


Open Opened 2 days ago by  [Rubén Montero](#)

Migraciones

Resumen

- Aprenderemos cómo generar una base de datos con `makemigrations` y `migrate`
- Cacharrearemos con la base de datos SQLite usando `sqlite3.exe`

Descripción

Tenemos dos modelos muy bonitos y relucientes en `models.py`.

Pero, ¿dónde está la base de datos?

Digno de recordar: Fichero de configuración

Si abres `settings.py` verás una parte que indica:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

No vamos a modificarlo, pero destacaremos que **ahí** está la configuración de la base de datos. Podríamos configurar una base de datos MySQL, PostgreSQL...

La nuestra es SQLite. Concretamente, se ubica en el archivo `db.sqlite3`.

Dos comandos muy sencillos

Las [migraciones](#) son la forma que tiene Django de *propagar* los cambios que hacemos en nuestro `models.py` a la base de datos.

Abre un terminal y navega hasta `WallAPI/`.

Lanzaremos el comando `makemigrations` que es responsable de crear nuevas *migraciones*.

```
python manage.py makemigrations
```

Esto ha generado **un nuevo** fichero en la carpeta `migrations/`. Ese fichero contiene la información de los *cambios en* `models.py` procesada por Django.

¿Y para qué?

Para que funcione nuestro segundo comando.

Ejecutemos:

```
python manage.py migrate
```

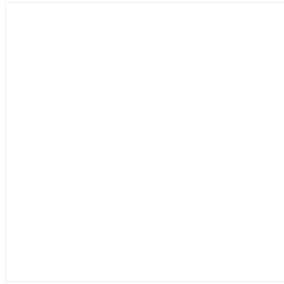
Esto *leerá* el contenido de `migrations/` y **ejecutará** las sentencias SQL necesarias para crear las tablas en la base de datos.

¡Migraciones realizadas!

¡Excelente!

¿El proceso inverso sería posible?

Sí. Con `python manage.py inspectdb`:



No lo pondremos en práctica.

Vale. Entonces... ¿Con **makemigrations** y **migrate** hemos creado tablas SQL?

Sí.

No lo creo...

Haces bien. Siempre se debe [comprobar lo que vamos haciendo](#)

sqlite3.exe

Es un cliente SQL muy sencillo. Se encuentra en tu repositorio dentro de `python-django/`, en el primer nivel.

Si haces **doble click** sobre él, se lanzará un terminal como este:

```
SQLite version 3.39.0 2022-06-25 14:57:57
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

(Debes abrirlo desde las carpetas de Windows, **no** desde PyCharm)

Vamos a ejecutar `.open` para abrir la base de datos del proyecto Django:

```
sqlite> .open apis/WallAPI/db.sqlite3
```

Este par de comandos ayudarán a mejorar la legibilidad de los resultados de las consultas:

```
sqlite> .mode column
sqlite> .headers on
```

Si ejecutamos `.tables` veremos las tablas de la base de datos. En concreto las interesantes son las correspondientes a los modelos que hemos creado nosotros.

```
sqlite> .tables
```

¿Distingues **cuáles** son?

Un sencillo **SELECT**

Podemos comprobar que dichas tablas están vacías haciendo **SELECT** :

```
sqlite> SELECT * FROM wallrest04app_entry;
sqlite> SELECT * FROM wallrest04app_comment;
sqlite>
```

Como ves no hay resultados, pero las tablas están ahí. Puedes comprobar su estructura con `.schema <NombreTabla>`

Un sencillo **INSERT**

¡Probemos a agregar una fila a `wallrest04app_entry` !

```
sqlite> INSERT INTO wallrest04app_entry (title, content, publication_date)
...> VALUES ("Hola", "Un primer ladrillo", datetime('now'));
```

Después de esto, el `SELECT` sí dará resultados:

¡Compruébalo!

¿No habíamos dicho que **no** escribiríamos SQL?

Sí.

Estos pasos **no** son necesarios para *desarrollar* el API REST, pero ayudan a entender la interrelación entre nuestro código ORM y la base de datos.

Terminando

Puedes teclear `.exit` cuando quieras para salir de `sqlite3.exe`.

Por último

Comprueba que tu código pasa el *test* asociado a la tarea.

Haz `commit` y `push` para subir los cambios al repositorio.



[Rubén Montero @ruben.montero](#) changed milestone to [%Sprint 4](#) 2 days ago