

Les modèles et migration

- ✓ L'ORM de Django permet de simplifier les opérations liées aux bases de données.
- ✓ Avec l'ORM Nous n'avons pas besoin d'utiliser des commandes SQL pour effectuer des opérations sur la base de données.
- ✓ La transformation du code Python en structures de base de données est appelée migration.

Configuration Par défaut

```
68 DATABASES = {
69     'default': {
70         'ENGINE': 'django.db.backends.sqlite3',
71         'NAME': BASE_DIR / 'db.sqlite3',
72     }
73 }
```



- La base de données utilisé par défaut est de type **SQLite**, un fichier sur votre ordinateur.
- La valeur par défaut du champ **NAME** est : **BASE_DIR/'db.sqlite3'**, ce fichier est stocké dans le répertoire de projet



Les deux réglages qui sont obligatoires sont **ENGINE** et **NAME** :

- **ENGINE** :

- 'django.db.backends.sqlite3',
- 'django.db.backends.postgresql',
- 'django.db.backends.mysql'
- 'django.db.backends.oracle'.

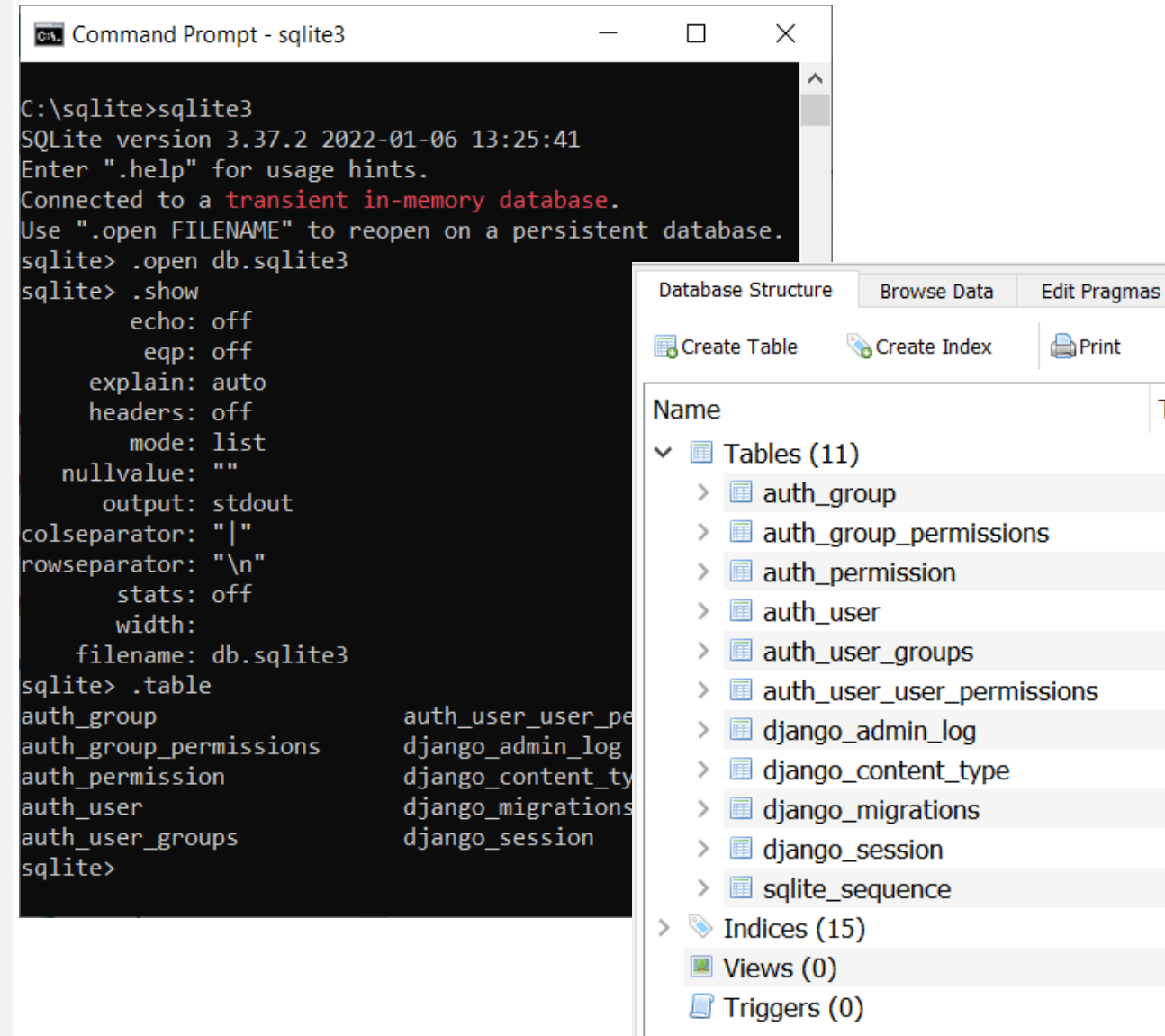
- **NAME** : Le nom de la base de données.

D'autres réglages supplémentaires peut être indiqués comme **USER**, **PASSWORD** et/ou **HOST**.

Pour créer les tables dans la base

```
python manage.py migrate
```

La commande **migrate** examine le réglage **INSTALLED_APPS** et crée les tables de base de données nécessaires en fonction des réglages de base de données dans le fichier **nomDuProjet/settings.py** et des migrations de base de données contenues dans l'application.



The screenshot shows a Command Prompt window titled "Command Prompt - sqlite3" and a database structure viewer. The Command Prompt shows the following commands and output:

```
C:\sqlite>sqlite3
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open db.sqlite3
sqlite> .show
      echo: off
       eqp: off
    explain: auto
   headers: off
        mode: list
 nullvalue: ""
    output: stdout
colseparator: "|"
rowseparator: "\n"
       stats: off
        width:
      filename: db.sqlite3
sqlite> .table
auth_group          auth_user_user_pe
auth_group_permissions  django_admin_log
auth_permission      django_content_ty
auth_user            django_migrations
auth_user_groups     django_session
sqlite>
```

The database structure viewer shows the following tables (11):

- auth_group
- auth_group_permissions
- auth_permission
- auth_user
- auth_user_groups
- auth_user_user_permissions
- django_admin_log
- django_content_type
- django_migrations
- django_session
- sqlite_sequence

Indices (15), Views (0), and Triggers (0) are also listed.

```
68 DATABASES = {
69     'default': {
70         'ENGINE': 'django.db.backends.mysql',
71         'NAME': 'djangoTuto',
72         'USER': 'root',
73         'PASSWORD': '',
74         'HOST': 'localhost',
75         'PORT': '3306',
76     }
77 }
```

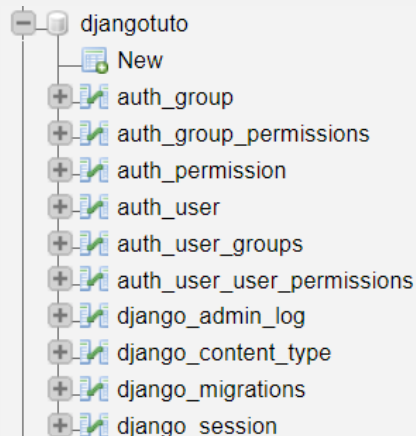
`python manage.py migrate`

- Pour connecter l'application à une base de données **mysql**, il faut installer un package du python **mysqlclient** avant de lancer la commande **migrate**.

`pip install mysqlclient`

- Ainsi, il faut y avoir un serveur web MySQL dans la machine.





- Un **modèle** est une **classe Python** qui contient le schéma de création d'une table dans une **base de données**. Elle hérite de **django.db.models.Model**
- Chaque **modèle** correspond à une seule **table de base de données**.
- Chaque **attribut** du modèle représente un **champ** de la base de données
- Le fichier **models.py** peut contenir **plusieurs modèles**.

Exemple :

```
models.py x
from django.db import models

class NomModel(models.Model):
    premier_champ = models.CharField(max_length=30)
    deuxieme_champ = models.CharField(max_length=30)
```

Chaque champ de votre modèle doit être une instance de la classe Field appropriée.

- **IntegerField** : pour des entiers
- **FloatField** : pour des nombre floeat
- **CharField** : utilisé pour des chaînes de caractères courtes.
- **EmailField** : pour les email
- **URLField** : pour des URL....

Une fois, le modèle est créé. Nous allons migrer les modèles Django vers la base de données.

À l'aide de l'utilitaire manage.py on lance la cette commande

python manage.py makemigrations

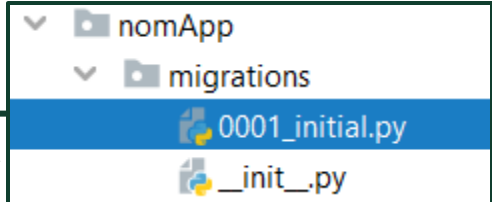
Cette commande va déterminer quelles modifications ont été apportées aux modèles.

python manage.py migrate

La commande **migrate** va réaliser des changements dans la base de données.

NB : Django ajoute un champ supplémentaire nommé **id** de type Integer pour la clé primaire.

```
PS D:\djangoTP\monSite\nomDuProjet> python manage.py makemigrations
Migrations for 'nomApp':
  nomApp\migrations\0001_initial.py
    - Create model NomModel
```



nomApp_nommodel			CREATE TABLE "nomApp_nommodel" ("id"
id	integer	"id" integer NOT NULL	
premier_champ	varchar(30)	"premier_champ" varchar(30) NOT NULL	
deuxieme_champ	varchar(30)	"deuxieme_champ" varchar(30) NOT NULL	



Pour mieux comprendre comment Django transforme un modèle en une table de base de données, exécutez la commande **sqlmigrate**

python manage.py sqlmigrate nomApp 0001_initial

Django nous offre un outil se forme d'un interpréteur interactif pour manipuler les modèles comme dans une vue.

Il suffit d'utiliser une autre commande de manage.py

python manage.py shell

```
PS D:\djangoTP\monSite\nomDuProjet> python manage.py shell
Python 3.10.2
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from nomApp.models import NomModel
>>> m = NomModel(1,'contenu premier champs','contenu 2eme champs')
>>> print(m.id)
1
>>> m.save()
>>> m = NomModel(2,'contenu 2 premier champs','contenu 2 2eme champs')
>>> m.save()
```

Database Structure

Browse Data

Edit Pragmas

Execute SQL

Table:

nomApp_nommodel

	id	premier_champ	deuxieme_champ
	Filter	Filter	Filter
1	1	contenu premier champs	contenu 2eme champs
2	2	contenu 2 premier champs	contenu 2 2eme champs

.save(): sauvegarder une entrée

.delete(): supprimer une entrée

.objects.create() : créer un enregistrement.

.objects.all(): obtenir toutes les entrées enregistrées.

.objects.filter() : obtenir les entrées enregistrées avec des critères.

.objects.exclude(): exclure les entrées par des critères.

.objects.order_by(): obtenir les entrées ordonnées ...



- Vous pouvez remplir votre base de données avec des données aléatoire à l'aide d'un package appelée Faker.
- Voici les étapes à suivre pour ajouter des données aléatoire à un modèle Django :

1. Install the Faker package
2. Ouvrir le shell de Django à l'aide de la commande
3. Importer les modèles nécessaires et le package Faker
4. Créer un instance de la classe Faker
5. Utiliser l'instance Faker pour générer des données et les enregistrer dans la base de données
6. Vérifiez que les données ont été ajoutées à la base de données

```
pip install faker
```

```
python manage.py shell
```

```
from .models import NomModel
from faker import Faker
f = Faker()
for i in range(10):
    m = NomModel(
        field1=f.name(),
        field2=f.address(),
        field3=f.phone_number(),)
    m.save()
NomModel.objects.all()
```


- Créez un **projet**, puis créer une **application** nommée **hopital**
- Créez un modèle **Patient**: chaque patient est défini par :
 - Son **nom** de type **CharField**
 - Son **date de naissance** de type **DateField**
 - **Malade** de type **BooleanField**
- En utilisant **shell** de la ligne de commande de Django :
 - Insérez 5 enregistrement dans la table `hopital_patient`
 - Manipulez d'autre opération CRUD.
- Créez un fichier de template qui présente les données de la forme motionné dans la figure.
- Créez une vue qui renvoie un fichier de template avec des données.
- Configurez les **URLs**, pour qu'une vue soit afficher comme une page d'accueil.

ID	Nom	Date	Malade
1	Hassan	2020-04-08	false
2	Mohammed	2020-04-08	true
3	Imane	2020-04-08	false
4	Yasmine	2020-04-08	true
5	Hassan	2020-04-08	false