

# **Speeding up a Django project**

Paweł Marczewski

# What this talk will be about

Django.

Some advice is Postgres-specific  
(but should be easy to adapt).

Mostly data-processing / database  
performance.

I hope for some advice from you as well!

# Codility

A service for testing programming skills (think olympiad/contests, but with simple problems).

The website uses Django and PostgreSQL.

No strong backend/frontend divide (yet).

No huge amounts of data, but we're running into performance problems from time to time.

Here's what we came up with...

The screenshot shows the Codility interface for a task. The task description is in English and asks for an equilibrium index in an array. A code editor on the right shows a C++ solution. Below the editor, there are buttons for HELP, RUN, SUBMIT THIS TASK, and QUIT. A status bar indicates 'Running solution...' and 'Compilation successful'. The returned value is 4, and the example test passes.

```
1 // you can write to stdout for debugging purposes, e.g.
2 // printf("this is a debug message\n");
3
4 int solution(int A[], int N) {
5     return 3;
6 }
```

Running solution...  
Compilation successful.  
Your test case: [4]  
Returned value: 3  
Example test: [-1, 3, -4, 5, 1, -6, 2, 1]  
OK  
Your code is syntactically correct and works properly on the example test.

The screenshot shows the Codility website interface for a user named Pawel. The page displays a list of test results for a specific task. The table has columns for Name, Test, Created, Taken, Status, Invitation, Result, Test link, and Similarity. The results show a list of anonymous users with their test status and results.

	Name	Test	Created	Taken	Status	Invitation	Result	Test link	Similarity
<input type="checkbox"/>	Anonymous		5 days ago	5 days ago	closed		50 / 100		
<input type="checkbox"/>	Anonymous	Bla	10 days ago	10 days ago	closed		0 / 100		
<input type="checkbox"/>	Anonymous		19 days ago	19 days ago	closed		0 / 100		
<input type="checkbox"/>	Anonymous		19 days ago	19 days ago	closed		0 / 100		
<input type="checkbox"/>	Anonymous		23 days ago	23 days ago	closed		0 / 100		
<input type="checkbox"/>	Anonymous		23 days ago	23 days ago	closed		0 / 100		
<input type="checkbox"/>	Anonymous		23 days ago	23 days ago	closed		0 / 100		
<input type="checkbox"/>	Anonymous		23 days ago		not started			Copy	
<input type="checkbox"/>	Anonymous		a month ago	a month ago	closed		0 / 100		
<input type="checkbox"/>	Anonymous		a month ago	a month ago	closed		0 / 100		
<input type="checkbox"/>	Anonymous	QA	a month ago	a month ago	closed		0 / 300		
<input type="checkbox"/>	Anonymous		a month ago	a month ago	closed		0 / 0		

# Use SQLite-in-memory for unit tests

```
DATABASES['default'] = {  
    'ENGINE': 'django.db.backends.sqlite3',  
    'NAME': ':memory:',  
}
```

Blazing-fast startup time!

# Use SQLite-in-memory for unit tests

It's good to test on production engine as well (your CI server can do both).

Your code has to support SQLite.

Alternative (Postgres): turn off *fsync* for tests.

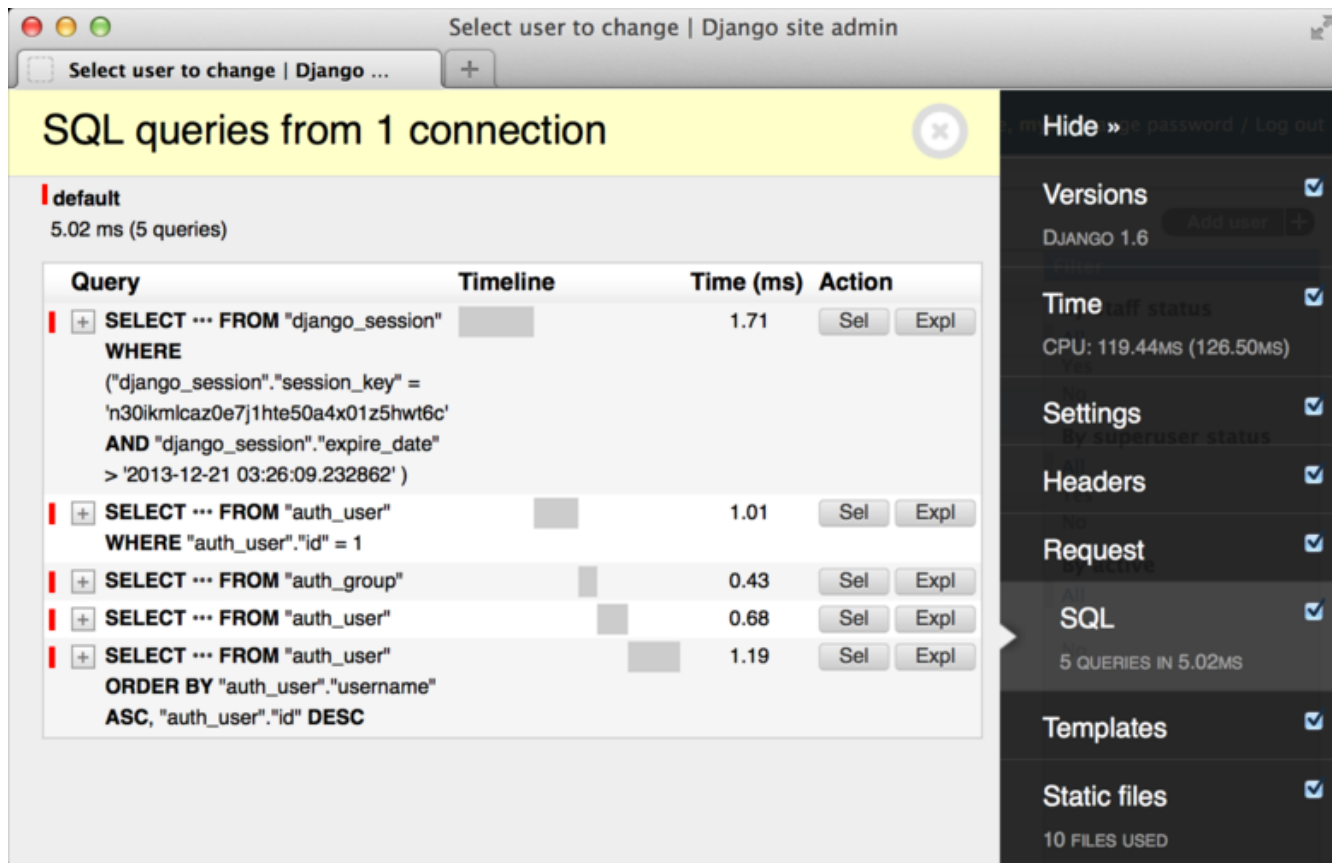
# Other testing tips

If migrations are a bottleneck, you can squash them.

Parallelize your builds (useful if you have many Selenium tests).

# Check your queries

*django-debug-toolbar*



The screenshot shows the Django Debug Toolbar interface. The main panel displays "SQL queries from 1 connection" for a "default" connection, which took 5.02 ms to execute (5 queries). The toolbar is currently set to "SQL" mode, showing 5 queries in 5.02ms. The right sidebar contains various toolbars: Versions (DJANGO 1.6), Time (CPU: 119.44ms (126.50ms)), Settings, Headers, Request, SQL (selected), Templates, and Static files (10 FILES USED).

Query	Timeline	Time (ms)	Action
<code>SELECT ... FROM "django_session" WHERE ("django_session"."session_key" = 'n30ikmlcaz0e7j1hte50a4x01z5hwt6c' AND "django_session"."expire_date" &gt; '2013-12-21 03:26:09.232862' )</code>		1.71	Sel Expl
<code>SELECT ... FROM "auth_user" WHERE "auth_user"."id" = 1</code>		1.01	Sel Expl
<code>SELECT ... FROM "auth_group"</code>		0.43	Sel Expl
<code>SELECT ... FROM "auth_user"</code>		0.68	Sel Expl
<code>SELECT ... FROM "auth_user" ORDER BY "auth_user"."username" ASC, "auth_user"."id" DESC</code>		1.19	Sel Expl

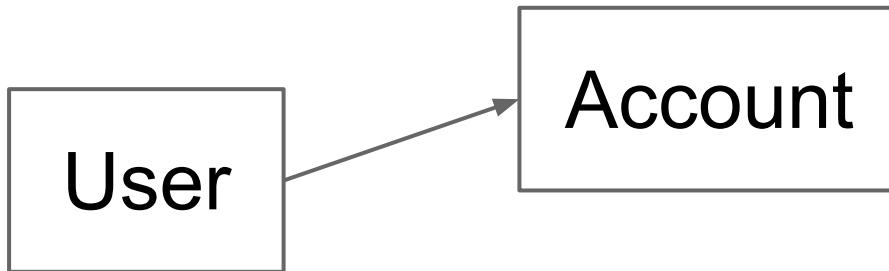
# Check your queries

Or, just look at plain Django logs  
(DEBUG level).

```
DEBUG 2015-04-28 19:02:26,290 utils 23888 140174780528384 (0.001) SELECT "auth_user"."id", "auth_user"."password", "auth_
DEBUG 2015-04-28 19:02:26,296 utils 23888 140174780528384 (0.004) SELECT COUNT(*) FROM "tickets_ticketmailstatus" INNER J
DEBUG 2015-04-28 19:02:26,298 utils 23888 140174780528384 (0.001) SELECT "tasks_task"."id", "tasks_task"."name", "tasks_t
DEBUG 2015-04-28 19:02:26,301 utils 23888 140174780528384 (0.001) SELECT "auth_user"."id", "auth_user"."password", "auth_
DEBUG 2015-04-28 19:02:26,315 utils 23888 140174780528384 (0.001) UPDATE "django_session" SET "session_data" = 'ZWMxYjg1Z
DEBUG 2015-04-28 19:02:27,458 utils 23888 140174780528384 (0.009) SELECT "django_session"."session_key", "django_session"
DEBUG 2015-04-28 19:02:27,464 utils 23888 140174780528384 (0.004) SELECT "auth_user"."id", "auth_user"."password", "auth_
DEBUG 2015-04-28 19:02:27,468 utils 23888 140174780528384 (0.002) SELECT "profiles_userdata"."account_id", "profiles_user
DEBUG 2015-04-28 19:02:27,472 utils 23888 140174780528384 (0.002) SELECT "profiles_userprofile"."id", "profiles_userprofi
INFO 2015-04-28 19:02:27,475 log 23888 140174780528384 user access, username=pawel@codility.com, method=GET, path=/dashbo
DEBUG 2015-04-28 19:02:27,479 utils 23888 140174780528384 (0.001) SELECT "profiles_userdata"."account_id", "profiles_user
DEBUG 2015-04-28 19:02:27,482 utils 23888 140174780528384 (0.001) SELECT "auth_user"."id", "auth_user"."password", "auth_
DEBUG 2015-04-28 19:02:27,484 utils 23888 140174780528384 (0.001) SELECT "auth_user"."id", "auth_user"."password", "auth_
DEBUG 2015-04-28 19:02:27,490 utils 23888 140174780528384 (0.004) SELECT COUNT(*) FROM "tickets_ticketmailstatus" INNER J
DEBUG 2015-04-28 19:02:27,492 utils 23888 140174780528384 (0.001) SELECT "tasks_task"."id", "tasks_task"."name", "tasks_t
DEBUG 2015-04-28 19:02:27,495 utils 23888 140174780528384 (0.001) SELECT "auth_user"."id", "auth_user"."password", "auth_
"email": "pawel@codility.com", "username": "pawel@codility.com", "password": "pawel@codility.com")
```



# Use select\_related



ID	User	Account
1	foo@example.com	Foo
2	foo2@example.com	Foo
3	bar@example.com	Bar

# Use `select_related`

Rendering `User.objects.all()`:

```
SELECT ... FROM users;  
SELECT ... FROM accounts WHERE id = 1;  
SELECT ... FROM accounts WHERE id = 2;  
SELECT ... FROM accounts WHERE id = 3;  
...
```

# Use `select_related`

Rendering

`User.objects.select_related('account')`:

```
SELECT ... FROM users
      JOIN accounts
      ON users.id = accounts.user_id;
```

**Much better!**

# Use prefetch\_related

ID	Account	Users
1	Foo	foo1@example.com, foo2@example.com, foo3@example.com
2	Bar	bar1@example.com, bar2@example.com

# Use prefetch\_related

Rendering Account.objects.all():

```
SELECT ... FROM accounts;  
SELECT ... FROM users WHERE account_id = 1;  
SELECT ... FROM users WHERE account_id = 2;  
SELECT ... FROM users WHERE account_id = 3;  
...
```

# Use prefetch\_related

Rendering

Account.objects.prefetch\_related('user\_set'):

```
SELECT ... FROM accounts;
```

```
SELECT ... FROM users
```

```
    WHERE account_id in (1,2,3,4);
```

# Or just drop to raw SQL

```
Blog.objects.extra(  
    select={  
        'entry_count': 'SELECT COUNT(*) FROM blog_entry  
WHERE blog_entry.blog_id = blog_blog.id'  
    }  
)
```

General rule:

**Make  $O(1)$  queries per page.**



# Do more in SQL than in your code

Example: data migrations

(convert data from one format to another).

Your database engine will be *WAY* more efficient at this than Python!

A complicated “UPDATE WHERE...” can be orders of magnitude faster than a for-loop.

# Check what your queries are doing

Example from Postgres documentation.

```
EXPLAIN SELECT *  
FROM tenk1 t1, tenk2 t2  
WHERE t1.unique1 < 10 AND t1.unique2 = t2.unique2;
```

## QUERY PLAN

---

```
Nested Loop  (cost=4.65..118.62 rows=10 width=488)  
-> Bitmap Heap Scan on tenk1 t1  (cost=4.36..39.47 rows=10 width=244)  
    Recheck Cond: (unique1 < 10)  
        -> Bitmap Index Scan on tenk1_unique1  (cost=0.00..4.36 rows=10 width=0)  
            Index Cond: (unique1 < 10)  
-> Index Scan using tenk2_unique2 on tenk2 t2  (cost=0.29..7.91 rows=1 width=244)  
    Index Cond: (unique2 = t1.unique2)
```

# Check what your queries are doing

In Postgres, EXPLAIN will give you a query plan.

EXPLAIN ANALYZE will also run the query and give you the timing.

Often, you'll find out you need another index!

Even better: run statistics on the production database.

**PostgreSQL 9.0 High Performance** contains a good explanation of query plans and Postgres internals.

# Memoize properties

```
class User(models.Model):  
  
    @property  
    def available_credits(self):  
        return self.query_for_credits()
```

# Memoize properties

```
@property
def available_credits(self):
    if not hasattr(self, '_available_credits'):
        self._available_credits = self.query_for_credits()
    return self._available_credits
```

Useful for complicated pages.

Downside: cache invalidation.

# Use cache

Cache arbitrary data using `cache.get()` and `cache.set()`.

Cache template fragments.

Use *django-cache-machine* to cache models (useful for data that changes rarely but is accessed often).

# Compute things asynchronously

Example: our real time map widget.

Just now, a new test session started in

**United States**



**2,123,406**

Assessments till date

# Compute things asynchronously

Don't do this:

```
def get_map_data():
    if not cache.get('map_data'):
        data = compute_map_data()
        cache.set('map_data', data, 5 * 60)

    return cache.get('map_data')
```



# Compute things asynchronously

Use a task queue, like Celery.

Support a “not ready yet” response and initiate re-computation.

Or just compute things periodically.

Good for pages with lots of views (like contest leaderboards).

# Don't auto-reload, poll

codility

PDF

**Grzegorz Jakacki**

<b>Candidate</b> E-mail: foo@example.com Last school attended: University of Warsaw (Poland) Academic degree: Master of Science (MSc) Field of study: Computer sciences Profile URL: http://www.codility.com/ Notes: Not defined, yet	<b>Session</b> ID: 5VM69K-BEF Time limit: 90 min. Report recipients: hr@example.com Accessed from: 198.51.100.0 Invited by: demo@codility.com [view account]	<b>Status: closed</b> Created on: 2014-04-30 08:24 UTC Started on: 2014-04-30 08:25 UTC Finished on: 2014-04-30 08:48 UTC
---	---	--

Tasks in test

	Correctness	Performance	Task score
1   Equi Submitted in: C	60%	54%	58%
2   BugfixingLeader Submitted in: Python	100%	100%	100%
3   PtrListLen Submitted in: Python	100%	not assessed	100%

Test score

# 86%

258 out of 300 points

**1. Equi** score: 58 of 100

Find an index in an array such that its prefix sum equals its suffix sum.

**Task description**

This is a demo task. You can read about this task and its solutions in [this blog post](#).

A zero-indexed array  $A$  consisting of  $N$  integers is given. An *equilibrium index* of this array is any integer  $P$  such that  $0 \leq P < N$  and the sum of elements of lower indices is equal to the sum of elements of higher indices, i.e.

$$A[0] + A[1] + \dots + A[P-1] = A[P+1] + \dots + A[N-2] + A[N-1].$$

Sum of zero elements is assumed to be equal to 0. This can happen if  $P = 0$  or if  $P = N-1$ .

For example, consider the following array  $A$  consisting of  $N = 8$  elements:

```
A[0] = -1
A[1] =  3
```

**Solution**

Programming language used: C

Total time used: 24 minutes

Effective time used: 15 minutes

Notes: not defined yet

**Task timeline**

08:25:11 08:48:51

Our report used to auto-reload until it was assessed and ready.

However, polling a single AJAX endpoint (“is it ready yet?”) made the page less straining for our servers.

**If all else fails...**

Just use a stronger server!

**codility**

WE BOUGHT AN SSD

# What comes next for us?

More aggressive HTML fragment caching.

Better frontend / backend split (serve static HTML and JS, pass data using JSON).

Sharding / horizontal database scaling.

# Questions? Comments?

You can reach me at [pwmarcz@gmail.com](mailto:pwmarcz@gmail.com)  
and <http://pwmarcz.pl>.