Model Manager

A Manager is the interface through which database query operations are provided to Django models. At least one Manager exists for every model in a Django application.

Model manager is used to interact with database.

By default, Django adds a Manager with the name objects to every Django model class.

django.db.models.manager.Manager

Change Manager Name

By default, Django adds a Manager with the name objects to every Django model class. However, if you want to use objects as a field name, or if you want to use a name other than objects for the Manager, you can rename it on a per-model basis.

To rename the Manager for a given class, define a class attribute of type models.Manager() on that model.

```
from django.db import models

class Student(models.Model):
 name = models.CharField(max_length=70)
 roll = models.IntegerField()
 students = models.Manager()


student_data = Student.students.all()
```

Custom Model Manager

You can use a custom Manager in a particular model by extending the base Manager class and instantiating your custom Manager in your model.

A custom Manager method can return anything you want. It doesn’t have to return a QuerySet.

- to modify the initial QuerySet the Manager returns
- to add extra Manager methods

Modify the initial QuerySet

A Manager's base QuerySet returns all objects in the system. You can override a Manager's base QuerySet by overriding the Manager.get_queryset() method. get_queryset() should return a QuerySet with the properties you require.

Write Model Manager

```
class CustomManager(models.Manager):
  def get_queryset(self):  # overriding Built-in method called when we call all()
     return super().get_queryset().order_by('name')
```

Associate Manager with Model

```
Class Student(models.Model):
  objects = models.Manager()          # Default Manager
  students = CustomManager()          # Custom Manager
```

You can associate more than one manager in one Model

views.py

```
Student_data = Student.objects.all()          # Work as per default Manager
Student_data = Student.students.all()         # Work as per Custom manager
```

Add extra Manager methods

Adding extra Manager methods is the preferred way to add “table-level” functionality to your models.

Write Model Manager

```
class CustomManager(models.Manager):
 def get_stu_roll_range(self, r1, r2):
  return super().get_queryset().filter(roll__range=(r1, r2))
```

Associate Manager with Model

```
Class Student(models.Model):
objects = models.Manager()
 students = CustomManager()
```

views.py

```
Student_data = Student.objects.all()
Student_data = Student.students.get_stu_roll_range(101, 103)
```