Video lecture 66

Code file 56

**Working with Database Returns New Queryset**

## **Working with Database Returns New QuerySet in Django**

When data is going into the dataset so how we set that like how we read, filter, order, and other task how we do that

In Django, whenever you interact with the database through the **ORM (Object Relational Mapper)**, the result is a **QuerySet**.

A **QuerySet** is a collection of Django model objects retrieved from the database. It allows you to **filter, order, slice, and chain queries** in a clean Pythonic way.

### ✅ Key Points:

* **Lazy Evaluation**: QuerySets are not executed immediately. The database query runs only when you access the data (e.g., iteration, converting to list, printing).
* **Immutable**: Every time you apply a method like .filter(), .exclude(), or .order\_by(), Django returns a **new QuerySet** without modifying the old one.
* **Chainable**: Because each call returns a new QuerySet, you can chain multiple methods.

### 🔹 Example

# Assume we have a model Student

from myapp.models import Student

# Get all students (returns a QuerySet)

students = Student.objects.all()

# Filter students (returns a new QuerySet)

passed\_students = students.filter(grade='A')

# Further filtering (still returns a new QuerySet)

top\_students = passed\_students.order\_by('-marks')

# Chaining directly

top\_students = Student.objects.filter(grade='A').order\_by('-marks')

### 🔹 Why Important?

* Efficient: Only fetches data when needed.
* Flexible: You can build complex queries step by step.
* Safe: Old QuerySets remain unchanged, avoiding accidental data modifications.

👉 In short: **Every database operation in Django ORM returns a new QuerySet, allowing lazy, safe, and chainable queries**

**OWN NOTES:**

**Mostly work inside the view**

**First make the model and do write code in admin to register and make migraitons and do code in view**

**And in views.py**

**View.py:**

from django.shortcuts import render

from school.models import Student, Teacher

def home(request):

    # ---------------------------

    # Basic QuerySet operations

    # ---------------------------

    # Get all Student records

    all\_data = Student.objects.all()

    # Filter: return only students from a specific city

    all\_data = Student.objects.filter(city="hasilpur")

    # Exclude: return all students except those from a specific city

    all\_data = Student.objects.exclude(city="hasilpur")

    # ---------------------------

    # Ordering QuerySets

    # ---------------------------

    # Order by marks (ascending)

    all\_data = Student.objects.order\_by('marks')

    # Order by marks (descending)

    all\_data = Student.objects.order\_by('-marks')

    # Order by name (alphabetical)

    all\_data = Student.objects.order\_by('name')

    # Random order

    all\_data = Student.objects.order\_by('?')

    # Reverse order (alphabetical reversed)

    all\_data = Student.objects.order\_by('name').reverse()

    # Slicing: get first 5 records

    all\_data = Student.objects.order\_by('name')[0:5]

    # ---------------------------

    # Working with values() and values\_list()

    # ---------------------------

    # Get all fields as dictionaries

    all\_data = Student.objects.values()

    # Get specific fields as dictionaries

    all\_data = Student.objects.values('name', 'city')

    # Get all fields as tuples

    all\_data = Student.objects.values\_list()

    # Get specific fields as tuples

    all\_data = Student.objects.values\_list('id', 'name')

    # Named tuples (easier to read)

    all\_data = Student.objects.values\_list('id', 'name', named=True)

    # ---------------------------

    # Combining QuerySets (UNION, INTERSECTION, DIFFERENCE)

    # ---------------------------

    qs1 = Student.objects.values\_list('id', 'name', named=True)

    qs2 = Teacher.objects.values\_list('id', 'name', named=True)

    # UNION → combines both QuerySets (removes duplicates by default)

    all\_data = qs2.union(qs1)

    # UNION (with duplicates allowed)

    all\_data = qs2.union(qs1, all=True)

    # INTERSECTION → only common records

    all\_data = qs1.intersection(qs2)

    # DIFFERENCE → records in qs2 but not in qs1

    all\_data = qs2.difference(qs1)

    # ---------------------------

    # Using AND (&) and OR (|) in filters

    # ---------------------------

    # AND condition (two ways of writing)

    all\_data = Student.objects.filter(name="rohit") & Student.objects.filter(city="ranchi")

    all\_data = Student.objects.filter(name="rohit", city="ranchi")

    # OR condition

    all\_data = Student.objects.filter(name="rohit") | Student.objects.filter(city="ranchi")

    # Print data in console (for debugging)

    print("All Data:", all\_data)

    # print("SQL Query:", all\_data.query)  # shows raw SQL query

    return render(request, 'school/home.html', {'all\_data': all\_data})

**models.py:**

from django.db import models

# Create your models here.

class Student(models.Model):

 name = models.CharField(max\_length=70)

 roll = models.IntegerField(unique=True, null=False)

 city = models.CharField(max\_length=70)

 marks = models.IntegerField()

 pass\_date = models.DateField()

class Teacher(models.Model):

 name = models.CharField(max\_length=70)

 empnum = models.IntegerField(unique=True, null=False)

 city = models.CharField(max\_length=70)

 salary = models.IntegerField()

 join\_date = models.DateField()

**admin.py:**

from django.contrib import admin

from .models import Student, Teacher

# Register your models here.

@admin.register(Student)

class StudentAdmin(admin.ModelAdmin):

 list\_display = ['id', 'name', 'roll', 'city', 'marks', 'pass\_date']

@admin.register(Teacher)

class TeacherAdmin(admin.ModelAdmin):

 list\_display = ['id', 'name', 'empnum', 'city', 'salary', 'join\_date']