May 01

CHAPTER-11

Django ORM:

ORM--->Object Relational Mapping
Java:Hibernate, SPring ORM etc......

To select all employees from the employee table
sql query:select * from employee
ORM:Employee.objects.all()

Ex:

```
D:\Django_20MAR_7PM>django-admin startproject ormproject1
D:\Django_20MAR_7PM>cd ormproject1
D:\Django_20MAR_7PM\ormproject1>py manage.py startapp testapp
```

-->Add app in settings.py

- **models.py**

```
class Employee(models.Model):
   eno = models.IntegerField()
   ename = models.CharField(max_length=30)
   esal = models.FloatField()
   eaddr = models.CharField(max_length=64)
```

-->makemigrations and migrate

- **admin.py**

```
from testapp.models import Employee
class EmployeeAdmin(admin.ModelAdmin):
   list_display = ['eno','ename','esal','eaddr']
admin.site.register(Employee,EmployeeAdmin)
```

-->create super user.

- **populate.py**

```
import os
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'ormproject1.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....')
```

- **base.html**

```
<body>
    <div class="container">
      {% block body_block %}
      {% endblock %}
    </div>
</body>
```

- **index.html**

```
<!DOCTYPE html>
{% extends 'testapp/base.html' %}
{% block body_block %}
<h1>Employee Information DashBoard</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>
{% endblock %}
```

- **views.py**

```
from testapp.models import Employee
def retrieve_view(request):
   emp_list = Employee.objects.all()
   return render(request,'testapp/index.html',{'emp_list':emp_list})
```

- **urls.py**

```
path('', views.retrieve_view),
```

To select all records:

```
Employee.objects.all()
```

The return type of all() method is:QuerySet

```
<class 'django.db.models.query.QuerySet'>
```

To get a particular record:

We have to use get() method.

```
D:\Django_20MAR_7PM\ormproject1>py manage.py shell

>>> from testapp.models import Employee
>>> emp = Employee.objects.get(id=1)
>>> emp #<Employee: Employee object (1)>
>>> emp.eno #5568
>>> emp.ename #'Lauren Griffin'
>>> type(emp) #<class 'testapp.models.Employee'>
```

-->The return type of get() method is Employee object.

How to find query associated with QuerySet:

Every ORM statement will be converted into sql query. We can find query from the QuerySet.

```
>>qs = Employee.objects.all()
>>> qs.query
<django.db.models.sql.query.Query object at 0x0000020FE9274D00>
>>> str(qs.query)
'SELECT "testapp_employee"."id", "testapp_employee"."eno",
"testapp_employee"."ename", "testapp_employee"."esal",
"testapp_employee"."eaddr" FROM "testapp_employee"'
```

How to filter records based on some condition

1).List out all employees whose salaries greater than 15000.

```
                emp_list= Employee.objects.filter(esal__gt=15000)
```

2).Salaries greater than or equal to 15000

```
                emp_list= Employee.objects.filter(esal__gte=15000)
```

Similarly we can use __lt and __lte also.

Ex:

1.exact:exact match

```
>>> emp = Employee.objects.get(id__exact=52)
>>> emp.ename #'Radhika'
>>> emp = Employee.objects.get(id=51)
>>> emp.ename #'Sunny'
```

2.contains:case sensitive containment test

```
                select .....where ename like '%jhon%'
        emp_list = Employee.objects.filter(ename__contains='jhon')
```

3.in:

In a given iterable like tuple or list

```
        emp_list = Employee.objects.filter(id__in=[1,51,52])
```

4).gt:greater than

5).gte:greater than or equal to

6).lt:less than

7).lte:less than or equal to

8).startswith:
select all employees where ename starts with 'S'
emp_list = Employee.objects.filter(ename__startswith='S')

9).endswith:
emp_list = Employee.objects.filter(ename__endswith='s')

10).range:
range test(inclusive)
To select all employees where esal in the range 12000 to 15000
emp_list = Employee.objects.filter(esal__range=[12000,15000])

Q1.Select all employees where ename starts with 'A'
emp_list = Employee.objects.filter(ename__startswith='A')

Q2.Select all employees whose sal <= 15000
emp_list= Employee.objects.filter(esal__lte=15000)

Q3.Select all employees where ename starts with 'A' or esla <= 15000.
We can implement OR queries in 2-ways.

1st way:
emp_list = queryset1 | queryset2
Ex:
emp_list = Employee.objects.filter(ename__startswith='A') |

Employee.objects.filter(esal__lte=11000)
2nd way:
filter(Q(condition1) | Q(condition2))
from django.db.models import Q
emp_list = Employee.objects.filter(Q(condition1) | Q(condition2))
emp_list = Employee.objects.filter(Q(ename__startswith='D') | Q(esal__lte=12000))