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

Querysetler veritabanındaki bir nesne kolleksiyonunu temsil eder. Mesela Ürünler. QuerySetler sıralanabilir., filtrelenebilir. Bu işlemler için Manager metodlarını kullanırız.

view/products.py dosyamızı açalım ve product metodunu veritabanından tüm ürünleri getirecek şekilde düzenleyelim.

Öncelike veritabanı ilişki kurabilmek için modeli import etmemiz gerekmektedir. product ve user modelimizi import edelim.

from learning.models import Product

Tüm nesneleri listelemek için **all()** methodunu kullanırız. Veritanbanındaki ürünler product\_list değişkenine atadık. Şimdi bunları template gönderelim.

def products(request):  
  
 product\_list = Product.objects.all()  
  
 context = {  
 'products': product\_list  
 }  
  
 return render(request=request**,** template\_name='product/list.html'**,** context=context)

Template sayfamızda {{ products }} elemanını sayfaya basalım. Sunucumuzu kontrol ettiğimiz ekrana QuerySet olarak basıldığını görüyoruz.

Bu sayfa ürün listeleme sayfasıdır.  
{{ products }}

Biz bu querysetimizi satır satır ekrana basmak için döngüleri kullanmamız gerekiyor. Django’da templatelerde etiketlerin, filtrelerin, değişkenlerin nasıl kullanılacabileceğimiz hakkında bilgi verelim.

Değişkenler iki tane süslü parantez arasında kullanılır.

{{ products }}

Etiketler ise süslü parantez ve yüzde işareti ile kullanılır. Etiketler fonksiyonlar olabilir döngüler, if – else karar yapıları olabilir. Python kodu yazarken biliyorsunuzki süslü parantez hiç kullanmıyoruz. Girintiler ile hangi kodun hangi blok içerisinde olduğu anlıyor fakat templateler bu şekilde değil. Etiketleri açıyoruz ve kapatıyoruz. (bir nevi süslü parantez) Mesela döngüler için for ile döngünün başladığını belirtirken end for ile bittiğini bildiriyoruz.

{% for product in products %}

{% endfor %}

Filtrelere ise pythondaki or | işareti kullanılır.

{{ products | filtre }}

Bu öğrendiklerimizi template sayfamızda hemen uygulayalım.

Bu sayfa ürün listeleme sayfasıdır.  
  
<table>  
{% for product in products %}  
 <tr>  
 <td>Ürün Adı : {{ product.name | upper }}</td>  
 <td>Ürün Açıklaması : {{ product.content }}</td>  
 </tr>  
{% endfor %}  
</table>

Sunucumuzu çalıştıralım ve sayfamızı kontrol edelim. <http://127.0.0.1:8000/learning/product/>

Gördüğünüz gibi ürünlerimiz satır satır gelmişler. Ürün isminde upper kullandığımız için ürün isimleri büyük gelmiş.

Şimdi ürün detay sayfası ve ürün arşiv sayfasınıda güncelleyelim. Ürün detay sayfası ile başlayalım view dosyamızı açalım ve veritabanından ürünümüzü çekelim. 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/#django.core.exceptions.MultipleObjectsReturned) hatası döner.

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

def product(request**,** pk=None):  
  
 product\_detail = Product.objects.get(pk=pk)  
  
 context = {  
 'product': product\_detail  
 }  
   
 return render(request=request**,** template\_name='product/detail.html'**,** context=context)

Sunucumuzu çalıştıralım ve sayfamızı kontrol edelim. <http://127.0.0.1:8000/learning/product/detail/2/>

Olmayan id’ye bakalım. <http://127.0.0.1:8000/learning/product/detail/20/> . Gördüğünüz gibi hata olarak [**DoesNotExist**](https://docs.djangoproject.com/en/2.1/ref/models/instances/#django.db.models.Model.DoesNotExist) hatası olarak döndü.

Biz eğer ürün yoksa bunu farklı bir template sayfası getirebiliriz. Bu try except yapısı ile yakalayabiliriz. exceptions/notexists.html dosyası oluşturalım ve eğer ürünü bulamazsak buraya yönlendirelim.

def product(request**,** pk=None):  
  
 try:  
 product\_detail = Product.objects.get(pk=pk)  
 context = {  
 'product': product\_detail  
 }  
 return render(request=request**,** template\_name='product/detail.html'**,** context=context)  
  
 except ObjectDoesNotExist:  
 return render(request=request**,** template\_name='exceptions/notexists.html')

Şimdide arşiv sayfamızı güncelleyelim. view dosyamızı açalım ve veritabanından yıl ve aya göre ürünlerimizi listeleyelim.

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

filter içerisinde verilen argümanlar ile eşleşen kayıtları getirir.

# WHERE active=1

Products.objects.filter(active=True)

# WHERE active=1 AND name='Ürün Adı'

Products.objects.filter(active=True, name='Ürün Adı’)

exclude ise filter tam tersidir. verilen paratmere ile eşleşmeyen kayıtlar getirilir. Örneğimizde veritabanında active True olmayan ve name=’Ürün Adı’ olmayan kayıtlar sorgu setine eklenecektir.

# WHERE active != 1 AND name !='Ürün Adı'

Products.objects.exclude(active=True, name='Ürün Adı’)

Exclude ve filter zincirleme olarak kullanılabilir. Filter() zincirleme kayıtları birbirine AND ile exclude() zincirleme kayıtları ise OR ile bağlanır.

# WHERE active != 1 OR name !='Ürün Adı'

Products.objects.exclude(active=True).exclude(name='Ürün Adı’)

DJANGO bize filter(), exclude() ve get() methodları için alan araması imkanı sunar. Field lookupslar anahtar kelimenin yanına iki tane alt çizgi ile kullanılır. 1 den büyük olan kayıtları da getirebiliriz. Veya 1 içeren kayıtları. Bu İngilizce olarak field lookups diye geçmektedir. fields\_lookupları sorgu setlerinde kullandıkça öğreneceğiz.

Ürünleri ay ve yıla göre filtreleyebilmek için year ve month field lookuplarını kullanacağız.

@require\_http\_methods(["GET"**,** "POST"])  
def product\_archive(request**,** year=None**,** month=None):  
  
 product\_list = Product.objects.filter(created\_\_year=year**,** created\_\_month=month)  
  
 context = {  
 'year': year**,** 'month': month**,** 'products': product\_list**,** }  
  
 return render(request=request**,** template\_name='product/archive.html'**,** context=context)

Verilerimizi context olarak template gönderdik. Şimdi template dosyamızda verilerimizi ekrana basacak html kodlarımızı yazalım.

Bu sayfa {{ year }} / {{ month }} için ürün arşiv sayfasıdır.  
<br>  
{% if products %}  
 <table>  
 {% for product in products %}  
 <tr>  
 <td>Ürün Adı : {{ product.name | upper }}</td>  
 <td>Ürün Açıklaması : {{ product.content }}</td>  
 </tr>  
 {% endfor %}  
 </table>  
{% else %}  
 Ürün bulunamadı  
{% endif %}

Şimdi arşiv sayfamızdaki ürünlere link vererek detail sayfasına ulaşmasını sağlayalım.Bunun için ürün adına link verelim ve tıklayınca ürün detay sayfası olan "/learning/product/detail/{{ product.id }}" linkine gitsin. Burada product.id yerine product.pk de kullanabilirdik.

<td><a href="/learning/product/detail/{{ product.id }}">Ürün Adı : {{ product.name | upper }}</a></td>

Linkleri bu şekilde manuel vermek yerine URL Configürasyonlarını da kullanabiliriz. Kullanabilmek için routelarımıza name vermemiz gerekmetktedir. Urls.py dosyamızı açalım ve name verelim.

urlpatterns = [  
 path('example/'**,** views.examples)**,** path('product/'**,** views.products**,** name='product-list')**,** path('product/detail/<int:pk>/'**,** views.product**,** name='product-detail')**,** path('product/archive/<int:year>/<int:month>/'**,** views.product\_archive**,** name='product-archive')**,**]

template sayfamızdan url etiketi kullanarak linklerimizi path name kullanarak verebiliriz.

<td><a href="{% url 'product-detail' product.id %}">Ürün Adı : {{ product.name | upper }}</a></td>

Örnek olması açısından arşiv sayfası ve ürün sayfası linki de verelim.

2019 / 5. ay arşivleri için <a href="{% url 'product-archive' **2019 5** %}">tıklayın</a>

<br>  
  
Ürün listesi için <a href="{% url 'product-list'%}">tıklayın</a>

Ürün listeleme sayfasında tüm ürünlerimiz gelmektedir.

Djangoda sorgu setlerinde indexing (indexleme), slicing (dilimleme) ve stepping (adımlama) desteklenir. Veritabanına eklenen son 5 ürünü getirelim. Django da limitleme işlemleri aynı Python’daki gibidir.

Örnek üzerinde görmek için products view metodumuzu açalım ve üzerinde yapalım.

product\_list = Product.objects.all()[:**5**]

Gördüğünüz gibi 5 tane ürün ile sınırlandırdı. 6 ve 10. Arası kayıtlar için [5:10] yapabiliriz. [6:10:2] yaparak 6 - 10 arası kayıtlar arasından 2. Kayıtları alabiliriz. Sadece 2. indexte yer alan kaydı isteseydik [2] yapabilirdik. [0] yaptığımızda ilk kaydı getirdiğini unutmayalım [2] yaptığımızda 5. Kaydı getirir.

verileri ada göre artan sıralamak için order\_by method içinde argüman olarak name kullanmamız yeterli. Azalan sıralamada ise başına tire (eksi) işareti koyaraız. Rasgele bir sıralama istersek argüman olarak ? kullanırız.

Verileri isme göre sıralayıp ilk 5 kaydı getirmek istersek.

product\_list = Product.objects.order\_by(‘name’)[:**5**]

Ters yönlü sıralamak için ise başına – işareti ekleriz.

product\_list = Product.objects.order\_by(‘-name’)[:**5**]