Merhaba arkadaşlar bu videoda model kalıtımını öğreneceğiz.

--------------- model kalıtımı ---------------

Django’da üç farklı şekikde modeller türetilebilir.

* Alt modeller için yazmak istemediğimiz bilgileri tutmak için ana sınıfı kullanmak. Bu tarz kalıtımlarda ana sınıf için veritabanında tablo oluşturulmaz. (model kalıtımı)
* Hiyerarşideki her modelin veritabanında tabloya sahip olduğu yapılar. Bu tarz kalıtımlarda ana model ile alt modeller arasında bire-bir ilişki kurulur. (çoklu tablo kalıtımı)
* Modelin vaysayılan yöneticisi gibi özelliklerini değiştirmek veya yeni yöntem eklemek için kullanılan yapılar. (vekil model)

Bir modeli ana sınıf gibi kullanıp sadece özelliklerini alt sınıflarına aktarmak istiyorsak o modelin meta ayarlarında sınıfı abstract olarak tanımlamalıyız. Bu sayede o model için veritabanında tablo oluşturulmaz.

Öncelikle yeni bir model oluşturalım. Modelimizin ismi CompanyGroup olsun. Bunun için model klasörünün altında bir adet learning\_inheritance.py dosyası oluşturuyorum. Daha önce söylediğim gibi modellerimiz Model sınıfndan türetilmek zorundadır.

from django.db import models  
  
  
class Company (models.Model):  
 pass

Modelimizi \_\_init\_\_.py dosyasından tanıtalım.

from .learning\_inheritance import Company

Bu modelimizin bizim ana sınıfımız olsun. Bunu tanıtmak için Meta ayarlarına abstrat=True eklemeliyiz. Bunu ekleyerek Model sınıfına bunun bir ana sınıf olduğunu bunun için veritabanında tablo oluşturmaması gerektiğini belirtmiş olduk.

class Meta:  
 abstract = True

Şimdi alanlarımızı oluşturalım. Şirketlerin isimleri olsun. CharField olarak tanımlayalım 200 karakter sınırı koyalım. Firma bilgileri için content alanı oluşturalım. Text Field olarak tanımlayalım. Uzun yazı girilebilsin. Şirketlerin sahibi olsun ve bu sahip sitemize üye olması gereksin. Foroeign key ile Kullanıcı tablosuna ilişki kuralım. Bunun için ilişki tablosunu import etmemiz gerekli. Son olarak oluşturulma tarihi ekleyelim. auto\_now\_add = True vererek kayıt eklendiğinde alanın otomatik doldurulmasını sağlayalım.

name = models.CharField(max\_length=**200**)  
content = models.TextField()  
author = models.ForeignKey(User**,** on\_delete=models.CASCADE)  
created = models.DateTimeField(auto\_now\_add=True)

from django.contrib.auth.models import User

Şimdi alt sınıflarımızı oluşturalım. 2 tane alt sınıfımız olsun bunlardan birtanesi BookCompany diğeri ise GameCompany.

class BookCompany(Company):  
 pass  
  
  
class GameCompany(Company):  
 pass

Bu iki alt sınıfımızda Company tablosunda ekleyeceğimiz aynı alanlara sahip olacağı için Company sınıfından türetelim. Yani bu name, content, author ve created alanları artık bu modellerimizde geçerli oldu. Tekrardan yazmamıza gerek kalmadı. Biz alt sınıflarımıza özel alanlarımızı ekleyebiliriz. Örnek olarak BookCompany için publisher\_name, GameCompany için ise platform olaranları ekleyelim.

Modelimizi \_\_init\_\_.py dosyasından tanıtalım.

from .learning\_inheritance import Company, BookCompany, GameCompany

Yeni oluşturduğumuz modellerin veritabanına eklenebilmesi için migration dosyalarını oluşturalım ve değişiklikleri veritabanı ile eşleyelim.

**python manage.py makemigrations   
python manage.py migrate**

Veritabanını açıp oluşturulan veritabanlarını kontrol edelim. BookCompany ve GameCompany modelleri için tablolar oluşturulduğunu ve Company için abstract sınıf olduğu için tablo oluşturulmadığını görüyoruz.

Model kalıtımında dikkat etmemiz gereken önemli bi kural ilişki alanlarında related\_name ve related\_query\_name kullanımıdır. Django bu alanların tüm proje dahilinde benzersiz olmasını zorunlı kılmaktadır. Eğer related\_name olarak authors kullanırsak bu alt modellereden aynı şekilde miras alınacak ve hata ile sonuçlanmasına neden olacaktır.

author = models.ForeignKey(User**,** on\_delete=models.CASCADE**,** related\_name='authors')

Böyle durumlar için bu alanları dinamik olarak vermemiz gerekir. Uygulama ve model adını kullanarak bunu benzersiz hale getirebiliriz. Bu şekilde kullandığımızda learnings\_bookcompanys\_related, learnings\_gamecompanys\_related şeklinde olacaktır.

author = models.ForeignKey(User**,** on\_delete=models.CASCADE**,** related\_name="%(app\_label)s\_%(class)s\_related"  
 )

Meta ayarlarını da miras şeklinde alt sınıflarımıza devredebiliriz. Bu şekilde kullandığımızda ana sınıfımızın meta ayarlarında yer alan abstract=True alanı miras alınsa bile False olarak davranır.

class Meta(Company.Meta):  
 pass

Eğer alt sınıfı da abstract sınıf olarak tanımayacaksanız tekardan abstract=True yazmanız gerekir.

class Meta(Company.Meta):  
 abstract = True

Alt sınıfımızın meta ayarlarında db\_table ekleyelim.

class Meta(Company.Meta):  
 db\_table = 'company\_alt\_sinifi\_book'

Şu anda sınıfımızın meta ayarlarında sadece db\_table gözükmesine rağmen üst sınıftan miras aldığı için ordering=[‘name’] özelliğine de sahiptir.

Şimdi hiyerarşideki tüm modellerin tabloya sahip olduğu durumu inceleyelim. Bir Book modeli ve Intro modeli oluşturalım. Intro, kitaplardaki ilk bölümdeki kısım olsun. Kitap sınıfımıza name ve page\_count alanlarını ekleyelim. Intro sınıfımızı Book sınıfından türetelim. Üst sınıfta dikkat ederseniz abstract=True ifadesini yazmadık. Book sınıfını abstract sınıf olmadı için alanları Intro sınıfımız için geçerli olmaz. Ayrıca Book modelimiz ve Intro modelimiz için veritabanında farklı tablolar oluşturulacaktır. Django bu iki sınıfı bire-bir ilişki ile bağlayacaktır. İlişkiler kısımında bir sınıftan diğer sınıf ile nasıl ilişki kurulacağını anlatacağım.

class Book(models.Model):  
 name = models.CharField(max\_length=**200**)  
 page\_count = models.PositiveSmallIntegerField()  
  
  
class Intro(Book):  
 content = models.TextField()

Modelimizi \_\_init\_\_.py dosyasından tanıtalım.

from .learning\_inheritance import Company, BookCompany, GameCompany, Book, Intro

Yeni modellerin veritabanına eklenebilmesi için migration dosyalarını oluşturalım ve değişiklikleri veritabanı ile eşleyelim.

**python manage.py makemigrations   
python manage.py migrate**

Veritabanını açıp oluşturulan veritabanlarını kontrol edelim. Book ve Intro modelleri için tablolar oluşturulduğunu görüyoruz. Çünkü iki sınıfta abstract sınıf değiller.

Modellerde anlatacağımız son kalıtım türü Vekil Sınıflardır Proxy Models diye geçiyor. Bu sınıflar bize ana sınıfımızı değiştirmeden model üzerinde değişiklik yapmamıza olanak verir. Modelin vekil model olduğunu belirtmek için meta ayarlarında proxy = True diye belirtiriz. Belirtilen model için veritabanında ayrıca tablo oluşturulmaz.

Biraz önce oluşturduğumuz Book modeli için bir vekil model oluşturalım. Model ismimizi ProxyBook olarak oluşturalım. Book sınıfını miras alsın. Meta ayarlarında vekil sınıf olduğunu belirtelim. Sıralamasını isme göre ayarlatalım ve yeni bir property ekleyelim. short\_name bize name alanını 10 harf ile sınırlandırarak versin. Vekil sınıf sayesinde ana modelimizi değiştirmeden ondan vekil bir model yaratarak farklı şekilde davranmasını sağladık.

class ProxyBook(Book):  
 class Meta:  
 proxy = True  
 ordering = ['name']  
   
 @property  
 def short\_name(self):  
 return self.name[:**50**]

Modelimizi \_\_init\_\_.py dosyasından tanıtalım.

from .learning\_inheritance import Company, BookCompany, GameCompany, Book, Intro, ProxyBook

Yeni modellerin veritabanına eklenebilmesi için migration dosyalarını oluşturalım.

**python manage.py makemigrations**

ProxyBook modeli vekil sınıf olduğu için hiçbir değişiklik bulunamadı.

Vekil sınıfımızın model yöneticisinide değiştirebiliriz. Yeni bir NewManager adında Model yöneticisi oluşturalım ve vekil modelimizde tanıtalım.

class NewManager(models.Manager):  
 # yapılacak işlemler  
 pass

class ProxyBook(Book):  
   
 objects = NewManager()  
   
 class Meta:  
 proxy = True  
 ordering = ['name']  
  
 @property  
 def short\_name(self):  
 return self.name[:**10**]

Kalıtım konusunda son olarak çoklu kalıtımdan bahsedelim. Bir sınıfımız birden fazla sınıftan miras alabilir fakat bu çok kullanılan bir durum değildir. Ama yinede biz örneğini verelim. Örneğimizi daha önce oluşturduğumuz Book sınıfımızdan verelim. Intro sınıfı Book sınıfından türüyordu. Birde intro yorumları için Comment sınıfı tanımlayam ve alan olarak content tanımlayalım. Hem Comment sınıfında hemde Book sınıfında primary\_key id olduğu için Intro sınıfında çakışma olacaktır onun için her iki sınıfta primary\_keyleri manuel olarak verelim.

class Comment(models.Model):  
 comment\_id = models.AutoField(primary\_key=True)  
 content = models.TextField()

class Book(models.Model):  
 book\_id = models.AutoField(primary\_key=True)  
 name = models.CharField(max\_length=**200**)  
 page\_count = models.PositiveSmallIntegerField()

Intro sınıfımızı Book ve Comment sınıfından türetelim.

class Intro(Book**,** Comment):  
 content = models.TextField()

Modelimizi \_\_init\_\_.py dosyasından tanıtalım.

from .learning\_inheritance import Company, BookCompany, GameCompany, Book, Intro, ProxyBook, Comment

Yeni modellerin veritabanına eklenebilmesi için migration dosyalarını oluşturalım ve değişiklikleri veritabanı ile eşleyelim.

**python manage.py makemigrations   
python manage.py migrate**

Veritabanını açıp oluşturulan veritabanlarını kontrol edelim Comment modeli için tablo oluşturulduğunu görüyoruz. Çünkü sınıfımız abstract sınıf değildir.