django

Wednesday, February 9, 2022 12:36 AM

Commands	output
section.1.Django Fundamentals 1. Introduction 2. Start Server 3. Django default apps 4. Create an app 5. Registering apps 6. view(works like request handler) 7. Mapping URLs to views 8. Django-templates **section.2.Building a Data Mode** 1. Data Model 2. Creating Models: 3. choice fields 4. One-to-one relationship 5. on_delete 6. One-to-many Relationship 7. One-to-many relationship of entire data model: 8. Many-to-Many relationship: 9. Resolving Circular Relationships(circular dependency): 10. Generic Relationships	
Introduction storefront is our project and playground is our app. django-admin startproject <project-name> .	creates project files at the current location storefront ├ asgi.py ├ settings.py ├ urls.py ├ wsgi.py └ __init__.py
Start Server python manage.py runserver # manage.py knows our settings	start server
Django default apps 'django.contrib.admin', #this app provides the admin interface for managing our data 'django.contrib.auth',#this is used for authentication of users 'django.contrib.contenttypes',# # 'django.contrib.sessions', #this is used for managing data on server it is notused these days. 'django.contrib.messages',#displaying one time notification to users. 'django.contrib.staticfiles',# this provides static files to user like images.	
Create an app python manage.py startapp <playground> #this creates app	
Registering apps Everytime you create an app register that app in the settings/INSTALLED_APPS list. INSTALLED_APPS = ['django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', # 'django.contrib.sessions', 'django.contrib.messages', 'django.contrib.staticfiles', **'playground'**]	
view(works like request handler) It takes a **request** and returns a **response**.It is also called request handler.	
Mapping URLs to views (views.py) from django.shortcuts import render from django.http import HttpResponse # Create your views here. def say_hello(requests): return HttpResponse('Hello World!') (urls.py)>app from django.urls import path from . import views	http://127.0.0.1:8000/playground/hello/ Hello World!

```
#URLConf
urlpatterns = [
    path('hello/', views.say_hello)
]

(urls.py)>project
"""storefront URL Configuration
The `urlpatterns` list routes URLs to views. For more information please
see:
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include,
path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path,include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('playground/',include('playground.urls'))
]
```

Django-templates

View in other frameworks is called template in django.

creates new folder in ./playground/templates

(views.py)

```
from urllib import request
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
def say_hello(request):
    return render(request, 'hello.html',{'name':'Django'})
```

(hello.html)

```
<h1>Hello {{name}}</h1>
```

```
playground
├migrations
|├__pycache__
||└__init__.cpython-37.pyc
|└__init__.py
├templates
|└hello.html
├__pycache__
|├admin.cpython-37.pyc
|├apps.cpython-37.pyc
|├models.cpython-37.pyc
|├urls.cpython-37.pyc
|├views.cpython-37.pyc
|└__init__.cpython-37.pyc
├admin.py
├apps.py
├models.py
├tests.py
├urls.py
├views.py
└__init__.py
```

section.2.Building a Data Mode

Data Model:
Tables joined together with foreign keys. Here tables are called entities.

Design your own data models according to your need with one-to-one,one-to-many,may-to-many relations.

Avoid Monolith app:
Don’t put all models in a single app rather make multiple apps and separate them.

Avoid coupling(minimal coupling or high cohesion/focus):
There should be minimum coupling between tables.And each app should handle one functionality properly.

Creating Models:
Django Field types:

- CharField
- TextField
- DecimalField
- IntegerField
- DateTimeField #give the exact time.
- DateField #gives the day not the exact time.
- EmailField
- PositiveSmallIntegerField

```
from django.db import models

#creating product model
class Product(models.Model):
    title=models.CharField(max_length=255)
    description=models.TextField()
    price=models.DecimalField(max_digits=6,decimal_places=2)
    inventory=models.IntegerField()
    last_update=models.DateTimeField(auto_now=True)
```

"auto_now_add":
will set time when an instance is created

"auto_now":
will set time when someone modified his feedback.

By default django automatically creates an **"id" field which is primary key for that table.** If you want to set your own primary key then define your column and make it primary key using **"primary_key=True"**

choice fields
This field type is used to set the **"possible input values"** for a field.
e.g:The membership can be gold, silver and bronze.

```
MEMBERSHIP_BRONZE='B'
MEMBERSHIP_SILVER='S'
MEMBERSHIP_GOLD='G'

MEMBERSHIP_CHOICE=[
    ("B","Bronze"),
    ("S","Silver"),
    ("G","Gold")
]

# this field only takes 3 values either 'B','S' or 'G'

membership=models.CharField(max_length=1,
choices=MEMBERSHIP_CHOICE,default=MEMBERSHIP_BRONZE)
```

One-to-one relationship

You only have to create relationship in one class django automatically create reverse relationship in other class.

Order of parent and child class sequence matter in code. Parent classes should be on top and child classes should be below parent classes.
But sometimes you cant follow this so **pass models name as string then**.e.g

```
    collection = models.ForeignKey('Collection', on_delete=models.PROTECT)

#relationship of customer with adresses.
ass Address(models.Model):
    street=models.CharField(max_length=255)
    city=models.CharField(max_length=255)
    customer=models.OneToOneField(Customer,on_delete=models.CASCADE,primar
y_key=True)
```

on_delete

```
on_delete=models.PROTECT # prevent deleting the child table fields
on_delete=models.CASCADE # delete all relating fields in other tables
on_delete=models.SET_NULL # set the relating fields to null.
on_delete=models.SET_DEFAULT
```

One-to-many Relationship

```
# customer(1)-----------Adresses(*)
A customer can have many addresses.

class Customer(models.Model):
    MEMBERSHIP_BRONZE = 'B'
    MEMBERSHIP_SILVER = 'S'
    MEMBERSHIP_GOLD = 'G'
    MEMBERSHIP_CHOICE = [
        (MEMBERSHIP_BRONZE, "Bronze"),
        (MEMBERSHIP_SILVER, "Silver"),
        (MEMBERSHIP_GOLD, "Gold")
    ]
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    email = models.EmailField(unique=True)
    phone = models.CharField(max_length=255)
    birth_date = models.DateField(null=True)
    # this field only takes 3 values either 'B','S' or 'G'
    membership = models.CharField(
        max_length=1, choices=MEMBERSHIP_CHOICE,

    default=MEMBERSHIP_BRONZE)

class Address(models.Model):
    street = models.CharField(max_length=255)
    city = models.CharField(max_length=255)

    #a customer can have multiple addresses
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
```

One-to-many relationship of entire data model:

```
class Collection(models.Model):
    title = models.CharField(max_length=255)

class Product(models.Model):
    title = models.CharField(max_length=255)
    description = models.TextField()
```

```
    price = models.DecimalField(max_digits=6, decimal_places=2)
    inventory = models.IntegerField()
    last_update = models.DateTimeField(auto_now=True)
    collection = models.ForeignKey(Collection, on_delete=models.PROTECT)

class Customer(models.Model):
    MEMBERSHIP_BRONZE = 'B'
    MEMBERSHIP_SILVER = 'S'
    MEMBERSHIP_GOLD = 'G'
    MEMBERSHIP_CHOICE = [
        (MEMBERSHIP_BRONZE, "Bronze"),
        (MEMBERSHIP_SILVER, "Silver"),
        (MEMBERSHIP_GOLD, "Gold")
    ]
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    email = models.EmailField(unique=True)
    phone = models.CharField(max_length=255)
    birth_date = models.DateField(null=True)
    # this field only takes 3 values either 'B','S' or 'G'
    membership = models.CharField(
        max_length=1, choices=MEMBERSHIP_CHOICE,
default=MEMBERSHIP_BRONZE)

class Order(models.Model):
    PAYMENT_STATUS_PENDING = "P"
    PAYMENT_STATUS_COMPLETED = "C"
    PAYMENT_STATUS_FAILED = "F"
    PAYMENT_STATUS_CHOICE = [
        (PAYMENT_STATUS_PENDING, "Pending"),
        (PAYMENT_STATUS_COMPLETED, "Completed"),
        (PAYMENT_STATUS_FAILED, "Failed")
    ]
    placed_at = models.DateTimeField(auto_now_add=True)
    payment_status = models.CharField(
        max_length=1, choices=PAYMENT_STATUS_CHOICE,
default=PAYMENT_STATUS_PENDING)
    customer = models.ForeignKey(Customer, on_delete=models.PROTECT)

class OrderItem(models.Model):
    order = models.ForeignKey(Order, on_delete=models.PROTECT)
    product = models.ForeignKey(Product, on_delete=models.PROTECT)
    quantity = models.PositiveSmallIntegerField()
    unit_price = models.DecimalField(max_digits=6, decimal_places=2)

class Address(models.Model):
    street = models.CharField(max_length=255)
    city = models.CharField(max_length=255)
    # here customer is a foreign key
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)

class Cart(models.Model):
    created_at = models.DateTimeField(auto_now_add=True)

class CartItem(models.Model):
    cart = models.ForeignKey(Cart, on_delete=models.CASCADE)
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    quantity = models.PositiveSmallIntegerField()
```

Many-to-Many relationship:

promotion(*)----------------products(*)

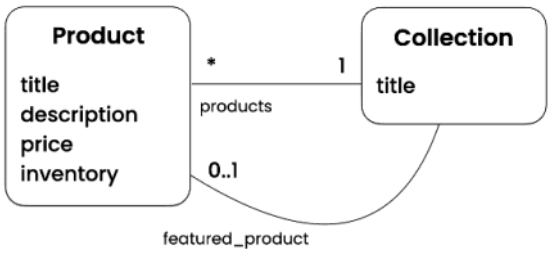
```
class Promotion(models.Model):
    description = models.CharField(max_length=255)
    discount = models.FloatField()

class Product(models.Model):
    title = models.CharField(max_length=255)
    description = models.TextField()
    price = models.DecimalField(max_digits=6, decimal_places=2)
    inventory = models.IntegerField()
    last_update = models.DateTimeField(auto_now=True)
    collection = models.ForeignKey(Collection, on_delete=models.PROTECT)

    promotions=models.ManyToManyField(Promotion)
```

Resolving Circular Relationships(circular dependency):

When two classes depend on each other. Circular dependency normally happens in this relation.
e.g:
like products depend on collection class and collection depend on products class.

Circular dependency happens here when we django creates a relation in the collection class it automatically creates reverse relation in the product class but we've already created a collection column/field.That's why our name clashes with the one django automatically created.

```
class Collection(models.Model):
    # title = models.CharField(max_length=255)
```



```
    featured_product=models.ForeignKey(Product,on_delete=models.SET_NULL)
class Product(models.Model):
    # title = models.CharField(max_length=255)
    # description = models.TextField()
    # price = models.DecimalField(max_digits=6, decimal_places=2)
    # inventory = models.IntegerField()
    # last_update = models.DateTimeField(auto_now=True)
    collection = models.ForeignKey(Collection, on_delete=models.PROTECT)
    # promotions=models.ManyToManyField(Promotion)
```

Solution:
So if we don't need django reverse relation we can ignore it using **related_name='+'** and also pass the name of table/model in strings.

```
class Collection(models.Model):
    # title = models.CharField(max_length=255)
    featured_product = models.ForeignKey(
        'Product', on_delete=models.SET_NULL, null=True, related_name='+')

class Product(models.Model):
    # title = models.CharField(max_length=255)
    # description = models.TextField()
    # price = models.DecimalField(max_digits=6, decimal_places=2)
    # inventory = models.IntegerField()
    # last_update = models.DateTimeField(auto_now=True)
    collection = models.ForeignKey(Collection, on_delete=models.PROTECT)
    # promotions=models.ManyToManyField(Promotion)
```

CIRCULAR DEPENDENCY

Generic Relationships

We can use generic relationship anywhere, so we design our own app having this general functionality.
For example tags app is created and we can use its ability to tag anywhere.
Django provide **"ContentType" model** that is specifically build to specify generic relations.

For this we need 3 things:
- content_type(to find the table)
- object_id(to find the row in the table)
- content_object(to find the actual object)

```
from django.contrib.contenttypes.models import ContentType
from django.contrib.contenttypes.fields import GenericForeignKey
# Create your models here.

class Tag(models.Model):
    label = models.CharField(max_length=255)

class TaggedItem(models.Model):
    # what tag is applied to what object
    tag = models.ForeignKey(Tag, on_delete=models.CASCADE)

    # Type (product,video,article)-using this we can find the table
    # ID - using this we can find the row/record
     #ContentType is a model just like our own models/tables
    content_type=models.ForeignKey(ContentType,
      on_delete=models.CASCADE)

    object_id=models.PositiveIntegerField()
    #now to get the actual product which is tagged
    content_object=GenericForeignKey()
```