

django

# Le framework Django

## Les modèles

Formation Haïti  
08/05/27 - 11/05/12

# Utilisation d'une BDD

- Django et GeoDjango :
  - MySQL, PostgreSQL, SQLite, Oracle ...
- Utilisation d'une ORM : Object Relational Mapping
- Description des modèles en Python avec l'API de Django

```
class Layer(models.Model):  
    name = models.CharField(max_length=100)  
    author = models.CharField(max_length=100)  
    abstract = models.TextField(null=True)  
    date = models.DateTimeField(  
        auto_now_add=True, auto_now=False, verbose_name="Date de  
création")
```

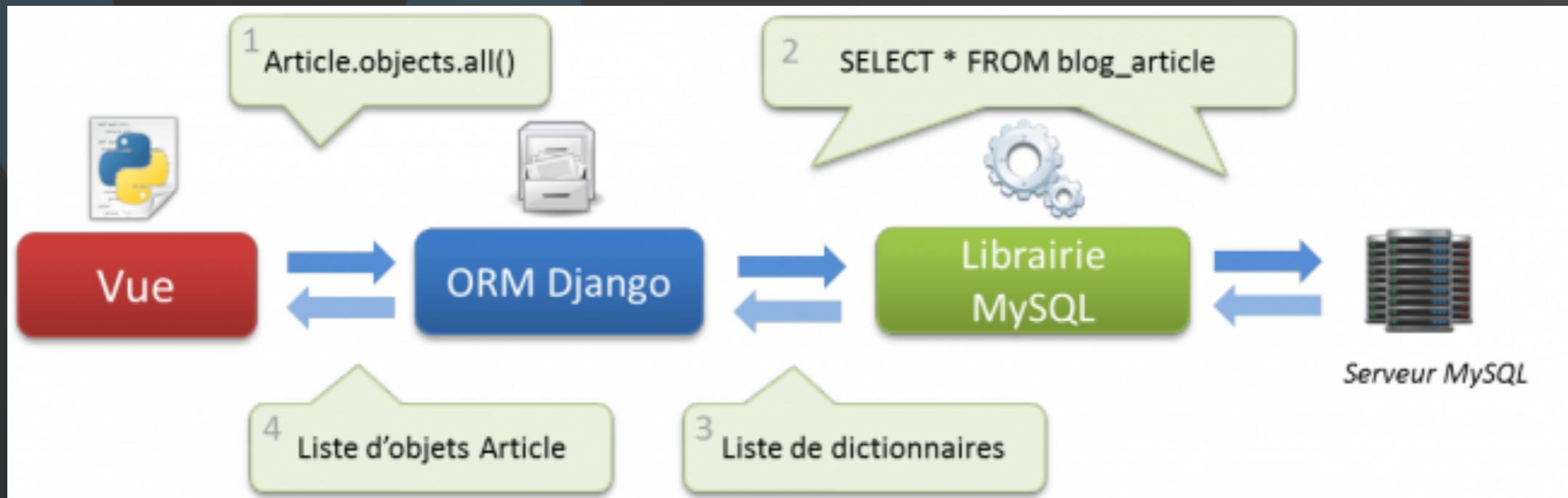
# Utilisation d'une BDD

- Ajout dans la base de données

```
Layer(name="Occupation du sol", auteur="Etienne").save()
```

- Lecture de la table

```
Layer.objects.all()
```



# Pourquoi une ORM ?

- Points positifs
  - Abstraction du langage SQL
  - Intégration dans Django (Python)
  - Fonctionne quelque soit le backend de BDD
- Points négatifs
  - Apprendre à manipuler l'ORM
  - Plus de SQL

# Les types de champ

- BooleanField
- CharField
- DateField
- EmailField
- FileField
- IntegerField
- TextField
- <https://docs.djangoproject.com/fr/1.11/ref/models/fields/>

# De l'ORM à la table

- 1. Écrire le fichier models.py

```
class Layer(models.Model):  
    name = models.CharField(max_length=100)  
    author = models.CharField(max_length=100)  
    abstract = models.TextField(null=True)  
    date = models.DateTimeField(  
        auto_now_add=True, auto_now=False, verbose_name="Date de  
création")
```

- 2. python manage.py makemigrations
- 3. python manage.py migrate

# Ajouter, supprimer dans la table

```
layer = Layer(name='Occupation du sol', author='Etienne')  
layer.save()  
layer.delete()
```

# Lire dans la table

# Tous les objets de la table

```
Layers.objects.all()
```

# Filtrer sur un champ

```
Layer.objects.filter(author="Tom")
```

```
Layer.objects.exclude(author="John")
```

# Filtrer sur plusieurs champs

```
Layer.objects.filter(name="Building", author="Tom")
```

# Filtrer plus finement

```
Layer.objects.filter(name__contains="sol")
```

```
Layer.objects.filter(date__lt=timezone.now())
```

# lt, gt, lte, gte

# Ordonner

```
Layer.objects.order_by('date')
```

```
Layer.objects.order_by('-date')
```

# Cumuler ces fonctions

```
Layer.objects.filter(date__lt=timezone.now()).order_by('date', 'name').reverse()
```

# Lire dans la table

Il existe plusieurs façons de lire des données dans une table.

```
# Ne demander que un seul objet
```

```
Layer.objects.get(name__contains="sol")
```

```
# Get or create
```

```
Layer.objects.get_or_create(name="MNT", author="Tom")
```

```
# Get an object or 404
```

```
from django.shortcuts import get_object_or_404
```

```
id=5
```

```
layer = get_object_or_404(Layer, id=id)
```

# Faire des liens entre table

- One to Many
- One to One
- Many to Many

# BDD Avancé

## Héritage entre table

```
class Layer(models.Model):
    name = models.CharField(max_length=255)
    date = models.DateTimeField(auto_now_add=True)
    author = models.CharField(max_length=255)

class Meta:
    abstract = True

class Raster(Layer):
    pixel_size = models.IntegerField()

class Vector(Layer):
    geometry_type = models.CharField()
```



L'administration de  
nos modèles

# Ajoutons nos modèles

```
from django.contrib import admin  
from .models import Layer, Category  
  
admin.site.register(Layer)  
admin.site.register(Category)
```

# Personnalisons notre administration

list_display	Liste des champs du modèle à afficher dans le tableau
list_filter	Liste des champs à partir desquels nous pourrons filter les entrées
date_hierarchy	Permet de filtrer par date de façon intuitive
ordering	Tri par défaut
search_fields	Configuration du champ de recherche

# Ajoutons nos modèles

```
class LayerAdmin(admin.ModelAdmin):
    list_display = ('name', 'author', 'category')
    list_filter = ('author', 'category',)
    date_hierarchy = 'date'
    ordering = ('date', )
    search_fields = ('name', 'abstract')
```