Staying sane while defying

Joel Spolsky with Django
Hola, me llamo Nick

@nickbruun
nickbruun
http://bruun.co/
I’m the CTO of Iconfinder
“... the single worst strategic mistake that any software company can make: They decided to rewrite the code from scratch.”
LAMP stack

Linux, Apache, MySQL, PHP and memcached

(dv) (-ish)

16 cores, 16 GiB RAM, death star
Restructure

Rebuild
Two simple goals:

Fast and flexible

(aka code base nirvana)
April 2013

3M monthly visitors

3M requests per day

30GiB data in PostgreSQL
April 2013

83 ms
average response time

329 ms
99th percentile response time
Long story short... We survived! - and learned a few lessons in the process.
1. ORMs are stupid.
Watch it **like a boss**

# In postgresql.conf:

```plain
log_destination = 'csvlog'
log_directory = 'pg_log'
logging_collector = on
log_filename = 'postgres-%Y-%m-%d_%H%M%S'
log_rotation_age = 1d
log_rotation_size = 1GB
log_min_duration_statement = 0ms
log_checkpoints = on
log_connections = on
log_disconnections = on
log_lock_waits = on
log_temp_files = 0
```

dump every statement and its execution time
Watch it **like a boss**

Follow the tail of the log file

```
$ tail -f "/usr/local/var/postgres/pg_log/$(ls -1t /usr/local/var/postgres/pg_log/ | head -1)"
```
Did I say stupid?

```python
SomeModel.objects.filter(
    other = None
)

SomeModel.objects.filter(
    other_id = None
)
```
Did I say stupid?

```python
SomeModel.objects.filter(
    other = None
)
```

```python
SomeModel.objects.filter(
    other_id = None
)
```

same exact thing!
Did I say stupid?

```python
SomeModel.objects.filter(
    other = None
)
```

```python
SomeModel.objects.filter(
    other_id = None
)
```

```sql
SELECT
    "myapp_somemodell"."id",
    "myapp_somemodell"."arbitrary",
    "myapp_somemodell"."other_id"
FROM
    "myapp_somemodell"
LEFT OUTER JOIN "myapp_othermodel" ON (  
    "myapp_somemodell"."other_id" =  
    "myapp_othermodel"."id"
)
WHERE
    "myapp_othermodel"."id" IS NULL
```

```sql
SELECT
    "myapp_somemodell"."id",
    "myapp_somemodell"."arbitrary",
    "myapp_somemodell"."other_id"
FROM
    "myapp_somemodell"
WHERE
    "myapp_somemodell"."other_id" IS NULL
```
ORMs are **leaky abstractions**

If we don't want to waste resources like this:

```python
count_by_name = {m.name: m.other_set.count() for m in SomeModel.objects.all()}
```

The essence of SQL leaks through:

```python
count_by_names = {m['name']: m['num_other'] for m in SomeModel.objects.all().annotate(num_other=Count('other')).values('name', 'num_other')}
```
I, for one, welcome our SQL overlords
2. Monitor *everything*.
No, your log file probably isn't good enough.
Side effect:
awesome deployment workflow

build stuff
Side effect:
awesome deployment workflow
Side effect:
awesome deployment workflow
Side effect:
awesome deployment workflow
3. Don’t assume. Profile.
Hot startup optimization strategy
Hot startup optimization strategy

1. Database indexes
Hot startup optimization strategy

1. Database indexes
2. Page caching
Hot startup optimization strategy

1. Database indexes
2. Page caching
3. Database relationship denormalization
Hot startup optimization strategy

1. Database indexes
2. Page caching
3. Database relationship denormalization
4. Fine grained caching
Hot startup optimization strategy

1. Database indexes
2. Page caching
3. Database relationship denormalization
4. Fine grained caching
5. Throw money at it
Hot startup optimization strategy

1. Database indexes
2. Page caching
3. Database relationship denormalization
4. Fine grained caching
5. Throw money at it
6. Go to step 5
91 ms

original response time
“… the worst strategic mistake that any software company can make: They decided to rewrite the code from scratch.”
Japanese temple
91 ms
original response time

29 ms
response time with Jinja2
… the worst strategic mistake that any software company can make: They decided to rewrite the code from scratch.

29.16% of time spent escaping HTML
91 ms
original response time

29 ms
response time with Jinja2

20 ms
response time with Jinja2 without auto-escaping
4. Everything prepackaged isn’t good, and everything good isn’t prepackaged.
Watch your transactions
Watch your transactions

Transactions + Caching
Watch your transactions

Transactions + Caching + Background tasks
Watch your transactions

Transactions + Caching + Background tasks = ANGER!!1
Enqueuing tasks in transactions

Request

Background worker
Enqueuing tasks in transactions
Enqueuing tasks in **transactions**

Request

- Begin transaction
- Create model

Background worker
Enqueuing tasks in **transactions**

Request

- Begin transaction
- Create model
- Add task to queue

Background worker
Enqueuing tasks in **transactions**

Request

- Begin transaction
- Create model
- Add task to queue

Background worker

- Dequeue task from queue
Enqueuing tasks in **transactions**

Request:
- Begin transaction
- Create model
- Add task to queue

Background worker:
- Dequeue task from queue
- Get model
Enqueuing tasks in **transactions**

Request:
- Begin transaction
- Create model
- Add task to queue
- Dequeue task from queue
- Get model
  - raises Model.DoesNotExist

Background worker:
Enqueuing tasks in **transactions**

Request:
- Begin transaction
- Create model
- Add task to queue
- Commit transaction

Background worker:
- Dequeue task from queue
- Get model
- Raises Model.DoesNotExist
With **django-celery-transactions**

Request

Background worker
With **django-celery-transactions**

Request

Begin transaction

Background worker
With **django-celery-transactions**

Request

- Begin transaction
- Create model

Background worker
With **django-celery-transactions**

Request

- Begin transaction
- Create model
- Locally add task to queue

Background worker
With **django-celery-transactions**

Request:
- Begin transaction
- Create model
- Locally add task to queue
- Commit transaction

Background worker
With **django-celery-transactions**

- Begin transaction
- Create model
  - Locally add task to queue
- Commit transaction
  - Actually add task to queue

---

**Request**

---

**Background worker**
With **django-celery-transactions**

Request:
- Begin transaction
- Create model
- Locally add task to queue
- Commit transaction
- Actually add task to queue
- Dequeue task from queue

Background worker
With **django-celery-transactions**

Request:
- Begin transaction
- Create model
- Locally add task to queue
- Commit transaction
- Actually add task to queue

Background worker:
- Dequeue task from queue
- Get model

Dequeue task from queue
With **django-celery-transactions**

- Request:
  - Begin transaction
  - Create model
  - Locally add task to queue
  - Commit transaction
  - Actually add task to queue

- Background worker:
  - Dequeue task from queue
  - Get model
  - Finish task
Django is awesome
django is a trap!
Frameworks are a trap!
Thanks!

It’s been awesome listening to me.