**May 07**

Employee model contains : 1000 records

employees = Employee.objects.all()-->To get all records based on insertion order.

employees = Employee.objects.all()-->To get all records based on ascending order of eno's

**How to define our own custom manager:**

-->We have to create child class for models.Manager class

-->Whenever we are calling all() method, internally it will call get\_queryset() method.

-->To customize behaviour, we have to override this method in our custom manager class.

**Ex:**To retrieve all employees data according to ascending order of eno, we have to define CustomManager class.

-->Create new project miproject2

-->Create an app

-->Add app in settings.py

* **models.py**

class CustomManager(models.Manager):

def get\_queryset(self):

qs = super().get\_queryset().order\_by('eno')

return qs

class Employee(models.Model):

eno = models.IntegerField()

ename = models.CharField(max\_length=30)

esal = models.FloatField()

eaddr = models.CharField(max\_length=30)

objects = CustomManager()

* **admin.py**

from testapp.models import Employee

class EmployeeAdmin(admin.ModelAdmin):

list\_display = ['eno','ename','esal','eaddr']

admin.site.register(Employee,EmployeeAdmin)

* **views.py**

from testapp.models import Employee

def display\_view(request):

emp\_list = Employee.objects.all()

return render(request,'testapp/index.html',{'emp\_list':emp\_list})

* **index.html**

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title></title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css" integrity="sha384-xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+N" crossorigin="anonymous">

<style media="screen">

body{

background:red;

color:white

}

</style>

</head>

<body>

<div class="container" align='center'>

<h1>Welcome To Employee List</h1>

<table border="3">

<thead>

<th>Employee Number</th>

<th>Employee Name</th>

<th>Employee Salary</th>

<th>Employee Address</th>

</thead>

{% for emp in emp\_list %}

<tr>

<td>{{emp.eno}}</td>

<td>{{emp.ename}}</td>

<td>{{emp.esal}}</td>

<td>{{emp.eaddr}}</td>

</tr>

{% endfor %}

</table>

</div>

</body>

</html>

* **populate.py**

import os

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'miproject2.settings')

import django

django.setup()

from testapp.models import Employee

from faker import Faker

from random import \*

faker = Faker()

def populate(n):

for i in range(n):

feno = randint(1001,9999)

fename = faker.name()

fesal = randint(10000,20000)

feaddr = faker.city()

emp\_record = Employee.objects.get\_or\_create(

eno = feno,

ename = fename,

esal = fesal,

eaddr = feaddr)

n = int(input('Enter number of employees:'))

populate(n)

print(f'{n} Records Inserted Successfully....')

* **urls.py**

path('data/', views.display\_view),

**-->**Based on our requirement, we can define our own new methods also inside CustomManager class.

* **models.py**

class CustomManager(models.Manager):

def get\_queryset(self):

qs = super().get\_queryset().order\_by('eno')

return qs

def get\_emp\_sal\_range(self,minsal,maxsal):

qs = super().get\_queryset().filter(esal\_\_range=(minsal,maxsal))

return qs

def get\_emp\_sorted\_by(self,param):

qs = super().get\_queryset().order\_by(param)

return qs

* **views.py**

def display\_view(request):

#emp\_list = Employee.objects.all()

#emp\_list = Employee.objects.get\_emp\_sal\_range(18000,20000)

#emp\_list = Employee.objects.get\_emp\_sorted\_by('ename')

emp\_list = Employee.objects.get\_emp\_sorted\_by('-esal')

return render(request,'testapp/index.html',{'emp\_list':emp\_list})

**5).Proxy Model Inheritance:**

**-->**For the same Model, we can provide a customized view without touching the database. This is possible by using proxy model inheritance.

**-->**In this table, a separate new table wont be created and new proxy model also pointing to the same old table.

class Employee:

fields

class ProxyEmployee(Employee):

class Meta:

proxy = True

Both Employee and ProxyEmployee are pointing to the same table only.

* **models.py**

class CustomManager1(models.Manager):

def get\_queryset(self):

return super().get\_queryset().filter(esal\_\_gte=19000)

class CustomManager2(models.Manager):

def get\_queryset(self):

return super().get\_queryset().filter(esal\_\_lte=11000)

class CustomManager3(models.Manager):

def get\_queryset(self):

return super().get\_queryset().order\_by('eno')

class Employee(models.Model):

eno = models.IntegerField()

ename = models.CharField(max\_length=30)

esal = models.FloatField()

eaddr = models.CharField(max\_length=30)

objects = CustomManager1()

class ProxyEmployee1(Employee):

objects = CustomManager2()

class Meta:

proxy = True

class ProxyEmployee2(Employee):

objects = CustomManager3()

class Meta:

proxy = True

* **admin.py**

from testapp.models import Employee,ProxyEmployee1,ProxyEmployee2

class EmployeeAdmin(admin.ModelAdmin):

list\_display = ['eno','ename','esal','eaddr']

class ProxyEmployee1Admin(admin.ModelAdmin):

list\_display = ['eno','ename','esal','eaddr']

class ProxyEmployee2Admin(admin.ModelAdmin):

list\_display = ['eno','ename','esal','eaddr']

admin.site.register(Employee,EmployeeAdmin)

admin.site.register(ProxyEmployee1,ProxyEmployee1Admin)

admin.site.register(ProxyEmployee2,ProxyEmployee2Admin)

* **views.py**

def display\_view(request):

#emp\_list = Employee.objects.all()

#emp\_list = ProxyEmployee1.objects.all()

emp\_list = ProxyEmployee2.objects.all()

return render(request,'testapp/index.html',{'emp\_list':emp\_list})

**CHAPTER-14**

**Deployment of our application to the live:**

**Q.Diff b/w str() and repr()?**

class Student:

def \_\_init\_\_(self,name,rollno):

self.name = name

self.rollno = rollno

def \_\_str\_\_(self):

return 'This is Student with Name:{} and Rollno:{}'.format(self.name,self.rollno)

s1 = Student('Radhika',101)

s2 = Student('Lilly',102)

print(s1)

print(s2)

**Ex:**

import datetime

today = datetime.datetime.now()

print(type(today))

s = repr(today)#converting datetime object to str

print(type(s))

d = eval(s)#converting str object to datetime

print(type(d))