



## Présentation ADRAR Pôle Numérique

Suivez-nous... LinkedIn  et Twitter  @Adrar\_Numerique



# ADRAR PÔLE NUMERIQUE

PÔLE NUMERIQUE DU CENTRE DE FORMATION ADRAR

- > SUPPORT, ADMINISTRATION SYSTEMES & RESEAUX
- > DEVELOPPEMENT D'APPLICATIONS WEB & MOBILES
- > WEBDESIGN & DEVELOPPEMENT INTERFACES
- > TRANSFORMATION NUMERIQUE DES ENTREPRISES

<http://www.adrar-numerique.com>

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)



Suivez-nous sur LinkedIn  Twitter  **@Adrar\_Numerique**

... et sur le web : **[www.adrar-numerique.com](http://www.adrar-numerique.com)**

# MISE EN PLACE DES DONNÉES

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Nous allons maintenant apprendre à mettre en place notre base de données, à l'intégrer dans notre projet et à la manipuler.

Les informations relatives à la connexion à la base de données se trouvent dans le fichier *monsite/settings.py*. Dans ce fichier vous avez une variable `DATABASES` qui est un dictionnaire comprenant les diverses informations de la BDD : le moteur de base de données utilisé ainsi que le nom de la base. Une base `sqlite3` est configuré par défaut, c'est ce que nous utiliserons pour ce tp mais vous pourriez très bien en utiliser une autre. Il faudrait pour cela modifier la clef `ENGINE` pour y indiquer le moteur de votre base et la clef `NAME` pour le nom de la base (dans le cas de `sqlite3` le nom de la base est un fichier sur votre machine. Voici un exemple de moteurs possibles (d'autres sont également disponibles sur la doc officielle de django) :

`django.db.backends.sqlite3`  
`django.db.backends.postgresql`  
`django.db.backends.mysql`  
`django.db.backends.oracle`

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



Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Tant que vous êtes dans le fichier *monsite/settings.py* dirigez vous tout en bas du fichier afin de modifier la valeur de la variable `TIME_ZONE` afin d'y indiquer 'Europe/Paris'. Cela permettra, lorsqu'on utilisera des fonctions de dates et d'heure, d'être sur le bon fuseau horaire.

```
TIME_ZONE = 'Europe/Paris'
```

Nous allons également rajouter notre application dans la liste des applications de notre projet, toujours dans le fichier *monsite/settings.py*. Repérer la variable liste nommé `INSTALLED_APPS` et rajoutez y la valeur : `'sondage.apps.SondageConfig'`,

```
INSTALLED_APPS = [  
    'sondage.apps.SondageConfig',  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]
```

Suivez-nous sur LinkedIn Twitter @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Ayant configuré notre projet nous pouvons maintenant mettre en place nos modèles servant à la mise en place de notre base de donnée. Cela se passe dans le fichier *sondage/models.py*. Notre projet va consister en un site visant à recueillir les avis de personnes grâce à des sondages. Il y aura donc dans notre base des questions ainsi que des choix. Voici le contenu de notre fichier *sondage/models.py* :

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date de publication')

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Nous commençons par import *models* du module *django.db*.

Nous créons ensuite une classe par tables de notre base héritant de la classe *Model*.

Nous définissons ensuite les champs de nos tables sous formes d'objets de classes *Charfield*, *DateTimeField*, *ForeignKey* ou encore *IntegerField*. Plusieurs paramètres sont précisables dans la construction. Le premier paramètre, facultatif, d'un objet *Field* est utilisé pour donner un nom plus lisible au champ (comme pour la variable *pub\_date*)

Suivez-nous sur LinkedIn  Twitter  **@Adrar\_Numerique**

... et sur le web : **[www.adrar-numerique.com](http://www.adrar-numerique.com)**

Maintenant que nos modèles sont faits nous pouvons indiquer à django de l'appliquer à notre base de données. Cela se fait en deux étapes : tout d'abord préparer la migration vers notre base, puis la migrer. Pour préparer la migration il faut utiliser la commande *makemigrations sondage* du script *manage.py*

```
PS C:\Users\maximeprzybylo\Documents\Cours\Django\TP\monsite> python .\manage.py makemigrations sondage
Migrations for 'sondage':
  sondage\migrations\0001_initial.py
    - Create model Question
    - Create model Choice
```

Suivez-nous sur LinkedIn  Twitter  **@Adrar\_Numerique**

... et sur le web : **[www.adrar-numerique.com](http://www.adrar-numerique.com)**

Suite à la première commande django créé un fichier de migration dans le dossier *migrations* de notre répertoire *sondage*. Vous pouvez visualiser le code sql de votre migration grâce à la commande *sqlmigrate sondage 0001*. 0001 ici symbolise le numéro de la migrations que vous voulez visualiser. /!\ Cette commande n'a aucune action sur votre base, elle permet juste de visualiser la requête.

```
PS C:\Users\maximeprzybylo\Documents\Cours\Django\TP\monsite> python .\manage.py sqlmigrate sondage 0001
BEGIN;
--
-- Create model Question
--
CREATE TABLE "sondage_question" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "question_text" varchar(200) NOT NULL, "pub_date" datetime NOT NULL);
--
-- Create model Choice
--
CREATE TABLE "sondage_choice" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "choice_text" varchar(200) NOT NULL, "votes" integer NOT NULL, "question_id" bigint NOT NULL REFERENCES "sondage_question" ("id") DEFERRABLE INITIALLY DEFERRED);
CREATE INDEX "sondage_choice_question_id_2421cbe7" ON "sondage_choice" ("question_id");
COMMIT;
```

Vous pouvez constater que django crée automatiquement un champ *id* comme clef primaire de chaque table. Le code que vous voyez affiché via cette commande varie en fonction du moteur que vous avez sélectionné.



Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Une fois la migrations créée vous devez l'appliquer grâce à la commande *migrate* du script *manage.py*

```
PS C:\Users\maximeprzybylo\Documents\Cours\Django\TP\monsite> python .\manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, sondage
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
  Applying sondage.0001_initial... OK
```

On voit ici que django applique les migrations de toutes les application renseignées dans le fichier *monsite/settings.py*

Suivez-nous sur LinkedIn Twitter @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Nous pouvons maintenant utiliser le shell fournit avec le scripte *manage.py* afin de tester nos tables. Ce shell est simplement un invité de commande python en liens directe avec votre projet.

```
PS C:\Users\maximeprzybylo\Documents\Cours\Django\TP\monsite> python .\manage.py shell
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from sondage.models import Choice, Question
>>> Question.objects.all()
<QuerySet []>
>>> from django.utils import timezone
>>> q = Question(question_text="Quel language préférez vous ?", pub_date=timezone.now())
>>> q.save()
>>> q.id
1
>>> q.question_text
'Quel language préférez vous ?'
>>> q.question_text="Quel language de programmation préférez vous ?"
>>> q.save()
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
```

Nous commençons par importer nos différents modèles.

Nos classes disposent de différentes méthodes, notamment `objects.all()` permettant d'afficher tous les éléments présents dans notre table.

On peut très bien faire des enregistrement dans notre base en créant un objet de la classe de notre table puis en utilisant la méthode `save()` sur notre objet pour l'enregistrer dans la base. Nous pouvons ensuite accéder aux champs de notre élément inséré en utilisant les attributs de notre objet. Nous pouvons également modifier les champs de notre enregistrement en modifiant ses attributs.

Suivez-nous sur LinkedIn Twitter @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Lorsque nous avons utilisé la méthode `objects.all()` sur notre objet créé l'affichage n'était pas parlant du tout

```
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
```

Pour bénéficier d'un affichage plus représentatif il faudra définir la super méthode `__str__` de notre classe. Cela se passe du coup dans notre fichier `sondage/models.py`

```
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date de publication')

    def __str__(self):
        return self.question_text
```

Maintenant, lorsqu'on utilisera la méthode `objects.all()` sur notre classe `Question` il affichera l'attribut `question_text`

```
>>> Question.objects.all()
<QuerySet [<Question: Quel language de programmation préférez vous ?>]>
```

Définissez la super méthode `__str__` pour votre classe `Choice` et relancez votre shell afin de constater le changement.

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Il nous est aussi possible de définir des méthode dans nos classes afin de les utiliser dans notre shell, et plus tard dans notre projet web. Rajoutons donc une méthode à notre classe Question permettant de vérifier si une question a été publié récemment:

```
def was_published_recently(self):  
    return self.pub_date >= timezone.now() - datetime.timedelta(days=1)
```

Il faudra bien sûr importer datetime et timezone pour faire fonctionner ces méthode

```
from django.utils import timezone  
import datetime
```

Cette méthode return donc True si la valeur l'attribut pub\_date de notre objet est supérieur ou égal à la date du jours -1 jour (donc hier)



Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Continuons de tester nos modèles avec le shell du script *manage.py*. Il est possible d'appliquer un filtre de recherche pour interroger notre base. Ce filtre peut prendre comme paramètre la clé primaire (*pk=*), recherche de chaîne de caractères dans un champ (*champ\_\_startswith=*) ou encore une correspondance de dates.

```
>>> from sondage.models import Question, Choice
>>> q = Question.objects.filter(pk=1)
>>> q
<QuerySet [<Question: Quel langage de programmation préférez vous ?>]>
```

```
>>> q = Question.objects.filter(question_text__startswith="Quel")
>>> q
<QuerySet [<Question: Quel langage de programmation préférez vous ?>]>
```

Le même genre de recherche peut s'effectuer sur un get, le type d'objets retourné n'est cependant pas le même.

```
>>> from django.utils import timezone
>>> current_year = timezone.now().year
>>> Question.objects.get(pub_date__year=current_year)
<Question: Quel langage de programmation préférez vous ?>
```

Suivez-nous sur LinkedIn  Twitter  [@Adrar\\_Numerique](https://twitter.com/Adrar_Numerique)

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

La différence entre le *get* et le *filter* est que le *get* nous retourne un objet unique correspondant au critères de recherche alors que le *filter* nous retourne un objet de type *QuesrySet* comprenant des objets correspondant aux critères. Cette différence change ce que nous pourrons faire avec et la manière de les manipuler. Déclarons une nouvelle question pour pouvoir tester le fonctionnement :

```
>>> q2 = Question(question_text="Que voulez vous faire ?", pub_date=timezone.now())
>>> q2.save()
```

Tentons maintenant d'appliquer notre filtre de recherche basé sur la présence d'une chaîne de caractères au début de notre attribut *question\_text* avec un *filter* puis avec un *get* :

```
>>> q = Question.objects.filter(question_text__startswith="Que")
>>> q
<QuerySet [<Question: Quel langage de programmation préférez vous ?>, <Question: Que voulez vous faire ?>]>
>>> for i in q :
...     print(i)
...
Quel langage de programmation préférez vous ?
Que voulez vous faire ?
```

La requête se passe correctement, il nous retourne bien un objet *QuerySet* comprenant les questions correspondantes dans lequel on peut itérer

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Faisons la même chose avec la fonction *get* :

```
>>> q = Question.objects.get(question_text__startswith="Que")
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "C:\Users\maximeprzybylo\AppData\Local\Programs\Python\Python310\lib\site-packages\django\db\models\manager.py", line 85, in manager_method
    return getattr(self.get_queryset(), name)(*args, **kwargs)
  File "C:\Users\maximeprzybylo\AppData\Local\Programs\Python\Python310\lib\site-packages\django\db\models\query.py", line 439, in get
    raise self.model.MultipleObjectsReturned(
sondage.models.Question.MultipleObjectsReturned: get() returned more than one Question -- it returned 2!
```

On constate que la fonction *get* nous retourne une erreur car plusieurs objets correspondent à la recherche. Nous pouvons cependant utiliser notre méthode *was\_published\_recently()* lorsque *get* nous retourne bien un seul objet :

```
>>> q = Question.objects.get(question_text__startswith="Quel")
>>> q.was_published_recently()
True
```

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Nous pouvons aussi nous servir de la relation clef étrangère pour manipuler nos tables :

```
>>> q = Question.objects.get(pk=1)
>>> q.choice_set.create(choice_text='Python', votes=0)
<Choice: Python>
>>> q.choice_set.create(choice_text='Php', votes=0)
<Choice: Php>
```

```
>>> c = q.choice_set.create(choice_text='Javascript', votes=0)
>>>
>>> c
<Choice: Javascript>
```

Nous pouvons du coup également manipuler notre set de clef étrangères de la même manière que nos objets :

```
>>> q.choice_set.all()
<QuerySet [<Choice: Python>, <Choice: Php>, <Choice: Javascript>]>
>>> q.choice_set.get(pk=1)
<Choice: Python>
>>> q.choice_set.filter(choice_text__startswith="P")
<QuerySet [<Choice: Python>, <Choice: Php>]>
```

```
>>> c = q.choice_set.get(choice_text__startswith="Java")
>>> c.delete()
(1, {'sondage.Choice': 1})
>>> q.choice_set.all()
<QuerySet [<Choice: Python>, <Choice: Php>]>
```



Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Nous avons également la possibilité d'utiliser la page d'administration générée par django pour manipuler notre BDD. Cette page est présente par défaut (vous l'avez peut-être vue dans la liste INSTALLED\_APPS du fichier *settings.py*) mais pour pouvoir l'utiliser il faut tout d'abord créer un utilisateur administrateur. Cela se fait encore une fois grâce au script *manage.py* avec l'option *createsuperuser* :

```
PS C:\Users\maximeprzybylo\Documents\Cours\Django\TP\monsite> python .\manage.py createsuperuser
Username (leave blank to use 'maximeprzybylo'): superadmin
Email address:
Password:
Password (again):
Superuser created successfully.
```

Nous pouvons maintenant nous connecter à la page d'administration de notre projet (en oubliant pas de démarrer notre serveur au préalable)

```
PS C:\Users\maximeprzybylo\Documents\Cours\Django\TP\monsite> python .\manage.py runserver 80
```

Comme indiqué dans le fichier *monsite/urls.py* la page d'administration est accessible à l'url <http://localhost/admin>

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

En tapant l'url vous arrivez sur la page de connexion de l'outil d'administration de votre projet. Saisissez les informations indiquées lors de la commande *createuser*.

### Django administration

Username:

Password:

[Log in](#)

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Par défaut vous ne voyez pas vos modèles sur la page d'administration. Vous ne voyez que ce qui est préconfiguré par django, c'est-à-dire les utilisateurs et les groupes. Nous allons ajouter l'administration de nos tables à cette page. Cela se passe dans le fichier *admin.py* de votre application. Rajoutez y les lignes suivantes :

```
from .models import Question, Choice

admin.site.register(Question)
admin.site.register(Choice)
```

## Django administration

WELCOME, **SUPERADMIN**. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

### Site administration

**AUTHENTICATION AND AUTHORIZATION**

**Groups**

[+ Add](#) [Change](#)

**Users**

[+ Add](#) [Change](#)

#### Recent actions

**My actions**  
None available

Suivez-nous sur LinkedIn  Twitter  @Adrar\_Numerique

... et sur le web : [www.adrar-numerique.com](http://www.adrar-numerique.com)

Rechargez la page pour voir que les catégories *Choices* et *Questions* se sont rajoutées.

Vous pouvez maintenant cliquer sur ces catégories pour créer, voir, mettre à jours ou supprimer des éléments de vos tables.

## Django administration

WELCOME, **SUPERADMIN**. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

### Site administration

#### AUTHENTICATION AND AUTHORIZATION

Groups	<a href="#">+ Add</a>	<a href="#">Change</a>
Users	<a href="#">+ Add</a>	<a href="#">Change</a>

#### SONDAGE

Choices	<a href="#">+ Add</a>	<a href="#">Change</a>
Questions	<a href="#">+ Add</a>	<a href="#">Change</a>

#### Recent actions

---

#### My actions

None available