```
#1.Real Estate Listing
class RealEstateListingManager(models.Manager):
    def by_property_type(self, property_type):
        # in the below case 'self' is 'objects'
        return self.filter(property_type=property_type)

    def in_price_range(self, min_price, max_price):
        # range lookup works inclusive
        return self.filter(price__range=(min_price, max_price))

    def with_bedrooms(self, bedrooms_count):
        return self.filter(bedrooms=bedrooms_count)

    def popular_locations(self):
        return self.values('location').annotate(
            location_count=Count('location')
        ).order_by('-location_count', 'id')[:2]
    # the slicing above on SQL level will be implemented as 'Limit 2'


#2. Video Games Library
#validators.py
def rating_validator(value):
    if not 0.0 <= value <= 10.0:
        raise ValidationError('The rating must be between 0.0 and 10.0')


def release_year_validator(value):
    if not 1990 <= value <= 2023:
        raise ValidationError('The release year must be between 1990 and 2023')


class VideoGameManager(models.Manager):
    def games_by_genre(self, genre):
        return self.filter(genre=genre)

    def recently_released_games(self, year):
        return self.filter(release_year__gte=year)

    def highest_rated_game(self):
        return self.annotate(max_rating=Max('rating')).order_by('-
max_rating').first()

    def lowest_rated_game(self):
        return
self.annotate(min_rating=Min('rating')).order_by('min_rating').first()

    def average_rating(self):
        average_rating = self.aggregate(average_rating=Avg('rating'))
["average_rating"]
        return f"{average_rating:.1f}"

#3.
class Invoice(models.Model):
    pass

    @classmethod
    def get_invoices_with_prefix(cls, prefix):
        return
```

```
cls.objects.select_related('billing_info').filter(invoice_number__startswith=prefix
)

    @classmethod
    def get_invoices_sorted_by_number(cls):
        return
cls.objects.select_related('billing_info').order_by('invoice_number')

    @classmethod
    def get_invoice_with_billing_info(cls, invoice_number):
        return
cls.objects.select_related('billing_info').get(invoice_number=invoice_number)


#4.IT Sector

def get_programmers_with_technologies(self):
    return self.programmers.prefetch_related('projects__technologies_used')


def get_projects_with_technologies(self):
    return self.projects.prefetch_related('technologies_used')


#5.Taskify
@classmethod
def overdue_high_priority_tasks(cls):
    return cls.objects.filter(
        Q(priority='High') & Q(is_completed=False) &
Q(completion_date__gt=F('creation_date'))
    )

@classmethod
def complete_mid_priority_tasks(cls):
    # return cls.objects.filter(priority='Medium', is_completed=True)
    return cls.objects.filter(
        Q(priority='Medium') & Q(is_completed=True)
    )

@classmethod
def search_tasks(cls, query):
    return cls.objects.filter(
        Q(title__icontains=query) | Q(description__icontains=query)
    )


def recent_completed_tasks(self, days):
    return Task.objects.filter(
        Q(is_completed=True) & Q(completion_date__gte=self.creation_date -
timedelta(days=days))
    )

#6. Gym Session

@classmethod
def get_long_and_hard_exercises(cls):
    return cls.objects.filter(
        duration_minutes__gt=30,
        difficulty_level__gte=10,
```

```
    )


@classmethod
def get_short_and_easy_exercises(cls):
    return cls.objects.filter(
        duration_minutes__lt=15,
        difficulty_level__lt=5,
    )


@classmethod
def get_exercises_within_duration(cls, min_duration, max_duration):
    return cls.objects.filter(
        duration_minutes__range=(min_duration, max_duration)
    )


@classmethod
def get_exercises_with_difficulty_and_repetitions(cls, min_difficulty,
min_repetitions):
    return cls.objects.filter(
        difficulty_level__gte=min_difficulty,
        repetitions__gte=min_repetitions,
    )
```