------- DERS6 : DJANGO SHELL VE VERİTABANI İŞLEMLERİ – EKLEME, GÜNCELLEME, SİLME, LİSTELEME-------

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. Daha sonra benzer işlemleri views dosyalarımızda da yapacağız. Kullanacağımız metodlar aynı olacak.

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()**

create() methodunu kullanarak da veritabanına ekleme yapabiliriz.

**User.objects.create(username=’personel’, email=’personel@learning.udm’)**

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() yaparak veritabanına kayıt edebiliriz.

Veritabanındaki verileri güncellemenin diğer bir yolu da update() methodunu kullanmak. 1 numaralı kullanıcının email adresini güncelleyelim. update() methodu geriye güncellenen nesne sayısını döndürür.

**User.objects.filter(id=1).update(email=’update@learning.udm’)**

------------------ veritabanından 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. Veritanbanındaki kullanıcıları users değişkenine atadık. Şimdi bunları ekrana bastıralım.

**users = User.objects.all()**

**for user in users:**

**print(user)**

Sorgu setinin ilk nesnesini getirmek için **first()** methodunu kullanırız. İlk nesnenin bilgilerini user değişkenine atayalım ve daha sonra elemanlarına ulaşalım.

**user = User.objects.first()**

**user.username**

**user.is\_staff**

Sorgu setinin son nesnesini getirmek için **last()** methodunu kullanırız.

**User.objects.last()**

Eğer kaydı nasıl çekeceğimizi biliyorsak **get()** kullanırız. Sadece bir tane kayıt gelmesi gerekmektedir.

Eğer birden fazla kayıt gelirse **[MultipleObjectsReturned](https://docs.djangoproject.com/en/2.1/ref/exceptions/" \l "django.core.exceptions.MultipleObjectsReturned" \o "django.core.exceptions.MultipleObjectsReturned)** hatası döner.

Eğer kayıt gelmez ise **[DoesNotExist](https://docs.djangoproject.com/en/2.1/ref/models/instances/" \l "django.db.models.Model.DoesNotExist" \o "django.db.models.Model.DoesNotExist)** hatası verir. Bunun için genellikle unique (benzersiz) alanlar için veya primary\_key ile birlikte kullanılır.

**User.objects.get(id=1)**

Sorgu setindeki kayıt tarihi en son olan nesneyi getirmek için latest() methodu kullanılır

**User.objects.latest(date\_joined)**

Sorgu setindeki kayıt tarihi en önce olan nesneyi getirmek için earliest() methodu kullanılır

**User.objects.earliest(date\_joined)**

Sorgu setinde kayıt olup olmadığını tespit etmek için exists() methodu kullanılır. Eğer kayıt varsa True yoksa False döner.

**User.objects.filter(id=1).exists()**

------------------ veritabanına veriyi çek, eğer veri yoksa ekle ------------------

**get\_or\_create()** metodu ile veritabanından bir veriyi çekmeye çalıştığımızda eğer varsa çekeriz yoksa çekmeye çalıştığımız veriyi veri tabanına ekletebiliriz. get\_or\_create() geriye tupple olarak döner.

obj olarak eklenen veya çekilen nesne bilgisini döndürür

created olarak da eğer eklendiyse True, çekildiyse False döndürür.

defaults kısmında eklenecek olan diğer alanları belirtiriz.

**obj, created = User.objects.get\_or\_create(**

**username='emre',**

**defaults={'email': ‘emre@learning.udm’},**

**)**

------------------ veritabanındaki veriyi güncelle, eğer veri yoksa ekle ------------------

**update\_or\_create()** methodu ile veritabanından bir veriyi güncellemeye çalıştığımızda eğer veri varsa güncellenir yoksa güncellemeye çalıştığımız veri veritabanına ekletebiliriz. update\_or\_create() geriye tupple olarak döner.

obj olarak eklenen veya çekilen nesne bilgisini döndürür

created olarak da eğer eklendiyse True, güncellendiyse False döndürür.

defaults kısmında eklenecek olan diğer alanları belirtiriz.

**obj, created = User.objects.update\_or\_create(**

**username ='Emre',**

**defaults={'email': ‘emre@learning.udm’},**

**)**

------------------ veritabanından toplu veri çekme ------------------

in\_bulk() methodu ile veritabanındaki kayıtları primary key veya benzersiz alan ile eşleştirerek çekebiliriz. dict olarak döner.

User tablosundaki tüm kayıtları getirir.

**User.objects.in\_bulk()**

User tablosundaki primary keyi 1 ve 2 olan kayıtlar

**User.objects.in\_bulk([1,2])**

Blog tablosunda slug alanı filtered\_word olan kayıtları getirir.

**Blog.objects.in\_bulk(['filtered\_word'], field\_name='slug')**

------------------ veritabanına toplu ekleme ------------------

bulk\_create(obj) methodu ile verilen nesne listesi bir tane sorgu çalıştırılarak veritabanına eklenir.

**User.objects.bulk\_create([**

**User(username='Emre'),**

**User(username ='Emre'),**

**], 1)**

------------------ veritabanında toplu güncelleme ------------------

bulk\_update() methodu ile verilen nesne listesi bir tane sorgu çalıştırılarak veritabanına eklenir. Veritabanından User modelindeki tüm nesneleri çekelim. Daha sonra 0 ve 1. Elemanlarının is\_staff bilgisini değiştirelim ve is\_staff bilgisini toplu olarak güncelleyelim.

**objects = User.objects.in\_bulk()**

**objects[0].is\_staff = 1**

**objects[1].is\_staff = 1**

**User.objects.bulk\_update(update.values(), ['is\_staff'])**

------------------ veritabanından veri silme ------------------

Veritabanından veri silmek için delete() methodunu kullanırız. Bu kod veritabanındaki primary\_key 2 olan kaydı siler ve geriye silinen nesne sayısını döndürür.

**user = User.objects.get(pk=2)**

**user.delete()**

Tüm ürünleri silmek için ise bu şekilde bir kod çalıştırabiliriz. Fakat veritabanına yeniden veri eklememiz gerekeceği için çalıştırmıyorum. Siz deneyebilirsiniz.

**products = Product.objects.all()**

**products.delete()**

------------------ veritabanındaki verileri sayma ------------------

count() methodu sorgu setindeki nesne sayısını döndürür.

**User.objects.count()**

veritabanındaki toplam kullanıcı sayısı

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

veritabanındaki is\_staff=1 olan kullanıcı sayısı

Eğer QuerySet oluşturulmuşsa count() methodunun çalıştırması ekstra performans kaybında neden olur böyle durumlarda len() kullanılması daha uygundur.

**users = User.objects.all()**

**user.count()** # yanlış kullanım

**len(users)** # doğru kullanım

Bu videomuzda veritabanına nasıl veri ekleyeceğimizi, güncelleyeceğimizi ve listeleyebileceğimizi öğrendik.

Bir sonraki videomuzda ise sorgu setlerindeki kayıtların nasıl filtreleneceğini öğreneceğiz.

------------------ DERS7 : DJANGO SHELL VE VERİTABANI İŞLEMLERİ – FİLTRELEME ------------------

Merhaba arkadaşlar,

Bir önceki videomuzda veritabanına nasıl veri ekleyeceğimizi, güncelleyeceğimizi ve listeleyebileceğimizi öğrendik.

Bu videomuzda ise sorgu setlerindeki kayıtların nasıl filtreleneceğini öğreneceğiz.

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.

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

Bu videomuzda sorgu setlerindeki kayıtların nasıl filtreleneceğini öğrendik.

Bir sonraki videomuzda sorgu setlerini sıralama ve diğer methodları öğreneceğiz.

----- DERS8 : DJANGO SHELL VE VERİTABANI İŞLEMLERİ – SIRALAMA – DİĞER METHODLAR------

Merhaba arkadaşlar,

Bir önceki videomuzda sorgu setlerindeki kayıtların nasıl filtreleneceğimizi öğrendik.

Bu videomuzda ise sıralama ve diğer methodları öğreneceğiz.

Umarım derslerden sonra tekrar ediyor ve methodları öğreniyorsunuzdur. Bu methodların çoğunu ileriki konularda kullanacağız.Diğer konularda methodları kullanırkende ne işe yaradıklarına değineceğim fakat eksik bilgi ile yola devam edersek ileriki konularda sorun çekeriz. Lütfen öğrendiklerimiz ile bol pratik yapalım.

------------------ 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 parameter 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

Parametre olarak ? girerek verileri rasgele sıralayabiliriz.

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

İlişkili alana göre sıralama için parametre 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 parametre 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()[6:10]**

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()**

------------------ değer metodları ------------------

Sorgu seti nesne olarak dönmektedir. values() methodu ile dict (sözlük) olarak döndürülmesi sağlanır.

**User.objects.values()**

Values methoduna parametre olarak alan isimleri girilerek istenilen alanlar filtrelenebilir.

**User.objects.values(‘username’, ‘email’)**

values\_list() methodu sorgu setinin tupple olarak döndürülmesini sağlar.

**User.objects.values\_list()**

values() methodu gibi values\_list() methodunda da istenilen alanlar filtrelenebilir. Flat=True kullanarak sadece değerlerin dönmesini sağlayabiliriz.

**User.objects.values\_list(‘username’, ‘email’)**

**User.objects.values\_list(‘id, flat=True)**

Spesifik bir alanın değerini alabiliriz. Örnek olarak 1. numaralı kullanıcının kullanıcı adını alabiliriz.

**User.objects.values\_list(‘username’, flat=True).get(pk=1)**

Birincil veritabanımızdan using default kullanarak da veri çekebiliriz.

**Blog.objects.using(‘default’)**

defer() metodu sorgu setinden çıkarmak istediğimiz alanlar için kullanılır. User sorgu setinden is\_staff ve active alanlarını çıkartmak için parameter olarak göndermemiz gerekmektedir.

**user = User.objects.defer(‘is\_staff’, ‘active’)**

Çıkartılan alanları sorgu setine geri eklemek için parameter olarak None göndermek yeterlidir.

**user.defer(None)**

only() metodu ile sorgu setinden sadece parametre olarak girilen kayıtlar getirilir. Sadece kullanıcı adını çekmek için parametre olarak username göndermemiz yeterlidir.

**user = User.objects.only(‘username’)**

none() methodu boş bir queryset döndürmeye yarar

**User.objects.none()**

------------------ karşılaştırma metodları ------------------

distinct() metodu ile bir sorgu setindeki benzer kayıtları temizleyebiliriz.

**User.objects.distinct()**

union() methodu birden fazla sorgu setini birleştirmeye yarar. Sorgu setleri birleştirilirken alan sayıları eşit olmalıdır. Eğer eşit olmaz ise OperationalError hatası döner.

User tablosundan çektiğimiz veriler ile Product tablosundan çektiğimiz verileri birleştirelim. Birleştirirken default olarak benzersiz kayıtları listeye ekler. Tüm verilerin eklenmesini istiyorsanız parametre olarak all = True girilmelidir. 2 den fazla sorgu setini de birleştirebiliriz.

**query\_set1 = Product.objects.values\_list('name)**

**query\_set2 = User.objects.values\_list(username)**

**query\_set1.union(query\_set2, all=True).order\_by('name’)**

**query\_set1.union(query\_set2, query\_set3)**

intersection() methodu birden fazla sorgu setindeki kesişen nesnelerin listesini döndürür. union() methodu gibi alan sayıları eşit olmalıdır. 2 den fazla sorgu setindeki kesişen nesne listesini de alabiliriz. Bir sorgu setinin kendisiyle intersectionu alınırsa tüm nesneler kesişeceği için aynı sorgu seti geri döner.

**query\_set1.intersection(query\_set2, query\_set3)**

difference() methodu geçerli sorgu setindeki nesneler ile diğer sorgu setindeki nesneleri karşılaştırır ve diğer sorgu setleri ile kesişmeyen nesneleri döndürür. Bir sorgu setinin kendisiyle difference alınırsa tüm nesneler kesişeceği için sorgu seti boş döner.

**query\_set1.difference(query\_set2, query\_set3)**

Birden çok veritabanı kullanıyorsak using() methodu ile istediğimiz veritabanından veri çekebiliriz. Önceki konularımızı hatırlarsan settings.py dosyasında veritabanı bilgilerimizi tutuyorduk. Orada default olarak belirlenen veritabanından veri çekmek için using() kullanımına gerek yoktur.

------------------ tarih metodları ------------------

dates() ve datetimes() methodu parametre olarak girilen alanın paramatre olarak girilen tür bazında benzersiz değerlerini sorgu seti olarak geri döndürür. Örnek ile daha iyi anlışacaktır. User tablosundaki kullanıcıların katıldıkları yılları tekil olarak azalan olarak sıralayacaktır. Order girilmezse default olarak artan sıralama yapılır. dates() methodu ile year, month, week, day parametre olarak kabul edilir. datetimes() methodu ile bunlara ilave olarak hour, minute, second kullanılabilir.

**User.objects.dates(‘date\_joined’, ‘year’, order=’DESC’)**

**extra()** methodu django query leri ile yapılamayacak çok karışık sorguların yapılması için eklenen bir methoddur. Django ekibi bu methodun kaldırılması için çalışmalar yürütmektedir. extra() methodu içerisine alınan parametreler sql injection güvenlik açığına karşı korumasızdır aldığınız verileri kontrol etmeniz gerekmektedir.

**raw()** methoduda extra() gibi karışık sorguların yapılması için eklenen bir başka method olup, bu da güvenlik açıklarına karşı korumasızdır.

Ben hiç bu methodları kullanma ihtiyacı duymadım. Bu eğitim dizisinde bu methodlar ile ilgili anlatım yapmayacağım. Eğer gerek olursa django dükümantasyondan bilgi alabilirsiniz.

----- DERS7 : DJANGO SHELL VE VERİTABANI İŞLEMLERİ – MODEL ALANLARI ------

**transaction da anlatalım**

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

**select\_for\_update()**

ilişkilerde anlatalım

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

annonate()

**select\_related()**

**prefetch\_related()**