Loading Django Data

John Strickler

Version 1.0, March 2021

Table of Contents

Chapter 1: Loading Django Data

Objectives

- Load data three ways
  - Create a data migration
  - Use **loaddata**
  - Create a new management command

Where do I start?

- Project database starts empty
- May need to add data for testing

Once you've created and migrated your models, your database is ready to use. And empty.

For a few records, you can add data via the admin interface or the Django shell.

However, you may want to add a number of records, for testing, or to migrate data from a previous incarnation of the project.

There are at least three ways to do this:

1. Create a data migration using the migration tools
2. Use **loaddata**
3. Create a new management command

NOTE	A fourth way is to write a script that is not part of your Django project, and that directly uses the database, but that is not recommended. It might create data that is incompatible with your app.

Creating a data migration

- Create an empty migration
- Implement code in migration file
- Call `manage.py migrate`
- Preferred approach

This is the preferred way to load data, as it becomes part of your data migration, so migrating on the deployment server is simpler.

To use migration tools, first create an empty migration with `manage.py makemigrations --empty <appname>`. This adds a migration file in the migrations folder, with no content. This will create a minimal migration script:

```
from django.db import migrations


class Migration(migrations.Migration):

    dependencies = [
        ('myapp', '0004_auto_20201031_2202'),
    ]

    operations = [
    ]
```

The **dependencies** list is pre-populated with the previous dependency

To perform a data migration, create a function that takes two arguments, an app registry (typically called `apps`) that manages previous versions of the app, and a schema editor. For normal use, you won't need the schema editor.

The app registry will be used to fetch objects from the database that match the current migration. Use `apps.get_model(<appname>, <model>)`. In the function, perform whatever tasks are needed for your data migration. Remember to save all modified objects. You may want to write helper methods so that your main function doesn't get unwieldy.

Call your function with `migrations.RunPython(<function>)`.

```
from django.db import migrations

def copy_data(apps, schema_editor):
    Model1 = apps.get_model('myapp', 'Model1')
    Model2 = apps.get_model('myapp', 'Model2')
    for model1 in Model1.objects.all():
        model1.some_field = "some value"
        model1.save()

class Migration(migrations.Migration):


    dependencies = [
        ('myapp', '0004_auto_20201031_2202'),
    ]

    operations = [
        migrations.RunPython(copy_data)
    ]
```

NOTE

If you make a mistake and need to re-run your migration, use the `--fake` option to `manage.py migrate`. Use this option with the migration *before* the one you want to re-run. Then call `manage.py migrate` as usual and it will re-run the migration:

```
manage.py migrate --fake <previous migration>
manage.py migrate
```

Using loaddata

- Create JSON or YAML data file(s)
- Use `manage.py loaddata`

You can use the manage.py command **loaddata** to insert data into your database. The data must be in JSON or YAML format. To see the correct format, add at least one record using the Django shell or the admin interface, then call `manage.py dumpdata <APP.MODEL> <FILENAME>' for JSON, or `manage.py dumpdate --format yaml <APP.MODEL> <FILENAME>` for YAML.

If you have related fields, you will have to load one table to get IDs, then write code to use that table to build the next table, etc.

Creating a new management command

- Add command to manage.py
- Create commands in *app*/management/command
- Inherit from BaseCommand
  - Add arguments
  - Define **handle()**

If you need to do some specific tasks relative to your databases (or any part of your app), you can add new commands to manage.py for convenience. These commands have access to your project's Django configuration.

To add a new commmand, create a folder called `management` in an app, then create a subfolder called `commands`. In that folder, you can create any number of modules (scripts), each of which will be a separate command. For example, `validate_cities.py` could then be called as `manage.py validate_cities`.

In each command script, create a class named `Command` which subclasses `BaseCommand`. Add a class level variable named **help** which is set to a description of what the command does.

If the command needs arguments, define the method `add_arguments()`, which has a parameter `parser`. This parser object will parse arguments passed in after your command name when it is called via `manage.py`. Usage is

```
parser.add_argument('<argument name>', type=<argument type>)
```

You must define a method named `handler()`. This will be the code that runs when you call your command. It takes two arguments. `args` is not normally needed, and `options` contains the options parsed in `add_arguments()`.

If your command needs to work with the database, you can import your models at the top of the script.

Chapter 1 Exercises

Exercise 1-1 (dogs/dogs_core/*)

Add a new field, **abbr** to Breed to hold the breed abbreviation. It should be a character field with a maximum size of 4. It will be created from the first two letters of the breed if it is a single word, or the first letters of each word if more than one word. Make the field nullable and migrate the changes.

Create a custom migration to populate the **abbr** field.

Exercise 1-2 (dogs/dogs_core/*)

Use **loaddata** to load data from the files **dog_data.yaml** and **breed_data.yaml** into your database. The yaml files are in the DATA folder.

Load the breeds first, and you will then have to dump the breeds table, and then manually copy the breed IDs into the dog file before loading it. In real life, you would write a script to do this.

Exercise 1-3 (dogs/dogs_core/*)

Create a new manage.py command named **addsnoopy** to add a dog to your database with the following data:

Name: Snoopy
Breed: Beagle
Weight: 30 Sex: M Neutered: Y

Run the command and confirm that Snoopy is added to your database.

NOTE	Your command should check to see if "Beagle" is in the breed table, and if not, add it.

Index