

Management dashboards with Django

or

30 minutes during which I hope someone has heard
of a great Django app to make this project redundant



Alex de Landgraaf

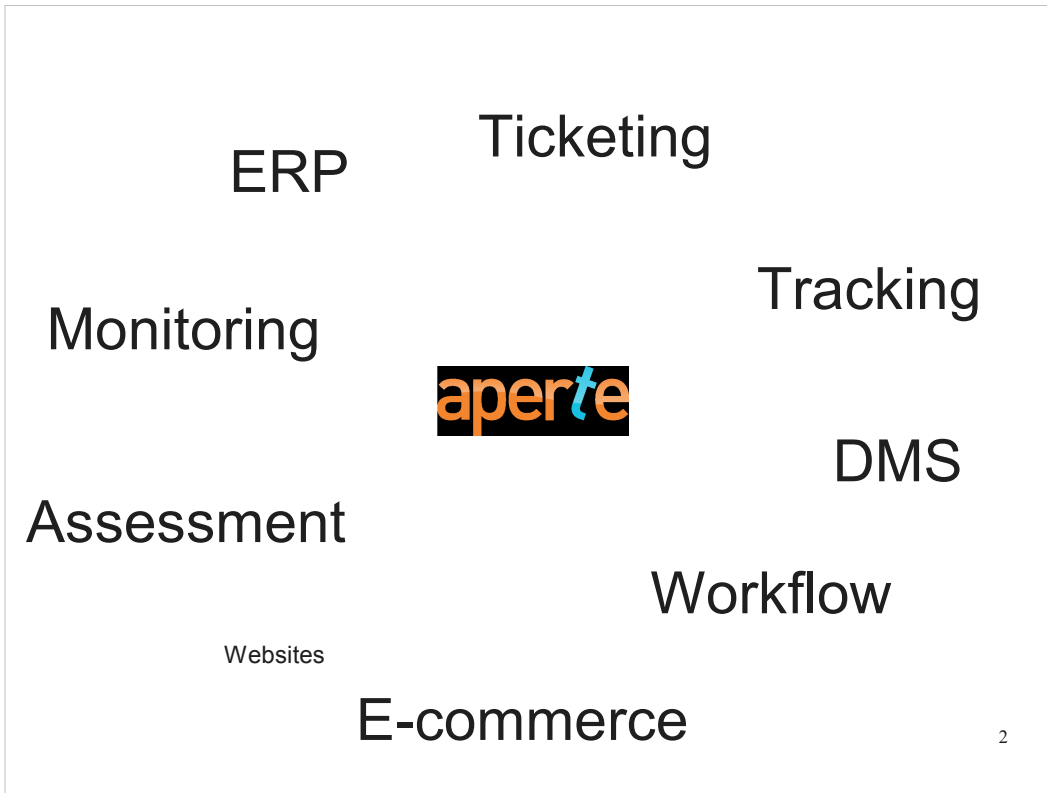


Django Meeting NL, 17/02/2010

1

- Introduction
 - Aperte, open source software, web applications
 - Use of Django for 3+ years
- Tell a little about the presentation
 - First the reason why this topic is interesting
 - Overview of the “Business Intelligence” world
 - Overview of the ultimate Django BI app and my current project status
 - Walkthrough of the current project using screenshots
 - Q&A session (two-directional)
 - Demo, if operational and we have the time

Note that the project is still very new and not yet released



Apologize for advertising Aperte and all the buzzwords

Projects completed or currently underway



All my projects involve Django in one way or another even though I didn't start out in that way

Last year increasingly more interest from companies in Django specifically

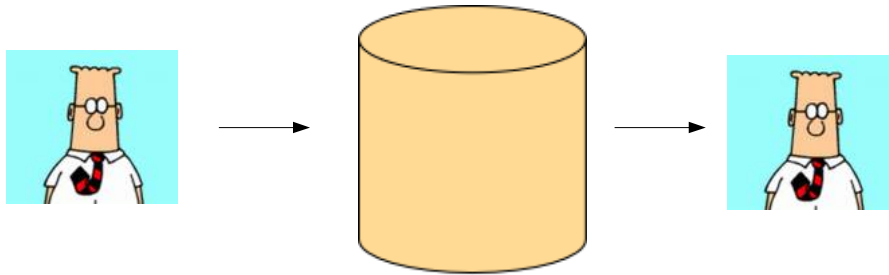
Q: How many people actively use Django?

Q: How many people rely on Django commercially?

Q: How many Django committers are in the room?

NOTE: Yes, I mixed up committers and contributors here. Would have been rare to have someone with trunk access with us.

CONFIDENTIAL
Summary of all projects



4

Companies pay good money for what is in theory very simple: Data in, data out

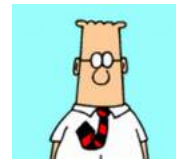
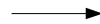
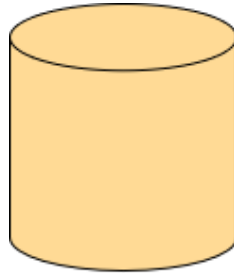
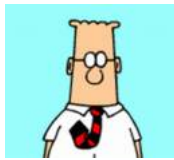
Of course clients don't like it when things are this simple, so we do our best to make the most out of it

Django is perfect for this

The Recurring Problem



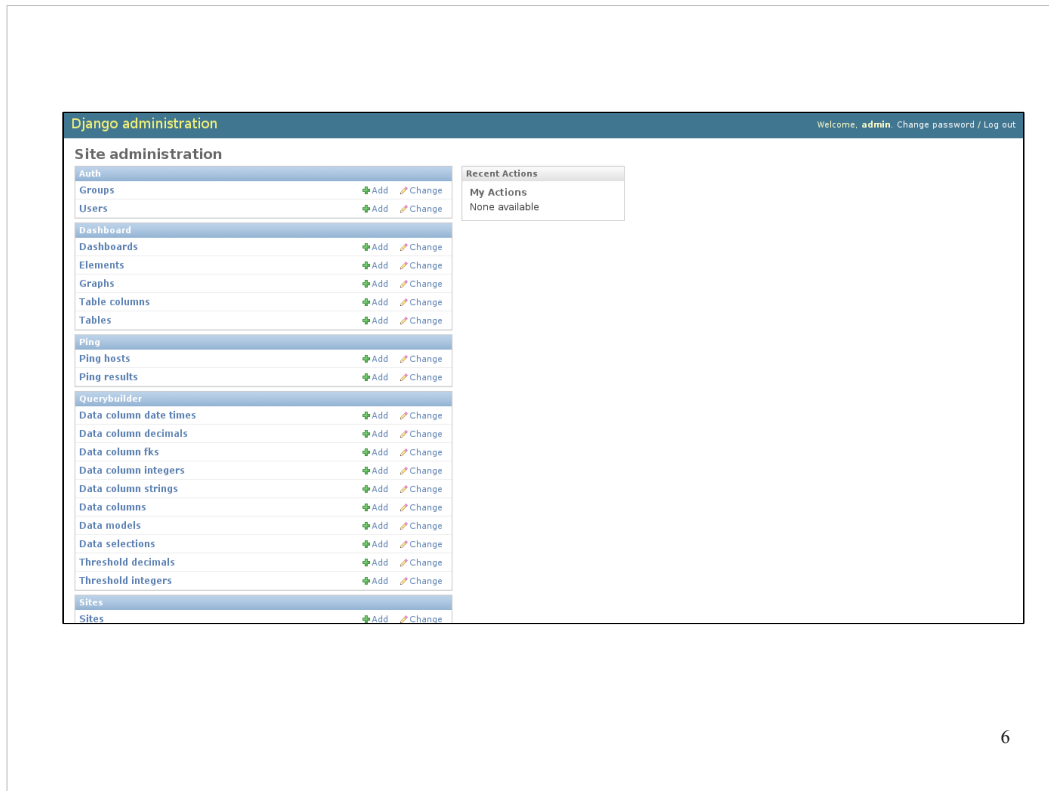
?



5

Managers (but actually all users) want to know what is going on inside your database

In the Django world this can be a problem...



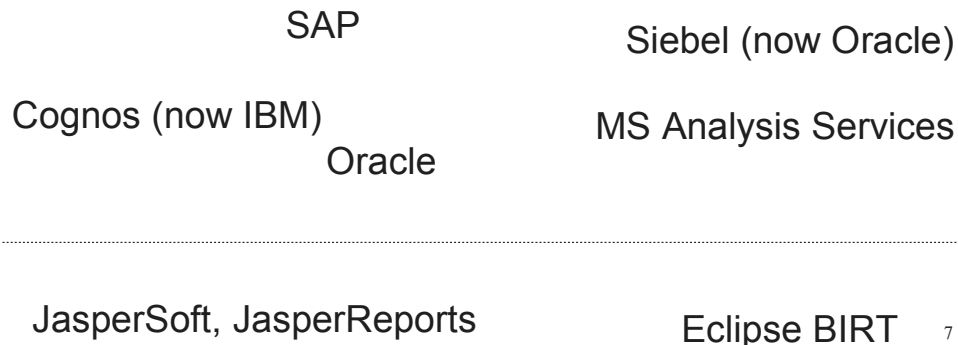
The Django admin, loved by all

Very easy to do stuff

Very hard to figure out what is going on

Solution? "Business Intelligence"

“Business Intelligence”



Very large scope, many large companies

Very expensive solutions that take a team of consultants to set up and operate

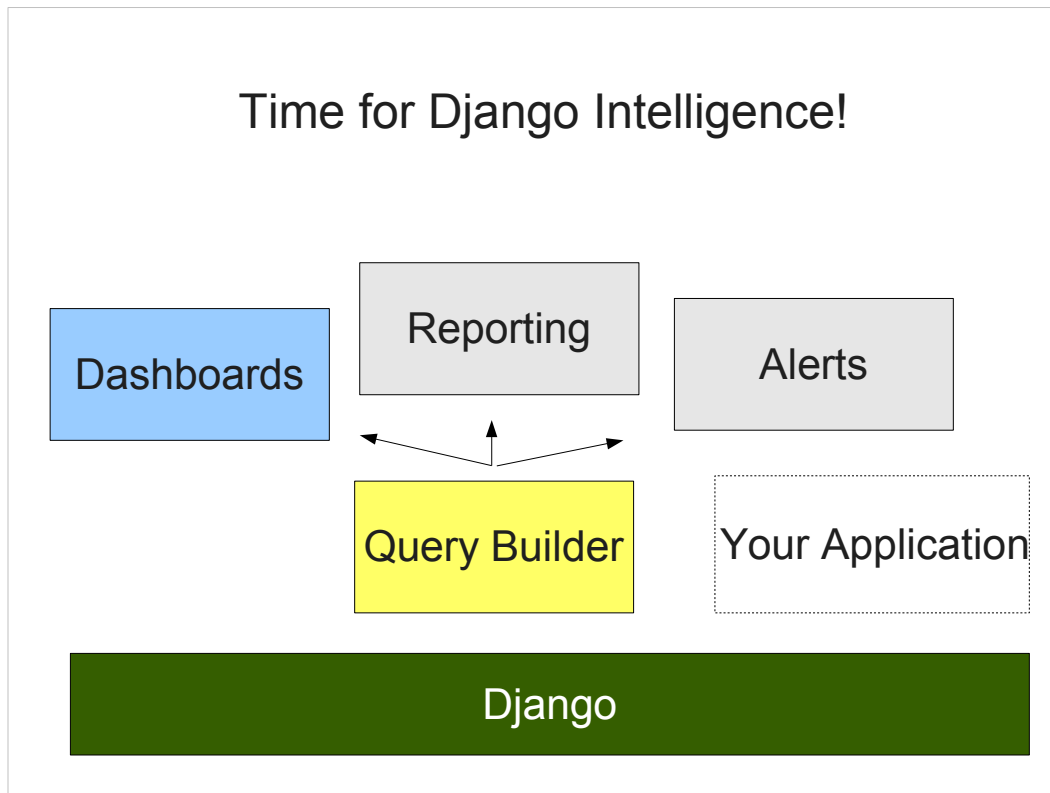
Q: Ask for experiences with proprietary and open source BI software (only have experience with Cognos) **(NOTES: others confirmed my views on BI)**

Sidenote: Cognos revenue 2007: \$980M

They do get the job done of opening up databases

Integration with Django:

- reuse models & code
- simpler to setup, use and maintain
- easier to hack
- integration into the Django admin



What I came up with, with my limited experience with Cognos and current needs

Query builder allows users to set up queries, using models & data from other Django applications

Query builder provides QuerySets and model details to Dashboard, Reporting, Alerts. Web-based data selections

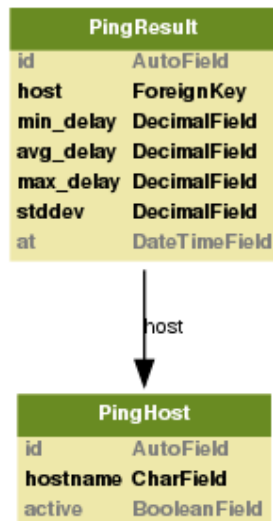
Dashboard > provide real-time data of current applications: ideally 1 screen per user

Reporting > provide recurring reports (PDF, Excel)

Alerts > notify people via e-mail, sms etc when stuff goes wrong (ie. Ability to set thresholds in Query Builder)

D+QB applications have a first prototype ready, grey = todo

Example Django application



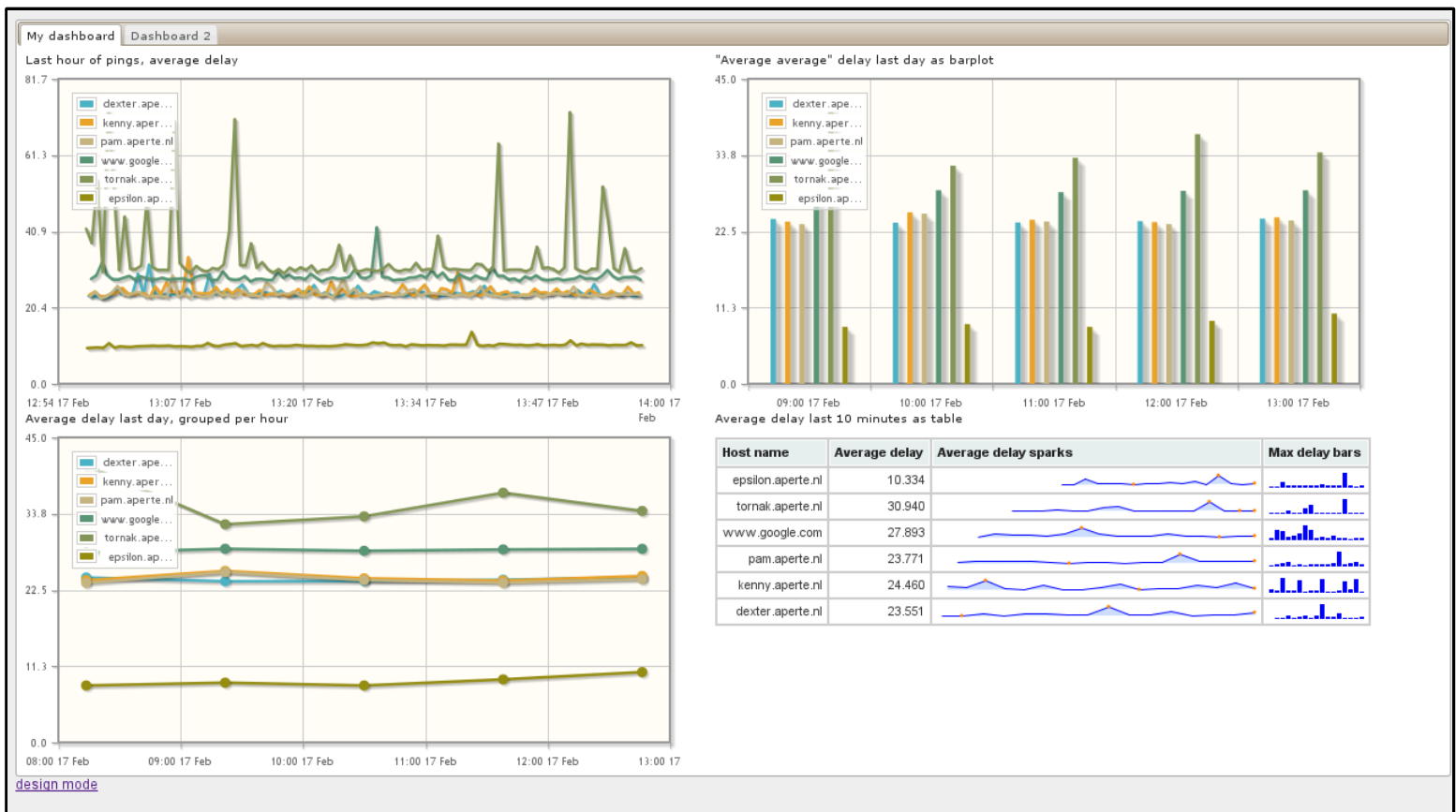
9

Very simple (non-blog!) application to collect data

Just 'ping' hosts in a round-robin fashion every minute or so.

Enough to demonstrate the basic functionalities of the current system

Collected a couple of days of data



Main dashboard view (note the tabs)

A lot of different layouts, provided you only want 2 or 3 columns :)

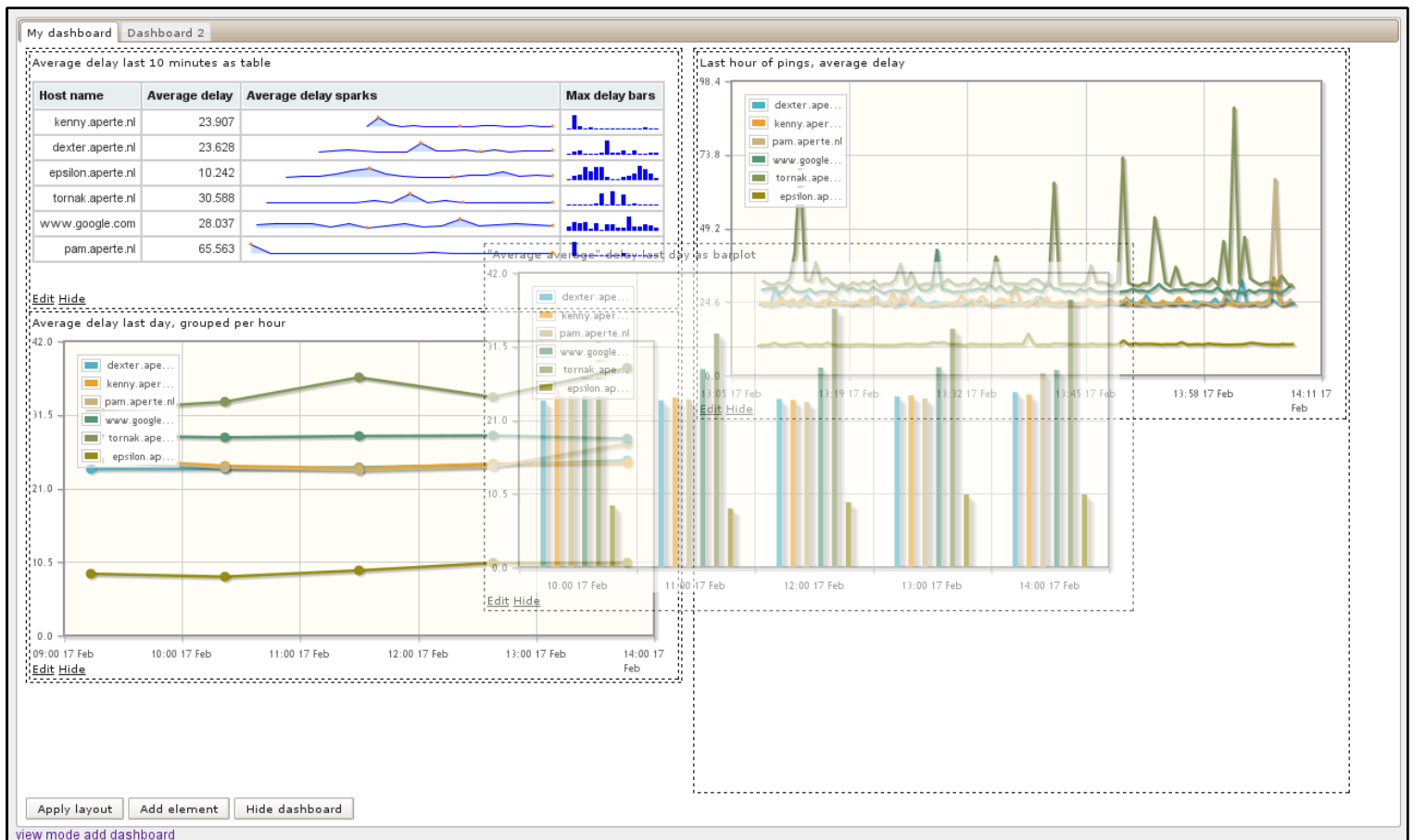
Currently support 2 types of elements (Graphs and Tables)

Each element is powered by a QuerySet, given to the Dashboard from the Query Builder

Graphs support various types of plotting (points, lines, bars) (jqPlot)

Tables support various embedded elements (sparklines), also bullets, boxes etc.

Complex queries: grouping over series, timesets



Here we hit the design-mode

Allows drag-and-drop placement of elements (jQuery)

Ability to add elements, hide/show dashboards etc

We'll now edit the top-right element, showing the 'raw' pings over the last hour

[Back to dashboard-design](#)
 Edit element Last hour of pings, average delay (left / 0)

Element form | Element data selections | Element view

Dashboard: My dashboard ▼

Name:

Height:

Refresh rate:

Active:

Series:

X axis:

Y axis:

Start y-axis at 0?

Graph type:

Fill:
If non-empty, the shape will be filled with this value being the opacity of the area. Bars are always filled.

Legend:

Count series:
Enable to simply plot the **number of entries** in the data selections. Implicitly done when assigning multiple data selections to a graph. Keep this off to plot the **data** from the data selection.

Data:

Hold down "Control", or "Command" on a Mac, to select more than one.

Here we're editing the graph itself.

Select from a list what data we want from Query Builder

Determine which columns from the query we use for which axis or series

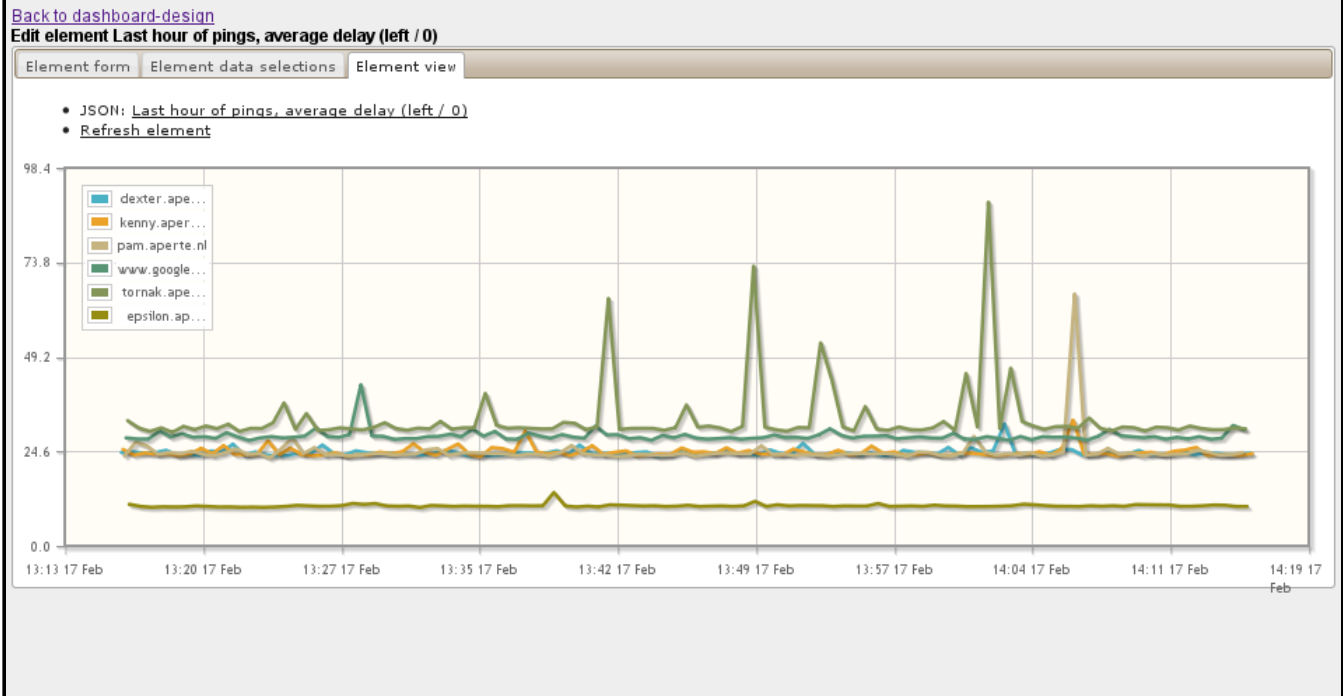
'at' the time of each ping

'avg_delay' the time it took

'host' the host to which the ping was sent

TODO: automate dashboard creation (automatically select datetime for x-axis, numerical values or count() for y-axis and plot against foreign keys / M2M where available **(NOTE: Didn't go into this, not enough time)**)

We go to the Element view tab



Simple view to be able to quickly check the element after modifications

Access to raw data as sent to the dashboard (JSON)

We hit Element data selection and go into the Query Builder

[Back Query builder home](#)

Data selection form | Data columns | Data queries & results

Name for this data selection:

Application / Data source:

Selected model:

Auto add columns:

Limit to:

Count series label:

Override the count series label for this data selection. If not defined, the model name will be used instead.

Extra SELECT SQL statements:

A JSON dictionary, assign attribute names to SQL clauses. Use the attribute names as a DataColumn "name" for usage in a DataElement. Each name and SQL clause must be a string. Example: {"foo": "bar > '20'", "foobar": "SELECT COUNT(*) FROM blog_entry WHERE blog_entry.blog_id = blog_blog.id"}

Extra FROM SQL statement:

A JSON list of strings. If you are using tables not normally retrieved in the QuerySet via the extra select/where/order_by clauses, add the table name(s) here. Do not add the table names of tables already used. Example: ["my_blog_posts", "my_blog_comments"]

Extra WHERE SQL statement:

A JSON list of strings. All strings are "AND"ed together to form the SQL Where statement. Example: ["my_blog_posts.blog_id > 200"]

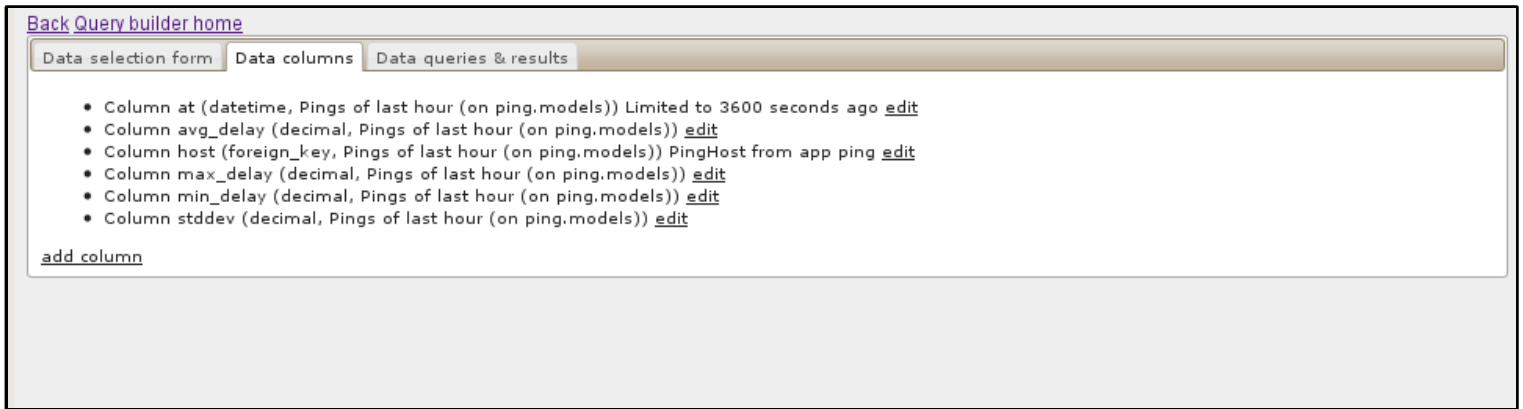
Goal of the Query Builder is to allow you to easily set up and store your queries

Flexibility of throwing SQL at the query (specifically ignoring security issues here)

When adding a new query, Query Builder will introspect your application:

- determine the models in your app
- when selecting a model, optionally populate your query with all the fields inside the model

This gives you the following list of columns for your query...



Here we see all related columns, and limitations per column set via the query builder

We'll get back to the data columns later on, first to the query tab...

[Back Query builder home](#)

Data selection form | Data columns | Data queries & results

Data queries

SQL:

```
SELECT `ping_pingresult`.`id`, `ping_pingresult`.`host_id`, `ping_pingresult`.`min_delay`, `ping_pingresult`.`avg_delay`, `ping_pingresult`.`max_delay`, `ping_pingresult`.`stddev`, `ping_pingresult`.`at` FROM `ping_pingresult` WHERE `ping_pingresult`.`at` >= 2010-02-17 13:16:47
```

Python:

```
qs = PingResult.objects.all().filter(at__gte = '2010-02-17 13:16:47.870021')
```

Data

- [Export to Excel](#)

```
pam.aperte.nl 24.837 at 2010-02-17 14:16:47
kenny.aperte.nl 23.717 at 2010-02-17 14:16:42
dexter.aperte.nl 24.240 at 2010-02-17 14:16:37
epsilon.aperte.nl 10.328 at 2010-02-17 14:16:27
tornak.aperte.nl 31.556 at 2010-02-17 14:16:22
www.google.com 27.675 at 2010-02-17 14:16:17
pam.aperte.nl 23.775 at 2010-02-17 14:16:12
kenny.aperte.nl 24.012 at 2010-02-17 14:16:07
dexter.aperte.nl 24.095 at 2010-02-17 14:16:01
epsilon.aperte.nl 10.255 at 2010-02-17 14:15:51
```

Here we see the generated SQL and the Python that is used under the hood.

This allows the user to fine-tune his query while seeing the raw data

Ability to export to Excel (using xlwt), will probably feed this directly to the dashboard for users wanting the raw data

Query looks simple, doesn't it? Well, things quickly get more complicated...

Data queries

SQL:

```
SELECT ((SELECT DISTINCT CAST(DATE_FORMAT(`at`, "%Y-%m-%d %H:00:00") AS DATETIME))) AS `at`, `ping_pingresult`.`id`, `ping_pingresult`.`host_id`, `ping_pingresult`.`min_delay`, `ping_pingresult`.`avg_delay`, `ping_pingresult`.`max_delay`, `ping_pingresult`.`stddev`, `ping_pingresult`.`at`, AVG(`ping_pingresult`.`avg_delay`) AS `avg_delay` FROM `ping_pingresult` WHERE `ping_pingresult`.at >= 2010-02-17 10:00:00 GROUP BY HOUR(at), host_id, (SELECT DISTINCT CAST(DATE_FORMAT(`at`, "%Y-%m-%d %H:00:00") AS DATETIME)) ORDER BY `at` ASC
```

Python:

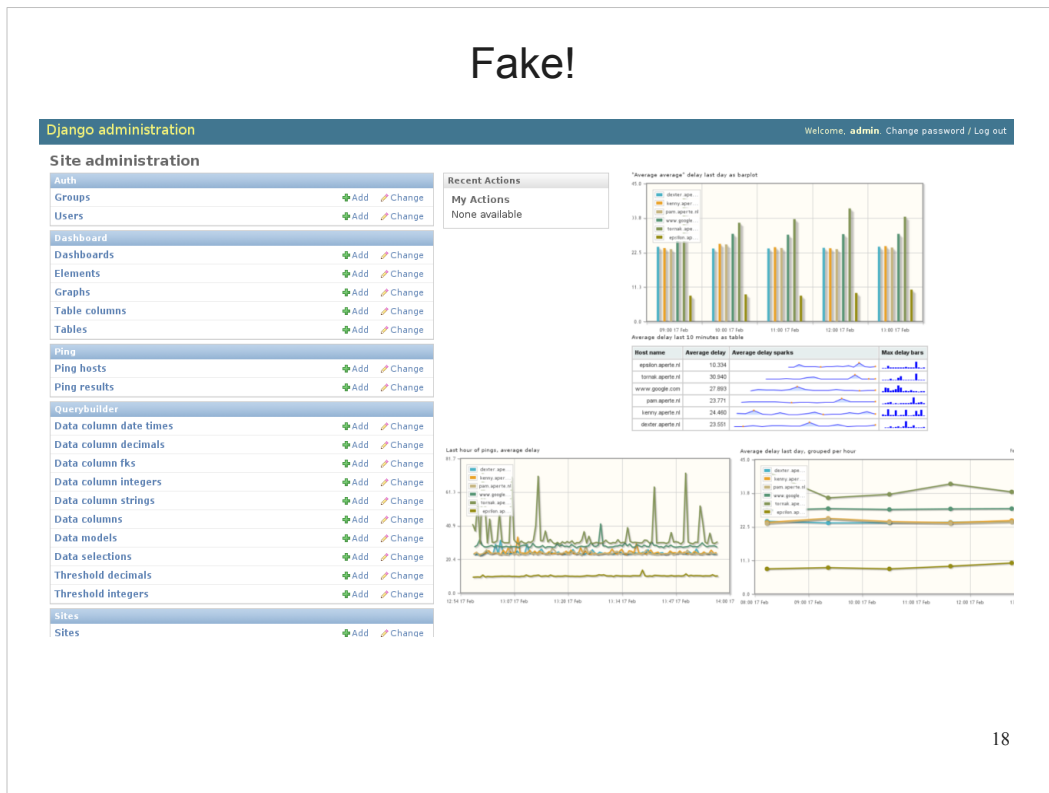
```
qs = PingResult.objects.all().filter(at__gte = '2010-02-17 10:00:00')
qs = qs.query.group_by = ['HOUR(at)', 'host_id']
qs = qs.extra(select = ['(SELECT DISTINCT CAST(DATE_FORMAT(`at`, "%Y-%m-%d %H:00:00") AS DATETIME)')])
qs = qs.annotate(avg_delay = Avg('avg_delay'))
qs = qs.order_by([at])
```

This is the same page but for a different query (namely, the one used for generating the bar-plot earlier).

Admit that it might be slightly less complicated when wrapped around with Python, but not much.

Just an example that for simple tasks you might need much more complex queries. Django gives us the flexibility to drop down to the SQL level in these cases (and Query Builder handles this for you)

Fake!



18

After getting the dashboard app finished, django admin integration should be very easy

Might go even further:

- clean up the left side and so provide more space for graphs and tables
- per-user dashboards > different data to different users
- **(NOTE: take a look at Grappelli, thanks Joeri!)**

Some things not yet demonstrated:

- Thresholds (nice red dot to grab your attention)
- Automation of query building using models
- Couple of other topics

Current status: first prototype done, not yet released

Q&A time

Q: What are your experiences with Django and reporting?

Have I missed the magic app or is everyone rolling their own?

Q: What would you expect to be able to do with such a tool?

Q: Beyond Dashboards, Reports and Alerts, do you see any other useful additions?

Q: Web-based queries versus “hard-coding” QuerySets. Advantages, disadvantages?

Q: Naming suggestions? “Django Goes BI”?

19

NOTES:

- Name: Django Bingo! (Business Intelligence Nurtures Grand Opportunities)
- API for Query Builder, Dashboard?
- Performance? (going to be a very important issue)
- Release! Now! Yesterday!
- But first security, we don't want Bobby Drop Tables
- Multiple Database support in Django 1.2 > investigate
- Grappelli > Django admin makeover (widget support?)

That's it!
Thanks for listening!



(this one is especially for Remcø)

Will upload slides + notes for those interested.
(No, I can't make all this stuff up on-the-fly. Srsly!)