# Exercise 11: Creating a Model

30 to 45 minutes

In this exercise, you will create a `Joke` model to store joke data. Before doing that though, look in the `jokes` folder for a folder named `migrations`. The important thing to notice is that it does not contain any Python files other than `__init__.py`. When you change models, you will need to make migration files, which will get stored in this directory. Every time you change the model, you need to make a new migration file. Let's get started…

1. Open `jokes/models.py` and add the highlighted code after the `import` statement:

### Exercise Code 11.1: jokes/models.py

```
from django.db import models

class Joke(models.Model):
    question = models.TextField(max_length=200)
    answer = models.TextField(max_length=100, blank=True)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.question
```

**Things to notice about the Joke model:**

A. It inherits from `models.Model`. All models will inherit from `models.Model` or from one of its subclasses.

B. It has four fields:

   i. `question` – A `TextField` that can hold up to 200 characters.

   ii. `answer` – A `TextField` that can hold up to 100 characters. Because `blank` is set to `True`, this field can be left empty in forms that are created from this model. This would be for jokes that don't require an answer.

   iii. `created` – A `DateTimeField`. By setting `auto_now_add` to `True`, the field will automatically be assigned the current date and time when the joke is inserted.

   iv. `updated` – A `DateTimeField`. By setting `auto_now` to `True`, the value of the field will be changed to the current date and time when the joke is updated.

   C. The `__str__()` method determines what gets output when a `Joke` instance is converted to a string, either explicitly with `str()` or implicitly as occurs when an object is passed to certain functions such as `print()`.

2. As you have changed the model, you need to make new migration files. With `djangojokes.com` open in the terminal, run the following:

```
(.venv) …/projects/djangojokes.com> python manage.py makemigrations
Migrations for 'jokes':
  jokes\migrations\0001_initial.py
    - Create model Joke
```

3. Look again in the `jokes/migrations` folder. You will see that a `0001_initial.py` file has been added.
4. Now that you have created the migration file, you need to run the migration. Run the following:

```
(.venv) …/projects/djangojokes.com> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, jokes, sessions
Running migrations:
  Applying jokes.0001_initial... OK
```

   The migration is complete and the `Joke` model has been added to the database.

## ❖ E11.1. Exploring the Joke Model

Let's play with the new `Joke` model in the shell:

1. Open the shell:

```
(.venv) …/projects/djangojokes.com> python manage.py shell
```

2. Import the Joke class and retrieve all the `Joke` instances by running:

```
>>> from jokes.models import Joke
>>> Joke.objects.all()
<QuerySet []>
```

As you can see, this returns an empty `QuerySet`, which is an iterable containing records returned from a database. It is empty, because you haven't created any jokes yet.

3. Create your first joke:

```
>>> q = 'Why did the chicken cross the road?'
>>> a = 'To get to the other side.'
>>> joke = Joke(question=q, answer=a)
```

4. Now, let's look at the `QuerySet` again:

```
>>> Joke.objects.all()
<QuerySet []>
```

It's still empty. That's because you haven't saved the new joke. Run this:

```
>>> joke.save()
```

Now, check the `QuerySet`:

```
>>> Joke.objects.all()
<QuerySet [<Joke: Why did the chicken cross the road?>]>
```

There is your joke! Notice that it only outputs the question. That comes from the value returned by the `__str__()` function of the `Joke` class. You can get the values of the other fields easily enough:

```
>>> joke.question
'Why did the chicken cross the road?'
>>> joke.answer
'To get to the other side.'
>>> joke.created
datetime.datetime(2020, 8, 4, 21, 27, 53, 277762, tzinfo=<UTC>)
>>> joke.updated
datetime.datetime(2020, 8, 4, 21, 27, 53, 277762, tzinfo=<UTC>)
```

5. In addition to the fields explicitly named in the model, an auto-incrementing `id` field is added to every model as the primary key for the table. It can be referenced as `id` or as `pk` (for **p**rimary **k**ey):

```
>>> joke.id
1
>>> joke.pk
1
```

6. Add another joke:

```
>>> q = 'What kind of music do windmills like?'
>>> a = 'Dude, they\'re big heavy metal fans.'
>>> joke = Joke(question=q, answer=a)
>>> joke.save()
```

7. Now, look at the jokes again, this time using a for loop:

```
>>> for joke in Joke.objects.all():
...     print('Q:', joke.question)
...     print('A:', joke.answer)
...
Q: Why did the chicken cross the road?
A: To get to the other side.
Q: What kind of music do windmills like?
A: Dude, they're big heavy metal fans.
```

8. I think "*huge* heavy metal fans" sounds better than "*big* heavy metal fans." Let's update that. Assign the last joke you entered to a variable:

```
>>> windmill_joke = Joke.objects.last()
```

9. Look at the current answer to that joke:

```
>>> windmill_joke.answer
"Dude, they're big heavy metal fans."
```

10. Change the answer and save:

```
>>> windmill_joke.answer = "Dude, they're huge heavy metal fans!"
>>> windmill_joke.save()
```

11. Now, loop through the jokes again:

```
>>> for joke in Joke.objects.all():
...     print('Q:', joke.question)
...     print('A:', joke.answer)
...
Q: Why did the chicken cross the road?
A: To get to the other side.
Q: What kind of music do windmills like?
A: Dude, they're huge heavy metal fans.
```

12. Let's see how many jokes you have written:

```
>>> Joke.objects.count()
2
```

13. That joke about the chicken crossing the road never was funny. Let's delete it. You could get it using:

```
Joke.objects.first()
```

But let's assume you don't know that it is the first joke you entered and get it using the `get()` method:

```
>>> chicken_joke = Joke.objects.get(question = 'Why did the chicken cross the
road?')
```

14. Confirm you have the right joke:

```
>>> chicken_joke
<Joke: Why did the chicken cross the road?>
```

15. Delete it:

```
>>> chicken_joke.delete()
(1, {'jokes.Joke': 1})
```

The `delete()` method returns a tuple. The first element contains the total number of objects that were deleted. The second element contains a dictionary showing the number of deletions per object type. In this case, just one object was deleted: a `jokes.Joke` object.

16. Finally, let's see how many jokes you have left:

    ```
    >>> Joke.objects.count()
    1
    ```

    It looks like the delete worked.

This shows how to create a model and how to create, save, and view instances of that model, but with Django, you really want to do all this using web pages. You will learn how to do that next.

Make sure to exit the shell:

```
>>> exit()
(.venv) …/projects/djangojokes.com>
```

### Git Commit

Commit your code to Git.

## 2.4. Types of Views

You used a `TemplateView` to create the home page. `TemplateView` is a class-based view. Django has *view functions* in addition to *class-based views*, but class-based views make development much easier, so you will (mostly) stick with them. You imported `TemplateView` from `django.views.generic`. There are many other view classes available in that module, including:

- `ListView` – A view for listing objects (records in a queryset).
- `DetailView` – A view for displaying details about an object (one retrieved from a queryset).
- `CreateView` – A view for creating a new object.
- `UpdateView` – A view for updating an object.
- `DeleteView` – A view for deleting an object.

You will work with all of these views in the next exercises to list jokes, show joke details, and create, edit, and delete jokes.