Arkadaşlar merhaba,

Bu videomuzda Django’da model ilişkilerinin nasıl kurulacağını anlatacağım. Öncelikle size Veritabanı İlişkileri hakkında bilgi vermek istiyorum.

Veritabanı İlişkilerini kullanarak;

* Tekrarlanan verilerin azaltılması
* Veritabanının tutarlı olması
* İş yükünün azalması sağlanır.

Tablolar ilişkilendirilirken normalizasyon kuralları uygulanır. Genel olarak kullanılan 3 normal form vardır. Bire-bir ilişkiler 1. normal Form, Çoka- bir ilişkiler 2. normal Form, Çoka - çok ilişkiler ise 3. normal Form sınıfındadır. Normalizasyon ile ilgili detaylı bilgilere google dan normalizasyon yazarak bilgil alabilirsiniz.

-------------- bire – bir ilişki --------------

Bire – bir ilişki nedir?

Veritabanımızdaki Users tablosunu düşünelim. Users tablosunda kullanıcı üye olurken kullanıcı adı, şifre, email bilgilerini alıyor ve oluşturulduğunda otomatik primary\_key tanımlanıyor.

Kullanıcı bilgileri adında bir tablomuz daha olduğunu düşünelim bu tabloda da kullanıcının adresi, yaşı vb. detay bilgileri tutulsun.

Her kullanıcının sadece bir kullanıcı detayı. Her kullanıcı detayına karşılıkta sadece bir tane kullanıcısı olabilir. Her iki tablodan karşılıklı eşleşen bir kayıt varsa bunu bire – bir ilişki diyoruz.

Hatırlarsanız model kalıtımı konusunda yaptığımız Kitap – Intro örneği bire bir ilişki örneğiydi. Her kitabın sadece bir Intro su vardı. Bir Intro’da sadece bir kitaba ait olabiliyordu.

Yukarıda verdiğimiz kullanıcı örneğini koda dökerek daha iyi anlamaya çalışalım. Zaten Django’da Users tablosu var. Biz UserDetail tablosu oluşturarak bu iki tablo arasında bire – bir ilişki kuracağız. Kullanıcı detayı kullanıcılara bağlı olduğu için ilişkiyi UserDetail modelinde oluşturacağız. Model klasörümüze learning\_relations.py dosyası oluşturalım ve içine User modelimizi import edelim. UserDetail modelimizi oluşturalım ve adress alanı ekleyelim. Şimdi iki tablo arasında ilişki kurmamız gerekli Bire bir ilişkiler için OneToOneField alan türü kullanılmaktadır. İlk parametre olarak ilişki kurulacak Model yazıyoruz, on\_delete alanı ilişkili nesne silindiğinde bu nesneye ne olacağıdır. models.CASCADE yazarsak bu nesne de veritabanından silinir. models.SET\_NULL yaparsak ise alan silindiğinde user alanı NULL olarak işaretlenir. Fakat bunlar bire – bir ilişkili olduğu için birinin silinince diğerininde silinmesi mantıklıdır. Alanı primary\_key olarak işaretleyebiliriz. Çünkü her kullanıcı için sadece bir tane detay bilgisi olacak. İlişkili alanlarda veritabanında user olarak yazdığımız alan adı user\_id olarak kayıt edilir. user alanına User modelimizi ilişkilendirmiş olduk. Model içerisinden self.user, model dışından UserDetail.user ile artık User modeline ulaşabiliriz.

Son olarak \_\_str\_\_ metodumuzu tanımlayalım. Bağlı olduğu kullanıcının usernamini alalım ve detail ile birleştirelim. self.user

from django.db import models  
from django.contrib.auth.models import User  
  
  
class UserDetail(models.Model):  
 address = models.TextField()  
 user = models.OneToOneField(User**,** on\_delete=models.CASCADE**,** primary\_key=True)

def \_\_str\_\_(self):  
 return self.user.username + ' detail'

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

from .learning\_relations import UserDetail

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. Gördüğünüz gibi userdetail tablomuzu ekledi ve auth\_user.id alanı ile userdetail.user\_id ilişkisi kuruldu. Şimdi shellden birkaç kullanıcı ve detay bilgisi ekleyerek bu modeller ile sorgulamalar yapalım.

**python manage.py shell**

**from django.contrib.auth.models import User**

**from learning.models import UserDetail**

**user1 = User(username=’user1’)**

**user1.save()**

**user2 = User(username =’user2)**

**user2.save()**

**user1\_detail = UserDetail(user=user1, adress=’user 1 adresi’)**

**user1\_detail.save()**

**UserDetail.objects.all()**

Kayıt işlemini model yöneticisi ile de yapabiliriz.

**user2\_detail = UserDetail.objects.create(user=user2, adress=’user 2 adresi’)**

**user1.userdetail**

**user1\_detail.user**

**User.objects.all()**

**User.objects.get(userdetail=user1\_detail)**

**User.objects.filter(userdetail\_\_address\_\_startswith="u")**

Daha önceden eklenen kullanıcılar için UserDetail nesnesi eklemedik. Detail bilgisine ulaşınca neler olacak inceleyelim.

**old\_user = User.objects.get(id=1)**

**hasattr(user1, 'userdetail')**

**hasattr(old\_user, 'userdetail')**

-------------- çoka – bir ilişki --------------

Çoka – bir ilişki nedir?

Çoka bir ilişkilere daha önce oluşturduğumuz Product ve User modeliini verebiliriz. Kullanıcılar birden fazla ürün oluşturabilmektedir. Fakat her ürünün sadece bir tane sahibi vardır. Çoka bir ilişkiler Foreign Key ile kurulur. Hatırlarsanız Product modelinde User tablosu ile author alanını kullanarak ilişki kurmuştuk. Product modelini açalım ve kontrol edelim.

author = models.ForeignKey(User**,** on\_delete=models.CASCADE)

Buradada görüldüğü üzere Product modelinden self.author ve model dışından Product.author diyerek User modeline ulaşabiliriz. shelli açıp ürün ekleyelim ve ilişkileri kontrol edelim.

**python manage.py shell**

**from django.contrib.auth.models import User**

**from learning.models import Product**

user1 kullanıcısı için 2 tane ürün ekleyelim, user2 kullanıcısı için 1 tane ürün ekleyelim.

**user1 = User.objects.get(id=3)**

**user2 = User.objects.get(id=4)**

**product1 = Product.objects.create(name='product1', author=user1)**

**TypeError: save() got an unexpected keyword argument 'force\_insert'**

Force\_insert komutunu bulamadığından dolayı hata ile karşılaşıldı. Biz product metodunu override etmiştik oraya parametre olarak force\_insert parametresini girmemiz gerekiyor. Hemen düzeltelim ve shellimizi tekrar açalım.

def save(self**,** force\_insert=False**,** force\_update=False**,** using=None):

**python manage.py shell**

**from django.contrib.auth.models import User**

**from learning.models import Product**

**user1 = User.objects.get(id=3)**

**user2 = User.objects.get(id=4)**

**product1 = Product.objects.create(name='product1', author=user1)**

**product2 = Product.objects.create(name='product2', author=user1)**

**product3 = Product.objects.create(name='product3', author=user2)**

**Product.objects.all()**

**product1.author**

**product1.author.username**

Kullanıcının eklediği ürünleri listelemeye çalışalım. İlişki product modelinde bulunduğu için \_set ile ters ilişki yapabiliriz.

**user1.product\_set.all()**

**user\_product = user1.product\_set.first()**

**user.product.name**

**new\_product = user1.product\_set.create(name=’new product’)**

product\_set yerine kendi atadığımız bir ismide kullanabiliriz. Bunun için ilişkili model alanına related\_name vermemiz gerekmektedir. related\_query\_name ise filter() metodunda kullanılır.

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

shell yeniden çalıştırarak ilişkilerimizi tekrar kontrol edelim.

**python manage.py shell**

**from django.contrib.auth.models import User**

**from learning.models import Product**

**user1 = User.objects.get(id=3)**

**product1 = Products.objects.get(id=3)**

**product1.author.name**

**user1.products.all()**

**user\_product = user1.products.first()**

**user\_product.name**

**User.objects.filter(product\_\_name\_\_startswith='p')** # related\_query\_name kontrol et !

**user2\_product = Product.objects.get(id=3)**

Ürün sahibini değiştirelim.

**user1.products.add(user2\_product)**

-------------- çoka – çok ilişki --------------

Çoka – çok ilişki nedir?

-------------- polimorfosizm --------------