----------- 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)

----------- AGGREGATION --------------

Aggregation, sorgularda toplam, ortalama, en çok, en az gibi hesaplama işlemleri yapmamızı sağlar.

aggregate() tablo için genel hesaplamalar, annotate() ise satır bazında hesaplamalar yapar. annotate() genelde ilişkili tablo hesaplamalarında kullanılır.

Örnekler üzerinden devam edelim.

from django.db.models import Avg, Count, Max, Min, Sum

# adet - 73

Product.objects.all().count()

############## AGGREGATE ##############

# ortalama – price\_\_avg : 23.34  
Product.objects.aggregate(Avg('price'))

# en çok – price\_\_max : 50

Product.objects.aggregate(Max('price'))

# en az – price\_\_min : 3

Product.objects.aggregate(Min('price'))

# toplam – price\_\_sum : 4

Product.objects.aggregate(Sum('price'))

# ortalama - istediğimiz isme atabiliriz – average\_price : 23.34

Product.objects.aggregate(average\_price=Avg('price'))

# birden fazla aggregate çalıştırabiliriz.

# {'price\_\_avg': 34.35, 'price\_\_max': Decimal('81.20'), 'price\_\_min': Decimal('12.99')}

Product.objects.aggregate(Avg('price')**,** Max('price')**,** Min('price'))

############## ANNOTATE ##############

# kullanıcılara ait ürün sayıları

users = Users.objects.annotate(Count('products'))

users[0].products\_\_count

# belirlediğimiz isme atayabiliriz.

users = Users.objects.annotate(num\_products=Count('products'))

users[0].num\_products

Aggregate ve filter() kullanımı

from django.db.models import Count  
Client.objects.aggregate(  
 regular=Count('pk'**,** filter=Q(account\_type=Client.REGULAR))**,** gold=Count('pk'**,** filter=Q(account\_type=Client.GOLD))**,** platinum=Count('pk'**,** filter=Q(account\_type=Client.PLATINUM))**,**)

**SELECT** **count**('id') FILTER (**WHERE** account\_type=1) **as** regular,  
 **count**('id') FILTER (**WHERE** account\_type=2) **as** gold,  
 **count**('id') FILTER (**WHERE** account\_type=3) **as** platinum  
**FROM** clients;

Birden fazla annotate birleştirilmesi hatalara sebep olur. Çoğu hesaplama için bunun çözümü yoktur fakat count için distinct ile çözülür.

# yanlış değerler döndürür.

users = Users.objects.annotate(Count('products'), Count('books'))

# doğru değerler döndürür.

users = Users.objects.annotate(Count('products' **,** distinct=True), Count('books' **,** distinct=True))

İlişkili işlemlerde annonate kullanılabilir. Kategorideki max ve min ürün fiyatlarını alalım. Birden fazla ilişki için de kullanılabilir. Çoka – çok ilişkiler için reverse relationship içinde kullanılabilir.

from django.db.models import Max**,** Min  
Category.objects.annotate(min\_price=Min('product\_\_price')**,** max\_price=Max('product\_\_price'))

# products\_\_authors\_\_age

# reverse relationship

Product.objects.annotate(Count('categories'))

filter() kullanımı.

Annotate filtrelenebilir. Toplam ürünü birden fazla olan kategoriler

Category.objects.annotate(num\_products=Count(products)).filter(num\_products\_\_gt=**1**)

Eğer birden fazla annotate varsa

Author.objects.annotate(num\_books=Count('book')**,** highly\_rated\_books= Count('book'**,** filter=Q(book\_\_rating\_\_gte=**7**)))

Annotate() ve filter() birlikte kullanımında sıralamaları önem arz etmektedir.

# Bu sorguda filtrenin annotate üzerinde bir etkisi yoktur. Her yayıncı için kitap sayılarını sayar.

a**,** b = Publisher.objects.annotate(num\_books=Count('book'**,** distinct=True)).filter(book\_\_rating\_\_gt=**3.0**)

# Bu sorgu her yayıncı için kitap ratingi 3 geçen kayıtları sayar.

a**,** b = Publisher.objects.filter(book\_\_rating\_\_gt=**3.0**).annotate(num\_books=Count('book'))

order\_by() kullanımı.

annotateler sıralanabilir

User.objects.annotate(num\_products=Count('products')).order\_by('num\_products')

values() kullanımı.

Annonate ile kullanıcılara ait ortalama kitap fiyatlarını seçelim. Bu sorgu her kullanıcı için ortalama kitap fiyatı hesaplayacaktır.

User.objects.annotate(average\_price=Avg('product\_\_price'))

Bu sorguyu first\_name alanına göre grupladığımızda durum biraz değişir. Aynı ismi kullanan birden fazla satır o varsa o satırların kayıtları birleştirilir.

User.objects.values('first\_name').annotate(average\_price=Avg('product\_price'))

values() metodu annonate() metodunundan sonra kullanılduğında ise annonate den gelen veriler üzerinden gelen alanlar alınır. Bu örneğimizde her bir kullanıcı için ayrı ayrı ürün fiyatı hesaplaması yapılır.

User.objects.annotate(average\_price=Avg('product\_price')).values('first\_name', 'average\_price')

ordering kullanımı :

User modelini name alanına göre sıraladığımızı düşünelim.

class Meta:  
 ordering = ["last\_name"]

Kullanıcı tablosunda kaçtane first\_name olduğunu saydıralım. Bu bize tam doğru sonucu vermez. Çünkü sorgu setimiz benzersiz (first\_name ve last\_name) göre gruplanır.

User.objects.values('first\_name').annotate(Count('id'))

İstediğimiz değeri alabilmek için verileri tekrardan first\_name göre sıralamamız gereklidir.

User.objects.values('first\_name').annotate(Count('id')).order\_by()

Annotate ile kullanılabilen djangonun builtin fonksiyonlar vardır. Bunlardan bazılarını inceleyelim.

**Şarta Bağlı Fonksiyonlar : When, Case**

----------- WHEN – CASE ------------

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')

Karşılaştırma ve Dönüşüm : **Cast**, **Coalesce**, **Greatest**, Least, **NullIf**

----------- CAST ------------  
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 ------------

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

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

from django.db.models.functions import Greatest

product = Product.objects.annotate(last\_created=Greatest('created'**,** 'comment\_\_created'))

----------- NULLIF ------------

İli tane arguman alır eğer argümanlar birbirine eşit ise None olarak geri döner değilse 1. argüman geri döner .

from django.db.models.functions import NullIf

product = Product.objects.annotate(new\_name= NullIf('name’**,** 'title'))

Tarih Fonksiyonları : **Extract**, **Now**, Trunc

----------- NOW ------------

Şu anki zamanı argümana atamak için kullanılır.

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

----------- EXTRACT ------------

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

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

# Ürünün eklenme yılı.  
product = Product.objects.annotate(  
 start\_year=Extract('created'**,** 'year')).get()  
product.start\_year  
**2015**

# Aynı yılda kaç tane sonuçlanmış ?  
Experiment.objects.filter(  
 start\_datetime\_\_year=Extract('end\_datetime'**,** 'year')).count() # 1

Matematiksel Fonksiyonlar : **Abs**, **Ceil**, Degrees, Exp, **Floor**, Ln, Log, Mod, Pi, **Power**, Radians, **Round**, **Sqrt**, ACos, ASin, ATan, ATan2, Cos, Sin, Tan, Cot

from django.db import models  
  
# MATEMATİKSEL FONKSİYONLAR  
  
class Vector(models.Model):  
 x = models.FloatField()  
 y = models.FloatField()

# Abs() : mutlak değer  
vector = Vector.objects.annotate(new\_x=Abs('x')).get()

# Ceil() : yukarı yuvarlama  
vector = Vector.objects.annotate(new\_x=Ceil('x')).get()

# Floor() : aşağı yuvarlama

vector = Vector.objects.annotate(new\_x=Floor('x')).get()

# Round() : en yakın değere yuvarlama  
vector = Vector.objects.annotate(new\_x=Round('x')).get()

# Power() : x üzeri y  
vector = Vector.objects.annotate(new\_value=Power('x'**,**'y')).get()

# Sqrt() : kare kök

vector = Vector.objects.annotate(new\_value=Sqrt('x')).get()  
  
# bunlar filter içerisinde kullanılabilecek şekilde ayarlanabilir.  
FloatField.register\_lookup(Ceil)  
FloatField.register\_lookup(Abs)

vectors = Vector.objects.filter(x\_\_abs\_\_lt=**1**) # mutlak değeri birden büyük olanlar

Text Fonksiyonları : Chr, Concat, Left, Length, Lower, Lpad, Ltrim, Ord, Repeat, Replace, Reverse, Right, Rpad, Rtrim, StrIndex, Substr, Trim, Upper,

# TEXT FONKSİYONLARI  
  
# name alanının ilk harfinin unicode sayı karşılığını verir.  
author = Author.objects.annotate(name\_code\_point=Ord('name')).get() # author.name\_code\_point = 77  
  
# Chr unicode sayı ile arama yapar. M harfi ile başlayan kayıtları getirir  
author = Author.objects.filter(name\_\_startswith=Chr(ord('M'))).get()  
  
# alanları birleştirir. screen\_name için name ve goes by alanları birleşir.  
author = Author.objects.annotate(  
 screen\_name=Concat(  
 'name'**,** V(' (')**,** 'goes\_by'**,** V(')')**,** output\_field=CharField()  
 )  
).get()  
  
# soldaki 1. harfi verir  
author = Author.objects.annotate(first\_initial=Left('name'**, 1**)).get()  
  
# sağdaki 1. harfi verir  
author = Author.objects.annotate(last\_letter=Right('name'**, 1**)).get()  
  
# name alanının karakter sayısını verir. register edip filterda kullanabiliriz.  
author = Author.objects.annotate(name\_length=Length('name')).get()  
  
# küçük harf yapar  
author = Author.objects.annotate(name\_lower=Lower('name')).get()  
  
# büyük harf yapar  
author = Author.objects.annotate(name\_upper=Upper('name')).get()  
  
# 8 karakter olana kadar sola abc ekler. 4 hafta ekleyeceği için abcaJohn  
Author.objects.update(name=LPad('name'**, 8,** Value('abc')))  
  
# 8 karakter olana kadar sağa abc ekler. 4 hafta ekleyeceği için Johnabca  
Author.objects.update(name=LPad('name'**, 8,** Value('abc')))  
  
# soldaki boşlukları temizleme  
Author.objects.update(name=LTrim('name'))  
  
# sağdaki boşlukları temizleme  
Author.objects.update(name=RTrim('name'))  
  
# sağ ve soldaki boşlukları temizleme  
Author.objects.update(name=Trim('name'))  
  
# name alanının 1. harfinden başlayıp 5. harfa kadar alır ve alias kaydeder  
Author.objects.update(alias=Lower(Substr('name'**, 1, 5**)))  
  
# eğer name içerisinde Smith geçiyorsa smith\_index 1 atar. ve smith\_index 1 den büyük olanları listeler  
authors = Author.objects.annotate(smith\_index=StrIndex('name'**,** V('Smith'))).filter(smith\_index\_\_gt=**0**)  
  
# name ters çevirir  
author = Author.objects.annotate(backward=Reverse('name')).get()  
  
# name alanındaki Margaret'i Margareth olarak değiştirir  
Author.objects.update(name=Replace('name'**,** Value('Margaret')**,** Value('Margareth')))  
  
# name alanındaki değer 3 kere yan yana yazar ve kayıt eder.  
Author.objects.update(name=Repeat('name'**, 3**))

Window Fonksiyonları : CumeDist, DenseRank, FirstValue, Lag, LastValue, Lead, NthValue, Ntile, PercentRank, Rank,