DBMS. Both are standardized query languages used for querying...

6 min read

Difference between Tuple Relational Calculus (TRC) and Domain Relational...

When you work on Relational Calculus which is the main concept in Database Theory, you will find two kinds of variations that play an important role. These...

5 min read

Intermediate fields in Django | Python

Prerequisite: Django models, Relational fields in DjangoIn Django, a many-to-many relationship exists between two models A and B, when one instance of A ...

2 min read

Django model data types and fields list

The most important part of a model and the only required part of a model is the list of database fields it defines. Fields are specified by class attributes. Be caref...

4 min read

Render Django Form Fields Manually

Django form fields have several built-in methods to ease the work of the developer but sometimes one needs to implement things manually for...

5 min read

Boolean Fields in Serializers - Django REST Framework

In Django REST Framework the very concept of Serializing is to convert DB data to a datatype that can be used by javascript. Every serializer comes with some...

4 min read

String Fields in Serializers - Django REST Framework

In Django REST Framework the very concept of Serializing is to convert DB data to a datatype that can be used by javascript. Every serializer comes with some...

5 min read

Core arguments in serializer fields - Django REST Framework

Serializer fields in Django are same as Django Form fields and Django model fields and thus require certain arguments to manipulate the behaviour of those...

6 min read

URL fields in serializers - Django REST Framework

In Django REST Framework the very concept of Serializing is to convert DB data to a datatype that can be used by javascript. Every serializer comes with some...

5 min read

Article Tags : DBMS Python Web Technologies Python Django

Practice Tags : python