In Search of Reduced Loading Times

Apostolis Bessas
mpessas@transifex.com
@mpessas

July 6, 2012

Transifex

Optimizing SQL

Measure

- ➜ django-debug-toolbar
- ➜ django-devserver
- ➜ `django.db.backends` logger
- ➜ Database logging
  - `log_min_duration_statement` in PostgreSQL

Evaluations

```
qs = TodoItem.objects.filter(owner__username='me')
nitems = qs.count()
return render_to_response(
    'template.html', {'nitems': nitems, 'items': qs},
    context_instance = RequestContext(request)
)
```

Less queries

- `select_related()` for OneToOneField and ForeignKeyField
- `prefetch_related()` for ManyToManyField (and the reverse direction of ForeignKey)

select_related

```
User.objects.select_related('profile').\
    filter(username='mpessas').query
```

select_related

```
User.objects.select_related('profile').\
    filter(username='mpessas').query
```

```
SELECT auth_user.*, txcommon_userenaprofile.*
FROM auth_user
LEFT OUTER JOIN txcommon_userenaprofile
ON (auth_user.id = txcommon_userenaprofile.user_id)
WHERE auth_user.username = 'mpessas'
```

prefetch_related

```
Pizza.objects.all().prefetch_related('toppings')
```

prefetch_related

```
Pizza.objects.all().prefetch_related('toppings')
```

```
SELECT "t_pizza".* FROM "t_pizza"

SELECT ("t_pizza_toppings"."pizza_id")
    AS "_prefetch_related_val",
    "t_topping"."id", "t_topping"."name"
FROM "t_topping"
INNER JOIN "t_pizza_toppings"
ON ("t_topping"."id" = "t_pizza_toppings"."topping_id")
WHERE "t_pizza_toppings"."pizza_id" IN (...)
```

.iterator()

Don’t cache database results unnecessarily.

annotate

Use `values()` before `annotate()`.

Example

```
Resource.objects.filter(
    project__slug='transifex'
).annotate(sw=Sum('wordcount'))
```

```
GROUP BY id, slug, name, i18n_type,
accept_translations, total_entities,
wordcount, category, created, last_update,
source_language_id, project_id, _order
```

Example

```
Resource.objects.filter(
    project__slug='transifex'
).values('slug').annotate(sw=Sum('wordcount'))
```

```
SELECT slug, SUM(wordcount) AS sw
FROM resources_resource INNER JOIN projects_project
ON (
  resources_resource.project_id = projects_project.id
)
WHERE projects_project.slug = 'transifex'
GROUP BY resources_resource.slug
```

Raw SQL

Don’t be afraid to use raw SQL queries.

Raw SQL

Don’t be afraid to use raw SQL queries.

- ➜ Manager.raw()
- ➜ `django.db.connection.cursor()`

RawQuerySet

✓ Something like `QuerySet`.

RawQuerySet

✓ Something like `QuerySet`.

✗ But it is not a `QuerySet`.

RawQuerySet

- ✓ Something like `QuerySet`.
- ✗ But it is not a `QuerySet`.
- ✓ Objects are valid models.

RawQuerySet

- ✓ Something like `QuerySet`.
- ✗ But it is not a `QuerySet`.
- ✓ Objects are valid models.
- ✓ Allows for complex queries.

Advanced Queries

```
SELECT id, string
FROM resources_translation tr
JOIN
(SELECT *
FROM crosstab(
'SELECT source_entity_id, language_id, string
FROM resources_translation
WHERE (language_id = 20 OR language_id = 19)
AND resource_id=4156
ORDER BY 1,2 DESC'
) AS t(row_name INTEGER, english text, greek text)
WHERE greek IS NULL
) AS ct
ON tr.source_entity_id = ct.row_name
WHERE language_id=20
```

defer() and only()

defer: Columns to omit from the SELECT list.

only: Columns to specify in the SELECT list.

True story

```
Project.objects.exclude(
    private=True
).distinct().order_by('name')
```

True story

```
Project.objects.exclude(
    private=True
).distinct().order_by('name')
```

```
SELECT DISTINCT projects_project.*
FROM projects_project
WHERE NOT (projects_project.private = True)
ORDER BY projects_project.name ASC
```

True story

```
Project.objects.exclude(
    private=True
).distinct().order_by('name')
```

```
SELECT DISTINCT projects_project.*
FROM projects_project
WHERE NOT (projects_project.private = True)
ORDER BY projects_project.name ASC
```

http://bit.ly/PgT2Am

Bulk operations

➜ `bulk_create`

Bulk operations

- `bulk_create`
- django-bulk

Bulk operations

- `bulk_create`
- django-bulk
- COPY (for PostgreSQL)

Bulk operations

- ➜ `bulk_create`
- ➜ django-bulk
- ➜ COPY (for PostgreSQL)

⇒ Take advantage of the native features of your database.

Denormalization

. . . denormalization is the process of attempting to optimise the read performance of a database by adding redundant data or by grouping data.

Wikipedia

Denormalization

. . . denormalization is the process of attempting to optimise the read performance of a database by adding redundant data or by grouping data.

Wikipedia

➜ Mostly for read-only data.

Meta.Options.ordering

Don’t use that.

Meta.Options.ordering

Don’t use that.

```
TodoItem.objects.filter(owner__username='me').delete()
```

Caching

Sessions

Don’t hit the database for sessions.

Template compilation

```
django.template.loaders.cached.Loader
```

Entity Tags/Last-Modified Dates

- ➜ Allow to use browser cache (304 HTTP status code).
- ➜ Worth, only if easier to calculate.

Entity Tags/Last-Modified Dates

- ➜ Allow to use browser cache (304 HTTP status code).
- ➜ Worth, only if easier to calculate.
- ➜ What about *personalized* pages?

Optimizing algorithms

Running times

```
for hash, s in translations:
    re.sub(hash, s, text)
```

Running times

```
for hash, s in translations:
    re.sub(hash, s, text)
```

```
hashes = {}
for hash, s in translations:
    hashes[s] = hash
re.sub(
    r'[0-9a-f]{32}', lambda s: return hashes[s], text
)
```

I/O

- ➜ Threads for I/O
- ➜ Async I/O

PJAX

django-pjax

Questions?