#### Searching for records

You can search for records that match certain criteria using the model's objects attribute (provided by the base class).

**Note**: Explaining how to search for records using "abstract" model and field names can be a little confusing. In the discussion below we'll refer to a Book model with title and genre fields, where genre is also a model with a single field name.

We can get all records for a model as a QuerySet, using objects.all(). The QuerySet is an iterable object, meaning that it contains a number of objects that we can iterate/loop through.

all\_books = Book.objects.all()

Django's filter() method allows us to filter the returned QuerySet to match a specified **text** or **numeric** field against particular criteria. For example, to filter for books that contain "wild" in the title and then count them, we could do the following.

wild\_books = Book.objects.filter(title\_\_contains='wild')

number\_wild\_books = wild\_books.count()

The fields to match and the type of match are defined in the filter parameter name, using the format: field\_name\_\_match\_type (note the double underscore between title and contains above). Above we're filtering title with a case-sensitive match. There are many other types of matches you can do: icontains (case insensitive), iexact (case-insensitive exact match), exact (case-sensitive exact match) and in, gt (greater than), startswith, etc. The [full list is here](https://docs.djangoproject.com/en/2.1/ref/models/querysets/#field-lookups).

In some cases you'll need to filter on a field that defines a one-to-many relationship to another model (e.g. a ForeignKey). In this case you can "index" to fields within the related model with additional double underscores. So for example to filter for books with a specific genre pattern, you will have to index to the name through the genre field, as shown below:

# Will match on: Fiction, Science fiction, non-fiction etc.

books\_containing\_genre = Book.objects.filter(genre**\_\_**name**\_\_**icontains='fiction')

**Note**: You can use underscores (\_\_) to navigate as many levels of relationships (ForeignKey/ManyToManyField) as you like. For example, a Book that had different types, defined using a further "cover" relationship might have a parameter name: type\_\_cover\_\_name\_\_exact='hard'.

There is a lot more you can do with queries, including backwards searches from related models, chaining filters, returning a smaller set of values etc. For more information see [Making queries](https://docs.djangoproject.com/en/2.1/topics/db/queries/) (Django Docs).