Django: Under the Hood
November 3rd, 2016
Aymeric Augustin

debugging
performance
👋 I’m Aymeric

Core Developer since 2011

- Time zones
- Python 3
- Transactions
- App loading
- Jinja2

Freelancing since 2015

Consulting on Django, big data, & architecture
👋 I’m Aymeric

Core Developer since 2011

- Time zones
- Python 3
- Transactions
- App loading
- Jinja2

Founders & CTO since 2015

First collaborative insurance broker in France

https://otherwise.fr/ [FR]
response times
Perception of response time

- **fast** = up to **0.1s** = reacting instantaneously
  - just display the result

- **normal** = up to **1s** = not interrupting the user's flow of thought
  - user will notice the delay but no feedback is necessary

- **slow** = up to **10s** = keeping the user's attention focused
  - feedback is important, especially if response time is unpredictable

Source: [https://www.nngroup.com/articles/website-response-times/](https://www.nngroup.com/articles/website-response-times/)
measuring
page load time
performance.timing
### Resource usage

<table>
<thead>
<tr>
<th>Resource</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>User CPU time</td>
<td>37.485 msec</td>
</tr>
<tr>
<td>System CPU time</td>
<td>11.221 msec</td>
</tr>
<tr>
<td>Total CPU time</td>
<td>48.706 msec</td>
</tr>
<tr>
<td>Elapsed time</td>
<td>56.434 msec</td>
</tr>
<tr>
<td>Context switches</td>
<td>17 voluntary, 5 involuntary</td>
</tr>
</tbody>
</table>

### Browser timing

<table>
<thead>
<tr>
<th>Timing attribute</th>
<th>Timeline</th>
<th>Milliseconds since navigation start (+length)</th>
</tr>
</thead>
<tbody>
<tr>
<td>domainLookup</td>
<td></td>
<td>11 (+0)</td>
</tr>
<tr>
<td>connect</td>
<td></td>
<td>11 (+1)</td>
</tr>
<tr>
<td>request</td>
<td></td>
<td>12 (+82)</td>
</tr>
<tr>
<td>response</td>
<td></td>
<td>94 (+0)</td>
</tr>
<tr>
<td>domLoading</td>
<td></td>
<td>101 (+428)</td>
</tr>
<tr>
<td>domInteractive</td>
<td></td>
<td>500</td>
</tr>
<tr>
<td>domContentLoadedEvent</td>
<td></td>
<td>505 (+11)</td>
</tr>
<tr>
<td>loadEvent</td>
<td></td>
<td>529 (+25)</td>
</tr>
</tbody>
</table>
In production

• Google Chrome Developer Tools
  • only affects the network

• Google Analytics Site Speed
  • sampled on 1% of users by default

• Application Performance Monitoring solutions
Performance timeline

redirects  app cache  DNS  TCP

request  response  processing  on load

Did you expect requestEnd?
The backend makes ~15% of the total page load time.
Fractal Ideas - $0.23s / 1.0s = 23\%$
Facebook - 2.8s / 14.2s = 20%
New York Times - 1.2s / 11.8s = 10%
HTTP/1.1 is bad at fetching many resources

- **Client-side**
  - DNS pre-fetch
  - TCP pre-connect
  - keep-alive & pipelining
  - parallel connections
  - caching

- **Server-side**
  - concatenation
  - spriting
  - inlining
  - domain sharding
  - allow caching
Let’s talk about the frontend.
Performance timeline

- navigationStart
- redirectStart
- redirectEnd
- fetchStart
- domainLookupStart
- domainLookupEnd
- connectStart
- secureConnectionStart
- connectEnd
- requestStart
- responseStart
- requestEnd
- responseEnd
- domLoading
- domInteractive
- domComplete
- loadEventStart
- loadEventEnd
- unloadEventStart
- unloadEventEnd

Did you expect requestEnd?
document.readyState & page load events

‘loading’
- domLoading
- No event (no JS yet!)
- Parse HTML & build DOM
- Download & run sync JS

‘interactive’
- domInteractive
- DOMContentLoaded event
- Download CSS & images
- Parse CSS & build CSSOM

‘complete’
- domComplete
- load event
- Page is fully loaded
document.readyState & page load events
Listen to DOMContentLoaded — not load

```javascript
(function(fractalideas_com)myk@mYk fractalideas_com %

commit 3b2e461dee9ff7d30be08cd45db47f8ebbfc2ce
Author: Aymeric Augustin <aymeric.augustin@fractalideas.com>
Date: Mon Oct 17 22:34:32 2016 +0200

I'm supposed to teach this stuff.

diff --git a/showcase/static/home.js b/showcase/static/home.js
index ab07c45..eb0b39f 100644
--- a/showcase/static/home.js
+++ b/showcase/static/home.js
@@ -270,7 +270,7 @@
     window.requestAnimationFrame run);
 
- window.addEventListener('load', debouncedRun);
+ window.addEventListener('DOMContentLoaded', debouncedRun);
     window.addEventListener('resize', debouncedRun);
 }

}(document, window));

(fractalideas_com)myk@mYk fractalideas_com % ~:/Documents/dev/fractalideas_com
```
client-side: loading pages
“an optimized web page”
Rendering pipeline

- HTML
- CSS
- JS

- DOM
- CSSOM

- render tree

- layout

- paint

- JS VM
- fonts
Critical path - visual edition

- HTML
- DOM
- sync JS
- JS VM
- CSS
- CSSOM
- render tree
Critical path - text edition

- Rendering a page requires a DOM and a CSSOM
  - Download and parse HTML
  - Download and parse CSS
- Building the DOM blocks on sync JS
  - Download and execute sync JS
- Executing JS blocks on the CSSOM
  - Wait until CSS is parsed to execute sync JS
Browsers optimize heavily page load time

- Parse HTML incrementally
- Paint while waiting for sync JS
  - After CSS is available
- Paint while waiting for web fonts
  - With a default font — browser dependent
- Preload scanner
Guidelines

1. Optimize HTML load time

2. Optimize CSS load time
   - Unblock first paint
   - Avoid blocking JS execution

3. Avoid sync JS, including inline JS, or put it at the bottom
   - Avoid blocking DOM construction
   - Trigger DOMContentLoaded as early as possible
client-side:
async scripts
“The JavaScript tracking snippet”

<!-- Google Analytics -->
<script>
(function(i,s,o,g,r,a,m)
{i['GoogleAnalyticsObject']=r;i[r]=i[r]||function()
{(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new
Date();a=s.createElement(o),m=s.getElementsByTagName(o)
[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m))
(window,document,'script','https://www.google-analytics.com/analytics.js','ga');

ga('create', 'UA-XXXXX-Y', 'auto');
ga('send', 'pageview');
</script>
<!-- End Google Analytics -->

https://developers.google.com/analytics/devguides/collection/analyticsjs/
Script-injected scripts

window.GoogleAnalyticsObject = 'ga'
window.ga = window.ga || function () {
  window.ga.q = window.ga.q || []
  window.ga.q.push(arguments)
}
window.ga.l = 1 * new Date()

var script = document.createElement('script')
script.async = 1
script.src = '
  'https://www.google-analytics.com/analytics.js''

var otherScript = \
  document.getElementsByTagName('script')[0]
otherScript.parentNode.insertBefore(script, otherScript)
“Alternative async tracking snippet”

<!-- Google Analytics -->
<script>
window.ga=window.ga||function(){(ga.q=ga.q||[]).push(arguments)};ga.l=new Date;
g(a('create', 'UA-XXXXX-Y', 'auto');
g(a('send', 'pageview');
</script>
<script async src='https://www.google-analytics.com/analytics.js'></script>
<!-- End Google Analytics -->

https://developers.google.com/analytics/devguides/collection/analyticsjs/
Async scripts

```javascript
window.ga = window.ga || function () {
    ga.q = ga.q || []
    ga.q.push(arguments)
}
```

```html
<script async src='https://www.google-analytics.com/analytics.js'></script>
```

// plus an async script:
New best practice?

<html>
  <head>
    <script> /* Async function queuing */ </script>
    <link rel="stylesheet" href="style.css">
    <script async src="critical.js"></script>
    <style> /* For above-the-fold content */ </style>
  </head>
  <body>
    <script async src="non-critical.js"></script>
  </body>
</html>
Credits: Ilya Grigorik

https://developers.google.com/web/fundamentals/performance/

https://www.igvita.com/2014/05/20/script-injected-async-scripts-considered-harmful/
https://www.igvita.com/2015/04/10/fixing-the-blank-text-problem/
server-side: serving requests
Request-response cycle

1. **request**
   - request_started
   - request

2. **middleware**
   - view

3. **view**
   - t. r.
   - mw.
   - tmpl. resp.

4. **middleware**
   - exc.
   - mw.

5. **response**
   - request_finished
Slow page?
SQL queries!
test
setup
pgbench (1/2)

$ pgbench -i --foreign-keys duth16dp
creating tables...
100000 of 100000 tuples (100%) done (elapsed 0.11 s, remaining 0.00 s)
vacuum...
set primary keys...
set foreign keys...
done.
$ pgbench -c 30 -T 180 duth16dp

starting vacuum...end.

transaction type: TPC-B (sort of)

scaling factor: 1

query mode: simple

number of clients: 30

duration: 180 s

number of transactions actually processed: 313568

latency average: 17.221 ms

tps = 1741.538334 (including connections establishing)
tps = 1741.569697 (excluding connections establishing)
Database structure

- Branch
  - 1

- Account
  - 100 000

- Teller
  - 10

- Transaction
  - 313 568
Testing environment

- https://github.com/aaugustin/duth16dp
- https://duth16dp.herokuapp.com/
- Hobby Dyno
- Hobby Basic Postgres

- pg_dump -O duth16dp | heroku pg:psql
Festival international des jardins, Chaumont, 8 mai 2015

select & prefetch related instances
select_related()

django.db.models.query.QuerySet.select_related
The $k \times N + 1$ queries problem
The $k \times N + 1$ queries problem

<table>
<thead>
<tr>
<th>QUERY</th>
<th>TIMELINE</th>
<th>TIME (MS)</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT ... FROM &quot;auth_user&quot; WHERE &quot;auth_user&quot;.&quot;id&quot; = 1</td>
<td></td>
<td>1.82</td>
<td>Sel, Expl</td>
</tr>
<tr>
<td>SELECT ... FROM &quot;pgbench_history&quot;</td>
<td>Duplicated 2 times.</td>
<td>28.93</td>
<td>Sel, Expl</td>
</tr>
<tr>
<td>SELECT ... FROM &quot;pgbench_history&quot;</td>
<td>Duplicated 2 times.</td>
<td>27.24</td>
<td>Sel, Expl</td>
</tr>
<tr>
<td>SELECT ... FROM &quot;pgbench_history&quot; ORDER BY &quot;pgbench_history&quot;.&quot;tid&quot; DESC LIMIT 100</td>
<td></td>
<td>1.70</td>
<td>Sel, Expl</td>
</tr>
<tr>
<td>SELECT ... FROM &quot;pgbench_tellers&quot; WHERE &quot;pgbench_tellers&quot;.&quot;tid&quot; = 10</td>
<td>Duplicated 100 times.</td>
<td>1.27</td>
<td>Sel, Expl</td>
</tr>
<tr>
<td>SELECT ... FROM &quot;pgbench_branches&quot; WHERE &quot;pgbench_branches&quot;.&quot;bid&quot; = 1</td>
<td>Duplicated 100 times.</td>
<td>1.61</td>
<td>Sel, Expl</td>
</tr>
<tr>
<td>SELECT ... FROM &quot;pgbench_accounts&quot; WHERE &quot;pgbench_accounts&quot;.&quot;aid&quot; = 3560</td>
<td>Duplicated 100 times.</td>
<td>1.50</td>
<td>Sel, Expl</td>
</tr>
<tr>
<td>SELECT ... FROM &quot;pgbench_tellers&quot; WHERE &quot;pgbench_tellers&quot;.&quot;tid&quot; = 4</td>
<td>Duplicated 100 times.</td>
<td>1.04</td>
<td>Sel, Expl</td>
</tr>
</tbody>
</table>

$k = 3$

$N = 100$
select_related() to the rescue

```python
@dth16dp  myk@mYk  dth16dp  %  git show
commit 0cdf807b1542b308856ed92fd2aec8d253fb09fc
Author: Aymeric Augustin <aymeric.augustin@m4x.org>
Date:   Sat Oct 29 12:20:22 2016 +0200

    select_related in transactions admin

diff --git a/pgbench/admin.py b/pgbench/admin.py
index 96a334f..ae04c7c 100644
--- a/pgbench/admin.py
+++ b/pgbench/admin.py
@@ -21,3 +21,7 @@
     class Teller(admin.ModelAdmin):
         @admin.register(models.Transaction)
         class Transaction(admin.ModelAdmin):
-            list_display = ['id', 'teller', 'branch', 'account', 'delta', 'mtime']
+            list_display = ['id', 'teller', 'branch', 'account', 'delta', 'mtime']
+    def get_queryset(self, request):
+        queryset = super().get_queryse(request)
+        return queryset.select_related('teller', 'branch', 'account')
```

select_related() to the rescue

```bash
(duth16dp) myk@myk duth16dp % git show ~:/Documents/dev/duth16dp
commit 93acbf4cf2f4ee7925ae04cb9f77d69aecdf3e38
Author: Aymeric Augustin <aymeric.augustin@m4x.org>
Date: Fri Oct 28 22:56:38 2016 +0200

    select_related in transactions admin

diff --git a/pgbench/admin.py b/pgbench/admin.py
index 96a334f..7b0241f 100644
--- a/pgbench/admin.py
+++ b/pgbench/admin.py
@@ -21,3 +21,4 @@ class Teller(admin.ModelAdmin):
     @admin.register(models.Transaction)
     class Transaction(admin.ModelAdmin):
         list_display = ['id', 'teller', 'branch', 'account', 'delta', 'mtime']
+        list_select_related = ['teller', 'branch', 'account']
(duth16dp) myk@myk duth16dp %
```
The $k \times N + 1$ queries problem, solved
The $k \times N + 1$ queries problem, solved
select_related()
Sprint idea

- Figure out if pathological performance of LEFT OUTER JOIN vs. INNER JOIN still happens these days

- Make the case for treating FKs identically in `select_related` regardless of whether they’re nullable

- Many projects would benefit from a default `select_related` that includes nullable FKs in the admin

- Think about backwards compatibility — no easy answer there

- Search: “site:code.djangoproject.com select_related nullable”
prefetch_related()

django.db.models.query.QuerySet.prefetch_related
The $k \times N + 1$ queries problem, again
The $k \cdot N + 1$ queries problem, again

<table>
<thead>
<tr>
<th>QUERY</th>
<th>TIMELINE</th>
<th>TIME (MS)</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ SELECT ′′′ FROM &quot;auth_user&quot; WHERE &quot;auth_user&quot;.&quot;id&quot; = 1</td>
<td></td>
<td>2.05</td>
<td>Sel</td>
</tr>
<tr>
<td>+ SELECT ′′′ FROM &quot;pgbench_accounts&quot;</td>
<td></td>
<td>11.14</td>
<td>Sel</td>
</tr>
<tr>
<td></td>
<td>Duplicated 2 times.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ SELECT ′′′ FROM &quot;pgbench_accounts&quot;</td>
<td></td>
<td>9.34</td>
<td>Sel</td>
</tr>
<tr>
<td></td>
<td>Duplicated 2 times.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ SELECT ′′′ FROM &quot;pgbench_accounts&quot; LEFT OUTER JOIN &quot;pgbench_branches&quot; ON (&quot;pgbench_accounts&quot;.&quot;bid&quot; = &quot;pgbench_branches&quot;.&quot;bid&quot;) ORDER BY &quot;pgbench_accounts&quot;.&quot;aid&quot; DESC LIMIT 100</td>
<td></td>
<td>1.82</td>
<td>Sel</td>
</tr>
<tr>
<td>+ SELECT ′′′ FROM &quot;pgbench_history&quot; WHERE &quot;pgbench_history&quot;.&quot;aid&quot; = 100000</td>
<td></td>
<td>32.90</td>
<td>Sel</td>
</tr>
<tr>
<td></td>
<td>Duplicated 100 times.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ SELECT ′′′ FROM &quot;pgbench_history&quot; WHERE &quot;pgbench_history&quot;.&quot;aid&quot; = 99999</td>
<td></td>
<td>31.67</td>
<td>Sel</td>
</tr>
<tr>
<td></td>
<td>Duplicated 100 times.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ SELECT ′′′ FROM &quot;pgbench_history&quot; WHERE &quot;pgbench_history&quot;.&quot;aid&quot; = 99998</td>
<td></td>
<td>31.08</td>
<td>Sel</td>
</tr>
<tr>
<td></td>
<td>Duplicated 100 times.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
prefetch_related() to the rescue

diff --git a/pgbench/admin.py b/pgbench/admin.py
index 4701ed6..244991c 100644
--- a/pgbench/admin.py
+++ b/pgbench/admin.py
@@ -17,6 +17,9 @@ class Account(admin.ModelAdmin):
    transactions = account.transaction_set.all()
    return ', '.join(str(transaction.delta) for transaction in transactions)

+    def get_queryset(self, request):
+        return super().get_queryset(request).prefetch_related('transaction_set')
+
@admin.register(models.Teller)
class Teller(admin.ModelAdmin):

The k * N + 1 queries problem, solved again
The $k \times N + 1$ queries problem, solved again
prefetch_related()

**first query**

ACCOUNT: 1

ACCOUNT: 2

**second query**

TRANSACTION: 1

TRANSACTION: 2

TRANSACTION: 3
Prefetch objects

```python
@duth16dp myk@Mk duth16dp % git show ~/Documents/dev/duth16dp
commit 51ecc754121c702823d5f5ef8f705c384fbc2ecf
Author: Aymeric Augustin <aymeric.augustin@m4x.org>
Date:   Sat Oct 29 09:22:27 2016 +0200

    order prefetched transactions

diff --git a/pgbench/admin.py b/pgbench/admin.py
index 244991c..e9d7750 100644
--- a/pgbench/admin.py
+++ b/pgbench/admin.py
@@ -18,7 +18,8 @@ class Account(ModelAdmin):
         return ', '.join(str(transaction.delta) for transaction in transactions)

     def get_queryset(self, request):
-        return super().get_queryset(request).prefetch_related('transaction_set')
+        return super().get_queryset(request).prefetch_related(Prefetch(
+            'transaction_set', models.Transaction.objects.order_by('mtime')))

 @admin.register(models.Teller)
@duth16dp myk@Mk duth16dp %
```

Prefetch objects

- **Prefetch(lookup, queryset=None, to_attr=None)**

- Specify a relation to follow

- Can filter, order, etc. the target queryset
  - Required to filter, order, etc. prefetched querysets

- Can attach the result to an attribute with another name
  - Recommended when the target queryset is filtered
prefetch_related_objects()

• Like prefetch_related

• Works on any iterable of model instances

• New in Django 1.10
select_related()

vs.

def prefetch_related()
<table>
<thead>
<tr>
<th></th>
<th>1/N to 1</th>
<th>1/N to N</th>
</tr>
</thead>
<tbody>
<tr>
<td>select_related()</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>prefetch_related()</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>
select_related() vs. prefetch_related()

• “Generally you’ll want to use select_related()”
  • It’s more elegant but it isn’t always faster
• select_related() fetches more data
  • Consequences depend on the database schema and content
• prefetch_related() makes several queries
  • Transactional consistency isn’t guaranteed (#27403)
• Depends mostly on the latency of database queries
select_related() vs. prefetch_related()

```python
from pgbench.models import Transaction
transactions = Transaction.objects.all()

# Many distinct related objects

transactions.select_related('account')
# Database = 1.32s - Total = 11.4s

transactions.prefetch_related('account')
# Database = 0.62s + 0.58s = 1s - Total = 14.5s (+26%)
```
select_related() vs. prefetch_related()

```python
from pgbench.models import Transaction
transactions = Transaction.objects.all()

# Few distinct related objects

transactions.select_related('teller')
# Database = 0.87s - Total = 12.6s

transactions.prefetch_related('teller')
# Database = 0.66s + ~0s = 0.66s - Total = 12.3s (-2.4%)```
“How do I check query patterns with Django Rest Framework?”

https://www.dabapps.com/blog/api-performance-profiling-django-rest-framework/
Log database queries to the console

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
        },
    },
    'loggers': {
        'django.db.backends': {
            'handlers': ['console'],
            'level': 'DEBUG',
        },
    },
}
Log database queries to the console

```
(0.000) SELECT "pgbench_branches"."bid", "pgbench_branches"."bbalance", "pgbench_branches"."filler" FROM "pgbench_branches" WHERE "pgbench_branches"."bid" = 1; args = (1,)
(0.000) SELECT "pgbench_branches"."bid", "pgbench_branches"."bbalance", "pgbench_branches"."filler" FROM "pgbench_branches" WHERE "pgbench_branches"."bid" = 1; args = (1,)
(0.000) SELECT "pgbench_branches"."bid", "pgbench_branches"."bbalance", "pgbench_branches"."filler" FROM "pgbench_branches" WHERE "pgbench_branches"."bid" = 1; args = (1,)
(0.000) SELECT "pgbench_branches"."bid", "pgbench_branches"."bbalance", "pgbench_branches"."filler" FROM "pgbench_branches" WHERE "pgbench_branches"."bid" = 1; args = (1,)
(0.000) SELECT "pgbench_branches"."bid", "pgbench_branches"."bbalance", "pgbench_branches"."filler" FROM "pgbench_branches" WHERE "pgbench_branches"."bid" = 1; args = (1,)
(0.000) SELECT "pgbench_branches"."bid", "pgbench_branches"."bbalance", "pgbench_branches"."filler" FROM "pgbench_branches" WHERE "pgbench_branches"."bid" = 1; args = (1,)
(0.000) SELECT "pgbench_branches"."bid", "pgbench_branches"."bbalance", "pgbench_branches"."filler" FROM "pgbench_branches" WHERE "pgbench_branches"."bid" = 1; args = (1,)
(0.000) SELECT "pgbench_branches"."bid", "pgbench_branches"."bbalance", "pgbench_branches"."filler" FROM "pgbench_branches" WHERE "pgbench_branches"."bid" = 1; args = (1,)
```
Log database queries to the console

```
(0.001) SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login", "auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name", "auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff", "auth_user"."is_active", "auth_user"."dateJoined" FROM "auth_user" WHERE "auth_user"."id" = 1; args=(1,)
(0.019) SELECT COUNT(*) AS "__count" FROM "pgbench_accounts"; args=()
(0.012) SELECT COUNT(*) AS "__count" FROM "pgbench_accounts"; args=()
(0.001) SELECT "pgbench_accounts"."aid", "pgbench_accounts"."bid", "pgbench_accounts"."abalance", "pgbench_accounts"."filler", "pgbench_branches"."bid", "pgbench_branches"."bbalance", "pgbench_branches"."filler" FROM "pgbench_accounts" LEFT OUTER JOIN "pgbench_branches" ON ("pgbench_accounts"."bid" = "pgbench_branches"."bid") ORDER BY "pgbench_accounts"."aid" DESC LIMIT 100; args=()
(0.324) SELECT "pgbench_history"."hid", "pgbench_history"."tid", "pgbench_history"."bid", "pgbench_history"."aid", "pgbench_history"."delta", "pgbench_history"."mtime", "pgbench_history"."filler" FROM "pgbench_history" WHERE "pgbench_history"."aid" IN (99901, 99902, 99903, 99904, 99905, 99906, 99907, 99908, 99909, 99910, 99911, 99912, 99913, 99914, 99915, 99916, 99917, 99918, 99919, 99920, 99921, 99922, 99923, 99924, 99925, 99926, 99927, 99928, 99929, 99930, 99931, 99932, 99933, 99934, 99935, 99936, 99937, 99938, 99939, 99940, 99941, 99942, 99943, 99944, 99945, 99946, 99947, 99948, 99949, 99950, 99951, 99952, 99953, 99954, 99955, 99956, 99957, 99958, 99959, 99960, 99961, 99962, 99963, 99964, 99965, 99966, 99967, 99968, 99969, 99970, 99971, 99972, 99973, 99974, 99975, 99976, 99977, 99978, 99979, 99980, 99981,
```
other ORM optimizations
Baseline

```python
from collections import defaultdict
balances = defaultdict(lambda: 0)

from pgbench.models import Transaction
txs = Transaction.objects.all()

for tx in txs:
    balances[tx.teller_id] += tx.delta

# Database = 630ms - Total = 6900ms
```
only() and defer()

```python
from collections import defaultdict
balances = defaultdict(lambda: 0)

from pgbench.models import Transaction
txs = Transaction.objects.only('teller_id', 'delta')

for tx in txs:
    balances[tx.teller_id] += tx.delta

# Database = 220ms (-65%) - Total = 7900ms (+15%)
```
Use only() or defer() when...

- you need model instances
- you don’t need all columns
  - especially columns containing large amounts of data
- Not a common use case
- Consider moving rarely needed data to a separate model
values_list() and values()

```python
from collections import defaultdict
balances = defaultdict(lambda: 0)

from pgbench.models import Transaction
txs = Transaction.objects.values_list('teller_id', 'delta')

for teller_id, delta in txs:
    balances[teller_id] += delta

# Database = 160ms (-75%) - Total = 550ms (-92%)
```
Use `values_list()` or `values()` when...

- you don’t need model instances
- you need to manipulate large amounts of data
  - large starts between 1,000 and 10,000 rows :-(

- Common use case: reports
- Huge, easy improvement for queries that return lots of rows
aggregate() and annotate()

```python
from django.db.models import Sum
from pgbench.models import Transaction

balances = dict(
    Transaction.objects.values_list('teller_id')
    .annotate(Sum('delta'))
)

# Database = 75ms (-88%) - Total = 77ms (-99%)
```
Pro-tip: print(queryset.query)

```sql
SELECT
    "pgbench_history"."tid",
    SUM("pgbench_history"."delta") AS "delta__sum"
FROM
    "pgbench_history"
GROUP BY
    "pgbench_history"."tid"
```
Use `aggregate()` or `annotate()` when...

- you can perform a calculation in the database
- you need to manipulate large amounts of data
  - large starts between 1,000 and 10,000 rows :-(
- Common use case: dashboards
- There’s a learning curve
iterator()

- Iterates over instances
- Doesn’t cache results
- Not a common use case
- The whole dataset is still fetched from the database at once
- Live references to instances still prevent garbage collection
Baseline

txs = Transaction.objects.all()

import tracemalloc
tracemalloc.start()

for tx in txs:
    balances[tx.teller_id] += tx.delta

tracemalloc.get_traced_memory()
# (166775984, 166824045) => current: 159MB, peak: 159MB
iterator()

txs = Transaction.objects.iterator()

import tracemalloc
tracemalloc.start()

for tx in txs:
    balances[tx.teller_id] += tx.delta

tracemalloc.get_traced_memory()
# (187159, 234230) => current: 183kB, peak: 229kB
Thank you!

Questions?