МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра автоматизированных систем управления



**ОТЧЁТ**

**по лабораторной работе №8**

**на тему:**

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

**по дисциплине:** Разработка программных приложений и WEB-программирование

Выполнили работу:

Студенты гр. АП-226, АВТФ

Ищенко С. Г.

«3» апреля 2025 г.

Проверил работу:

Эстрайх И. В.

«\_\_\_» \_\_\_\_\_\_ 20\_\_г

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**Цель**

Изучить типы связей между таблицами и способы их установки;

освоить применение методов выбора и обработки записей из таблиц.

**Задание**

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.

**Описание хода выполнения работы**

1. Добавить таблицы в БД, установить связи между таблицами

(использовать все виды связей).

**Новые модели и связи:**  
В файле **cs\_ty/cs/models.py** добавлены новые модели и поля для установления различных типов связей:

* **Модель FieldOfStudy:**  
  Создана для хранения информации об областях компьютерных наук.

*# Модель для областей компьютерных наук*

class FieldOfStudy(models.Model):

    name = models.CharField(*max\_length*=255, *verbose\_name*="Область науки")

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

    class Meta:

        verbose\_name = "Область науки"

        verbose\_name\_plural = "Области науки"

    def \_\_str\_\_(*self*):

*return* *self*.name

* **Связь один-ко-многим (ForeignKey):**  
  В модель **ComputerScienceConcept** добавлено поле field\_of\_study, ссылающееся на модель FieldOfStudy:

*# Связь один-ко-многим с FieldOfStudy*

    field\_of\_study = models.ForeignKey(

        FieldOfStudy,

*on\_delete*=models.CASCADE,

*related\_name*='concepts',

*verbose\_name*="Область науки",

*null*=True,

*blank*=True

    )

* **Модель ConceptDetail (OneToOne):**Создана для хранения расширенной информации об концепции, например, данные о ключевых технологиях, предварительных условиях и примерном времени изучения.

class ConceptDetail(models.Model):

    concept = models.OneToOneField(

        ComputerScienceConcept,

*on\_delete*=models.CASCADE,

*related\_name*='detail',

*verbose\_name*="Концепция"

    )

    core\_technologies = models.TextField(*blank*=True, *verbose\_name*="Ключевые технологии")

    prerequisites = models.TextField(*blank*=True, *verbose\_name*="Предварительные условия")

    estimated\_learning\_time = models.PositiveIntegerField(*verbose\_name*="Примерное время изучения (часы)", *null*=True, *blank*=True)

    class Meta:

        verbose\_name = "Детали концепции"

        verbose\_name\_plural = "Детали концепций"

    def \_\_str\_\_(*self*):

*return* f"Детали {*self*.concept.title}"

* **Связь многие-ко-многим (ManyToMany):**  
  Добавлено поле tags в модель **ComputerScienceConcept** для связи с моделью **Tag:**

class Tag(models.Model):

    name = models.CharField(*max\_length*=50, *unique*=True, *verbose\_name*="Тег")

    slug = models.SlugField(*max\_length*=50, *unique*=True, *db\_index*=True, *verbose\_name*="URL тега", *default*='default-slug') *# Добавляем slug для тегов*

    class Meta:

        verbose\_name = "Тег"

        verbose\_name\_plural = "Теги"

    def \_\_str\_\_(*self*):

*return* *self*.name

    def get\_absolute\_url(*self*):

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

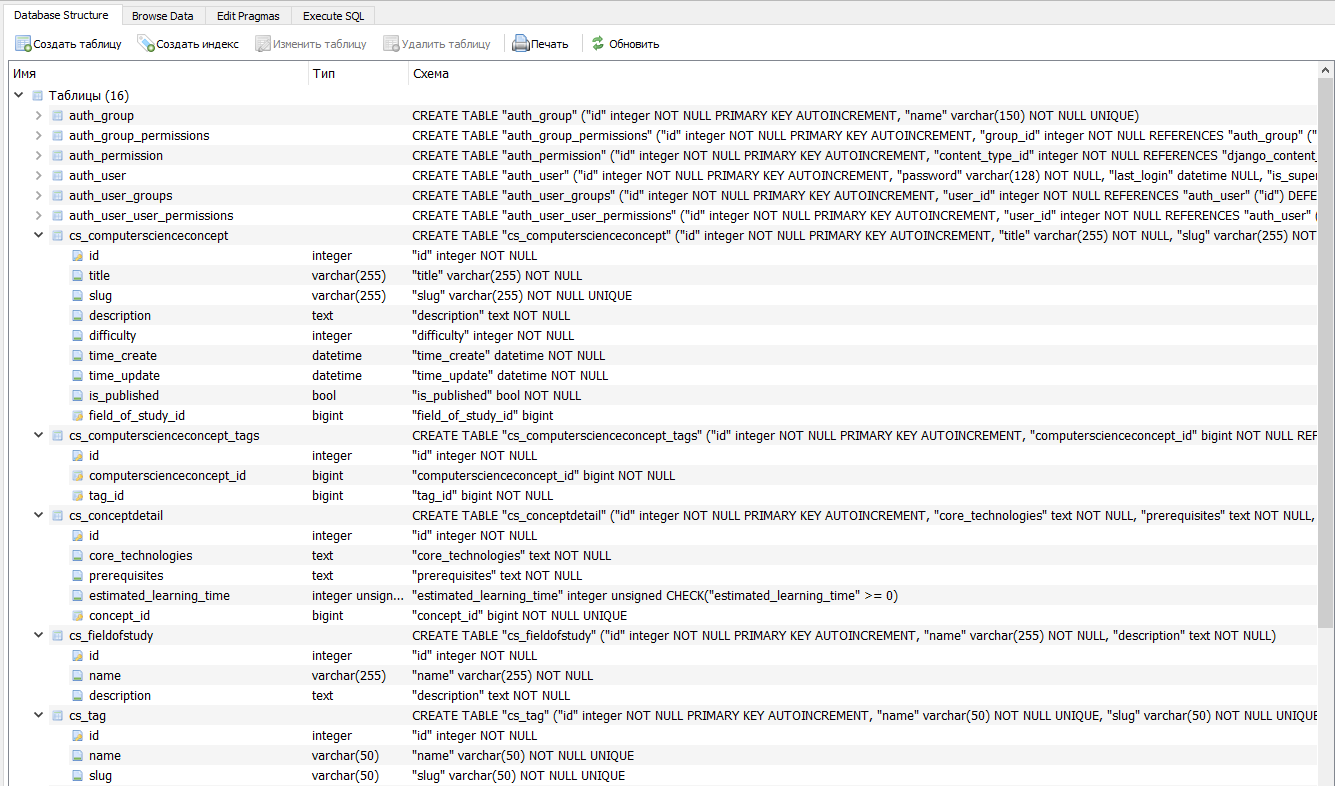


Рисунок 1 – Структура новых таблиц.

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

использованием тегов.

 **Модель Tag:**  
Модель **Tag** определена в файле **Cs\_ty/cs/models.py** .

class Tag(models.Model):

    name = models.CharField(*max\_length*=50, *unique*=True, *verbose\_name*="Тег")

    slug = models.SlugField(*max\_length*=50, *unique*=True, *db\_index*=True, *verbose\_name*="URL тега", *default*='default-slug') *# Добавляем slug для тегов*

    class Meta:

        verbose\_name = "Тег"

        verbose\_name\_plural = "Теги"

    def \_\_str\_\_(*self*):

*return* *self*.name

    def get\_absolute\_url(*self*):

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

**Связь Many-to-Many с концепциями:**  
Поле tags в модели **ComputerScienceConcept** позволяет привязывать несколько тегов к одной концепции.

 **Представление для тегов:**В файле**Cs\_ty/cs/views.py** добавлена функция tags\_list, которая выбирает все теги и передаёт их в шаблон.

def tags\_list(*request*):

    tags = Tag.objects.all()

    data = {

        'title': 'Теги',

        'tags': tags,

    }

*return* render(*request*, 'cs/tags\_list.html', *context*=data)

 **Шаблон для отображения тегов:**Создан шаблон для вывода тегов.  
**Путь:** Cs\_ty/cs/templates/cs/tags\_list.html

{% extends 'cs/base.html' %}

{% block title %}{{ title }}{% endblock %}

{% block content %}

  <h2>Список тегов</h2>

  <ul>

    {% for tag in tags %}

      <li><a href="{{ tag.get\_absolute\_url }}">{{ tag.name }}</a></li>

    {% endfor %}

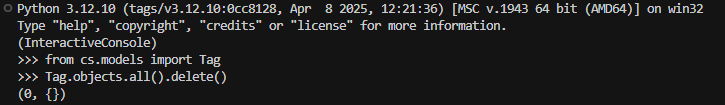
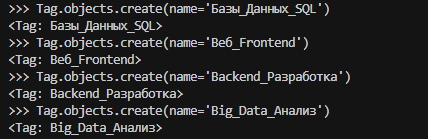
  </ul>

{% endblock %}

 **Маршрут для тегов:**  
В файле **Cs\_ty/cs/urls.py** добавлен URL для представления тегов:

path('tags/', views.tags\_list, *name*='tags\_list'),

Теперь создадим тестовые теги и проверим работоспособность.

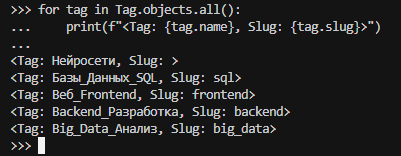


Рисунок 2 – Создание тестовых тегов.

Так же проверим записи в базе данных.

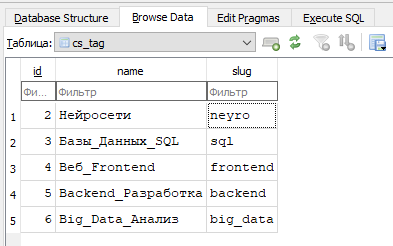


Рисунок 3 – Тестовые теги в базе данных.

И на сайте:

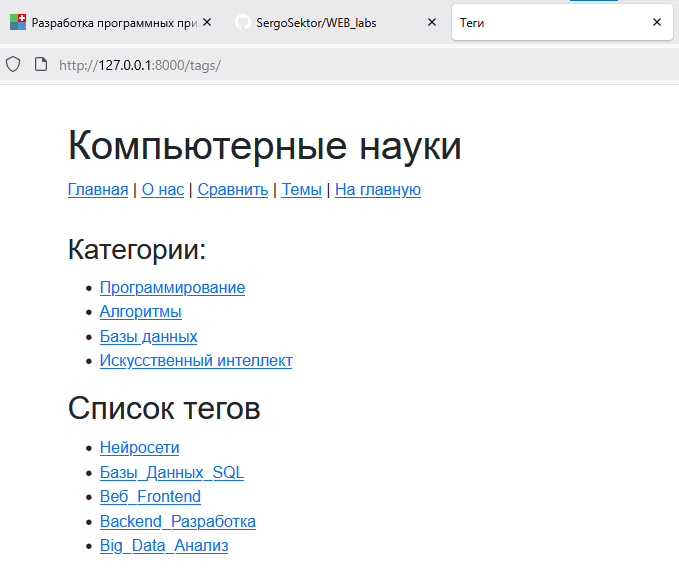


Рисунок 4 – Тестовые теги на сайте.

3. Использовать в программе описанные методы выбора записей,

классы Q, F и Value, вычисляемые поля, агрегирующие функции,

группировку записей, вычисления на стороне СУБД.

#### 3.1. Класс Q – составные условия фильтрации

Класс Q позволяет комбинировать условия с логическими операциями (AND, OR, NOT). Это особенно полезно, когда нужно построить запрос с условием «ИЛИ» или сложными составными фильтрами.

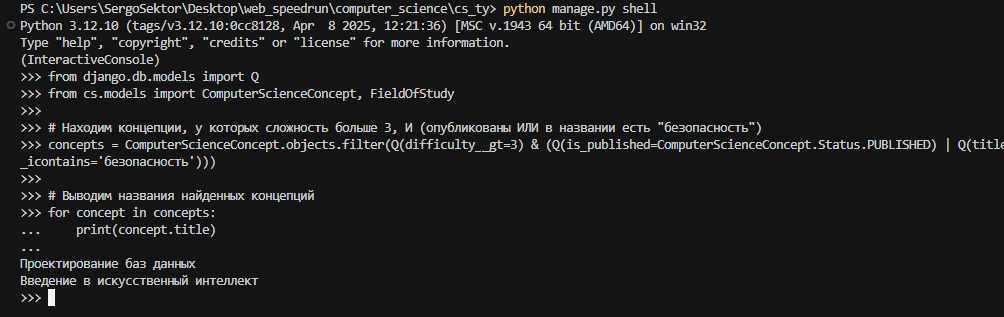


Рисунок 5 - Использование класса Q.

Этот запрос вернет "Введение в искусственный интеллект" (так как difficulty 4 > 3 и is\_published = 1) и "Основы кибербезопасности" (так как difficulty 5 > 3 и в названии есть "безопасность").

#### 3.2. Класс F – ссылки на значения полей

Выражения F позволяют ссылаться на поля модели в запросах, что полезно для сравнений между полями или вычисляемых полей.

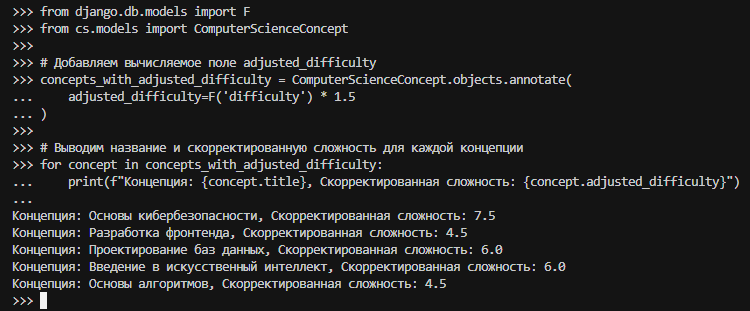


Рисунок 6 – Использование класса F.

Этот запрос выведет список всех концепций с их исходным названием и новым вычисляемым полем adjusted\_difficulty. Например, если у "Основы алгоритмов" сложность 3, то adjusted\_difficulty будет 4.5.

#### 3.3. Функция Value – добавление константных значений

Функция Value позволяет добавить в запрос постоянное значение как вычисляемое поле. Это полезно для создания вычисляемых полей, которые не зависят от значений других столбцов.

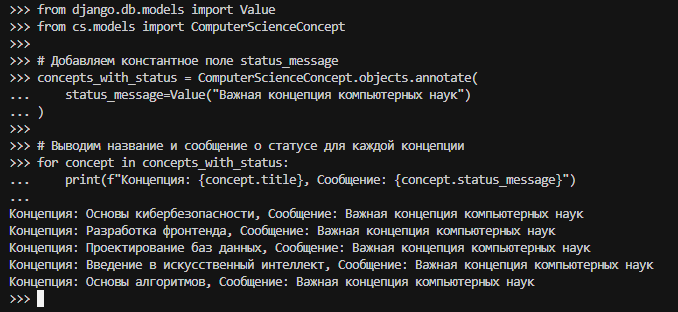


Рисунок 7 – Функция value.

В каждом объекте QuerySet появляется дополнительное поле status\_message с постоянным значением "Важная концепция компьютерных наук".

3.4. Агрегирующие функции и группировка записей

Агрегирующие функции позволяют выполнять вычисления над группами записей (например, суммирование, подсчёт, нахождение среднего значения).

Примеры:

* Подсчёт количества концепций:

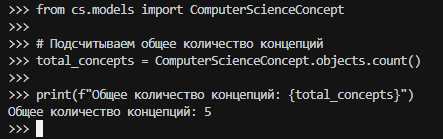


Рисунок 8 – Функция count.

* Средняя сложность концепций:

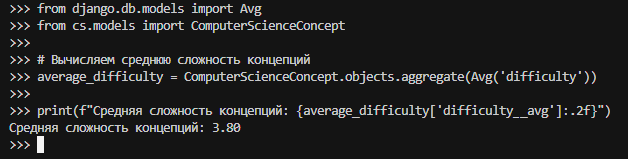


Рисунок 9 – Функция Avg.

Группировка по областям науки с подсчётом концепций:

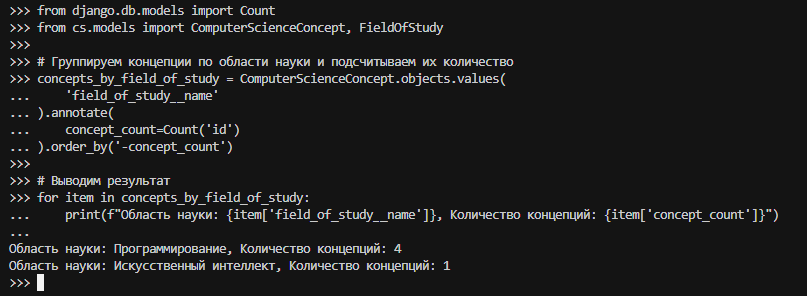


Рисунок 10 – Функция count.

Каждая группа по областям науки будет содержать подсчитанное количество концепций, отсортированное по убыванию.

Группировка по тегам:

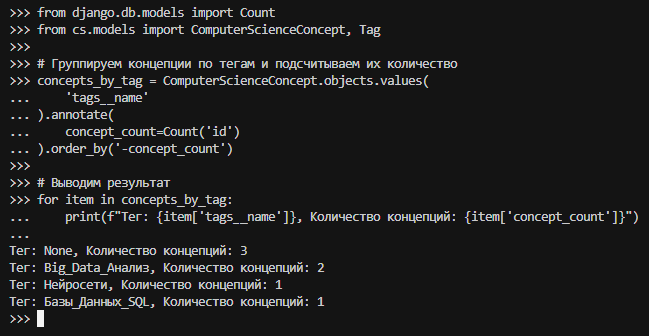


Рисунок 11 – Группировка по тегам.

#### 3.5. Вычисляемые поля

Аннотации (annotate) позволяют добавить в QuerySet вычисляемые поля, которые рассчитываются на стороне базы данных. Это улучшает производительность при работе с большим объёмом данных.

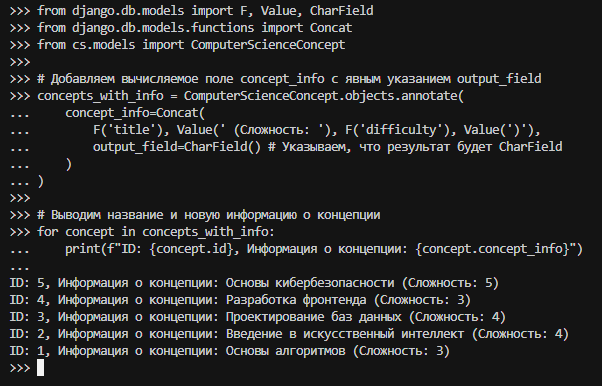


Рисунок 12 – Использование аннотации.

Листинг программ

Файл urls.py:

*from* django.urls *import* path

*from* . *import* views

*from* . *import* converters

*from* django.urls *import* register\_converter

register\_converter(converters.FourDigitYearConverter, 'year4')

urlpatterns = [

    path('', views.index, *name*='home'),           *# Главная страница*

    path('about/', views.about, *name*='about'),      *# О нас*

    path('compare/', views.compare, *name*='compare'), *# Сравнение*

    path('cs/', views.cs\_list, *name*='cs'),      *# Список тем*

    path('cs/<int:topic\_id>/', views.cs\_detail, *name*='cs\_detail'),  *# Детальная страница темы*

    path('concept/<slug:concept\_slug>/', views.cs\_detail, *name*='concept\_detail'),  *# Детальная страница концепции*

    path('archive/<year4:year>/', views.archive, *name*='archive'),

    path('go-home/', views.redirect\_example, *name*='go\_home'),

    path('hello/', views.get\_params\_example, *name*='hello'),

    path('category/<int:cat\_id>/', views.category, *name*='category'),

    path('tags/', views.tags\_list, *name*='tags\_list'),

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

]

handler404 = 'cs.views.custom\_404'

Файл views.py

*from* django.shortcuts *import* render, redirect, get\_object\_or\_404

*from* django.http *import* Http404, HttpResponse

*from* .models *import* ComputerScienceConcept, Tag *# Импортируем нашу новую модель и Tag*

def index(*request*):

    concepts = ComputerScienceConcept.published.all()  *# Получаем только опубликованные концепции*

    data = {

        'title': 'Главная страница',

        'concepts': concepts, *# Передаем концепции в шаблон*

    }

*return* render(*request*, 'cs/index.html', *context*=data)

def about(*request*):

*return* render(*request*, 'cs/about.html')

def compare(*request*):

*return* render(*request*, 'cs/compare.html')

def cs\_list(*request*):

    all\_cs\_topics = [

        {'id': 1, 'name': 'Искусственный интеллект', 'description': 'Область компьютерных наук, изучающая методы создания интеллектуальных агентов.'},

        {'id': 2, 'name': 'Машинное обучение', 'description': 'Подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться на данных.'},

        {'id': 3, 'name': 'Веб-разработка', 'description': 'Процесс создания веб-сайтов и веб-приложений.'},

        {'id': 4, 'name': 'Базы данных', 'description': 'Организованная коллекция данных, предназначенная для эффективного хранения и извлечения.'},

    ]

*return* render(*request*, 'cs/cs\_list.html', {'cs\_topics': all\_cs\_topics})

def cs\_detail(*request*, *concept\_slug*):

    concept = get\_object\_or\_404(ComputerScienceConcept, *slug*=*concept\_slug*)

    data = {

        'title': concept.title,

        'concept': concept,

    }

*return* render(*request*, 'cs/concept\_detail.html', *context*=data)

def tags\_list(*request*):

    tags = Tag.objects.all()

    data = {

        'title': 'Теги',

        'tags': tags,

    }

*return* render(*request*, 'cs/tags\_list.html', *context*=data)

def concepts\_by\_tag(*request*, *tag\_slug*):

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

    concepts = ComputerScienceConcept.published.filter(*tags*=tag)

    data = {

        'title': f'Концепции по тегу: {tag.name}',

        'concepts': concepts,

        'tag': tag,

    }

*return* render(*request*, 'cs/concepts\_by\_tag.html', *context*=data)

def archive(*request*, *year*):

*if* *year* > 2025:

*raise* Http404("Нет данных за этот год")

*return* HttpResponse(f"<h1>Архив за {*year*} год</h1>")

def redirect\_example(*request*):

*return* redirect('home')

def get\_params\_example(*request*):

    name = *request*.GET.get('name', 'Гость')

*return* HttpResponse(f"<h1>Привет, {name}!</h1>")

def category(*request*, *cat\_id*):

*return* HttpResponse(f"<h1>Страница категории</h1><p>ID категории: {*cat\_id*}</p>")

Файл model.py:

*from* django.db *import* models

*from* django.urls *import* reverse

*from* django.utils.text *import* slugify *# Импортируем slugify*

*# Модель для областей компьютерных наук*

class FieldOfStudy(models.Model):

    name = models.CharField(*max\_length*=255, *verbose\_name*="Область науки")

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

    class Meta:

        verbose\_name = "Область науки"

        verbose\_name\_plural = "Области науки"

    def \_\_str\_\_(*self*):

*return* *self*.name

*# Пользовательский менеджер для выборки только опубликованных записей*

class PublishedManager(models.Manager):

    def get\_queryset(*self*):

*# Фильтруем записи по полю публикации с использованием перечисления*

*return* super().get\_queryset().filter(*is\_published*=ComputerScienceConcept.Status.PUBLISHED)

class ComputerScienceConcept(models.Model):

*# Класс-перечисление для статуса публикации*

    class Status(models.IntegerChoices):

        DRAFT = 0, 'Черновик'

        PUBLISHED = 1, 'Опубликовано'

    title = models.CharField(*max\_length*=255, *verbose\_name*="Название концепции")

    slug = models.SlugField(*max\_length*=255, *unique*=True, *db\_index*=True, *verbose\_name*="URL")

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

    difficulty = models.IntegerField(default=1, *verbose\_name*="Сложность (от 1 до 5)") *# Новое поле для сложности*

    time\_create = models.DateTimeField(*auto\_now\_add*=True, *verbose\_name*="Время создания")

    time\_update = models.DateTimeField(*auto\_now*=True, *verbose\_name*="Время обновления")

    is\_published = models.BooleanField(*choices*=Status.choices, default=Status.DRAFT, *verbose\_name*="Публикация")

*# Связь один-ко-многим с FieldOfStudy*

    field\_of\_study = models.ForeignKey(

        FieldOfStudy,

*on\_delete*=models.CASCADE,

*related\_name*='concepts',

*verbose\_name*="Область науки",

*null*=True,

*blank*=True

    )

    objects = models.Manager()         *# Стандартный менеджер*

    published = PublishedManager()       *# Пользовательский менеджер для опубликованных записей*

    class Meta:

        ordering = ['-time\_create']

        indexes = [models.Index(*fields*=['-time\_create'])]

        verbose\_name = "Концепция компьютерных наук"

        verbose\_name\_plural = "Концепции компьютерных наук"

    def \_\_str\_\_(*self*):

*return* *self*.title

    def get\_absolute\_url(*self*):

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

*# Модель для расширенной информации о концепции (OneToOne)*

class ConceptDetail(models.Model):

    concept = models.OneToOneField(

        ComputerScienceConcept,

*on\_delete*=models.CASCADE,

*related\_name*='detail',

*verbose\_name*="Концепция"

    )

    core\_technologies = models.TextField(*blank*=True, *verbose\_name*="Ключевые технологии")

    prerequisites = models.TextField(*blank*=True, *verbose\_name*="Предварительные условия")

    estimated\_learning\_time = models.PositiveIntegerField(*verbose\_name*="Примерное время изучения (часы)", *null*=True, *blank*=True)

    class Meta:

        verbose\_name = "Детали концепции"

        verbose\_name\_plural = "Детали концепций"

    def \_\_str\_\_(*self*):

*return* f"Детали {*self*.concept.title}"

*# Модель для тегов (Many-to-Many)*

class Tag(models.Model):

    name = models.CharField(*max\_length*=50, *unique*=True, *verbose\_name*="Тег")

    slug = models.SlugField(*max\_length*=50, *unique*=True, *db\_index*=True, *verbose\_name*="URL тега") *# Убрал default='default-slug'*

    class Meta:

        verbose\_name = "Тег"

        verbose\_name\_plural = "Теги"

    def \_\_str\_\_(*self*):

*return* *self*.name

    def save(*self*, \**args*, \*\**kwargs*):

*if* not *self*.slug:

*self*.slug = slugify(*self*.name)

        super().save(\**args*, \*\**kwargs*)

    def get\_absolute\_url(*self*):

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

*# Добавляем ManyToManyField в ComputerScienceConcept после определения Tag*

ComputerScienceConcept.add\_to\_class('tags', models.ManyToManyField(

    Tag,

*related\_name*='concepts',

*verbose\_name*="Теги",

*blank*=True

))

Файл tags\_list.html:

{% extends 'cs/base.html' %}

{% block title %}{{ title }}{% endblock %}

{% block content %}

  <h2>Список тегов</h2>

  <ul>

    {% for tag in tags %}

      <li><a href="{{ tag.get\_absolute\_url }}">{{ tag.name }}</a></li>

    {% endfor %}

  </ul>

{% endblock %}

Вывод

В ходе лабораторной работы было выполнено расширение структуры базы данных проекта. Были созданы новые модели для представления областей компьютерных наук (FieldOfStudy), дополнительных деталей концепции (ConceptDetail) и тегов (Tag). В модели ComputerScienceConcept добавлены связи: ForeignKey к FieldOfStudy, OneToOneField к ConceptDetail и ManyToManyField к Tag, что позволило моделировать все виды отношений между таблицами. Эти изменения подтвердились появлением соответствующих таблиц и связей в базе данных (например, таблицы cs\_fieldofstudy, cs\_conceptdetail, cs\_tag и промежуточной таблицы для тегов).

Также на сайте добавлен функционал работы с тегами. Реализовано представление и шаблон для вывода списка тегов, доступное по специальному маршруту. Пользователь может перейти на страницу, где отображаются все созданные теги.

Для работы с данными использованы продвинутые методы Django ORM. С помощью класса Q были построены составные фильтры. С использованием F-выражений и функции Concat были добавлены вычисляемые поля (например, поле adjusted\_difficulty — скорректированная сложность, равная сложности, умноженной на 1.5, а также concept\_info, объединяющее название и сложность). Применены агрегирующие функции и группировка записей для подсчёта количества концепций по областям науки и по тегам. Это позволяет выполнять вычисления и агрегацию данных непосредственно на стороне базы данных, улучшая производительность приложения.

В итоге, лабораторная работа продемонстрировала успешное расширение базы данных, установление связей между таблицами, интеграцию тегов и применение сложных запросов для динамического анализа данных. Результаты работы подтверждаются визуально через менеджеры баз данных и страницы сайта, а также успешно выполненными примерами в Django Shell.