Creating models

Now we'll define your models – essentially, your database layout, with additional metadata.

> Philosophy
>
> A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing. Django follows the DRY Principle. The goal is to define your data model in one place and automatically derive things from it.
>
> This includes the migrations - unlike in Ruby On Rails, for example, migrations are entirely derived from your models file, and are essentially a history that Django can roll through to update your database schema to match your current models.

In our poll app, we'll create two models: `Question` and `Choice`. A `Question` has a question and a publication date. A `Choice` has two fields: the text of the choice and a vote tally. Each `Choice` is associated with a `Question`.

These concepts are represented by Python classes. Edit the `polls/models.py` file so it looks like this:

`polls/models.py` ¶

```
from django.db import models


class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')


class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Here, each model is represented by a class that subclasses `django.db.models.Model`. Each model has a number of class variables, each of which represents a database field in the model.

Each field is represented by an instance of a `Field` class – e.g., `CharField` for character fields and `DateTimeField` for datetimes. This tells Django what type of data each field holds.

The name of each `Field` instance (e.g. `question_text` or `pub_date`) is the field's name, in machine-friendly format. You'll use this value in your Python code, and your database will use it as the column name.

You can use an optional first positional argument to a `Field` to designate a human-readable name. That's used in a couple of introspective parts of Django, and it doubles as documentation. If this field isn't provided, Django will use the machine-readable name. In this example, we've only defined a human-readable name for `Question.pub_date`. For all other fields in this model, the field's machine-readable name will suffice as its human-readable name.

Some `Field` classes have required arguments. `CharField`, for example, requires that you give it a `max_length`. That's used not only in the database schema, but in validation, as we'll soon see.

A `Field` can also have various optional arguments; in this case, we've set the `default` value of `votes` to 0.

Finally, note a relationship is defined, using `ForeignKey`. That tells Django each `Choice` is related to a single `Question`. Django supports all the common database relationships: many-to-one, many-to-many, and one-to-one.

Revision #1
Created 22 February 2023 18:01:53 by Admin
Updated 23 February 2023 06:23:16 by Admin