খুব ভালো প্রশ্ন 👍
DRF **class-based ViewSet** ব্যবহার করলে `queryset` **attribute** আর `get_queryset()` **method**—এই দুইটার ব্যবহার বুঝলে অনেক clean ও optimized API লেখা যায়।

নিচে পরিষ্কারভাবে ব্যাখ্যা করছি 👇

🔹 `queryset` (attribute) — কখন ব্যবহার করবে

ব্যবহার করবে যখন:

- queryset **static** (সব request-এ একই)
- request / user / params এর উপর depend করে না
- শুধু basic CRUD দরকার

Example:

```
class ProductViewSet(ModelViewSet):
    queryset = Product.objects.filter(is_active=True)
    serializer_class = ProductSerializer
```

✅ ভালো যখন:

- list / retrieve same data
- simple API
- দ্রুত লেখা

❌ ব্যবহার কোরো না যখন:

- user-specific data
- query param অনুযায়ী filter
- permission অনুযায়ী data বদলায়

🔹 `get_queryset()` (method) — কখন ব্যবহার করবে

ব্যবহার করবে যখন:

- queryset **dynamic**
- `request.user` দরকার
- URL param / query param লাগবে

- role / permission অনুযায়ী data দেখাতে হবে

Example:

```
class OrderViewSet(ModelViewSet):
    serializer_class = OrderSerializer

    def get_queryset(self):
        user = self.request.user
        if user.is_staff:
            return Order.objects.all()
        return Order.objects.filter(user=user)
```

✅ Best practice:

- production API তে বেশি ব্যবহার হয়
- security & flexibility বেশি

🔁 দুটো একসাথে ব্যবহার করা যাবে?

হ্যাঁ, এবং **এটাই সবচেয়ে clean approach** 👇

```
class OrderViewSet(ModelViewSet):
    serializer_class = OrderSerializer
    queryset = Order.objects.select_related('user')

    def get_queryset(self):
        qs = super().get_queryset()
        if not self.request.user.is_staff:
            qs = qs.filter(user=self.request.user)
        return qs
```

👉 এতে:

- base optimization attribute এ
- dynamic filter method এ

🔥 Real-world guideline

Situation	Use
Simple CRUD	`queryset`

Situation	Use
Auth-based filtering	`get_queryset()`
Query param filter	`get_queryset()`
Performance optimization	`queryset`
Role-based access	`get_queryset()`

⚠️ Common mistake

```
queryset = Model.objects.filter(user=self.request.user)
```

❌ এটা কাজ করবে না — class load হওয়ার সময় `request` থাকে না

🎯 Bottom line

- **Static → queryset**
- **Dynamic → get_queryset()**
- **Optimization আগে, condition পরে**

চাও তো তুমি তোমার ViewSet code দিলে আমি বলে দেব
কোথায় attribute ভালো, কোথায় method লাগবে 👌

সংক্ষেপে উত্তর 👉 **হ্যাঁ,**
condition অনুযায়ী query বদলাতে হলে `get_queryset()` method ব্যবহার করাই সঠিক ✅

এখন পরিষ্কার করে বুঝাই 👇

❓ কেন `queryset` attribute দিয়ে condition দেওয়া যায় না?

কারণ `queryset` **class load হওয়ার সময় evaluate হয়**, তখন:

- `request`
- `request.user`
- query params
  👉 **কিছুই থাকে না**

তাই condition কাজ করবে না ❌

✅ সঠিক উপায়: `get_queryset()`

Example 1: user অনুযায়ী data

```
class OrderViewSet(ModelViewSet):
    serializer_class = OrderSerializer

    def get_queryset(self):
        user = self.request.user
        if user.is_staff:
            return Order.objects.all()
        return Order.objects.filter(user=user)
```

Example 2: query params অনুযায়ী condition

```
def get_queryset(self):
    qs = Order.objects.all()
    status = self.request.query_params.get('status')

    if status:
        qs = qs.filter(status=status)

    return qs
```

Example 3: queryset + get_queryset (BEST PRACTICE)

```
class OrderViewSet(ModelViewSet):
    serializer_class = OrderSerializer
    queryset = Order.objects.select_related('user')

    def get_queryset(self):
        qs = super().get_queryset()

        if not self.request.user.is_staff:
            qs = qs.filter(user=self.request.user)

        return qs
```

👉 এতে:

- base optimization (`select_related`) attribute এ

- condition logic method এ

❌ ভুল উদাহরণ

```
queryset = Order.objects.filter(user=self.request.user)
```

❌ কাজ করবে না

📌 Summary

✔ condition থাকলে → `get_queryset()`
✔ static হলে → `queryset`
✔ দুটো একসাথে ব্যবহার করা যায় (recommended)

চাও তো তুমি তোমার actual condition / code পাঠাও,
আমি সেটাকে clean & optimized করে সাজিয়ে দেব 👍

এটা DRF/Django ORM-এর **সবচেয়ে গুরুত্বপূর্ণ performance প্রশ্নগুলোর একটা** 👍
আমি **simple rule → example → mistake → real-life guideline** এইভাবে বুঝাচ্ছি।

🔑 মূল পার্থক্য (১ লাইনে)

- `select_related` → **ForeignKey / OneToOne** → SQL **JOIN**
- `prefetch_related` → **ManyToMany / reverse FK** → **আলাদা query + Python join**

1️⃣ `select_related` কখন ব্যবহার করবে

✔ ব্যবহার করবে যখন:

- relationship **ForeignKey** বা **OneToOne**
- child object থেকে parent এর field দরকার
- একসাথে JOIN করলেও data বেশি বড় না

Example (Bus Ticket):

```
Booking.objects.select_related(
    'user',      # FK
    'trip',      # FK
    'seat'       # FK
)
```

👉 এখানে:

- Booking → User
- Booking → Trip
- Booking → Seat
  সবই **ForeignKey**

🟢 **১টা query তেই সব data**

❌ ভুল ব্যবহার

```
Booking.objects.select_related('payments')
```

❌ `payments` যদি ManyToMany / reverse FK হয় → কাজ করবে না

2️⃣ `prefetch_related` কখন ব্যবহার করবে

✔ ব্যবহার করবে যখন:

- **ManyToMany**
- **reverse ForeignKey** (parent → children)
- child object এর list দরকার

Example (Trip → Seats):

```
Trip.objects.prefetch_related('seat_set')
```

👉 Queries:

- 1 query → Trip
- 1 query → Seat
  Python এ matching হয়

Another example:

```
Bus.objects.prefetch_related('trips')
```

✔ One Bus → Many Trips

3️⃣ একসাথে ব্যবহার (Most Common & Best)

```
Trip.objects.select_related(
    'bus', 'route'
).prefetch_related(
    'seat_set'
)
```

✔ Parent data JOIN
✔ Child list prefetch

4️⃣ কখন কোনটা? (Cheat Table)

Situation	Use
FK / OneToOne	`select_related`
ManyToMany	`prefetch_related`
Reverse FK	`prefetch_related`
Single object detail API	`select_related`
List + child list	`prefetch_related`
Large child list	`prefetch_related`

5️⃣ DRF ViewSet-এ Best Practice

```
class BookingViewSet(ModelViewSet):
    serializer_class = BookingSerializer
    queryset = Booking.objects.select_related(
        'trip', 'seat'
    ).prefetch_related(
        'payments'
    )
```

👉 Base optimization attribute এ
👉 Condition → `get_queryset()`

6️⃣ Common Performance Mistake 🚨

```
for booking in bookings:
    print(booking.trip.bus.name)
```

❌ যদি `select_related('trip__bus')` না থাকে
👉 N+1 query

7️⃣ Debug করার উপায়

```
django-debug-toolbar
```

বা

```
print(len(connection.queries))
```

🎯 Golden Rules (মুখস্থ করে ফেলো)

✔ FK → `select_related`
✔ List of children → `prefetch_related`
✔ Doubt হলে → relation type দেখো
✔ Loop এর ভিতরে query দেখলে 🚨

চাও তো তুমি তোমার **model relationship পাঠাও**,
আমি exact বলে দেব—কোথায় কোনটা বসবে 🔥

```
from django.db import models



# ==========================
```

```
# Patient

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

class Patient(models.Model):

CHOOSE_GENDER = [

('male', 'Male'),

('female', 'Female')

]


CHOOSES_BLOOD_GROUP = [

('A+', 'A+'), ('B+', 'B+'), ('O+', 'O+'), ('AB+', "AB+"),

('A-', 'A-'), ('B-', 'B-'), ('O-', 'O-'), ('AB-', "AB-")

]


first_name = models.CharField(max_length=50)

last_name = models.CharField(max_length=50)

email = models.EmailField(unique=True)

phone = models.CharField(max_length=20)

gender = models.CharField(max_length=6, choices=CHOOSE_GENDER)

dob = models.DateField()

blood_group = models.CharField(max_length=5, choices=CHOOSES_BLOOD_GROUP)

address = models.TextField()

emergency_contact = models.CharField(max_length=20)

created_at = models.DateTimeField(auto_now_add=True)


class Meta:
```

```
ordering = ['first_name', 'last_name']


def __str__(self):

return f"{self.first_name} {self.last_name}"



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

# Department

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

class Department(models.Model):

name = models.CharField(max_length=200)

description = models.TextField()


class Meta:

ordering = ['name']


def __str__(self):

return self.name



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

# Doctor

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

class Doctor(models.Model):

name = models.CharField(max_length=100)

department = models.ForeignKey(Department, on_delete=models.CASCADE,
```

```
related_name='doctors')

specialization = models.CharField(max_length=200)

phone = models.CharField(max_length=20)

email = models.EmailField(unique=True)

qualification = models.CharField(max_length=100)

is_active = models.BooleanField(default=True)

active_time = models.TimeField()



def __str__(self):

return self.name




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

# Appointment

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

class Appointment(models.Model):

STATUS_CHOICES = [

('pending', 'PENDING'),

('confirmed', 'CONFIRMED'),

('cancelled', 'CANCELLED'),

('completed', 'COMPLETED')

]



patient = models.ForeignKey(Patient, on_delete=models.CASCADE,
related_name='appointments')

doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE,
related_name='appointments')
```

```
date = models.DateField()

time = models.TimeField()

status = models.CharField(max_length=10, choices=STATUS_CHOICES,
default='pending')

notes = models.TextField(blank=True, null=True)



class Meta:

ordering = ['-date', '-time']



def __str__(self):

return f"Appointment: {self.patient.first_name} with {self.doctor.name}"




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

# Schedule

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

class Schedule(models.Model):

WEEKDAYS = [

('mon', 'MONDAY'),

('tue', 'TUESDAY'),

('wed', 'WEDNESDAY'),

('thu', 'THURSDAY'),

('fri', 'FRIDAY'),

('sat', 'SATURDAY'),

('sun', 'SUNDAY'),

]
```

```
doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE,
related_name='schedules')

weekday = models.CharField(max_length=3, choices=WEEKDAYS)

start_time = models.TimeField()

end_time = models.TimeField()



def __str__(self):

return f"{self.doctor.name} - {self.weekday}"




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

# Ward

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

class Ward(models.Model):

WARD_CHOOSE = [

('general', 'GENERAL'),

('icu', 'ICU'),

('cabin', 'CABIN')

]



name = models.CharField(max_length=100)

type = models.CharField(max_length=10, choices=WARD_CHOOSE)



def __str__(self):

return self.name
```

```
# ============================

# Room

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

class Room(models.Model):

ward = models.ForeignKey(Ward, on_delete=models.CASCADE,
related_name='rooms')

room_no = models.CharField(max_length=10)

bed_count = models.PositiveIntegerField()

is_available = models.BooleanField(default=True)



def __str__(self):

return f"Room {self.room_no} ({self.ward.name})"



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

# Admission

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

class Admission(models.Model):

STATUS_CHOOSE = [

('admitted', 'ADMITTED'),

('discharged', 'DISCHARGED'),

]



patient = models.ForeignKey(Patient, on_delete=models.CASCADE,
related_name='admissions')
```

```
room = models.ForeignKey(Room, on_delete=models.CASCADE,
related_name='admissions')

admitted_at = models.DateTimeField(auto_now_add=True)

discharged_at = models.DateTimeField(null=True, blank=True)

status = models.CharField(max_length=10, choices=STATUS_CHOOSE,
default='admitted')



def __str__(self):

return f"Admission: {self.patient.first_name}"




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

# Treatment

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

class Treatment(models.Model):

admission = models.ForeignKey(Admission, on_delete=models.CASCADE,
related_name='treatments')

doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE,
related_name='treatments')

description = models.TextField()

treatment_date = models.DateField()



def __str__(self):

return f"Treatment for {self.admission.patient.first_name}"




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

# Medication
```

```
# ============================

class Medication(models.Model):

treatment = models.ForeignKey(Treatment, on_delete=models.CASCADE,
related_name='medications')

medicine_name = models.CharField(max_length=100)

dosage = models.CharField(max_length=100)

frequency = models.CharField(max_length=100)



def __str__(self):

return self.medicine_name




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

# Nurse

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

class Nurse(models.Model):

name = models.CharField(max_length=100)

phone = models.CharField(max_length=20)

department = models.ForeignKey(Department, on_delete=models.CASCADE,
related_name='nurses')

assign_room = models.ForeignKey(Room, on_delete=models.SET_NULL,
null=True, blank=True, related_name='nurses')



def __str__(self):

return self.name
```

```
# ============================

# Lab Test

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

class LabTest(models.Model):

test_name = models.CharField(max_length=200)

description = models.TextField()

price = models.PositiveIntegerField()



def __str__(self):

return self.test_name




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

# Lab Report

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

class LabReport(models.Model):

patient = models.ForeignKey(Patient, on_delete=models.CASCADE,
related_name='lab_reports')

doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE,
related_name='lab_reports')

test = models.ForeignKey(LabTest, on_delete=models.CASCADE,
related_name='lab_reports')

report_file = models.FileField(upload_to='reports/')

created_at = models.DateTimeField(auto_now_add=True)



def __str__(self):

return f"Lab Report for {self.patient.first_name}"
```

```
# ============================

# Prescription

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

class Prescription(models.Model):

appointment = models.ForeignKey(Appointment, on_delete=models.CASCADE,
related_name='prescriptions')

doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE,
related_name='prescriptions')

patient = models.ForeignKey(Patient, on_delete=models.CASCADE,
related_name='prescriptions')

notes = models.TextField()

created_at = models.DateTimeField(auto_now_add=True)



def __str__(self):

return f"Prescription for {self.patient.first_name}"



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

# Invoice

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

class Invoice(models.Model):

STATUS_CHOOSE = [

('paid', 'PAID'),

('unpaid', 'UNPAID'),

]
```

```
patient = models.ForeignKey(Patient, on_delete=models.CASCADE,
related_name='invoices')

admission = models.ForeignKey(Admission, on_delete=models.SET_NULL,
null=True, blank=True, related_name='invoices')

total_amount = models.FloatField()

status = models.CharField(max_length=10, choices=STATUS_CHOOSE,
default='unpaid')

created_at = models.DateTimeField(auto_now_add=True)



def __str__(self):

return f"Invoice - {self.patient.first_name}"




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

# Payment

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

class Payment(models.Model):

METHOD_CHOOSE = [

('cash', 'CASH'),

('card', 'CARD'),

('online', 'ONLINE'),

]



invoice = models.ForeignKey(Invoice, on_delete=models.CASCADE,
related_name='payments')

amount = models.FloatField()

method = models.CharField(max_length=10, choices=METHOD_CHOOSE)
```

```
    paid_at = models.DateTimeField(auto_now_add=True)



    def __str__(self):

        return f"Payment for {self.invoice.id}"
```

চমৎকার 👍
তুমি যেহেতু **সব models** দিয়ে দিয়েছ, এখন আমি একদম **এই Hospital Management System অনুযায়ী** বলছি—

👉 **কোন model relation এ `select_related`**
👉 **কোন relation এ `prefetch_related`**
👉 **Real DRF queryset example**

🔑 Golden Rule (আবার মনে করাই)

- **ForeignKey / OneToOne →** `select_related`
- **reverse ForeignKey / Many list →** `prefetch_related`

🔗 তোমার Model Relationship Analysis

1️⃣ Appointment

```
Appointment
 ├── patient (FK)
 ├── doctor (FK)
```

✅ Use `select_related`

```
Appointment.objects.select_related(
    'patient',
    'doctor',
    'doctor__department'
)
```

📌 কেন?

- Appointment → Patient (FK)
- Appointment → Doctor (FK)

- Doctor → Department (FK)

🟢 ১টা JOIN query তেই সব data

2️⃣ Doctor

```
Doctor
 ├── department (FK)
 ├── schedules (reverse FK)
 ├── appointments (reverse FK)
 ├── treatments (reverse FK)
```

✅ Correct way

```
Doctor.objects.select_related(
    'department'
).prefetch_related(
    'schedules',
    'appointments'
)
```

📌 কারণ:

- department → FK → `select_related`
- schedules / appointments → list → `prefetch_related`

3️⃣ Patient

```
Patient
 ├── appointments
 ├── admissions
 ├── lab_reports
 ├── prescriptions
 ├── invoices
```

✅ Use `prefetch_related`

```
Patient.objects.prefetch_related(
    'appointments',
    'admissions',
    'lab_reports',
```

```
    'prescriptions',
    'invoices'
)
```

📌 Patient থেকে সবই **reverse FK (one-to-many)**

4️⃣ Admission

```
Admission
 ├── patient (FK)
 ├── room (FK)
 ├── treatments (reverse FK)
 ├── invoices (reverse FK)
```

✅ Best

```
Admission.objects.select_related(
    'patient',
    'room',
    'room__ward'
).prefetch_related(
    'treatments'
)
```

5️⃣ Treatment

```
Treatment
 ├── admission (FK)
 ├── doctor (FK)
 ├── medications (reverse FK)
```

✅ Correct

```
Treatment.objects.select_related(
    'admission',
    'doctor'
).prefetch_related(
    'medications'
)
```

6️⃣ Room & Ward

```
Room
 ├── ward (FK)
 ├── admissions
 ├── nurses
```

```
Room.objects.select_related(
    'ward'
).prefetch_related(
    'admissions',
    'nurses'
)
```

7️⃣ LabReport

```
LabReport
 ├── patient (FK)
 ├── doctor (FK)
 ├── test (FK)
```

✅ Only `select_related`

```
LabReport.objects.select_related(
    'patient',
    'doctor',
    'test'
)
```

8️⃣ Invoice

```
Invoice
 ├── patient (FK)
 ├── admission (FK)
 ├── payments (reverse FK)
```

✅ Correct

```
Invoice.objects.select_related(
    'patient',
```

```
    'admission'
).prefetch_related(
    'payments'
)
```

9️⃣ Prescription

```
Prescription
 ├── appointment (FK)
 ├── doctor (FK)
 ├── patient (FK)
```

```
Prescription.objects.select_related(
    'appointment',
    'doctor',
    'patient'
)
```

🚀 DRF ViewSet Real Example

AppointmentViewSet

```
class AppointmentViewSet(ModelViewSet):
    serializer_class = AppointmentSerializer
    queryset = Appointment.objects.select_related(
        'patient',
        'doctor',
        'doctor__department'
    )
```

Patient Detail API (Heavy but optimized)

```
class PatientViewSet(ModelViewSet):
    serializer_class = PatientDetailSerializer

    def get_queryset(self):
        return Patient.objects.prefetch_related(
            'appointments__doctor',
            'appointments__doctor__department',
```

```
            'admissions__room__ward',
            'lab_reports__test',
            'prescriptions'
        )
```

⚠️ Common Mistake (তোমার project এ হতে পারে)

```
for appointment in appointments:
    print(appointment.doctor.department.name)
```

❌ যদি `select_related('doctor__department')` না থাকে
👉 N+1 query problem

🎯 Final Cheat Sheet (এই project অনুযায়ী)

Model	Use
Appointment	`select_related(patient, doctor)`
Doctor	`select_related(department)` + `prefetch(schedules)`
Patient	`prefetch(all reverse)`
Admission	`select_related(patient, room)`
Treatment	`select_related(admission, doctor)`
Room	`select_related(ward)`
Invoice	`select_related(patient)` + `prefetch(payments)`
LabReport	`select_related(all)`

যদি চাও 👉
আমি **একটা complete Patient Dashboard API** বানিয়ে দিতে পারি
যেখানে ১–২ query তেই সব data আসবে 🔥