

Informe de migración de datos - CodeProject.

Introducción, objetivos y justificación.

Introducción.

El informe de migración de datos busca documentar y exponer al personal técnico, por medio de una información detallada, el paso a paso para la configuración de los SGBD y los archivos **Django** del proyecto para el control de inventario.

Objetivo.

Elaborar una guía paso a paso orientada al personal técnico encargado, que especifique los procesos de migración de datos a los distintos SGBD.

Objetivos específicos.

- Identificar a quién va dirigido el manual de instalación.
- Explicar, de forma muy detallada y gráfica, el proceso de migración de datos y las herramientas que se emplearon para desarrollar dicha actividad.

Justificación.

El presente informe de migración de datos le permitirá personal técnico saber qué herramientas se emplearon y cómo se hicieron las migraciones de datos. La función de este manual también es documentar el proceso de desarrollo del sistema informático dirigido a la comercializadora de ropa infantil Violetas.

Configuración de los SGBD.

Lo primero que debemos hacer es tener configurado nuestro ambiente virtual, nuestro proyecto **Django** y las librerías y controladores. Si no sabe cómo hacerlo, véase estas entradas:

→ **Instalación del ambiente virtual.**

[/manual-de-instalacion-codeproject./instalacion-del-ambiente-virtual.](#)

→ **Instalación del framework Django.**

[/manual-de-instalacion-codeproject./instalacion-del-framework.](#)

→ **Instalación de las librerías y controladores.**

[/manual-de-instalacion-codeproject./instalacion-de-las-librerias.](#)

Comprobación de dependencias y controladores.

- Bien, encendemos nuestro ambiente y ejecutamos el siguiente comando para poder saber qué dependencias y controladores tiene instalado nuestro proyecto.

```
pip freeze
```

- Asegúrese de tener instalados los siguientes controladores: ***psycopg2==2.9.1*** y ***mysqlclient==2.0.3***.

Creación archivo db.py.



Cabe aclarar que los controladores de SQLite vienen por defecto y no debe instalarlos manualmente.

- El siguiente paso es crear el archivo **db.py** en el mismo área de trabajo donde se encuentra ubicado el archivo **settings.py**. El archivo **db.py** debe contener la siguiente información dentro de sí.

```
1  import os
2
3  BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
4
5  SQLITE = {
6      'default': {
7          'ENGINE': 'django.db.backends.sqlite3',
8          'NAME': os.path.join(BASE_DIR, 'db.sqlite3')
9      }
10 }
11
12 POSTGRESQL = {
13     'default': {
14         'ENGINE': 'django.db.backends.postgresql_psycopg2',
15         'NAME': 'pydb',
16         'USER': 'postgres',
17         'PASSWORD': 'root',
18         'HOST': 'localhost',
19         'PORT': '5432'
20     }
21 }
22
23 MYSQL = {
24     'default': {
25         'ENGINE': 'django.db.backends.mysql',
26         'NAME': 'modelosdjango',
27         'USER': 'root',
28         'PASSWORD': 'root',
29         'HOST': 'localhost',
30         'PORT': '3306'
31     }
32 }
```

Explicación del archivo db.py.

- Lo primero es importar las funciones específicas del sistema operativo con el siguiente bloque de código:

```
1 import os
2
3 BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
```

- A continuación, debemos agregar los diccionarios de datos con la información de nuestros SGBD. Tomaré como ejemplo el diccionario de datos de **MySQL**.



Dentro de cada diccionario debemos tener en cuenta las siguientes cosas.

- **NAME:** Hace referencia al nombre de nuestra base de datos.
- **USER:** Hace referencia al usuario que tengamos registrado en nuestra base de datos.
- **PASSWORD:** Hace referencia a la contraseña que hayamos decidido para el usuario.
- **LOCALHOST** y **PORT:** Hacen referencia a la ubicación y el servidor donde se despliega el SGBD.

```
1 MYSQL = {
2     'default': {
3         'ENGINE': 'django.db.backends.mysql',
4         'NAME': 'modelosdjango',
5         'USER': 'root',
6         'PASSWORD': 'root',
7         'HOST': 'localhost',
8         'PORT': '3306'
9     }
10 }
```

- Debemos agregar los tres SGBD que manejamos y asegurarnos de que los datos del diccionario estén bien escritos.

Configuración settings.py.

- El primer paso es dirigirnos al archivo llamado **settings.py** e importar el archivo que creamos previamente (**db.py**).



```
1 import os
2 from django.urls import reverse_lazy
3 from pathlib import Path
4 import CodeProject.db as db
```



Cabe aclarar que en el apartado que dice **CodeProject** debe ir el nombre del proyecto.

```
1 import os
2 from django.urls import reverse_lazy
3 from pathlib import Path
4 import CodeProject.db as db
```

- El siguiente paso que debemos hacer es, ubicados en el mismo archivo de **settings.py**, agregar el siguiente comando que nos permitirá decirle al proyecto **Django** que tome ahora como archivo para la conexión con la base de datos nuestro archivo **db.py**.

```
DATABASES = db.MYSQL
```

- Con esto ya tendríamos nuestra conexión a los tres diferentes SGBD.

Cambiar de SGBD.

- Lo que debemos hacer para cambiar de SGBD es, después del igual y el db. escribir a cuál SGBD queremos conectarnos. Por ejemplo:

```
DATABASES = db.SQLITE
```

```
DATABASES = db.POSTGRESQL
```

Evidencias de la migración de datos en los SGBD.

A continuación, encontrará las diferentes migraciones en los distintos SGBD.

→ **Migración MySQL Workbench.**

/informe-de-migracion-de-datos-
codeproject./migracion-de-datos-en-los-
sgbd./migracion-mysql-workbench.

→ **Migración SQLite.**


/informe-de-migracion-de-datos-
codeproject./migracion-de-datos-en-los-
sgbd./migracion-sqlite.

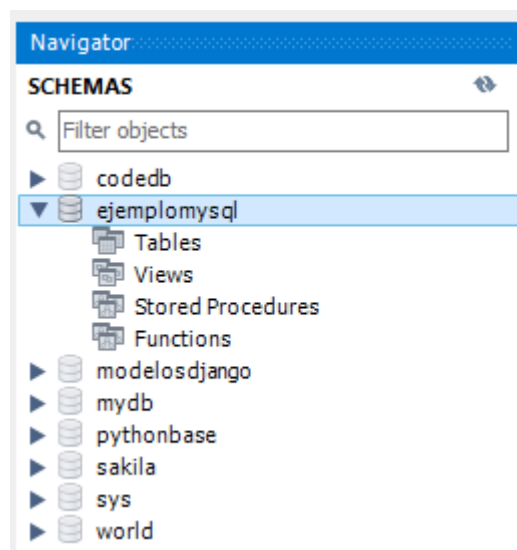
→ **Migración PostgreSQL.**

/informe-de-migracion-de-datos-
codeproject./migracion-de-datos-en-los-
sgbd./migracion-postgresql.

Migración MySQL Workbench.

- Lo primero que debemos hacer es crear una schema vacío en **MySQL Workbench**. En este caso, el nombre de la base de datos es **ejemplomysql**.

 Es válido aclarar que las tablas están vacías y que cuando realicemos la migración, la base de datos pasará a tener 10 tablas.



- Después de esto, cambiamos el nombre de la base de datos en nuestro **db.py**.

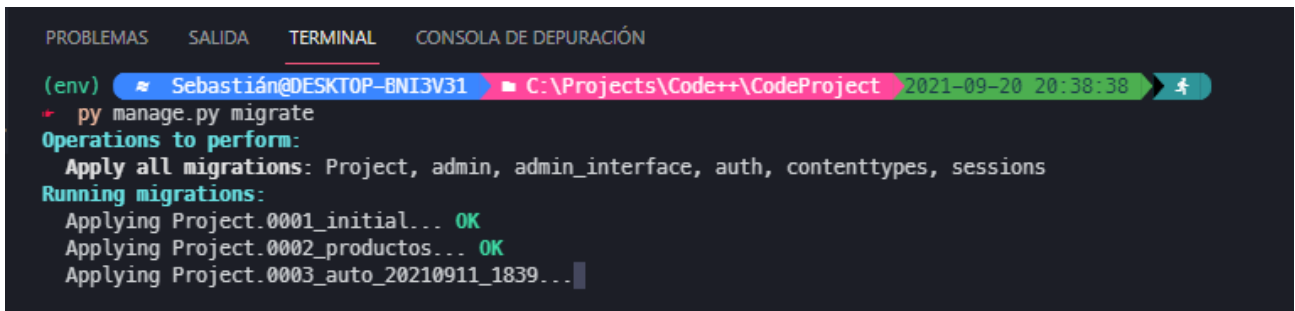
```
1  MYSQL = {  
2      'default': {  
3          'ENGINE': 'django.db.backends.mysql',  
4          'NAME': 'ejemplomysql',  
5          'USER': 'root',  
6          'PASSWORD': 'root',  
7          'HOST': 'localhost',  
8          'PORT': '3306'  
9      }  
10 }
```

- Continuando con el proceso, debemos ahora iniciar el ambiente virtual y con éste encendido, ejecutar el siguiente comando que nos permitirá hacer las respectivas

migraciones y posteriormente tener tablas predeterminadas en nuestra base de datos.

```
py manage.py migrate
```

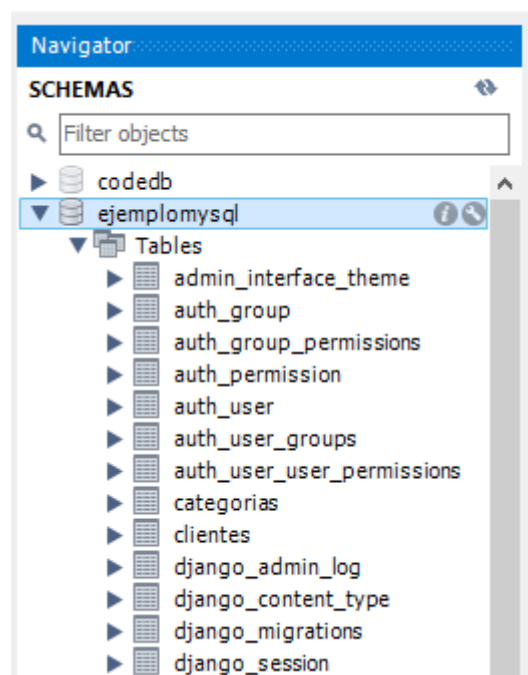
- El proceso de migraciones se vería similar al siguiente:



```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

(env) Sebastián@DESKTOP-BNI3V31 C:\Projects\Code++\CodeProject 2021-09-20 20:38:38
py manage.py migrate
Operations to perform:
  Apply all migrations: Project, admin, admin_interface, auth, contenttypes, sessions
Running migrations:
  Applying Project.0001_initial... OK
  Applying Project.0002_productos... OK
  Applying Project.0003_auto_20210911_1839...
```

- Al finalizar el proceso por consola, debemos dirigirnos a **MySQL Workbench** y comprobar que ahora existan las tablas en nuestra base de datos.



- Con esto habrá finalizado el proceso de migración para **MySQL Workbench**.

Migración SQLite.

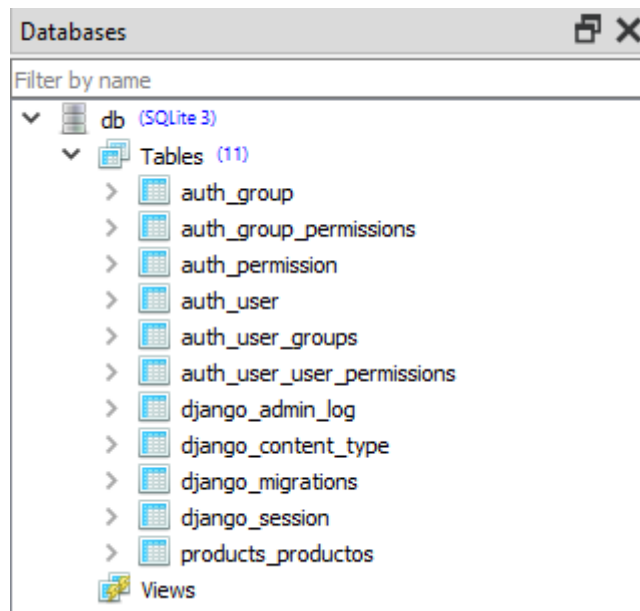
- Al haber creado un proyecto en Django, éste por defecto nos crea una base de datos, así que lo único que debemos ajustar es el archivo ***settings.py*** configurando la conexión con el SGBD que queremos, en este caso, **SQLite**.

```
DATABASES = db.SQLITE
```

- Después de eso, ejecutamos de nuevo, con el ambiente virtual activo, el siguiente comando que nos permitirá hacer las respectivas migraciones.

```
python manage.py migrate
```

- Comprobamos que se hayan creado nuestras tablas, y listo, el proceso de migración de datos hacia **SQLite** habría acabado.



Migración PostgreSQL.

- Lo primero que debemos hacer es crear una schema vacío en **pgAdmin 4**. En este caso, el nombre de la base de datos es **pydb**.
- Después de esto, cambiamos el nombre de la base de datos en nuestro **db.py**.

```
1  POSTGRESQL = {  
2      'default': {  
3          'ENGINE': 'django.db.backends.postgresql_psycopg2',  
4          'NAME': 'pydb',  
5          'USER': 'postgres',  
6          'PASSWORD': 'root',  
7          'HOST': 'localhost',  
8          'PORT': '5432'  
9      }  
10 }
```















- Lo siguiente que debemos ajustar es el archivo **settings.py** configurando la conexión con el SGBD que queremos, en este caso, **PostgreSQL**.

```
DATABASES = db.POSTGRESQL
```

- Después de eso, ejecutamos de nuevo, con el ambiente virtual activo, el siguiente comando que nos permitirá hacer las respectivas migraciones.

```
python manage.py migrate
```

- Nos dirigimos a **pgAdmin 4** y deberían verse reflejadas nuestras tablas ahora. Con esto finalizaría el proceso de migración de datos hacia **PostgreSQL**.

- ▼  Tables (13)
 - >  auth_group
 - >  auth_group_permissions
 - >  auth_permission
 - >  auth_user
 - >  auth_user_groups
 - >  auth_user_user_permissions
 - >  category
 - >  django_admin_log
 - >  django_content_type
 - >  django_migrations
 - >  django_session
 - >  product
 - >  ventas