How to Filter for Empty or Null Values in a Django QuerySet

Posted by **AJ Welch**

Like Ruby on Rails and other high-level web frameworks, Django features a great many methods and functions to simplify even the most complicated tasks so developers can focus on writing applications that work. If you find yourself working with Django models but need to ensure the dataset you retrieve contains (or *doesn't* contain) empty or `NULL` values for a field, Django has you covered.

In this brief tutorial we'll explore the simple methods and chain functionality Django offers to make model filtering of fields a breeze.

Understanding Django Field Lookup Syntax

Before we get into the specific code examples to properly filter QuerySets, it's important to understand how Django handles keyword argument syntax and how it parses that information in a useful way.

While the official documentation provides far more detail, the crux of the syntax is the `double-underscore`, which is used to separate the `field` name from the `lookuptype` or 'function', followed by the value to compare to. Per the documentation, the syntax is written like so: `field__lookuptype=value`

Therefore, if are using field lookup/double-underscore syntax to find values where the `name` field of your model contains the word `Smith`, you'd use this syntax:

```
name__contains='Smith'
```

The critical part here is the syntax of the field name first, followed by a `double-underscore` and then the `lookuptype`. Things can be far more complex than this, but this knowledge will guide us through the rest of the tutorial.

Easily the most important method when working with Django models and the underlying QuerySets is the `filter()` method, which allows you to generate a QuerySet of objects that match a particular set of filtered parameters.

For example, our application has a `Book` model with a few basic fields: `title`, `author`, and `date_published`. We can quickly see that our database contains `20` books in total by using the `count()` method:

```
>>> Book.objects.count()
20
```

Now, by using `filter()`, we can retrieve a QuerySet of just those books that were published within the last 90 days period, like so:

```
>>> from datetime import datetime, timedelta
>>> Book.objects.filter(date_published__gte=datetime.now() - timedelta(days=90)).count()
3
```

With `filter()`, we can determine that merely 3 of our 20 total books were published within the last 90 day period.

Working with Exclude

Similar to the `filter()` method but acting as the mirror opposite is the `exclude()` method. As the name implies, `exclude()` functions by *excluding* all the matches from the supplied parameters, and returning the QuerySet that remains.

Thus if we use the exact same `date_published` example above but swap out `filter()` for `exclude()`, we'd expect the resulting book count to be the inverse: from `3` of `20` to now `17` of `20`:

```
>>> from datetime import datetime, timedelta
>>> Book.objects.exclude(date_published__gte=datetime.now() - timedelta(days=90)).count()
17
```

Sure enough that's exactly what we get: Every book in the system that wasn't *excluded* by our filtered date range – in other words, everything published more than 90 days ago.

Filtering and Excluding Empty or Null Values

Now that we understand the basics of using `filter()` and `exclude()` to retrieve a modified QuerySet in Django, we can use these methods to look for field values that are either empty or `NULL`.

Filtering Null Fields

As discussed above, there are a number of potential `field lookups` that can be used with `filter()` and `exclude()`, and when working with field values that are `NULL`, the field lookup of choice should be `isnull`.

Here we can use `isnull` to find all `Books` where the `author` field is `NULL`:

We've quickly determined that `2` of our books contain a `NULL` value for the `author` field.

Filtering Empty Fields

While there isn't a specific `field lookup` to locate empty fields (such as string fields that contain nothing but are not `NULL` in the database), we can approximate that functionality with the `exact` field lookup instead:

```
>>> Book.objects.filter(title__exact='').count()
1
```

In the above example we see that there is currently `1 Book` that has an empty string (`''`) in the `title` field.

Combining Filters

One final tip is that it is possible to combine multiple field lookups by chaining together `filter()` or `exclude()` calls.

Here, we'll use `exclude()` to remove all `Books` with *either* a `NULL` author or an empty `title` field. Since we know there are `2` and `1` books in each category, respectively, we'd expect to get a result of `17` out of the `20` total books:

```
>>> Book.objects.exclude(author__isnull=True).exclude(title__exact='').count()
17
```

There we have it! Some simple uses of `filter()` and `exclude()` to retrieve (or ignore) both `NULL` and empty values in Django QuerySets.

SIMILAR ARTICLES

What is Ad Hoc Analysis and How Does it Work?

Ad hoc analysis (aka ad hoc reporting) is the process of using business data to find specific answers to in-the-moment, often one-off, questions. It introduces flexibility and spontaneity to the traditionally rigid process of BI reporting (occasionally at the expense of accuracy).

Where to Find Free Datasets & How to Know if They're Good Quality

There is a lot of free data out there, ready for you to use for school projects, for market research, or just for fun. Before you get too crazy, though, you need to be aware of the quality of the data you find. Here are a few great sources for free data and a few ways to determine their quality.

Distinguishing Data Roles: Engineers, Analysts, and Scientists

Learn about the responsibilities that data engineers, analysts, scientists, and other related 'data' roles have on a data team.