Abstract Base Class

این مدل حالتی از ارث بری می باشد که می توانیم برای صفات یا متعلقاتی از کلاس ها که در چند مورد تکرار شده اند یک کلاس والد بنویسیم و سایر کلاس های دارای آن صفت مشابه را با ارث بری از کلاس والد که خود از کلاس models ارث می برد تکمیل میکنیم.

به طور مثال:

```
from django.db import models


class Student(models.Model):  # STUDENT
    name = models.CharField(max_length=100)
    rollno = models.IntergerField()


class Teacher(models.Model):  # TEACHER
    name = models.CharField(max_length=100)
    ID = models.IntergerField()
```

در مثال بالا دو کلاس Student و Teacher دارای جزء مشابه name میباشند که برای عدم تکرار این جزء از یک کلاس والد دارای این جزء مشابه استفاده میکنیم.

```
from django.db import models


class common(models.Model):  # COMM0N
    name = models.CharField(max_length=100)

    class Meta:
        abstract = True
```

کلاس common همان کلاس والد است که دارای جزء name میباشد. همچنین از کلاس Meta درون کلاس والد استفاده میکنیم که به پایگاه داده بفهمانیم که این کلاس abstract است و آن را یک مدل جداگانه در نظر نگیرد و برای آن جدول درست نکند. در نهایت دو کلاس ما به شکل زیر ارث بری می کنند.

```
class Student(common):  # STUDENT
    rollno = models.IntegerField()


class Teacher(common):  # TEACHER
    ID = models.IntegerField()
```

Multi Table Model

در این مدل از نوعی ارث بری چند لایه استفاده میکنیم.در این مدل یک کلاس کلی در نظر گرفته می شود که مورد استفاده است اما ما کلاسی داریم که زیر مجموعه کلاس اصلی است و بدلیل پر استفاده بودن آن زیر مجموعه به طوری که از کلاس اصلی ارث بری کند ایجاد میکنیم و صفات جدیدی هم به آن اضافه می کنیم.

به طور مثال:

```python
from django.db import models

# Create your models here.


class Place(models.Model):
    name = models.CharField(max_length=30)
    address = models.CharField(max_length=30)


    def __str__(self):
        return self.name


class Restaurant(Place):
    serves_tuna = models.BooleanField(default=False)
    serves_pizza = models.BooleanField(default=False)


    def __str__(self):
        return self.serves_tuna
```

Proxy Model

در این مدل از نوعی ارث بری استفاده میکنیم برای اینکه به کلاس اصلی ویژگی و جزء های جدیدی را به صورت تفکیک شده اضافه کنیم و از خود کلاس های ارث بری شده نمیتوانیم استفاده مستقیم کنیم و برای استفاده هر کدام از ویژگی های اصلی حتما باید از کلاس اصلی استفاده کنیم. برای اینکه به پایگاه داده بفهمانیم که این کلاس ارث بری شده صرفا یک ویژگی جدید برای کلاس اصلی است در کلاس ارث برای شده در قسمت Meta واژه proxy=true را قرار می دهیم.

به طور مثال:

ما در این مثال از مدل پروکسی بهره بردیم برای اینکه Manager های جدیدی را به کلاس اصلی خود به صورت تفکیک شده استفاده کنیم.

```
STORY_TYPES = (
    ('f', 'Feature'),
    ('i', 'Infographic'),
    ('g', 'Gallery'),
)

class Story(models.Model):
    type = models.CharField(max_length=1, choices=STORY_TYPES)
    title = models.CharField(max_length=100)
    body = models.TextField(blank=True, null=True)
    infographic = models.ImageField(blank=True, null=True)
    link = models.URLField(blank=True, null=True)
    gallery = models.ForeignKey(Gallery, blank=True, null=True)
```

```
class FeatureStory(Story):
    objects = FeatureManager()
    class Meta:
        proxy = True

class InfographicStory(Story):
    objects = InfographicManager()
    class Meta:
        proxy = True

class GalleryStory(Story):
    objects = GalleryManager()
    class Meta:
        proxy = True
```

در این قسمت Manager های اسفاده شده را تعریف کردیم:

```
class FeatureManager(models.Manager):
    def get_queryset(self):
        return super(FeatureManager, self).get_queryset().filter(
            type='f')

class InfographicManager(models.Manager):
    def get_queryset(self):
        return super(InfographicManager, self).get_queryset().filter(
            type='i')

class GalleryManager(models.Manager):
    def get_queryset(self):
        return super(GalleryManager, self).get_queryset().filter(
            type='g')
```