Django SQLite 3 Tutorial
Created alongside ticket 21-implement-sqlite-into-project
——
When originally creating/modifying a database in Django, consider the following steps:

Step 1 - Configuration
Tells Django that we have another application (web-app) that has some dependencies we need to set up.

1. Modify settings.py in helloworld directory
2. Navigate to INSTALLED_APPS
3. Add “app-name.apps.app-nameConfig”
    1. Ie “main.apps.MainConfig”

Step 2 - Creating Model
Our model(s) serve as our base data classes that we will manipulate, load data from, receive user input to, etc. They will represent objects or ideas in our system.
1. Navigate to your app-name> models.py
    1. Ie voting>models.py
2. Create new model class and define the data fields the model will have

Model Applicable Datatypes available in documentation:
https://docs.djangoproject.com/en/4.2/ref/models/fields/

Step 3 - Saving The Model
After defining the model, we must leverage Django’s framework to generate some needed implementation for the model. Similar to Git’s “staging area” we must make “migrations” to the database (and automatically generate files), and then “commit” them to the project.

1. Run “python manage.py makemigrations app-name”
    1. Ie “python manage.py makemigrations voting”

```
(django-dev) corey@Coreys-MacBook-Air Summer-Project-2023 % python manage.py makemigrations voting
Migrations for 'voting':
  voting/migrations/0001_initial.py
    - Create model Neighbourhoods
```

2. Run “python manage.py migrate”

```
(django-dev) corey@Coreys-MacBook-Air Summer-Project-2023 % python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, voting
Running migrations:
  Applying voting.0001_initial... OK
```

Step 4 - Registering To Admin
Our model is now functional within the system, and we are able to read and write from the database model we have just created. The database (by default sqlite3) is created for these read/write operations when you run the migrate command.

We must register the model to our "admin system" in order to manipulate and view the data from our panel. Assuming we already have an admin superuser setup, we can do the following to enable our model in the admin settings.

1. Navigate your app-name>admin.py
2. Include import statement to your class
    1. "from .models import ModelName"
    2. Ie "from .models import User"
3. After importing, register your model with:
    1. Add: "admin.site.register(ModelName)"
    2. Ie "admin.site.register(User)

Step 5a - Adding to Database (admin)

Now that our models are created and registered in our system, we can add to our database by simply filling our the forms that are automatically generated in the admin panel.

1. Run "python manage,py runserver"
2. Navigate to your url/admin
3. Login with admin credentials
    1. Username: admin
    2. Password: password
4. Navigate to your created model and add+

Step 5b - Adding to Database (local)

Now that our models are created, we can actually create new data entries directly within the terminal.

1. In the terminal, type the command "python manage.py shell"
    1. This will open up a python shell in our project that will let us execute commands - with this we can read and write to the database.
2. We must import our defined model into the shell:
    1. Type "from app-name.models import ModelName)
    2. Ie "from voting.models import User"
3. Now that our model is imported, we can create a new instance of our defined object:
    1. Type "t = User(first_name = "FirstName", last_name = "LastName")
    2. This will create an instance of our object
4. Now we can save this object to the database:
    1. Type "t.save()"

Note we can exit the shell with the shortcut Control + D

Step 6a - Viewing the Database (admin)

By navigating to our admin panel we can view the entires of our models.

Step 6b - Viewing the Database (local)

We can also view entires by running commands within the shell. Once the shell is activated, run the command:

User.objects.all()