Django ORM Optimizasyonu

Fatih Erikli

Python & Django geliştiricisiyim.
Hipo'da çalışıyorum.

http://fatiherikli.com
http://github.com/fatiherikli/
http://twitter.com/fthrkl/

Optimizasyon

ORM optimizasyonundan önce standart veritabanı optimizasyonları yapılmalıdır.

- İndeksler
- Uygun veri tipleri
- ...

Profilleme

Sonraki iş ise hangi sorguların optimize edileceğine karar vermektir.

- Django-debug-toolbar
- Sql printing middleware
- Django test case

Queryset'leri anlamak

Queryset'ler tembeldirler

Bu güzel bir şey.

Queryset'ler tembeldirler

Sadece çalışması gerektiği zaman çalışırlar.

```
>>> qs = Foo.objects.all()
>>> qs = qs.filter(published=True)
```

Çalışan sorgu sayısı: 0

Ne zaman çalışırlar

- boolean(qs)
- repr(qs)
- qs[10:20]
- len(qs)
- iter(qs)

Queryset'ler tembeldirler

Aynı şeyleri tekrar tekrar hesaplamak yerine zaten hesapladığı şeyi size tekrar verirler.

```
>>> qs = Foo.objects.all()
>>> qs = qs.filter(published=True)
>>> list(qs)
    ....
>>> list(qs)
    ...
```

Çalışan sorgu sayısı: 1

Foreign-key'ler de tembel

```
>>> post = Post.objects.get(id=1) # ilk sorgu
>>> post.category # ikinci sorgu
    <Category: Personal>
>>> post.category # burada sorgu çalışmadı
    <Category: Personal>
```

N+1 sorgu problemi

Queryset'leri listelerken yazdırılan her Foreign-Key şeklinde ilişkilendirilmiş kayıt için ayrı sorgu çalıştırılması durumu.

Örnek: 10 adet blog post'umuz var

```
posts = Post.objects.all()
for post in posts:
    print post.category
```

Çalışan sorgu sayısı: 11

select_related

N+1 problemini çözmek için bu queryset metodu kullanılabilir.

```
posts = Post.objects.select_related("category")
for post in posts:
    print post.category
```

Çalışan sorgu sayısı: 1

Many-to-Many
One-to-Many problemleri

Yine bir queryset'i listelerken bir kaydın diğer tablodaki birden çok kaydını yazdırmak istediğimizde karşımıza çıkan problem.

Örnek: 10 adet blog post'umuz var.

```
posts = Post.objects.all()
for post in posts:
    print post.tags.all()
```

Çalışan sorgu sayısı: 11

prefetch_related

Bu problemi çözmek için ise bu queryset metodunu kullanabiliriz.

Örnek: 10 adet blog post'umuz var.

```
posts = Post.objects.prefetch_related("tags")
for post in posts:
    print post.tags.all()
```

Çalışan sorgu sayısı: 2

Prefetch_related iki adet sorgu çalıştırıyor.

- İlk sorgu queryset'in kendi sorgusu
- İkinci sorgu ise ilk sorguda aldığı id'ler ile ilişkili tablodaki kayıtları getirmek için yaptığı `IN` sorgusu.

Tek sorguya indirebilir miyiz?

Hayır, çünkü gerek yok.

Sorgu Sayısı != Performans

Gereksiz veriler

Yine Django ORM ile çok fazla alakası olmayan bir durum. Optimizasyon yapılacak bir view'da gereksiz hiç bir veri bırakmamak gerekir.

Ancak Django ORM'de bunu kolaylaştıracak bazı teknikler bulunmaktadır.

values metodu

Sadece istediğiniz alanların verilerini dictionary şeklinde listesini getirir.

```
posts = [{
    "id": post.pk
    "title": post.title
} for post in Post.objects.all()]
```

Pitonik :) fakat mükemmel değil.

```
posts = Post.objects.values("id", "title")
# [{"id": 2, "title": "test"}, ... ]
```

```
posts = Post.objects.values_list("id", flat=True)
# [2,3,4,5 ... ]
```

Mükemmel

Aggregation

Aggregation (sum, count, average vb.) işlemleri her zaman veritabanı katmanında daha hızlıdır.

```
posts = Post.objects.all()
print len(posts)
# tüm queryset çalıştırılıp python tarafında hesaplanır.
```

```
posts = Post.objects.all()
print posts.count()
# select count(1) from foobar;
# gibi bir sql sorgusu çalıştırılır.
```

Aggregation sayılmaz, ancak önemli bir ayrıntı.

```
posts = Post.objects.all()
print len(posts)
# tüm queryset çalıştırılıp python tarafında hesaplanır.
```

```
posts = Post.objects.all()
print posts.count()
# select count(1) from foobar;
# gibi bir sql sorgusu çalıştırılır.
```

Aggregation sayılmaz, ancak önemli bir ayrıntı.

```
categories = Category.objects.all()

for category in categories.all():
    print category.name, category.posts.count()
```

Kötü bir örnek

```
categories = Category.objects.prefetch_related("posts")
for category in categories.all():
    print category.name, category.posts.count()
```

Biraz daha iyi

```
categories = Category.objects.aggregate(
    post_count = Count("posts")
)
for category in categories.all():
    print category.name, category.post_count
```

Mükemmel

Testing

Sql sorgularını test etmek iyi bir pratik olabilir.

django.db.queries

Testler Debug=False şeklinde çalıştığı için Django'nun sorgu loglarını bu şekilde alamazsınız.

```
from django.test import TestCase


class FooTest(TestCase):
   def test_foo(self):
       self.assertNumQueries(0, Post.objects.all)
       self.assertNumQueries(1, Post.objects.count)


       def _test():
            for post in Post.objects.all():
                print post.title
       self.assertNumQueries(1, _test)
```

Django'nun TestCase sınıfı ile bu problemi aşabilirsiniz.

Ufak tefek optimizasyon ipuçları

```
post.category.id # burada bir sorgu çalışır
```

post.category_id # bu şekilde alınırsa sorgu çalışmaz

Foreign-Key'ler veritabanında foo_id şeklinde saklanır.

```
{% with posts.count as post_count %}
    {% if post_count %}
       {{ post_count }}
    {% endif %}
{% endwith %}
```

Template üzerinde `with` ile caching yapabilirsiniz.

```
from django.db.models import F
Post.objects.update(view_count = F("view_count") + 1)
```

`F` ile update ve filter metodunda modeldeki başka bir field'ı referans olarak gösterebilirsiniz.

Teşekkürler