Merhaba arkadaşlar,

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

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

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

Birden fazla argüman girebiliriz. Argümanlar 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.

Field lookupslar anahtar kelimenin yanına iki tane alt çizgi ile kullanılır. Kullanabileceğimiz fields lookuplara hızlı bir şekilde bakalım.

exact tam eşleşen kayıtları listeler

User.objects.get(username\_\_exact = ‘emre’) # where username=emre  
User.objects.get(username\_\_exact =None) # where username is null

iexact tam eşleşen kayıtları listeler, büyük-küçük harfe duyarlı değildir

User.objects.get(username\_\_iexact = ‘emre’) # EmRe emre

Büyüktür ve küçüktür filtrelemeleri için lt ve gt kullanılır.

1 den küçük (less then)

User.objects.filter(is\_staf\_\_lt = **1**)

1 den küçük veya eşit

User.objects.filter(is\_staf\_\_lte = **1**)

1 den büyük (greater then) :

User.objects.filter(is\_staf\_\_gt = **1**)

1 den büyük veya eşit

User.objects.filter(is\_staf\_\_gte = **1**)

sql tercümesi LIKE olan işlemler için (başlayan, biten, içeren) startswith, endswith, contains kullanılır

küçük-büyük harf duyarlı olmayacaksa başına i koyulur.

e ile başlayan :

User.objects.filter(username\_\_startswith = ‘e’) #sadece e  
User.objects.filter(username\_\_startswith = ‘e’) # e ve E

e ile biten :

User.objects.filter(username\_\_endsswith = ‘e’) # sadece e  
User.objects.filter(username\_\_ endsswith = ‘e’) # e ve E

e içeren :

User.objects.filter(username\_\_contains = ‘e’) # sadece e  
User.objects.filter(username\_\_ icontains = ‘e’) # e ve E

range iki değer arasındaki kayıtları filtreler. sql tercümesi BETWEEN’dir. ID si 2 ve 5 arasındaki kullanıcıları listelemek için kullanırız.

start = **2**end = **5**User.objects.filter(id\_\_range=(start**,** end))

in bir soruda birden fazla değere sahip kayıtların listelenmesini sağlar. IN kullanımı bazı durumlarda performans açısından hayat kurtarınıcı olabilir. İlişkili tablolar ile çalışırken tek sorguda yapmak yerine birinci sorguda ilk tablodaki kayıtların ID lerini almak daha sonra bu ID leri ikinci tabloda filtre olarak kullanmak büyük verilerle çalışıyorsanız performans açısından çok önemlidir.

User.objects.filter(id\_\_in=[**1, 3, 4**])

isnull filtrelenen alanın boş olup olmasını sorgulamak için kullanılır

User.objects.filter(email\_\_isnull=True) # WHERE email IS NULL;

regex ve iregex düzenli ifadeler ile sorgu yapmayı sağlar. iregex de küçük – büyük harf duyarlılığı yoktur. veritabanına göre kullanılacak düzenli ifadelerde ufak nüsans farklılıkları vardır. Kullanıcı adında sadece harf içeren kayıtları çeken bir sorgu:

User.objects.get(username\_\_regex= r'^[a-zA-Z]+$')

date, datetime ve time alanları için yıl, çeyrek, ay, gün, hafta, saat bazında filtreleme yapılabilir. Bu field lookupsları ile birlikte lt, gt, lte ve gte kullanılabilir. Bunların kullanımları hep aynı olup birkaç tane örnek yapalım.

User.objects.filter(date\_joined\_\_quarter=**2**) # yılın 2. çeyreğinde kayıt olan kullanıcılar  
User.objects.filter(date\_joined \_\_year\_\_gte=**2019**) # 2019 yılından sonra kayıt olan kullanıcılar

date: tarih (sadece datetime alanı için)

time: zaman (sadece datetime alanı için)

date ve datetime alanları için

year : yıl

month: ay

day: gün

week: haftay

week\_day: hafta günü (1 Pazar, 7 Cumartesi)

quarter: mevsim (1 ve 4 arası)

time ve datetime alanları için

hour: saat

minute: dakika

second: saniye

------------------ verileri sıralama ve limit ------------------

Sorgu setindeki verileri sıralamak için order\_by() methodunu kullanıyoruz.

Kullanıcı tablosundaki verileri kullanıcı adına göre artan sıralamak için order\_by method içinde argüman olarak username kullanmamız yeterli. Azalan sıralamada ise başına tire (eksi) işareti koyaraız.

User.objects.order\_by('username’) # artan sıralama  
User.objects.order\_by('-username’) # azalan sıralama

Argüman olarak ? girerek verileri rasgele sıralayabiliriz.

User.objects.order\_by(‘?’) # rasgele sıralama

İlişkili alana göre sıralama için argüman olarak öncelikle bizim tablomuzdaki alanı yazarız daha sonra iki tane alt çizgi ekler ilişki tablosundaki alan adını yazarız. User tablosundaki kullanıcı adına göre ürünleri sıralamasını söylüyoruz.

Bir önceki dersten hatırlarsınız fields lookupstada alttan iki çizgi ekliyorduk. Biz veri yapılarımızı tanımlarken alan adı olarak fields\_lookup da yer alan özellikleri aynı isimleri veremiyoruz. Django buna izin vermiyor.

İleriki konularda eğer unutmazsak model oluştururken bunu deneriz.

Product.objects.order\_by(‘author\_\_username’)

Eğer zincirleme olarak sıralama yaparsak zincirin en son halkasına göre sıralama yapılır. Diğerleri yok sayılır. Birden çok alana göre sıralama telk method içerisinde birden fazla argüman göndermeliyiz.

User.objects.order\_by('username').order\_by('date\_joined’) # sadece date\_joined göre sıralanır  
User.objects.order\_by('username’,’date\_joined’) # önce username sonra date\_joined göre sıralanır

Verileri tersten sıralamak için reverse() methodu kullanılır. Eğer veriler daha önce sıralanmamışsa hiçbir etkisi olmaz. Ürünler sorgu setini ters çevirelim.

products = Product.objects.order\_by(‘name’)  
products.reverse()

Djangoda sorgu setlerinde indexing (indexleme), slicing (dilimleme) ve stepping (adımlama) desteklenir. Örneklerle inceleyelim Öncelikle sorgu setindeki 2. İndexte yer alan kaydı alalım. Unutmayalım index 0 dan başlamaktadır.

Products.objects.all()[**2**]

ilk 10 kaydı alabiliriz.

Products.objects.all()[:**10**]

6. ve 10. Arası kayıtları çekelim.

Products.objects.all()[**5**:**10**]

İlk 11. ve 20. arası kayıtlar arasından 2. Kayıtları alalım

Products.objects.all()[**10**:**20**:**2**]

Ürünleri isme göre sıralayıp 0. indexteki nesneyi getirelim.

Product.objects.order\_by(‘name')[0]

Bir önceki sorgu seti ile bu sorgu seti aynı sonucu getirir. Eğer kayıt bulunamazsa birincisi IndexError hatası ikincisi ise DoesNotExist hatası döndürür.

Product.objects.order\_by(‘name')[0:1].get()