----------- DATABASE OPTIMISATION -----------

Model katmanına son olarak Django’nun veritabanı optimizasyonu için bizelere vermiş olduğu tavsiyelerden bahsedeceğim. Bildiğiniz gibi bir veritabanı işlemini Django ile birden fazla şekilde yapabiliriz. Kayıtları saymak için count() mu kullanmalıyız yoksa len() mi ? Şimdi başlıklar altında veritabanı optimizasyonu hakkında bilgiler vereceğim.

Öncelikle oluştuduğumuz sorgu setlerinin hangi sorguları olşturduğunu bilmeliyiz. Sorgu setlerinin veritabanı için hangi sorgulara dönüştüğünü explain() metodunu kullanabiliriz. Diğer bir yöndemde django debug toolbar kullanmaktadır.

Veritabanı optimizasyonu tek başına yeterli değildir. Bazen veritabanını optimize ederken Python tarafında yapılan işlemlere zarar verebilir. Her ikisini birden dengeli bir şekilde yürütmek gereklidir.

**Standart Veritabanı Optimizasyon Teknikleri**

* index kullanımı

veritabanı optimizasyonu konusunda en önemli konulardan birisidir. exclude(), filter(), order\_by() ve benzeri methodların kullanıldığı alanlar için index oluşturulabilir.

* alan tiplerinin doğru kullanılması

misal olarak sayı tuttuğumuz alanları ayarlarken small integer kabul edilecek bir alan için integer kullanmak gereksiz yüke neden olacaktır.

**Sorgu Setlerini Anlamak**

Sorgu setlerinin işleyişini anlamak optimizasyon için önemlidir.

**sorgu setleri lazy çalışır.**

Sorgu seti oluşturulduğunda herhangi bir veritabanı işlemi yapılmaz. Sorgu sonuçları kullanılana kadar veritabanına erişim sağlanmaz. Sanki üç tane veritabanı işlemi yapmışız gibi ancak print(q) satırına kadar herhangi bir veritabanı işlemi yapmaz. Genel olarak bir sorgu seti siz isteyene kadar veri tabanından alınmaz.

q = Product.objects.filter(nane\_\_startswith="Product”)  
q = q.filter(created\_\_lte=datetime.date.today())  
q = q.exclude(content\_\_icontains="mutfak")  
print(q)

* Sorgu setleri ne zaman veritabanında değerlendirilir.
  + iteration yapıldığında

for e in Entry.objects.all():  
 print(e.headline)

* + limitleme yapıldığında genellikle sorgu setleri veritabanında değerlendirilmez. Fakat adımlama özelliği kullanılırsa veritabanına erişim sağlanır.
  + repr(), len(), list() metodlarının kullanılması ile veritabanına erişim sağlanır.

entry\_list = list(Entry.objects.all())

* + Sorgu setininden dönen kayıt olup / olmadığının tespit edildiği durumlarda veritabanına erişim sağlanır.

if Entry.objects.filter(headline="Test"):  
 print("There is at least one Entry with the headline Test")

* + Sorgu setininden dönen kayıt olup / olmadığının tespit edildiği durumlarda veritabanına erişim sağlanır.
* Veriler bellekte nasıl saklandığı / cachelendiği

Django, sorgu setlerini ilk defa çalıştırıp sonuçları getirdikten sonra bu sonuçları önbellekte saklar. Bu şekilde veritabanından iki kez sorgu yapılır

print([product.headline for product in Product.objects.all()])  
 print([product.pub\_date for product in Product.objects.all()])

tüm ürünleri önbelleğe alıp ordan kullanarak gereksiz yere ikinci bir sorgu yapmasını önleyebiliriz.

# Aktivite yok.  
products = Product.objects.all()  
  
# Sonuçlar veritabanından alınıp, önbelleğe yazılıyor.  
names = [product.name for product in products]  
  
# Önbellekteki sonuçlar kullanılıyor.  
slugs = [product.slug for product in products]

sorgu setleri her zaman ön belleğe atılmaz. Örneğin limitleme işlemleri önbelleğe atılmaz.

# Aktivite yok.  
products = Product.objects.all()  
  
# 5. Kayıt için sonuçlar veritabanından alınıp, önbelleğe yazılmaz.  
print(products[5])

# 5. Kayıt için yeniden veritabanı sorgusu yapılır, önbelleğe yazılmaz.  
print(products[5])

# ilk 5 kayıt için yeniden veritabanı sorgusu yapılır, önbelleğe yazılmaz.  
  
print(products[:5])

Sorgu setini önce önbelleğe alsak daha sonra limitleme işlemlerini yaparsak sürekli veritabanından sorgulama yapmaya gerek kalmaz.

# Aktivite yok.  
products = Product.objects.all()  
  
# veritabanından sorgulama yapıldı, ön belleğe alındı.

[product for product in products]

# ön bellekteki sonuçlar kullanılır

print(products[5])

print(products[:5])

çağıralabilir olmayan özellikler ön belleğe alınır.

# Aktivite yok.  
product = Product.objects.get(id=1)  
  
# veritabanından sorgulama yapıldı, ön belleğe alındı.

print(product.name)

print(product.name) # ön bellekteki sonuçlar kullanılır

print(product.title) # ön bellekteki sonuçlar kullanılır

print(product.authors.all()) # yeniden sorgulama yapılır

print(product.authors.all()) # yeniden sorgulama yapılır

**İşleri Python’da değil Veritabanı kısmında halledin**

* verileri filtrelemek için filter() ve exclude() kullanın
* aynı modeldeki diğer alanlara göre filtreleme için F() nesnesi kullanın
* annonate ve aggretion kullanın

**Verileri veritabanından benzersiz ve indekslenmiş sütunlardan almaya özen gösterin. Bu sorguların daha hızlı çalışmasını sağlayacaktır.**

**İhtiyacınız olan tüm verileri bir kerede almaya özen gösterin**

Biraz önce örneklerde çağıralabilen alanların yeniden sorgulama yaptığını söylemiştik. Eğer neleri alacağımızı biliyorsak select\_related() ve prefetch\_related() ile tek seferde alabiliriz.

**İhtiyacınız olmayan alanları almaktan kaçının**

values(), values\_list(), defer() ve only() alanlarını kullanarak sadece kullanacağız alanları veritabanından getirelim.

**count() ve exitsts() kullanalım.**

Sadece kayıt sayısını istiyorsak len() yerine count() kullanmalıyız. Eğer en az bir kayıt olduğunun tespitini istiyorsak if yerine exists() kullanmalıyız. Fakat sorgu seti daha önce yüklendiyse count ve exists kullanımı ek yük getireceğinden len ve if kullanmak daha uygun olur.

**update() ve delete() kullanalım.**

Öncelikle kayıdı getirip sonra onu tek tek güncellemek ve silmek yerine direk toplu update() ve delete() metodlarının kullanılması daha uygun olur.   
 **Eğer gerekmiyorsa order\_by ve meta ayarlarınındaki sıralamaların kullanımından kaçının.**