ИНДЕКСИ

indexes = [models.Index(fields=['Menu'], name="main\_app\_menu\_review\_menu\_id")]

МИКСИНИ

class RestaurantReview(ReviewMixin):  
 reviewer\_name = models.CharField(max\_length=100)  
 restaurant = models.ForeignKey("Restaurant", on\_delete=models.CASCADE)  
  
 class Meta(**ReviewMixin.Meta**):  
 abstract = True  
 verbose\_name = "Restaurant Review"  
 verbose\_name\_plural = "Restaurant Reviews"  
 unique\_together = ["reviewer\_name", "restaurant"]

error\_message

email = models.EmailField(error\_messages=['invalid', "Enter a valid email address"])

Проверява дали всичките символи са буква или спейс

def validator(value):  
 for char in value:  
 if not char.isalpha() or char.isspace():  
 raise ValidationError("Name can only contain letters and spaces")  
  
Проверява чрез regex тел номер  
def phone\_validator(value):  
 if not re.match(r'^\+359\d{9}$', value):  
 raise ValidationError("Phone number must start with a '+359' followed by 9 digits")

**auto\_now\_add** 🡪 Every time a **new** **record** is created it should save the **current time** of the creation of the **record**.

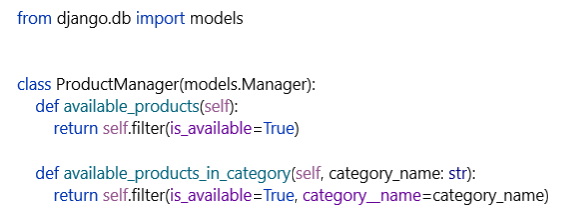
**Proxy = True** 🡪 The model "**DiscountedProduct**" shares the same **database** **table** as its parent model "**Product**" and provides **additional** or **customized** functionality.

**3-та задача** – **CUSTOM MODEL MANAGER**

1. Правим си нов файл manager.py
2. From django.db import models
3. Създаваме клас с име името на модела долепено до Manager 🡪 class DirectorManager(models.Manager):
4. Създаваме метода 🡪 def get\_directors\_by\_movies\_count(self):
5. return self.annotate(num\_movies=models.Count(“movies”)).order\_by(“-num\_movies”, “full\_name”)

**moveis e related\_name!!!**

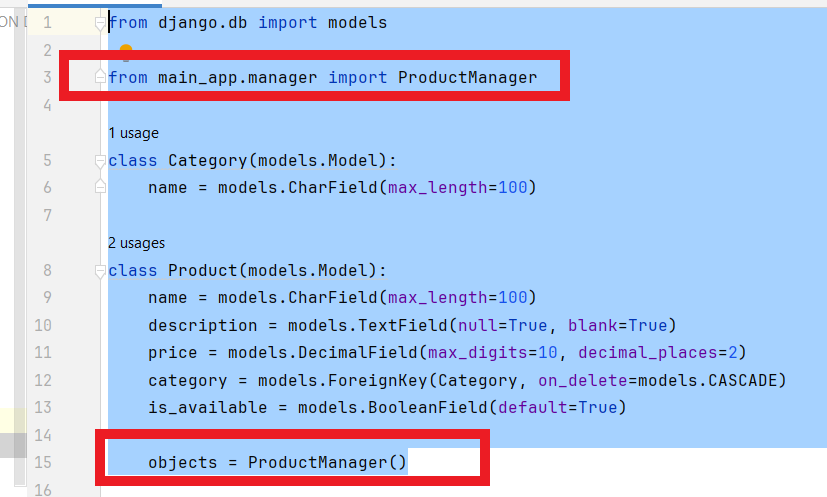
1. **Добавяме manager-a в class-a (models.py) 🡪 from main\_app.manage import DirectorManager**
2. **След filed-a на класа пишем objects = DirectorManage()**

****

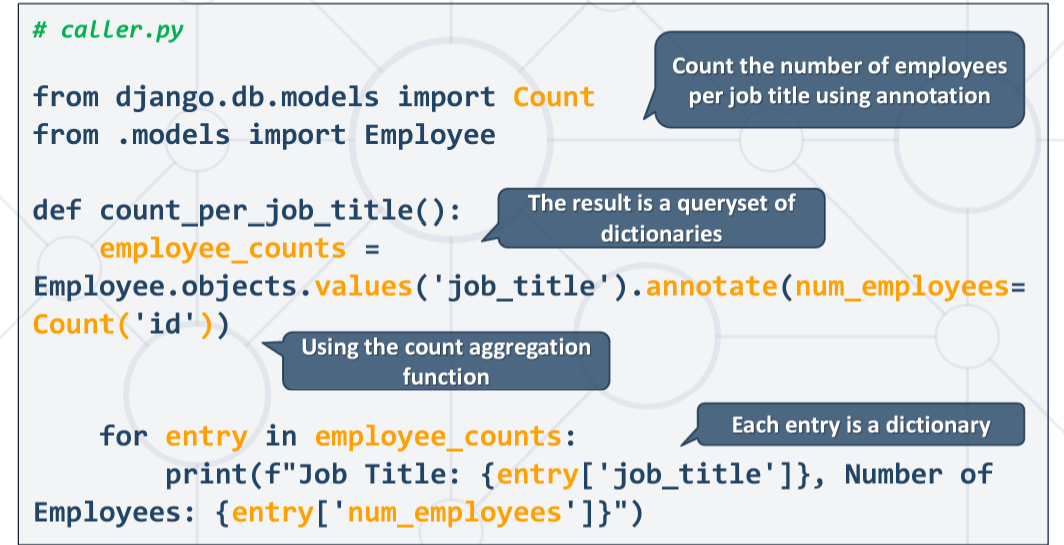
from django.db import models  
from django.db.models import Count  
  
  
class ProfileManager(models.Manager):  
 def get\_regular\_customers(self):  
 return self.annotate(orders\_count=Count('orders')).filter(orders\_count\_\_gt=2).order\_by('-orders\_count')

**ТУКА СЕ НАЛОЖИ ДА ПРАВЯ related\_name=’orders’ В МОДЕЛ ORDER, ИНАЧЕ НЕ МИНАВА**

**След това в модела:**

****

**ANNOTATION**



**Q**

**def filter\_products():  
 query = Q(is\_available=True) & Q(price\_\_gt=3)  
 products = Product.objects.filter(query).order\_by('-price', 'name')  
  
 result = []  
  
 for product in products:  
 result.append(f"{product.name}: {product.price}lv.")  
  
 return '\n'.join(result)  
  
print(filter\_products())**

**F**

**Пример**

****

**Променя всички стойности в дадено поле.**

**ЗАЯВКИ**

**4.1.**

from main\_app.models import Profile, Product, Order  
  
  
def get\_profiles(search\_string=None):  
 if not search\_string:  
 return ""  
  
 profiles = Profile.objects.filter(  
 Q(full\_name\_\_icontains=search\_string)  
 |  
 Q(email\_\_icontains=search\_string)  
 |  
 Q(phone\_number\_\_icontains=search\_string)  
 ).order\_by('full\_name')  
  
 result = []  
  
 for p in profiles:  
 result.append(f"Profile: {p.full\_name}, email: {p.email}, phone number: {p.phone\_number}, orders: {p.orders.count()}")  
  
 return '\n'.join(result)

**4.2.**

def get\_loyal\_profiles():  
 profiles = Profile.objects.get\_regular\_customers()  
  
 result = []  
  
 for p in profiles:  
 result.append(f"Profile: {p.full\_name}, orders: {p.orders\_count}")  
  
 return '\n'.join(result)

**4.3.**

def get\_last\_sold\_products():  
 last\_order = Order.objects.prefetch\_related('products').last()  
 if last\_order is None or not last\_order.products.exists():  
 return ""  
  
 product\_names = [product.name for product in last\_order.products.all()]  
  
 return f"Last sold products: {', '.join(product\_names)}"

**5.**

**Order** them **by the number of times** the **product** has been **sold** (included in an order), **descending**, **then** **ascending** by **product** **name**.

top\_products = Product.objects.annotate(  
 orders\_count=Count('order').filter(orders\_count\_\_gt=0).order\_by('-orders\_count', 'name')  
)