----------- Q NESNESİ İLE GELİŞMİŞ FİLTRELEME ------------

Daha önce öğrendiğimiz gibi Django ORM sorguları AND (&) ile birleşiyordu. OR (|) kullanabilmek için Q Nesnesinden yararlanırız.

# WHERE question LIKE 'Who%'

Q(question\_\_startswith='Who')

# WHERE question LIKE 'Who%' OR question LIKE 'What%'

Q(question\_\_startswith='Who') | Q(question\_\_startswith='What')

# SELECT \* from polls WHERE question LIKE 'Who%' AND (pub\_date = '2005-05-02' OR pub\_date = '2005-05-06')

Poll.objects.get(  
 Q(question\_\_startswith='Who')**,** Q(pub\_date=date(**2005, 5, 2**)) | Q(pub\_date=date(**2005, 5, 6**))  
)

# Q tilde (yaklaşık) işareti ile olumsuz yapılır

# WHERE question NOT LIKE 'Who%'

~Q(question\_\_startswith='Who')

----------- F NESNESİ ------------

F() bir model alanının değerini almamızı sağlar.

Product tablomuzda sipariş adedi ürün adedinden çok olan kayıtları çekelim. Burada F nesnesi ile Product tablosunda num\_stocks alanına direkt ulaşım sağladık.

Product.objects.filter(num\_orders\_\_gt=F('num\_stocks))

İşlemler Python tarafında değil veritabanı tarafında gerçekleşir. Bizim bir Book nesnemiz olsun ve view\_count alanımız olsun. Alan her görüntülendiğinde bir artsın. Sayfa görüntülendiğinde view\_count 1 artıralım. Burada veritabanından book nesnesini belleğe aldık ve Python operatörlerini kullanarak bir artırdık ve veritabanına tekrar kaydettik.

book = Book.objects.get(id=**1**)

book.view\_count += 1

book.save()

Bu örneğimizde ise **book.view\_count = F('view\_count') + 1** işlemi görüntü olarak Python işlemi gibi görünse de aslında veritabanında gerçekleşmektedir. Django F() gördüğünde Python operatörlerin geçersiz kılar. Python hiçbir zaman view\_count alanı hakkında bilgi sahibi olmaz. Bu durumda yeni veriye ulaşabilmek için veritabanından gelen veri yenilenmelidir.

from django.db.models import F  
  
book = Book.objects.get(id=1)  
book.view\_count = F('view\_count') + **1** # veritabanı işlemibook.save()

book.refresh\_from\_db() # view\_count yeni değerine artık ulaşabiliriz

view\_count artırma işlemini update() metodu ile de yapabilirdik.

book = Book.objects.get(id=1)  
book.update(view\_count=F('view\_count') + **1**)

F nesneleri model kaydedildikten sonra yeni bir kayıtta tekrar uygulanır. Bunu önlemek için kayıttan sonra refresh\_from\_db kullanmak gerekir.

book = Book.objects.get(id=1)  
book.view\_count = F('view\_count') + **1** # veritabanı işlemibook.save()

# burada refresh\_from\_db yapılırsa aşağıdaki kod sadece ismi günceller

book.name = 'Yeni kitap adı'  
book.save() # view\_count burda tekrar artar

F nesneleri annotate ile birlikte kullanılabilir. Her ürün için gerekli ilave stok miktarını hesaplayalım.

products = Product.objects.annotate(needed\_stock=F('num\_orders') – F('view\_stocks'))

F nesnesi ile boş kayıtları sorgu setinin en başına veya sonuna sıralayabiliriz.

Product.objects.order\_by(F('name').desc(nulls\_last=True)

----------- WHEN – CASE NESNESİ ------------

Şartlar gerçekleştiğinde then argümanındaki değeri ana argümana atar.

When(Q(name\_\_startswith="John") | Q(name\_\_startswith="Paul")**,** then='name')

Genellikle CASE ile birlikte kullanılır. CASE if – else kullanımına benzer. Hangi WHEN içerisindeki şart gerçekleşirse onun içerisindeki then atanır.

Client.objects.annotate(  
 discount=Case(  
 When(account\_type=Client.GOLD**,** then=Value('5%'))**,** When(account\_type=Client.PLATINUM**,** then=Value('10%'))**,** default=Value('0%')**,** output\_field=CharField()**,** )**,**).values\_list('name'**,** 'discount')

----------- CAST NESNESİ ------------

Veritabanından çektiğimiz alan türünü değiştirmeye yarar. annotate ile birlikte kullanılır. Örneğimizde integer olan bir alanın float olarak almamızı sağlar.

value = Value.objects.annotate(as\_float=Cast('integer'**,** FloatField())).get()  
print(value.as\_float) # 4.0

----------- COALESCE NESNESİ ------------

Listeye eklenen alanlardan ilk dolu bulduğu alanı argümana atar. Product modelimiz olsun alanlarıda name, title, slug olduğunu düşünelim.

yeni\_alan = Coalesce(‘name’,’title’,’slug’) yapığımızda yeni\_alan’a eğer name dolu ise name atanır. Boş ise title atanır. Title boş ise slug atanır.

Tüm alanların aynı türde olması gerekir.

from django.db.models import Sum**,** Value as V  
from django.db.models.functions import Coalesce  
  
product = Product.objects.annotate(new\_name=Coalesce('name'**,** 'title'**,** 'slug')).get()  
  
print(product.new\_name)

----------- GREATEST NESNESİ ------------

Listeye eklenen alanlardan en büyüğünü argümana atar. En küçüğü için ise Least kullanılır.

from django.db.models.functions import Greatest  
blog = Blog.objects.create(body='Greatest is the best.')  
comment = Comment.objects.create(body='No, Least is better.'**,** blog=blog)  
comments = Comment.objects.annotate(last\_updated=Greatest('modified'**,** 'blog\_\_modified'))  
annotated\_comment = comments.get()

----------- NULLIF NESNESİ ------------

Accepts two expressions and returns None if they are equal, otherwise returns expression1.

----------- EXTRACT NESNESİ ------------

Tarihlerin istenilen bileşenini almak için kullanılır. (yıl, ay vs.)

from datetime import datetime  
from django.db.models.functions import Extract

# Ürünün eklenme yılı.  
experiment = Experiment.objects.annotate(  
 start\_year=Extract('start\_datetime'**,** 'year')).get()  
experiment.start\_year  
**2015**# Aynı yılda kaç tane sonuçlanmış ?  
Experiment.objects.filter(  
 start\_datetime\_\_year=Extract('end\_datetime'**,** 'year')).count() # 1

----------- NOW NESNESİ ------------

Şu anki zamanı argümana atamak için kullanılır.

from django.db.models.functions import Now  
Article.objects.filter(published\_\_lte=Now())

----------- TRUNC NESNESİ ------------

from datetime import datetime  
from django.db.models import Count**,** DateTimeField  
from django.db.models.functions import Trunc  
Experiment.objects.create(start\_datetime=datetime(**2015, 6, 15, 14, 30, 50, 321**))  
Experiment.objects.create(start\_datetime=datetime(**2015, 6, 15, 14, 40, 2, 123**))  
Experiment.objects.create(start\_datetime=datetime(**2015, 12, 25, 10, 5, 27, 999**))  
experiments\_per\_day = Experiment.objects.annotate(  
 start\_day=Trunc('start\_datetime'**,** 'day'**,** output\_field=DateTimeField())  
).values('start\_day').annotate(experiments=Count('id'))  
for exp in experiments\_per\_day:  
 print(exp['start\_day']**,** exp['experiments'])  
  
# 2015-06-15 00:00:00 2  
# 2015-12-25 00:00:00 1  
experiments = Experiment.objects.annotate(  
 start\_day=Trunc('start\_datetime'**,** 'day'**,** output\_field=DateTimeField())  
).filter(start\_day=datetime(**2015, 6, 15**))  
for exp in experiments:  
 print(exp.start\_datetime)  
   
# 2015-06-15 14:30:50.000321  
# 2015-06-15 14:40:02.000123