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.cs
s" 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))
```