Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования  
«Новосибирский государственный технический университет»

Кафедра Автоматизированных систем управления



**Отчет**

**по лабораторной работе №8**

**Тема**: Связи между таблицами. Методы выбора и обработки записей

**Дисциплина**: Разработка программных приложений и WEB-программирование

|  |  |  |
| --- | --- | --- |
| **Выполнили:** |  | **Проверил:** |
| Студенты | *Вашурина С.И.* | Старший преподаватель кафедры АСУ Эстрайх И.В. |
| Факультет | *АВТФ* |
| Направление (специальность) подготовки | *09.03.03 – Прикладная информатика* |
| Группа | *АП-226* |

Новосибирск 2025

**Цель работы:** изучить типы связей между таблицами и способы их установки; освоить применение методов выбора и обработки записей из таблиц.

**Задание к лабораторной работе №8**

1. Добавить таблицы в БД, установить связи между таблицами (использовать все виды связей).

2. Добавить на сайт теги. Обеспечить вывод информации на странице с использованием тегов.

3. Использовать в программе описанные методы выбора записей, классы Q, F и Value, вычисляемые поля, агрегирующие функции, группировку записей, вычисления на стороне СУБД.

**Ход выполнения**

**1. Добавление всех видов связей**

Для выполнения первого пункта задания были добавлены две новые модели: Tag и Manufacturer.

**Модель Tag (ManyToManyField)**

Модель Tag представляет теги, которые могут быть присвоены товарам. Связь с моделью Product реализована через поле ManyToManyField, что позволяет одному товару иметь несколько тегов, и одному тегу быть связанным с несколькими товарами.

Код модели Tag в models.py:

class Tag(models.Model):

    name = models.CharField(max\_length=100, verbose\_name="Название тега")

    slug = models.SlugField(max\_length=100, unique=True, verbose\_name="URL-идентификатор")

    def \_\_str\_\_(self):

        return self.name

    def get\_absolute\_url(self):

        return reverse('tag', kwargs={'tag\_slug': self.slug})

    class Meta:

        verbose\_name = "Тег"

        verbose\_name\_plural = "Теги"

        ordering = ['name']

В модель Product добавлено поле:

tags = models.ManyToManyField('Tag', blank=True, related\_name='products', verbose\_name="Теги")

**Модель Manufacturer (OneToOneField)**

Модель Manufacturer хранит информацию о производителе. Связь с моделью Product реализована через OneToOneField, что означает, что у каждого товара может быть только один производитель.

Код модели Manufacturer в models.py:

class Manufacturer(models.Model):

    name = models.CharField(max\_length=100, verbose\_name="Название производителя")

    country = models.CharField(max\_length=100, blank=True, verbose\_name="Страна")

    description = models.TextField(blank=True, verbose\_name="Описание")

    def \_\_str\_\_(self):

        return self.name

    class Meta:

        verbose\_name = "Производитель"

        verbose\_name\_plural = "Производители"

В модель Product добавлено поле:

manufacturer = models.OneToOneField('Manufacturer', on\_delete=models.SET\_NULL, null=True, blank=True, related\_name='product', verbose\_name="Производитель")

**Связь ForeignKey**

Связь ForeignKey уже была реализована в исходной модели Product через поле category:

category = models.ForeignKey(Category, on\_delete=models.CASCADE, related\_name='products', verbose\_name="Категория")

После добавления моделей были созданы и применены миграции:

**python manage.py makemigrations**

**python manage.py migrate**

Для тестирования были добавлены данные через Django shell:

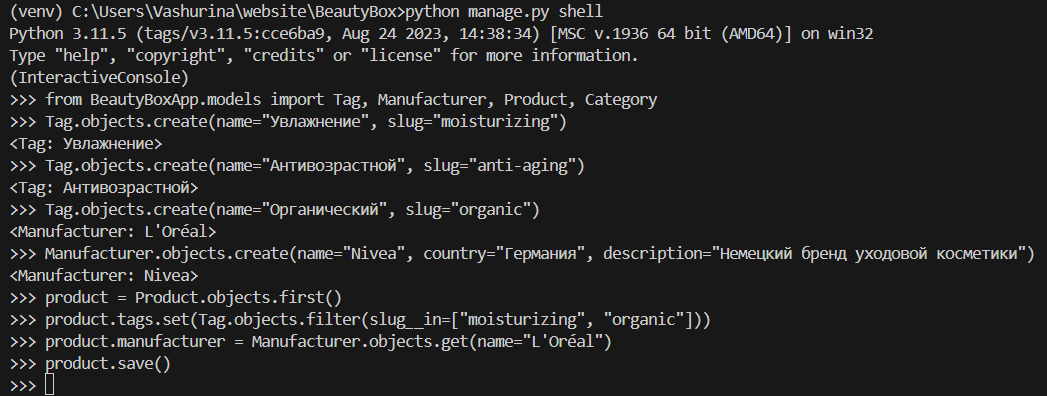


Рисунок 1 – Добавление данных

**2. Добавление тегов на сайт**

Для вывода тегов на сайте были выполнены следующие шаги:

**2.1. Маршрут для тегов**

В urls.py добавлен маршрут для отображения товаров по тегу:

**path('tag/<slug:tag\_slug>/', views.show\_tag, name='tag'),**

**2.2. Представление для тегов**

В views.py добавлено представление show\_tag:

def show\_tag(request, tag\_slug):

    tag = get\_object\_or\_404(Tag, slug=tag\_slug)

    products = Product.objects.filter(tags\_\_slug=tag\_slug, status=Product.ProductStatus.AVAILABLE.name)

    data = {

        'title': f'Тег: {tag.name}',

        'products': products,

        'cat\_selected': 0,

    }

    return render(request, 'cosmetics/catalog.html', data)

**2.3. Шаблонный тег для тегов**

Создан файл templatetags/cosmetics\_tags.py с тегом get\_tags:

@register.inclusion\_tag('cosmetics/list\_tags.html')

def show\_tags():

    tags = Tag.objects.annotate(total=Count('products')).filter(total\_\_gt=0)

    return {'tags': tags}

**2.4. Интеграция тегов в навигационное меню**

Теги добавлены в выпадающее меню "Теги" в includes/nav.html:

<li class="dropdown">

            <a href="#" class="dropdown-toggle">Теги</a>

            <ul class="dropdown-menu">

                {% show\_tags %}

            </ul>

        </li>

**2.4. Шаблон для тегов**

Добавлен шаблон list\_tags.html:

{% if tags %}

    <ul class="tags-list">

        {% for tag in tags %}

            <li><a href="{% url 'tag' tag\_slug=tag.slug %}">{{ tag.name }}</a></li>

        {% endfor %}

    </ul>

{% endif %}

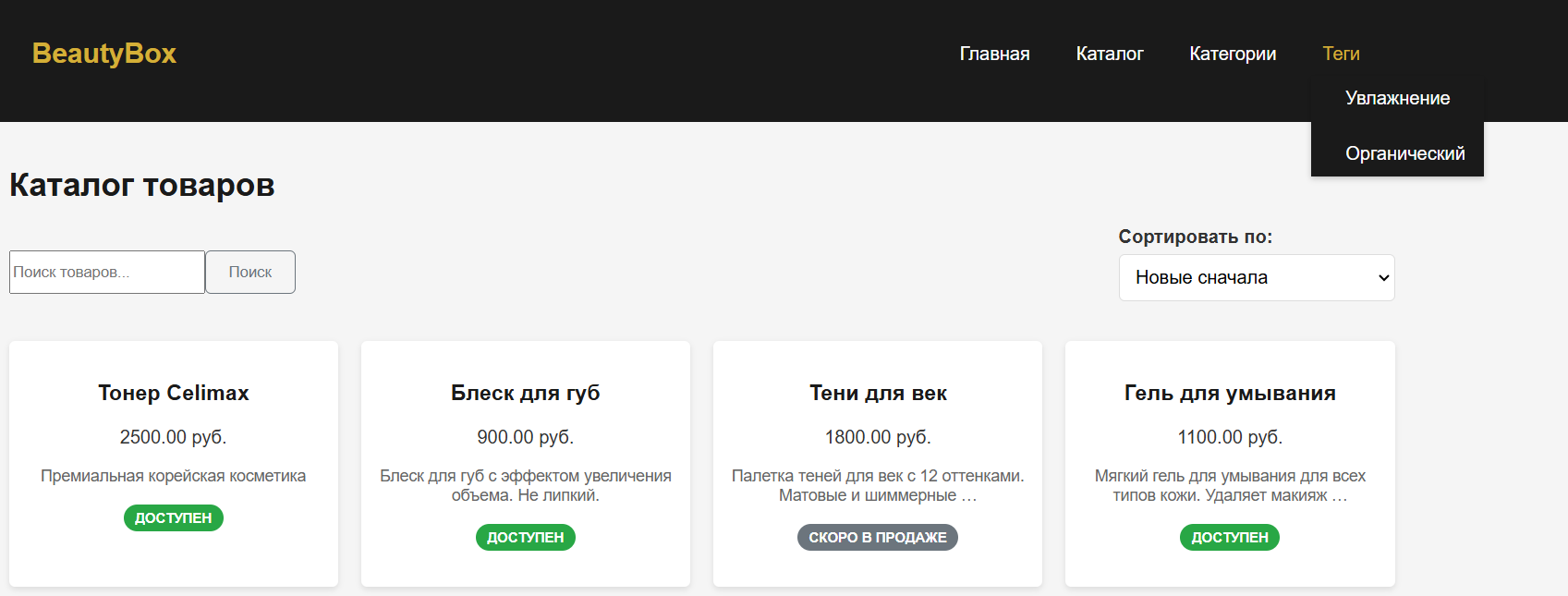


Рисунок 2 – Изменение на странице сайта

**3. Использование методов ORM Django**

Рассмотрим некоторые методы извлечения записей. Например, чтобы взять первую запись из выборки, используется метод first(); Берется первая запись в соответствии с порядком сортировки модели. Мы можем поменять этот порядок и с помощью этого же метода first() выбирать разные записи, например, так.

Или же воспользоваться аналогичным методом last() для выбора последней записи из набора:

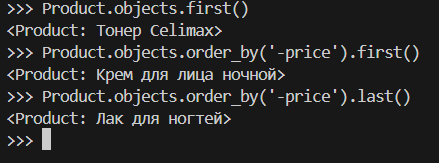


Рисунок 3 – Работа с методом first() и last()

Если в таблице присутствуют поля с указанием даты и времени, то для таких записей и таких таблиц можно применять методы: latest() – выбор записи с самой поздней датой (наибольшей); earliest () – выбор записи с самой ранней датой (наименьшей). Например:

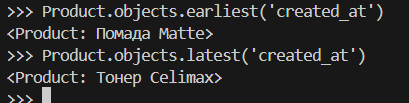


Рисунок 4 – Использование latest и earliest

Для получения предыдущей записи относительно текущей, можно записать:

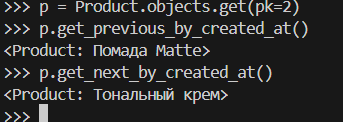


Рисунок 5 – Использование get\_previous\_by\_, get\_next\_by\_

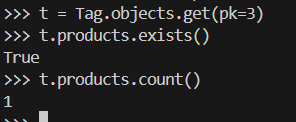


Рисунок 6 – Использование exists и count

Если в условии нужно использовать логическое ИЛИ, а также НЕ, то вместо перечисления критериев отбора через запятую, следует использовать специальный класс Q. С помощью этого класса можно описывать более сложные критерии (условия), используя специальные операторы:

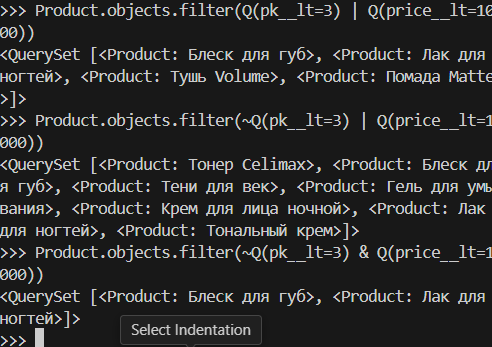


Рисунок 7 – Работа с классом Q

Для того, чтобы использовать значение другого поля таблицы нужно использовать специальный класс F, позволяющий нам выполнять подобные операции. Расположен он в ветке django.db.models:

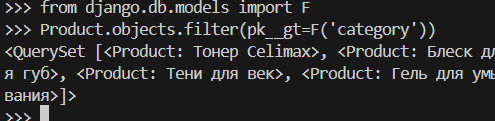


Рисунок 8 – Работа с классом F

При формировании выборки записей из таблиц БД ORM Django предоставляет возможность формировать дополнительные вычисляемые поля. Что это такое? Предположим, что мы бы хотели автоматически формировать новое булево поле is\_test для таблицы techspec, со значениями «5+4» (9). Для этого следует воспользоваться специальным методом, который называется annotate(), следующим образом:

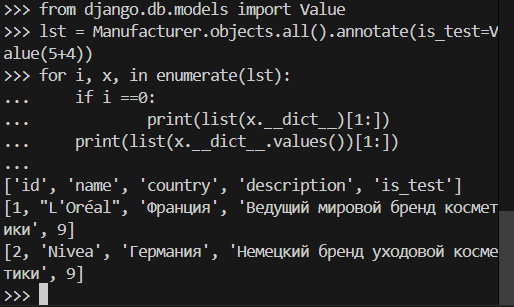


Рисунок 9 – Работа с классом Value

Здесь Value – это специальный класс, который используется для формирования вычисляемых значений полей таблицы. В данном случае мы просто указали выражение 5+4.

**Метод Values():** Для указания нужных полей в выборке, используется метод values() с указанием названий полей, например, так:

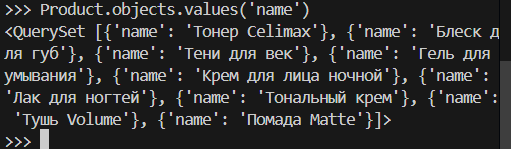


Рисунок 10 – Работа с классом Values

**Группировка записей:**

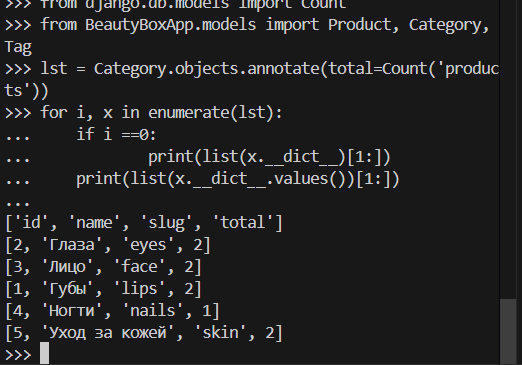
****

Рисунок 11 – Группировка записей

**Агрегирующие функции:**

Рассмотрим несколько агрегирующих методов. С одним из них мы уже знакомы – это метод count(), который подсчитывает число записей. Например, можно определить число записей в таблице Watch:



Рисунок 12 – Работа метода Count

ORM Django поддерживает следующие основные команды агрегации: Count, Sum, Avg, Max, Min. Сначала их следует импортировать из ветки django.db.models:

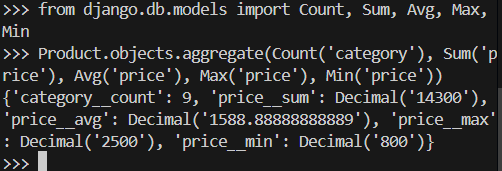


Рисунок 13 – Работа с агрегирующими функциями

**Вычисления на стороне СУБД**

Для примера рассмотрим использование функции Length для вычисления длины строки. Сначала нам нужно ее импортировать:

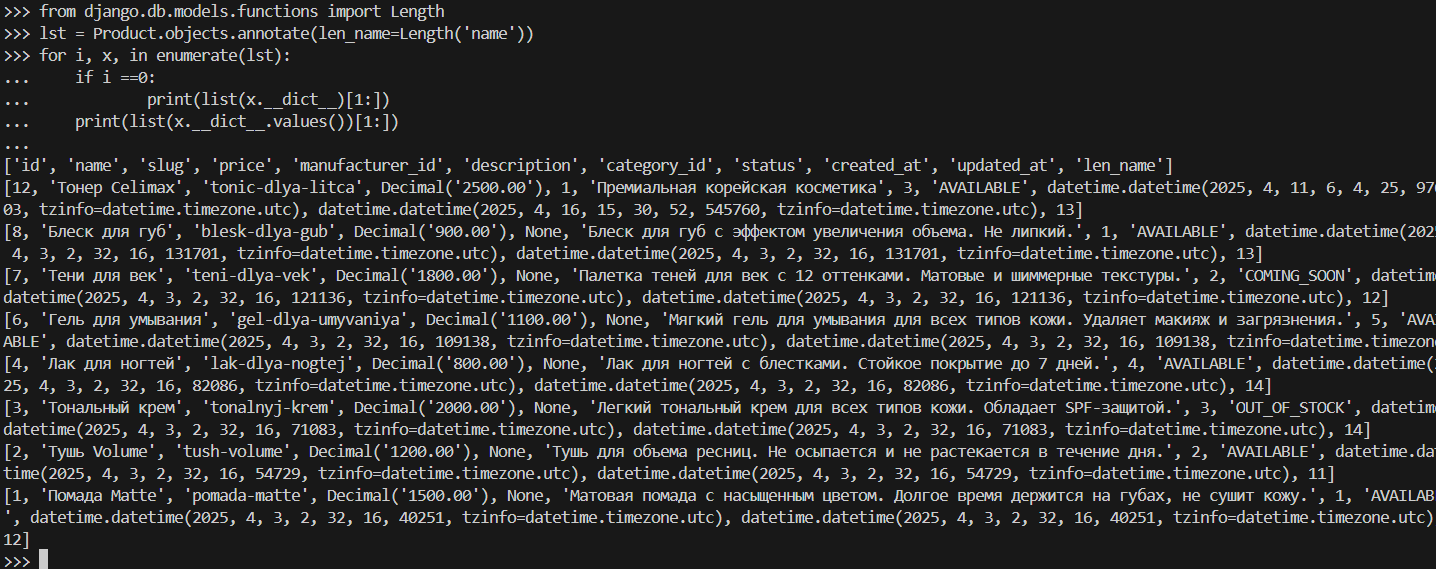


Рисунок 14 – Работа с вычислением на стороне СУБД

**Вывод**

В процессе выполнения лабораторной работы были рассмотрены следующие аспекты:

 Типы связей между таблицами в Django: ForeignKey (категории), ManyToManyField (теги), OneToOneField (производители).

Использование шаблонных тегов для динамической генерации данных (например, get\_tags для тегов).

Методы ORM Django: фильтрация с использованием Q, вычисления с F, аннотации с Value, агрегирующие функции (Count, Avg), группировка (values).

В ходе работы были получены следующие навыки:

Разработка веб-приложений на Django с использованием моделей, представлений и шаблонов.

Оптимизация запросов к базе данных с помощью ORM.