Session 12 - 14/10/2025

ORM (object - Relational Mapping)

ORM : هي تقنية برمجية تحول البيانات بين لغات OOP وقواعد البيانات العلائقية
فهي تشكل طبقة بين قاعدة البيانات والكود تسمح للمطورين بالتفاعل مع قواعد
البيانات باستخدام objects و classes بدلاً من raw sql queries بدقة عالية

كل class من نموذج Model يقابل جدول table

كل attribute يقابل عمود column

الطبقة التي تحول الجداول إلى objects

حلقة الوصل بين البايثون والـ sql.

Django Developers لا يتعامل مع قواعد البيانات مباشرة ... يتعامل مع الـ ORM

الصف class هو row ضمن قاعدة البيانات.

مثل student هو row ضمن جدول

والـ attributes هي column ضمن جدول

بالـ views.py

```
class PostViewset(ModelViewset):
    queryset = Post.objects.filter(published=True, category='news')
```

[1]

filter → صفي النتائج

filter queryset

هو الـ query التي ستنفذ بالـ database

الـ queryset السابقة تكافئ:

```
select * from Post where published = True and category = 'news'
```

serializer_class = PostSerializer

users = User.objects.get(id=5) 2 get:

تجيبلي سجل واحد فقط

Post.objects.order_by('title') 3 order by

ترتيب تصاعدي من A ← Z

Post.objects.order_by('-created_at')

ترتيب تنازلي من الأحدث إلى الأقدم.

User.objects.values('id', 'email') 4

لإرجاع بيانات محددة من الـ object بدلاً من إحضار الـ object كاملاً

"استعلام أخف على الـ database".

Post.objects.values('category').distinct() 5 distinct:

عرض النتائج بدون تكرار.

6

العلاقات في الـ ORM

يوجد نوعين من قواعد البيانات

Relational

Non Relational

أنواع الارتباط بين جدول وآخر

1 one to one 1:1

سطر واحد من جدول له علاقة بسطر واحد من جدول آخر

- لكل ستديم جواز سفر واحد

- كل جواز سفر تابع لستديم واحد فقط

/ /

2 One to Many 1:M

سطر واحد من جدول له علاقة بأكثر من سطر من جدول آخر.

- في كل محافظة يوجد العديد من المستخدمين

- كل مستخدم يتبع لمحافظة واحدة فقط

بال user table يوجد column لل city_id

3 Many to Many N:M

الكثير من القيم من جدول سترتبط بالكثير من القيم من جدول آخر

- تصنيف مقال معين ← ممكن مقال يتبع لأكثر من تصنيف

↙ ممكن تصنيف يكون تحته أكثر من مقالة

المفروض نعمل جدول ثالث فيه ids التصنيف و ids المقالة

*1 (1:M)

ضمن ال models.py (تحت Post class)

```
author = models.ForeignKey(User, on_delete = models.CASCADE)
```

تحديد ما الذي سيحدث للابن في حال حذف الأب و CASCADE تعني اذا حُذف الأب

فـ تحذف الأبناء التابعين له.

ضمن ال views.py

```
Post.objects.filter(author__username = 'somar')
```

الانتقال ما بين الجداول المرتبطة ببعضها

جلب كل ال Posts الي كاتبها ال user الي اسمه Somar.

N:M

models.py من ال # *2

```
tags = models.ManyToManyField(Tag)
```

views.py من ال #

```
Post.objects.filter(tags__name = 'Django')
```

1:1

models.py من ال # *3

```
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete = models.CASCADE)
```

Aggregate 7

views.py من ال #

```
Post.objects.aggregate(average_count = Avg('views'))
```

aggregate: تجميع (قيمة اجمالية عامة) حساب قيمة معينة على كامل الجدول.

Avg: دالة method جاهزة تحسب المتوسط الحسابي للعمود اللي اسمه views

output

```
>> {'average_count': 15.2}
```

Annotate 8

```
category.objects.annotate(post_count = count('post'))
```

بترجعلي ال Category وعدد المنشورات التابعة لكل ال Category

/ /

9 Q object

يساعدني على التعامل مع البوابات المنطقية

- البوابات المنطقية مثل (AND , OR , NOT ...)

```
Post.objects.filter(Q(published=True) | Q(views_gt=100))
```

OR ≡ |

يجلب كل الـ Posts المنشورة أو عدد مشاهدات أكثر من 100

OR ≡ |	شرط واحد تحقق على الأقل
AND ≡ &	جميع الشروط يجب أن تكون محققة

10 F

تساعدني في العمليات المباشرة بين الحقول داخل قاعدة البيانات.

```
Post.objects.update(views=F('views')+1)
```

كل الـ Posts سيتم زيادة قيمة الـ views الخاصة بها بمقدار واحد

دون أن تحمل هذه الـ data ← أداء عالي جداً.

```
User.objects.filter(profile__isnull=True)
```

يجلب جميع الـ users الذين ليس لديهم profile

Session 13 - 15/10/2025

في هذه الجلسة سنبني أول CRUD

- عملنا new project اسمه library-api

وتحته catalog app

وطبعاً أول شي:

```
pip install django
pip install mysqlclient
```

- ال database اللي مستخدمة في django ال default هي SQLite
  بس الآن عندنا نستطيع mysql

1. لذلك أول خطوة تنزيل mysqlclient ب pip كما في الخطوة السابقة
2. ثانياً: في settings.py سنقوم بتعديل بيانات الاتصال كما يلي:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'library_db',
        'USER': 'root',
        'PASSWORD': 1122334455
        'HOST': '127.0.0.1',
        'PORT': '3306',
        'OPTIONS': {'charset': 'utf8mb4'}
    }
```

ضمن ملف models.py

```python
class Author(models.Model):
    name = models.CharField(max_length=200, db_index=True)   # [1]
    country = models.CharField(max_length=100, blank=True)   # [2]
    birthday_year = models.IntegerField(blank=True, null=True)   # [3]

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

[1] db_index عمود مهم جداً بالنسبة الي لا بحث فغالباً رح استخدم الاسم
← دائماً بعطي أهمية كبيرة للاسم.
في ضمن (جدول) Author Model بعتبر الـ name هو الـ index.

[2] blank=True تتعلق بصفحة النموذج للسماح بقيمة فارغة
يفضل استخدامها مع الحقول النصية لأن قاعدة البيانات في django
تخزن سلاسل نصية فارغة بدلاً من NULL "يمكن ادخال قيمة default"

[3] null=True تتعلق بحفظ قاعدة البيانات.
يستخدم للحقل الذي يتم تخزين NULL في قاعدة البيانات مثل حقل
الأعداد أو العلاقات Foreign Keys

عند ذكر الشرطين blank=True و null=True للتعبير أن الحقل اختياري
ويتم أيضاً تخزين NULL في قاعدة البيانات.

```
class Book (models.Model):
    author = models.ForeignKey(Author, on_delete = models.CASCADE)
    title = models.CharField(max_length = 250, db_index = True)
    published_year = models.IntegerField(blank = True, null = True)
    price = models.DecimalField(max_digits = 8, decimal_places = 2,
                                default = 0)
    in_stock = models.PositiveIntegerField(default = 0)

    class Meta:
        indexes = [models.Index(fields = ['title'])]

    def __str__(self):
        return self.title

class Review (models.Model):
    book = models.ForeignKey(Book, on_delete = models.CASCADE,
                             related_name = 'reviews')
    reviewer_name = models.CharField(max_length = 120)
    rating = models.PositiveSmallIntegerField(
        validators = [MinValueValidator(1), MaxValueValidator(5)]
    )
    Comment = models.TextField(blank = True)
    Created_at = models.DateTimeField(auto_now_add = True)

    class Meta:
        ordering = ['-created_at']
```

```
def __str__(self):
    return f'{self.review_name}-{self.rating}'
```

★ أهمية وضع الـ Validator هنا ضمن الـ model:
طبقة أمان وحماية إضافية لمنع القدرة على تعديل القيم من مكان آخر..
بالطريقة الحديثة نفسه عليه طبقة أمان و ليس الـ [illegible] الـ Form فقط.

ضمن الـ serializers.py نعمل Serializer لكل class لأنه كل class
له بيانات خاصة فيه.

```
from rest_framework import serializers
from .models import Author, Book, review

class AuthorSerializer(serializers.ModelSerializer):
    books_count = serializers.IntegerField(read_only=True)

    class Meta:
        __module__ = Author
        fields = ['id', 'name', 'country', 'birth_year', 'books_count']

class BookSerializer(serializers.ModelSerializer):
    auth_name = serializers.ReadOnlyField(source='author.name')  →
    reviews = ReviewSerializer(many=True, read_only=True)  →

    class Meta:
        __module__ = Book
        fields = __all__
```

لعرض اسم الكاتب بجانب الكتاب للقراءة فقط ← (auth_name)

إظهار الـ Reviews كلها تحت كل كتاب ← (reviews)

```
class ReviewSerializer(serializers.ModelSerializer):
    class Meta:
        __module__ = Review
        fields = __all__
        read_only_fields = ['id', 'created_at']
```

ضمن ال views.py شغل ال CRUD لل models الحقيقي.

```
from djángo.db.models.aggregates import count
from .models import Author, Book, Review
from .serializers import AuthorSerializer, BookSerializer, ReviewSerializer
from rest_framework import viewsets, status

class AuthorViewSet(viewsets.ModelViewSet):
    serializer_class = AuthorSerializer
    search_fields = ['name', 'country']
    ordering_fields = ['name', 'birth_year']

    def get_queryset(self):
        qs = Author.objects.all().annotate(books_count = Count('books'))   # عدد الكتب لكل كاتب
        Country = self.request.query_params.get('country')
        has_books = self.request.query_params.get('has_books')
        if country:
            qs = qs.filter(country__iexact = country)   # iexact: غير حساسة لحالة الأحرف / exact: مطابقة كاملة
        if has_books:
            qs = qs.filter(books_count__gt = 0)
```