Lecture 15: Django Database Intro + Templates

Database Interaction

Managers

- `Manager` is a class
- It's the interface between the database and django
- Various methods, including `filter()`, `exclude()`, and `order_by()`
- Also has `get_query_set`, which returns a `QuerySet` object

QuerySets

- `QuerySet` is a class
- Does not initiate the database interaction until told to
- Also has similar methods including `filter()`, `exclude()`, and `order_by()`

Getting all data

- `Blog.objects.get_query_set.all()`
- Shorthand: `Blog.objects.all()`
- Gets all the data associated with the model but does NOT execute the query

Filtering Data

- exact: gets an exact match
  - `Blog.objects.filter(title__exact='cool')`
  - `Blog.objects.filter(title='cool')` # __exact is implied
- contains: find if a match is contained inside a field
  - `Blog.objects.filter(blog_text__contains='cool')`
- icontains: case insensitive contains
  - `Blog.objects.filter(author__icontains='smith')`
- More here: https://docs.djangoproject.com/en/1.3/ref/models/querysets/#field-lookups

Ordering

- `Blog.objects.order_by('-pub_date', 'title')`
  - First orders by `pub_date` in descending order (hence the negative sign). If there are `pub_dates` that are equivalent, then `title` is ordered in ascending order.

Values

- `Blog.objects.values()`
  - Returns a `ValueQuerySet`, which returns a list of dictionaries *when executed*
- `Blog.objects.values('title', 'body')`
  - Returns only the fields `title` and `body` in the dictionary

Distinct

- `Blog.objects.distinct()`
  - If there are any duplicate rows, only one is returned
  - This will rarely work like this, because you often will already have a distinct field, like an id
- `Blog.objects.values('title', 'body').distinct()`
  - This will get all unique `title-body` combinations
  - Notice the **chaining** here

Slicing

- `Blog.objects.all()[:5]`
  - Gets the first 5 blog objects
  - The limit happens in the sql query
    - ex: SELECT * FROM users LIMIT 5

Get

- Gets a single row
- raises `MultipleObjectsReturned` if more than one object was found. The `MultipleObjectsReturned` exception is an attribute of the `model` class.
- raises a `DoesNotExist` exception if an object wasn't found for the given parameters. This exception is also an attribute of the `model` class.

`Get` continued

- `Blog.objects.get(id=5)`
  - Returns a single `QuerySet` if there is a row that exists, otherwise an error ensues
- `Blog.objects.filter(id=5)[0]`
  - Similar, except no exceptions are thrown

When are QuerySets Evaluated?

- Iteration

```
for e in Entry.objects.all():
    print e.headline
```

- Boolean

```
if Entry.objects.filter(headline="Test"):
   print "There is at least one Entry with the
headline Test"
```

Lookups that span relationships

- `Blog.objects.filter(comment__title__contains='Lennon')`
  - Retrieves all `Blog` objects with a comment whose `title` contains `'Lennon'`

Other Syntax

URLs

```
urlpatterns = patterns('',
    url(r'^$', 'blog.views.home'),
    url(r'^list/(₩d+)?$', 'blog.views.blog_list'),
    url(r'^search/(.*)$', 'blog.views.blog_search'),
    url(r'^(detail|info)/(?P<id>₩d+)/((?P<showComments>.*)/) ?$',
'blog.views.blog_detail'),
)
```

Views

```
def store_list(request, limit=100):
    store_list = Store.objects.all()[:limit]
    print store_list # [<Store: phones>, <Store: food>]
    return HttpResponse('going to give a list')
```

weather.html

```
<html>
    <head>
        <title> Weather </title>
    </head>
    <body>
        <p>Today’s weather in {{ city }} is {{ description }}.</p>
        <div id=“temperature”>
        {% for day in thisWeek %}
                <li> On {{ day.date }}, the temperature will be {{ day.temperature }}. </li>
        {% endfor %}

        </div>
        <div id="ads">
        {% block ads %}
        Click on these ads!
        {% endblock %}
        </div>
    </body>
</html>
```

Context

```
city = 'Accra‘
description = 'sunny‘
thisWeek = [dict(date='Thursday', temperature=20),
            dict(date='Friday', temperature=25),
            dict(date='Saturday', temperature=22)]
```

Displayed by browser

Today’s weather in Accra is sunny.

- On Thursday, the temperature will be 20.
- On Friday, the temperature will be 25.
- On Saturday, the temperature will be 22.

Click on these ads!

Templates

- A text-based template for HTML, CSS, XML, JavaScript, etc.
- Mixture between hard-coded text and abstractions
- Abstractions
  - Variables
  - Tags
- Re-useable and extensible

Hard-coded Text in weather.html

```
<html>
    <head>
        <title> Weather </title>
    </head>
    <body>
        <p>Today's weather in {{ city }} is {{ description }}.</p>
        <div id="temperature">
        {% for day in thisWeek %}
            <li> On {{ day.date }}, the temperature will be {{ day.temperature }}. </li>
        {% endfor %}

        </div>
        <div id="ads">
        {% block ads %}
        Click on these ads!
        {% endblock %}
        </div>
    </body>
</html>
```

Variables

- `{{ variable }}`
  - If variable doesn't exist, then output `TEMPLATE_STRING_IF_INVALID` (default: empty string "")
- `{{ variable.attribute }}`
  1. Dictionary Lookup. `variable["attribute"]`
  2. Attribute Lookup. `variable.attribute`
  3. Method Call. `variable.attribute()`
  4. List-index Call. `variable[attribute]`

Variables in weather.html

```
<html>
    <head>
        <title> Weather </title>
    </head>
    <body>
        <p>Today's weather in {{ city }} is {{ description }}.</p>
        <div id="temperature"
        {% for day in thisWeek %}
            <li> On {{ day.date }}, the temperature will be {{ day.temperature }}. </li>
        {% endfor %}

        </div>
        <div id="ads">
        {% block ads %}
        Click on these ads!
        {% endblock %}
        </div>
    </body>
</html>
```

Filters

- Modify the output of variables
- `{{ variable|filter }}`

```
foo := “Hello World”
bar := [‘a’, ‘b’, ‘c’]

{{ foo|lower }} --> hello world
{{ bar|length }} --> 3
{{ bar|slice:“:2” }} --> [‘a’, ‘b’]
{{ baz|default:“error!” }} --> error!
```

Tags

- for loops
- if clauses
- comments
- blocks
- and many more built-in tags (look them up!)

- {% tag %} … {% endtag %}

Tags in weather.html

```
<html>
    <head>
        <title> Weather </title>
    </head>
    <body>
        <p>Today's weather in {{ city }} is {{ description }}.</p>
        <div id="temperature"
        {% for day in thisWeek %}
            <li> On {{ day.date }}, the temperature will be {{ day.temperature }}. </li>
        {% endfor %}

        </div>
        <div id="ads">
        {% block ads %}
        Click on these ads!
        {% endblock %}
        </div>
    </body>
</html>
```

For loops

```
{% for x in y %}
    … logic …
{% endfor %}

fruit_basket := {‘apples’, ‘oranges’, ‘pineapples’}

{% for fruit in fruit_basket %}
  <li>{{ fruit }}</li>
{% endfor}
```

```
      <li>apples</li>
-->   <li>orange</li>
      <li>pineapples</li>
```

If clauses

{% if <condition> %}

… logic …

{% else %}

… logic …

{% endif %}

```
{% if rain > 1 }
     Buy an umbrella for {{ price1 }}
{% else %}
     Buy sunglasses for {{ price2 }}
{% endif %}
```

Comments

```
{% comment %}
    This comment won't be displayed!
{% endcomment}
```

- Ignore everything inside tag
  - For inline comments, use {# blah blah blah #}

Template Inheritance

- Define extensible parts of a template with block tags

```
{% block name %}
   …
{% endblock %}
```

- Create child templates that can extend blocks
- Load parent template with

```
{% extends “parent_template” %}
```

weather.html

```
<html>
    <head>
        <title> Weather </title>
    </head>
    <body>
        <p>Today's weather in {{ city }} is {{ description }}.</p>
        <div id="temperature">
        {% for day in thisWeek %}
            <li> On {{ day.date }}, the temperature will be {{ day.temperature }}. </li>
        {% endfor %}

        </div>
        <div id="ads">
        {% block ads %}
        Click on these ads!
        {% endblock %}
        </div>
    </body>
</html>
```

ads.html

```
{% extends "weather.html" %}
{% block ads %}
{% if rain > 1 }
    Buy an umbrella!
{% else %}
    Buy sunglasses!
{% endif %}
{% endblock %}
```

Context

```
city = 'Accra‘
description = 'sunny‘
thisWeek = [dict(date='Thursday',temperature=20),
                dict(date='Friday', temperature=25),
                dict(date='Saturday', temperature=22)]
rain = 3
```

Displayed by browser

Today’s weather in Accra is sunny.

- On Thursday, the temperature will be 20.
- On Friday, the temperature will be 25.
- On Saturday, the temperature will be 22.

Click on these ads!

Buy an umbrella!

Template Inheritance

- In child template, redefine contents of the parent's block tag
  - similar to overriding methods in class inheritance
- If a block tag is not redefined, then use contents of block tag in parent
- {{ block.super }} explicitly refers to contents of block tag in parent

ads.html

```
{% extends "weather.html" %}
```

Context

```
city = 'Accra‘
description = 'sunny‘
thisWeek = [dict(date='Thursday',temperature=20),
                dict(date='Friday', temperature=25),
                dict(date='Saturday', temperature=22)]
rain = 3
```

Displayed by browser

Today’s weather in Accra is sunny.

- On Thursday, the temperature will be 20.
- On Friday, the temperature will be 25.
- On Saturday, the temperature will be 22.

Click on these ads!

Templates

- Mixture of hard-coded text and abstractions
- Abstractions often look like and function like Python code, but you can't run arbitrary Python code
  - Lookup list of built-in filters and tags in Django
  - Customize your own filters and tags
- Complex logic with arbitrary Python should be performed by views.py and only the processed variables should be passed to a template

Templates

Remember to specify where your templates are in TEMPLATE_DIRS in settings.py

Django Architecture