QuerySet. Conjunto de consultas Django

Un QuerySet es una colección de datos de una base de datos.

Un QuerySet se construye como una lista de objetos.

QuerySets facilita la obtención de los datos que realmente necesita, al permitirle filtrar y ordenar los datos.

Datos de la tabla Members

Members:

id	firstname	lastname
1	Emil	Refsnes
2	Tobias	Refsnes
3	Linus	Refsnes
4	Lene	Refsnes
5	Stalikken	Refsnes

Querying data. Consulta de datos

En **views.py**, tenemos una vista para pruebas llamada **testing** donde probaremos diferentes consultas.

En el siguiente ejemplo, usamos el método **.all()** para obtener todos los registros y campos del modelo Members:

View
members/views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Members

def testing(request):
  mydata = Members.objects.all()
  template = loader.get_template('template.html')
  context = {
    'mymembers': mydata,
  }
  return HttpResponse(template.render(context, request))
```

El objeto se coloca en una variable llamada **mydata** y se envía a la plantilla a través del objeto **context** como **mymembers** y se ve así:

```
<QuerySet [
 <Members: Members object (1)>,
 <Members: Members object (2)>,
 <Members: Members object (3)>,
 <Members: Members object (4)>,
 <Members: Members object (5)>
]>
```

Como puede ver, nuestro modelo de miembros contiene 5 registros y se enumeran dentro del QuerySet como 5 objetos.

En la plantilla puedes usar el mymembers objeto para generar contenido:

Template
template.html:

```
<table border='1'>
 <tr>
  <th>ID</th>
  <th>Firstname</th>
  <th>Lastname</th>
 </tr>
 {% for x in mymembers %}
  <tr>
   <td>{{ x.id }}</td>
    <td>{{ x.firstname }}</td>
   <td>{{ x.lastname }}</td>
  </tr>
 {% endfor %}
</table>
```

Django QuerySet Get Data. Obtener datos

Existen diferentes métodos para obtener datos de un modelo en un QuerySet.

El método values()

El método **values()** le permite devolver cada objeto como un diccionario de Python, con los nombres y valores como pares clave/valor:

Template

template.html

```
<!DOCTYPE html>
<html>
<body>

<p>The queryset object:</p>
{{ mymembers }}

<p>Loop through the items:</p>

<table border='1'>
 <tr>
  <th>ID</th>
  <th>Firstname</th>
  <th>Lastname</th>
 </tr>
 {% for x in mymembers %}
  <tr>
   <td>{{ x.id }}</td>
    <td>{{ x.firstname }}</td>
   <td>{{ x.lastname }}</td>
  </tr>
 {% endfor %}
</table>

</body>
</html>
```

View

members/views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Members

def testing(request):
  mydata = Members.objects.all().values()
  template = loader.get_template('template.html')
  context = {
    'mymembers': mydata,
  }
  return HttpResponse(template.render(context, request))
```

The queryset object:

<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'}, {'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes'}, {'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes'}, {'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes'}, {'id': 5, 'firstname': 'Stalikken', 'lastname': 'Refsnes'}]>

Loop through the items:

ID	Firstname	Lastname
1	Emil	Refsnes
2	Tobias	Refsnes
3	Linus	Refsnes
4	Lene	Refsnes
5	Stalikken	Refsnes

Devolver columnas específicas

El método **values_list()** le permite devolver solo las columnas que especifique.

Template

template.html

```
<!DOCTYPE html>
<html>
<body>

<p>The queryset object:</p>

{{ mymembers }}

<p>Loop through the items:</p>

<table border='1'>
 {% for x in mymembers %}
   <tr>
     <td>{{ x }}</td>
   </tr>
 {% endfor %}
</table>

</body>
</html>
```

View

members/views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Members

def testing(request):
 mydata = Members.objects.values_list('firstname')
 template = loader.get_template('template.html')
 context = {
   'mymembers': mydata,
 }
 return HttpResponse(template.render(context, request))
```

The queryset object:

<QuerySet [('Emil',), ('Tobias',), ('Linus',), ('Lene',), ('Stalikken',)]>

Loop through the items:

Emil
Tobias
Linus
Lene
Stalikken

Devolver filas específicas

Puede filtrar la búsqueda para que solo devuelva filas/registros específicos, utilizando el método **filter().**

Template

template.html

```
<!DOCTYPE html>
<html>
<body>

<p>The queryset object:</p>

{{ mymembers }}

<p>Loop through the items:</p>

<table border='1'>
 <tr>
  <th>ID</th>
  <th>Firstname</th>
  <th>Lastname</th>
 </tr>
 {% for x in mymembers %}
  <tr>
   <td>{{ x.id }}</td>
    <td>{{ x.firstname }}</td>
   <td>{{ x.lastname }}</td>
  </tr>
 {% endfor %}
</table>

</body>
```

```
</html>
```

View
members/views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Members

def testing(request):
 mydata = Members.objects.filter(firstname='Emil').values()
 template = loader.get_template('template.html')
 context = {
  'mymembers': mydata,
 }
 return HttpResponse(template.render(context, request))
```

The queryset object:

<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'}]>

Loop through the items:

ID	Firstname	Lastname
1	Emil	Refsnes

Django QuerySet Filter. Filtro

El método filter()se usa para filtrar su búsqueda y le permite devolver solo las filas que coinciden con el término de búsqueda.

Como aprendimos en el capítulo anterior, podemos filtrar por nombres de campo como este:

Ejemplo
Devuelve solo los registros donde el nombre es 'Emil':

```
mydata = Members.objects.filter(firstname='Emil').values()
```

En SQL, la declaración anterior se escribiría así:

```
SELECT * FROM members WHERE firstname = 'Emil';
```

and
El método filter()toma los argumentos como **kwargs (argumentos de palabras clave), por lo que puede filtrar en más de un campo separándolos con una coma.

Ejemplo
Devolver registros donde el apellido es "Refsnes" y la identificación es 2:

```
mydata = Members.objects.filter(lastname='Refsnes', id=2).values()
```

En SQL, la declaración anterior se escribiría así:

```
SELECT * FROM members WHERE   lastname = 'Refsnes' AND id = 2;
```

or
Devolver registros donde el nombre es Emil o el nombre es Tobias (es decir, devolver registros que coincidan con cualquiera de las consultas, no necesariamente con ambas) no es tan fácil como el ejemplo AND anterior.

Podemos usar múltiples filter()métodos, separados por un |carácter de canalización. Los resultados se fusionarán en un modelo.

Ejemplo
Devolver registros donde el nombre sea "Emil" o Tobias":

```
mydata = Members.objects.filter(firstname='Emil').values() |
Members.objects.filter(firstname='Tobias').values()
```

Otro método común es importar y usar expresiones Q:

Ejemplo
Devolver registros donde el nombre sea "Emil" o Tobias":

```
from django.http import HttpResponse
from django.template import loader
from .models import Members
from django.db.models import Q

def testing(request):
 mydata = Members.objects.filter(Q(firstname='Emil') | Q(firstname='Tobias')).values()
 template = loader.get_template('template.html')
 context = {
  'mymembers': mydata,
 }
 return HttpResponse(template.render(context, request))
```

En SQL, la declaración anterior se escribiría así:

```
SELECT * FROM members WHERE   firstname = 'Emil' OR firstname = 'Tobias';
```

Búsquedas de campo

Django tiene su propia forma de especificar sentencias SQL y cláusulas WHERE.

Para hacer cláusulas where específicas en Django, use "Búsquedas de campo".

Las búsquedas de campo son palabras clave que representan palabras clave específicas de SQL.

Ejemplo:

```
.filter(firstname__startswith='L');
```

Es lo mismo que la sentencia SQL:

```
WHERE firstname LIKE 'L%'
```

La declaración anterior devolverá registros donde el nombre comienza con 'L'.

Sintaxis de búsquedas de campo

Todas las palabras clave de búsqueda de campo deben especificarse con el nombre del campo, seguido de dos (!) caracteres de subrayado y la palabra clave.
En nuestro modelo de miembros, la declaración se escribiría así:

Ejemplo

Devuelve los registros donde el nombre comienza con la letra 'L':

```
mydata = Members.objects.filter(firstname__startswith='L').values()
```

Referencia de búsquedas de campo

Una lista de todas las palabras clave de búsqueda de campo:

Keyword	Description
contains	Contains the phrase
icontains	Same as contains, but case-insensitive
date	Matches a date
day	Matches a date (day of month, 1-31) (for dates)
endswith	Ends with
iendswith	Same as endswidth, but case-insensitive
exact	An exact match
iexact	Same as exact, but case-insensitive
in	Matches one of the values
isnull	Matches NULL values
gt	Greater than
gte	Greater than, or equal to
hour	Matches an hour (for datetimes)
lt	Less than
lte	Less than, or equal to
minute	Matches a minute (for datetimes)
month	Matches a month (for dates)
quarter	Matches a quarter of the year (1-4) (for dates)
range	Match between
regex	Matches a regular expression
iregex	Same as regex, but case-insensitive
second	Matches a second (for datetimes)
startswith	Starts with
istartswith	Same as startswith, but case-insensitive
time	Matches a time (for datetimes)
week	Matches a week number (1-53) (for dates)
week_day	Matches a day of week (1-7) 1 is sunday
iso_week_day	Matches a ISO 8601 day of week (1-7) 1 is monday
year	Matches a year (for dates)
iso_year	Matches an ISO 8601 year (for dates)

Django QuerySet Order by

ordenar por

Para ordenar QuerySets, Django usa el método order_by():

Ejemplo
Ordene el resultado alfabéticamente por nombre:

```
mydata = Members.objects.all().order_by('firstname').values()
```

En SQL, la declaración anterior se escribiría así:

```
SELECT * FROM members ORDER BY firstname;
```

Orden descendiente

Por defecto, el resultado se ordena de forma ascendente (primero el valor más bajo), para cambiar la dirección a descendente (primero el valor más alto), utilice el signo menos (NOT), - delante del nombre del campo:

Ejemplo
Ordene el primer nombre del resultado de forma descendente:

```
mydata = Members.objects.all().order_by('-firstname').values()
```

En SQL, la declaración anterior se escribiría así:

```
SELECT * FROM members ORDER BY firstname DESC;
```

Ordenamientos Múltiples

Para ordenar por más de un campo, separe los nombres de los campos con una coma en el order_by()método:

Ejemplo
Ordene el resultado primero por apellido ascendente, luego descendente en id:

```
mydata = Members.objects.all().order_by('lastname', '-id').values()
```

En SQL, la declaración anterior se escribiría así:

```
SELECT * FROM members ORDER BY lastname ASC, id DESC;
```