**Videofile: 84**

**Code file: ch75**

**Django Async ORM**

First of all do all the setting for Async using unicron also their css settings

Now , Make models:

Models.py:

from django.db import models

class Student(models.Model):

    name = models.CharField(max\_length=100)

    age = models.IntegerField()

    email = models.EmailField()

    def \_\_str\_\_(self):

        return self.name

admin.py:

from django.contrib import admin

from myapp.models import Student

@admin.register(Student)

class StudentAdmin(admin.ModelAdmin):

  list\_display=['id', 'name', 'age', 'email']

before moving forward lets see these tings

important notes:

# Django Asynchronous ORM Methods – Notes

### ⚡ QuerySet Methods

* **Return new QuerySets** → Non-blocking, don’t need async versions.  
  Examples:
  + all()
  + filter()
  + exclude()
  + annotate()
  + order\_by()
  + distinct()

### ⚡ Blocking Methods (Async Versions Available)

These methods perform DB operations and therefore have asynchronous equivalents (usually prefixed with a).

## 🔹 Retrieval Methods

* **aget()** → Asynchronously retrieves a single object.
* **acount()** → Asynchronously counts objects in a queryset.
* **afirst()** → Asynchronously gets the first object.
* **alast()** → Asynchronously gets the last object.
* **aexists()** → Asynchronously checks if any objects exist.
* **alatest()** → Asynchronously gets the latest object by a field (like latest()).
* **aearliest()** → Asynchronously gets the earliest object by a field (like earliest()).
* **ain\_bulk()** → Asynchronously fetches multiple objects by primary keys.
* **aiterator()** → Provides an async iterator over queryset (efficient for large datasets).

## 🔹 Creation & Saving

* **acreate()** → Asynchronously creates a new object.
* **aget\_or\_create()** → Asynchronously gets or creates an object.
* **abulk\_create()** → Asynchronously creates multiple objects in one query.
* **asave()** → Asynchronously saves an object.

## 🔹 Updating

* **aupdate()** → Asynchronously updates objects in the DB.
* **aupdate\_or\_create()** → Asynchronously updates or creates an object.
* **abulk\_update()** → Asynchronously updates multiple objects in one query.

## 🔹 Deletion

* **adelete()** → Asynchronously deletes objects.

## 🔹 Aggregation & Query Analysis

* **aaggregate()** → Asynchronously performs aggregation (SUM, AVG, etc.).
* **aexplain()** → Asynchronously shows SQL query execution plan (useful for performance).

## 🔹 Related Object Optimization

* **aprefetch\_related\_objects()** → Asynchronously prefetches related objects to minimize queries.

✅ **Summary Rule**:

* If a method only **builds a queryset** → No async version.
* If a method **hits the database** → Async version exists (a...).

Now

Views.py:

this is working in synronous but we can make it async

from django.shortcuts import render

from myapp.models import Student

# Create your views here.

def home(request):

    students = Student.objects.all()

    for student in students:

        print(f"name: {student.name}  age: {student.age}  email: {student.email}")

    return render(request, 'myapp/home.html',{'students':students})

now here are proper example of these :

from django.shortcuts import render

from myapp.models import Student

# ================================

#   Sync View Example

# ================================

def home(request):

    # Synchronous Query

    students = Student.objects.all()

    for student in students:

        print(f"name: {student.name}  age: {student.age}  email: {student.email}")

    return render(request, 'myapp/home.html', {'students': students})

# ================================

#   Async View Example - Method 1

# ================================

async def home(request):

    # Async iteration directly (❌ not supported on plain QuerySet)

    async for student in Student.objects.all():

        print(f"name: {student.name}  age: {student.age}  email: {student.email}")

    return render(request, 'myapp/home.html')

# ================================

#   Async View Example - Method 2

# ================================

async def home(request):

    # Using aiterator() ✅ (correct way)

    students = Student.objects.all()

    async for student in students.aiterator():

        print(f"name: {student.name}  age: {student.age}  email: {student.email}")

    return render(request, 'myapp/home.html', {'students': students})

# ================================

#   Async View Example - Other Async ORM Methods

# ================================

async def home(request):

    # Example: Async iteration

    async for student in Student.objects.all().aiterator():

        print(f"name: {student.name}  age: {student.age}  email: {student.email}")

    # Example: Async create

    await Student.objects.acreate(

        name='kunal',

        age=30,

        email='kunal@example.com'

    )

    # Example: Async count

    total\_students = await Student.objects.acount()

    print(total\_students)

    # Example: Async get by primary key

    student = await Student.objects.aget(pk=1)

    print(f"name: {student.name}  age: {student.age}  email: {student.email}")

    # Example: Async delete

    student = await Student.objects.aget(pk=4)

    await student.adelete()

    return render(request, 'myapp/home.html')