Arkadaşlar merhaba,

Bu videomuzda model meta ayarlarını anlatacağım.

Meta ayarlarının modellerde class Meta: olarak kullanacağını önceki videolarımızda öğrenmiş alabileceği bazı ayarları uygulamıştık.

**abstract** :

Daha önce model kalıtımında kullanmıştık. True olduğunda modelimiz ana sınıf oluyor ve bu model için veritabanına tablo oluşturulmuyor.

class Meta:  
 abstract = True

**app\_label :**

eğer modelimiz uygulama içerisinde değilse hangi uygulamaya ait olduğunu belirtmek için kullanırız.

class Meta:  
 app\_label = 'learning'

**db\_table:**

veritabanına oluşturulacak tablo adını değiştirmek için kullanılır. Daha önce örneklerimizde kullanmıştık.

class Meta:  
 db\_table = 'products'

**base\_manager\_name**:

Modelde kullanılacak base manager adı.

class Meta:  
 default\_manager\_name = 'base\_objects'

**default\_manager\_name:**

Modelde kullanılacak default manager adı. Varsayılan olarak objects dir.

class Meta:  
 default\_manager\_name = 'default\_objects'

**default\_related\_name:**

varsayılan olarak <model adı>\_set ‘tir. Reverse relation için kullanılmaktadır.

class Meta:  
 default\_related\_name = 'reverse\_relationship\_name\_here'

**get\_latest\_by:**

latest() ve earlist() metodları için kullanılacak sıralama ayarları için kullanılır.

class Meta:  
 get\_latest\_by = ['name']

**managed:**

varsayılan olarak true seçilidir. Eğer false seçilirse bu model için veriabanına tablo ekleme, silme işlemleri yapılmaz. Migration işlemlerinden muaf olur.

class Meta:

managed = False

**order\_with\_respect\_to:**

Verilen alana göre sıralama yapılmasına olanak sağlar. Genellikle ForeignKey alanları için kullanılır.

class Product(models.Model):  
  
 author = models.ForeignKey(User**,** on\_delete=models.CASCADE)   
  
 class Meta:  
 order\_with\_respect\_to='author'

get\_product\_order() ile sıralaması alınır. get\_product\_set() ile de sıralama ayarları yapılır. Product nesnemizin adıdır. Sıralama yapılacak alanın modeline göre değişir. Sıralaramalar veritabanında \_order isminde bir alanda saklanır. Meta bilgisi ekledikten ve sildikten sonra migration yapmak gereklidir.

user = User.objects.get(id=**1**)  
user.get\_product\_order()  
# [1, 2, 3]

user.set\_product\_order([**3,1,2**])

İlişkili nesne içinde iki adet kullanılabilir metod vardır. get\_next\_in\_order() ile bir sonraki kayıt, get\_previous\_in\_order() ile bir önceki kayıt seçilir.

product = Product.objects.get(id=**2**)  
product.get\_next\_in\_order()  
# < Product 3>  
product.get\_previous\_in\_order()

# < Product 1>

ordering ve order\_respect\_to beraber kullanılamaz.

**ordering**

Nesne için verilan alana göre sıralama sağlar. Varsayılan olarak azalan sıralama yapar. Artan sıralama için başına – işareti konulur. Rasgele sıralama yapmak için ? kullanılır.

ordering = ['name']  
  
ordering = ['-name']  
  
ordering = ['-name'**,** 'author']

F() gibi sorgu ifadelerini kullanılabilir isme göre artan sırala ve boş kayıtları en son sırala.

ordering = [F('name').asc(nulls\_last=True)]

**default\_permissions**

Django ile kullanılabilecek varsayılan izinlerin ayarlandığı bölümdür. Varsayılan olarak ('add', 'change', 'delete', 'view') dir. Eğer listeyi boş bırakırsanız modelin hiçbir izne gerek duymaz.

**permissions**

modele varsayılan izinler harici izin girmek için kullanılır. Tupple şeklinde kullanılır (izin adı, izin detayı)

permissions = (  
 ("can\_edit\_product"**,** "Ürün güncelleyebilir")**,** ("can\_attach\_product"**,** "Ürün ilişkilendirebilir")**,**)

**proxy**

Daha önce model kalıtımında kullanmıştık. True olduğunda modelimiz proxy sınıf oluyor ve bu model için veritabanına tablo oluşturulmuyor.

**indexes**

İndexler genel olarak where cümleciği içeren select, update ve delete sorgularında ciddi performans artışları sağlarlar. insert sorgularında ve indexli alanalar üzerinde yapılacak update ve delete sorgularında ise performance düşürücü bir etkiye sahiptirler.

Genelde where sorgusunda kullanılacak alanlar için indexleme yapılır. Djangoda Meta ayarları altında indexleme yapabiliriz.

first\_name alanı için index oluşturduk ve isim olarak first\_name\_idx verdik. Eğer isim vermezsek django kendisi oluşturur. İki alan için birlikte de index oluşturabiliriz. First\_name ve last\_name için birlikte index oluşturmuş olduk. Bunu unit\_together ile de yapabilirdik.

class Meta:  
 indexes = [  
 models.Index(fields=['first\_name']**,** name='first\_name\_idx')**,**

models.Index(fields=['first\_name', 'last\_name'])**,** ]

**index\_together**

Birden fazla alan için index oluşturmamızı sağlar.

index\_together = ["first\_name"**,** "last\_name"]

Eğer birden fazla oluşturacaksak;

index \_together = (  
 ["first\_name"**,** "last\_name"]**,** ["username"**,** "email"]**,**)

**unique\_together**

Birlikte benzersiz olacak alanlar için kullanılır.

unique\_together = ("first\_name"**,** "last\_name")

Eğer birden fazla oluşturacaksak;

unique\_together = (  
 ("first\_name"**,** "last\_name")**,** ("username"**,** "email")**,**)

**verbose\_name**

Model adının görüntülenen ismidir varsayılan olarak model ismi atanır. verbose\_name\_plural ise çoğul halidir kullanılmaz ise verbose\_name s takısı gelir. Admin bölümünde kullanılmaktadır.

class Meta:  
 verbose\_name = 'Ürün'  
 verbose\_name\_plural = 'Ürünler'