## Operators in Querysets

- | (OR)
- & (AND)

They are more SQL based and add more functionality to django. And to use these features you have to write following code in shell you have opened

In [5]: from django.db.models import Q

**Using |(OR) operator**

In [6]: details.objects.filter(Q(last_name='pari')& Q(age=21)).all()

Out[6]: <QuerySet [<details: pari is 21 years old.>]>

Note- Using this |(OR) operator we can do same as we did above using filter chaining

## Field Lookups

Our model.py looks like:

```
from django.db import models

# Create your models here.

class details(models.Model):
    first_name=models.CharField(max_length =30)
    last_name=models.CharField(max_length =30)
    age = models.IntegerField()
    def __str__(self):
        return f"{self.last_name} is {self.age} years old."
```

**All the data we have are:**

In [2]: from office.models import details

In [3]: details.objects.all()

Out[3]: <QuerySet [<details: pathak is 22 years old.>, <details: pari is 20 years old.>, <details: Y is 44 years old.>, <details: B is 33 years old.>, <details: pari is 21 years old.>]>

Using Operator | (OR) and & (AND) in querysets

In [17]: from django.db.models import Q

In [18]: details.objects.filter(Q(last_name='pari') & Q(age=21)).all()

Out[18]: <QuerySet [<details: pari is 21 years old.>]>

**FIeld Lookups:**

Note- We use field look ups for more complex filtering like in range, starting and ending strings and so on. Lets see them in example below

Some of the field lookups are:

- in
- startswith
- endswith
- gt
- gte
- lt
- lte
- range
- date
- year
- month
- time

In [19]: details.objects.filter(last_name__startswith="p").all()

Out[19]: <QuerySet [<details: pathak is 22 years old.>, <details: pari is 20 years old.>, <details: pari is 21 years old.>]>

Note-

In [20]: details.objects.filter(age__in=[20,21,44]).all()

Out[20]: <QuerySet [<details: pari is 20 years old.>, <details: Y is 44 years old.>, <details: pari is 21 years old.>]>

Note-

In [21]: details.objects.filter(age__gte=30).all()

Out[21]: <QuerySet [<details: Y is 44 years old.>, <details: B is 33 years old.>]>

**order_by()**

In [5]: details.objects.order_by('age')

Out[5]: <QuerySet [<details: pari is 20 years old.>, <details: pari is 21 years old.>, <details: pathak is 22 years old.>, <details: B is 33 years old.>, <details: Y is 44 years old.>]>

## Updating Entries into Model and Saving Changes in Database

Lets try to add new field "user_id" in our model details,

user_id=models.IntegerField(default=60) #Correct

user_id=models.IntegerField(null=True) #Correct

user_id=models.IntegerField() #Wrong

Note- While adding new field in our entries we have to compulsary give some default value to it either default value or set null value, else our makemigration command wont work.

Now our model.py looks like

**Step 1:**

```
from django.db import models

# Create your models here.

class details(models.Model):
    first_name=models.CharField(max_length =30)
    last_name=models.CharField(max_length =30)
    age = models.IntegerField()
    user_id=models.IntegerField(default=60)
    def __str__(self):
        return f"{self.last_name} is {self.age} years old."
```

**Step 2:**

python manage.py makemigrations office

Note- This will save the change in database and we can see that in migration folder. It creates new file in migrations folder eg. 0002_details_user_id.py and we can see the details in that file.

**Updating Entries**

**Step 1**: Selecting or grabbing the entry

```
In [6]: details.objects.get(pk=2)
Out[6]: <details: pari is 22 years old.>

In [7]: y=details.objects.get(pk=2)
In [8]: y.last_name='hero'
```

**Step 2**: Saving the change. It wont work if it is not saved before

In [9]: details.objects.get(pk=2)

Out[9]: <details: pari is 22 years old.>

In [10]: y.save()

In [11]: details.objects.get(pk=2)

Out[11]: <details: hero is 22 years old.>

**Deleting Entries**

**Step 1**: Selecting

In [14]: z=details.objects.get(pk=1)

In [16]: z

Out[16]: <details: a is 23 years old.>

**Step 2**: Delete

Database before delete()

In [12]: details.objects.all()

Out[12]: <QuerySet [<details: a is 23 years old.>, <details: hero is 22 years old.>, <details: pari is 21 years old.>, <details: pdl is 23 years old.>]>

Database after delete()

In [17]: z.delete()

Out[17]: (1, {'office.details': 1})

In [19]: details.objects.all()

Out[19]: <QuerySet [<details: hero is 22 years old.>, <details: pari is 21 years old.>, <details: pdl is 23

years old.>]>

**Connecting Database to Template**

**Step 1**: Create relevent view.py

```
from django.shortcuts import render
from requests import request
from . import models
# Create your views here.
def list_details(request):
    all_data=models.details.objects.all()
    x ={'detail_contents':all_data}
    return render(request,'office/abc.html',context=x)
```

**Step 2**: Connect in app level urls.py

```
from django.urls import path
from . import views
urlpatterns = [
        path('',views.list_details,name="LP")
]
```

**Step 3**: Connect to project level urls.py

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
```

```
    path("admin/", admin.site.urls),
    path("office", include('office.urls')),
]
```

**Step 4**: Create relevent abc.html in template folder within app level

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Details list:</h1>
    {{ detail_contents }}
    <hr>
    <ul>
        {% for p in detail_contents %}
        <li>
        {{p}}
        </li>
        {% endfor %}
    </ul>
</body>
```

```
</html>
```

Note-By following above steps we completed our mission to create model, create templates and connecting them to our database