Merhaba arkadaşlar,

Bir önceki konumuzda model kavramına giriş yaptık ve model oluşturup genel özeliklerini öğrendik.

Şu anda temel bilgiler anlatıyorum. Konular ilerledikçe anlaşılması daha zor olacak onun için temel bilgileri lütfen tekrar edelim.

Bu videomuzda veritabanına nasıl veri ekleyeceğimizi, güncelleyeceğimizi ve listeleyebileceğimizi öğreneceğiz. İlk aşamada veritabanındaki verilere ulaşmak için django shell kullanacağız.

Django shelll açmak için terminalde: **python manage.py shell** yazıp entere basıyoruz.

Öncelike veritabanı ilişki kurabilmek için modeli import etmemiz gerekmektedir.

product ve user modelimizi shelle import edelim.

**from learning.models import Product**

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

------------------ veritabanına ekleme ve güncelleme ------------------

Veritabanına veri eklemek ve güncellemek için Model sınıfının save() metodunu kullanıyoruz.

Bir adet kullanıcı ekleyelim.

**user = User(username=’editor’, email=’editor@learning.udm’)**

Kullanıcı bilgilerini user nesnesine atamış olduk. Bunları veritabanına işlemek için save() methodunu kullanmalıyız.

**user.save()**

Veritabanından kontrol ettiğimizde eklenmiş olduğunu görüyoruz.

Kullanıcı ismini değiştirelim. Hala user nesnemizde kayıt ettiğimiz kullanıcı bilgileri durmaktadır.

**user.username = ‘yonetici’**

**user.save()**

Bir tanede ürün eklemesi yapalım. Ürünlerde bildiğiniz gibi User modeli ile ilişkikli author alanı bulunmaktadır.

**product = Product(name=’Shell Product’, content=’Shellden yeni ürün ekliyoruz’, author=user)**

Evet şimdi product değişkenine ürün nesnemizi atadık. save() yaparal veritabanına kayıt edebiliriz.

------------------ veritabanına veri çekme ------------------

Veritabanından veri çekmek için Model Manager (model yöneticisi) ile QuerySet yani sorgu seti oluştururuz. Her modelin en az bir adet Manager’ı vardır ve modelismi.objects olarak ulaşılır.

**Product.objects**

QuerySet nedir?

Querysetler veritabanındaki bir nesne kolleksiyonunu temsil eder. Mesela Ürünler.

QuerySetlerin sıralanabilir., filtrelenebilir. Bu işlemler için Manager metodlarını kullanırız. Şimdi bu metodları inceleyelim.

Tüm nesneleri listelemek için all() methodunu kullanırız.

**User.objects.all()**

------------------ filtreleme ------------------

Tüm kullanıcıları değilde bazı filtrelerden geçen kullanıcıları listelemek için filter() ve exclude() kullanılır.

filter içerisinde verilen parametrelere ile eşleşen kayıtları getirir. Örneğimizde veritabanında is\_staff bir olan kayıtlar sorgu setine eklenecektir.

**User.objects.filter(is\_staff=1)**

Birden fazla parametrede girebiliriz. Parametreler birbirine AND ile bağlanır.

sql sorugusuna tercüme edersek WHERE is\_staff=1 AND active=1

**User.objects.filter(is\_staff=1, active=1)**

exclude ise filter tam tersidir. verilen paratmere ile eşleşmeyen kayıtlar getirilir. Örneğimizde veritabanında is\_staff bir olmayan ve is\_superuser=1 olmayan kayıtlar sorgu setine eklenecektir.

**User.objects.exclude(is\_staff=1, is\_superuser=1)**

filter ve exclude zincirleme olarak kullanıma uygundur. Örneğimizde aktif alanı 1 olan is\_staff alanı 1 olmayan ve is\_superuser alanı 0 olan kayıtlar gelecektir.

**User.objects.filter(active=1).exlude(is\_staff=1).filter(is\_superuser=0)**

İki exludenin zincirleme kullanılmasında olay biraz farklı. Yukarıda bu örneği tek exclude içerisinde vermiştik ve is\_staff bir olmayan ve is\_superuser=1 olmayan kayıtları getirdiğini söylemiştik. Zincirleme kullandığımızda ise is\_staff bir olmayan veya is\_superuser=1 olmayan kayıtları getirir. Bu ve / veya ayrımı önemli.

**User.objects.exclude(is\_staff=1).exclude(is\_superuser=1)**

DJANGO bize filter(), exclude() ve get() methodları için alan araması imkanı sunar.

Biraz önce örneklerimizde is\_staff = 1 olan kayıtları getirmiştik. 1 den büyük olan kayıtları da getirebiliriz. Veya 1 içeren kayıtları. Bu İngilizce olarak field lookups diye geçmektedir.

gt

lt

exact

iexact

contains

icontains

in

gt

gte

lt

lte

startswith

istartswith  
endswith

iendswith

range  
date

year

month

day

week

week\_day

quarter

time

hour

minute

second

isnull

regex

iregex

------------------ sıralama ------------------

get : primary key ile arama. Sadece bir kayıt getirir. Birden çok kayıt gelirse hata verir.

First: ilk kaydı getirir

----------------------------------------------------------------------------------------------

User.objects.order\_by('-username’).all()

User.objects.order\_by('-username’, ‘email’).all()

User.objects.order\_by(‘?’).all() -> rasgele sıralama

Product.objects.order\_by(‘author\_\_id’).all() <- ilişkili alana göre sıralama

Entry.objects.order\_by('headline').order\_by('pub\_date’) <- sadece pub\_date göre sıralanır

Limitleyerek gösterelim :

----------------------------------------------------------------------------------------------

İlk 5 kayıt -> [:5]

6. Kayıttan başlayıp 10 .kayda kadar -> [5:10]

İlk 10 kayıttaki 2. kayıtlar -> [:10:2]

Product.objects.order\_by(‘name')[0] <- yoksa index hatası verir

Product.objects.order\_by(‘name')[0:1].get() <- yoksa DoesNotExist hatası verir.

----------------------------------------------------------------------------------------------

Tersten Sıralama : reverse()

----------------------------------------------------------------------------------------------

my\_queryset.reverse()[:5]

Çoklayanları Temizleme : distinct()

----------------------------------------------------------------------------------------------

Author.objects.distinct()

values() : Verileri object olarak değil dict olarak döndürür

----------------------------------------------------------------------------------------------

Blog.objects.filter(name\_\_startswith='Beatles').values()

Blog.objects.values('id', 'name’) -> istenelen alanlar filtrelenebilir

values\_list() – verileri tupple olarak döndürür

----------------------------------------------------------------------------------------------

Entry.objects.values\_list('id', 'headline')

Entry.objects.values\_list('id', flat=**True**).order\_by('id’) <- sadece değerler döner

**>>>** Entry.objects.values\_list('headline', flat=**True**).get(pk=1) <- özel bir alan değerini döndürmek için

dates() - **datetimes()**

----------------------------------------------------------------------------------------------

order, which defaults to 'ASC', should be either 'ASC' or 'DESC'. This specifies how to order the results.

Entry.objects.dates('pub\_date', 'year')   
Entry.objects.dates('pub\_date', 'day', order='DESC')

"year" returns a list of all distinct year values for the field.

"month" returns a list of all distinct year/month values for the field.

"week" returns a list of all distinct year/week values for the field. All dates will be a Monday.

"day" returns a list of all distinct year/month/day values for the field.

none() **– boş queryset döndürür**

----------------------------------------------------------------------------------------------

**>>>** Entry.objects.none() <QuerySet []>

**union()**

----------------------------------------------------------------------------------------------

Uses SQL’s UNION operator to combine the results of two or more QuerySets.

The UNION operator selects only distinct values by default. To allow duplicate values, use the all=True argument.

qs1 = Author.objects.values\_list('name)

qs2 = Entry.objects.values\_list('headline’)

qs1.union(qs2).order\_by('name’)

**intersection()**

----------------------------------------------------------------------------------------------

Uses SQL’s INTERSECT operator to return the shared elements of two or more QuerySets. For example:

qs1.intersection(qs2, qs3)

**difference()**

----------------------------------------------------------------------------------------------

Uses SQL’s EXCEPT operator to keep only elements present in the QuerySet but not in some other QuerySets.   
qs1.difference(qs2, qs3)

**extra()**

----------------------------------------------------------------------------------------------  
**select**

Entry.objects.extra(select={'is\_recent': "pub\_date > '2006-01-01'"})   
SELECT blog\_entry.\*, (pub\_date > '2006-01-01') AS is\_recent FROM blog\_entry;

Blog.objects.extra( select={ 'entry\_count': 'SELECT COUNT(\*) FROM blog\_entry WHERE blog\_entry.blog\_id = blog\_blog.id' }, )

SELECT blog\_blog.\*, (SELECT COUNT(\*) FROM blog\_entry WHERE blog\_entry.blog\_id = blog\_blog.id) AS entry\_count FROM blog\_blog;

**where**

Entry.objects.extra(where=["foo='a' OR bar = 'a'", "baz = 'a'"])

SELECT \* FROM blog\_entry WHERE (foo='a' OR bar='a') AND (baz='a')

**order\_by**

q = Entry.objects.extra(select={'is\_recent': "pub\_date > '2006-01-01'"})

q = q.extra(order\_by = ['-is\_recent’])

**params**

Entry.objects.extra(where=['headline=**%s**'], params=['Lennon'])

**defer()**

----------------------------------------------------------------------------------------------

This is done by passing the names of the fields to not load to defer():  
Entry.objects.defer("headline", "body")

*# Defers both the body and headline fields.*

Entry.objects.defer("body").filter(rating=5).defer("headline")

Blog.objects.select\_related().defer("entry\_\_headline", "entry\_\_body")

If you want to clear the set of deferred fields, pass None as a parameter to defer():

*# Load all fields immediately.*

my\_queryset.defer(**None**)

**only()**

----------------------------------------------------------------------------------------------

Suppose you have a model with fields name, age and biography.

The following two querysets are the same, in terms of deferred fields:

Person.objects.defer("age", "biography")

Person.objects.only("name")

*# Final result is that everything except "headline" is deferred.*

Entry.objects.only("headline", "body").defer("body")

*# Final result loads headline and body immediately (only() replaces any*

*# existing set of fields).*

Entry.objects.defer("body").only("headline", "body")

**using()**

----------------------------------------------------------------------------------------------

This method is for controlling which database the QuerySet will be evaluated against if you are using more than one database.

The only argument this method takes is the alias of a database, as defined in [DATABASES](https://docs.djangoproject.com/en/2.1/ref/settings/).

*# queries the database with the 'default' alias.*

Entry.objects.all()

*# queries the database with the 'backup' alias*

Entry.objects.using('backup’)

**Raw Queries de anlatalım**

----------------------------------------------------------------------------------------------

**raw()**

**transaction da anlatalım**

----------------------------------------------------------------------------------------------

**select\_for\_update()**

ilişkilerde anlatalım

----------------------------------------------------------------------------------------------

annonate()

**select\_related()**

**prefetch\_related()**