May 06

CHAPTER-12

Working with Advanced Model concepts:

Model Inheritance:

1.Abstract Base Class model inheritance
2.Multi table inheritance
    3.Multi level inheritance
    4.Multiple inheritance
5.Proxy model inheritance

1.Abstract Base Class model inheritance:

-->If several model classes having some common fields, then it is not recommended to write these fields in every model class separately, because it increases length of the code and reduces readability.

-->We have to separate those common fields into a separate model class which is nothing but Base class. If we extend Base class then automatically common fields will be inherited to every child class.

Ex:
1.Create project miproject1
2.Create app
3.Add app in settings.py

- **settings.py**

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.mysql',
      'NAME': 'midb_7pm',
      'USER': 'root',
      'PASSWORD': 'root'
   }
}
```

- **models.py**

```
class ContactInfo(models.Model):
   name = models.CharField(max_length=30)
   email = models.EmailField()
   address = models.CharField(max_length=30)
   class Meta:
       abstract = True
class Student(ContactInfo):
   rollno = models.IntegerField()
   marks = models.IntegerField()
class Teacher(ContactInfo):
   subject = models.CharField(max_length=30)
   salary = models.FloatField()
```

- **admin.py**

```
from testapp.models import Student,Teacher
admin.site.register(Student)
admin.site.register(Teacher)
```

-->Makemigrations and Migrate.

Note:

ConcatInfo class is a abstract class, hence table wont be created. This type of inheritance is applicable only at code elevel but not at database level.

<u>2).Multi table inheritance:</u>

-->If the Base class is not abstract, then such type of inheritance is called as multi table inheritance.

-->This type of inheritance applicable at code level and DB level.

-->In multi table inheritance, inside database, for both parent and child tables will be created.

- **models.py**

```
class ContactInfo1(models.Model):
    name = models.CharField(max_length=30)
    email = models.EmailField()
    address = models.CharField(max_length=30)
class Student1(ContactInfo1):
    rollno = models.IntegerField()
    marks = models.IntegerField()
class Teacher1(ContactInfo1):
    subject = models.CharField(max_length=30)
    salary = models.FloatField()
```

- **admin.py**

```
admin.site.register(Student1)
admin.site.register(Teacher1)
admin.site.register(ContactInfo1)
```

-->In this case 3-tables will be created and child table will maintain pointer to the parent table to refer the common properties.

<u>3).Multi level inheritance:</u>

Inheritance at multiple levels.

<u>Ex:</u>

```
class Person(models.Model):
   name = models.CharField(max_length=30)
   age = models.IntegerField()
class Employee(Person):
   eno = models.IntegerField()
   esal = models.FloatField()
class Manager(Employee):
   exp = models.IntegerField()
   team_size = models.IntegerField()
```

- **admin.py**

```
admin.site.register(Person)
admin.site.register(Employee)
admin.site.register(Manager)
```

<u>4).Multiple Inheritance:</u>

If model class extends multiple parent classes simultaneously then such type of inheritance is called as multiple inheritance.

```
class Parent1(models.Model):
   f1 = models.CharField(max_length=30)
   f2 = models.CharField(max_length=30)
class Parent2(models.Model):
   f3 = models.CharField(max_length=30,primary_key=True)
   f4 = models.CharField(max_length=30)
class Child(Parent1,Parent2):
   f5 = models.CharField(max_length=30)
   f6 = models.CharField(max_length=30)
```

Note:

1.Parent classes hould not contain common fields, otherwise we eill get an error.

2.Internally this inheritance also multi table inheritance.

-->Makemigrations and Migrate.

<u>Model Manager:</u>

-->We can use ModelManager to interact with database.
-->We can get default ModelManager by using Model.objects property
-->Model.objects is of type:django.db.models.manager.Manager

```
manager = Employee.objects
employees = manager.all()
```

Q1.What is the purpose of ModelManager?
Ans: To interact with database.
Q2.How to get default Model Manager?
Ans: By using Model.objects property
Q3.ModelManager is of what type?
Ans: django.db.models.manager.Manager

go to shell:

```
>>> from testapp.models import Person
>>> type(Person.objects)
<class 'django.db.models.manager.Manager'>
```

-->Based on our requirement, we can define and use our own custom model managers.

<u>How to define our own custom manager:</u>

<u>How to access private variables outside of the class</u>

```
class Test:
        __x = 100 #static variable
        def __init__(self):
                self.__y = 200 #instance variable
```

```
t = Test()
#print(t.__dict__)
print(t._Test__y)
#print(Test.__dict__)
print(Test._Test__x)
```