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 №7**

**на тему:**

**Подключение базы данных к проекту**

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

Выполнили работу:

Студенты гр. АП-226, АВТФ

Ищенко С. Г.

«31» марта 2025 г.

Проверил работу:

Эстрайх И. В.

«\_\_\_» \_\_\_\_\_\_ 20\_\_г

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

(подпись)

**Цель**

Изучить принципы подключения и использования в проекте базы

данных.

**Задание**

1. Создать таблицу БД, заполнить несколькими записями с помощью

фреймворка.

2. Зайти в БД с помощью менеджера DB Browser for SQLite и убедиться, что

данные в таблице действительно появились

3. Обеспечить наполнение страницы информацией из базы данных.

4. Реализовать работу с записями БД из своей программы: создание,

изменение, удаление, выборку, фильтрацию, сортировку. Проверить,

что все эти операции работают корректно.

5. Добавить в БД слаги и обеспечить отображение записей по их слагу.

6. Создать пользовательский менеджер модели.

7. Использовать в программе класс перечисления.

**Описание хода выполнения работы**

1. Создать таблицу БД, заполнить несколькими записями с помощью

фреймворка.

В файле Cs\_ty/cs/models.py была создана модель ComputerScienceConcept, которая описывает таблицу для хранения информации об концепциях.

Определены поля:

* title – название концепции,
* slug – уникальное поле для формирования удобного URL,
* description – описание концепции,
* difficulty – цена,
* time\_create и time\_update – время создания и обновления записи,
* is\_published – флаг публикации.

В модели используется вложенный класс Status, наследуемый от models.IntegerChoices, для понятного задания состояний (черновик или опубликовано).

Также был создан пользовательский менеджер PublishedManager (назначенный в атрибуте published), который возвращает только опубликованные записи.

Файл models.py:

*from* django.db *import* models

*from* django.urls *import* reverse

*# 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")

    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})

Затем нужно было выставить миграции командами:

python3 manage.py makemigrations cs

python3 manage.py migrate

После выполнения команд в базе данных появляется таблица cs\_computerscienceconcept с нужной структурой.

В Django Shell (с помощью команды python3 manage.py shell) добавлены несколько записей в таблицу с помощью метода create(), а также проведены другие действия.

*from* cs.models *import* ComputerScienceConcept

*# Создаем три новые концепции*

concept1 = ComputerScienceConcept.objects.create(

*title*="Основы алгоритмов",

*slug*="osnovy-algoritmov",

*description*="Введение в базовые алгоритмы и структуры данных.",

*difficulty*=3,

*is\_published*=ComputerScienceConcept.Status.PUBLISHED

)

print(f"Создана концепция: {concept1.title}")

concept2 = ComputerScienceConcept.objects.create(

*title*="Введение в искусственный интеллект",

*slug*="vvedenie-v-iskusstvennyj-intellekt",

*description*="Обзор ключевых концепций и методов ИИ.",

*difficulty*=4,

*is\_published*=ComputerScienceConcept.Status.PUBLISHED

)

print(f"Создана концепция: {concept2.title}")

concept3 = ComputerScienceConcept.objects.create(

*title*="Базы данных",

*slug*="bazy-dannyh",

*description*="Организованная коллекция данных, предназначенная для эффективного хранения и извлечения.",

*difficulty*=2,

*is\_published*=ComputerScienceConcept.Status.DRAFT

)

print(f"Создана концепция: {concept3.title} (как черновик)")

*# Выбираем концепцию по ее слагу*

print("\n--- ИЗМЕНЕНИЕ ЗАПИСИ ---")

concept\_to\_update = ComputerScienceConcept.objects.get(*slug*="osnovy-algoritmov")

print(f"Исходная сложность '{concept\_to\_update.title}': {concept\_to\_update.difficulty}")

*# Изменяем ее сложность*

concept\_to\_update.difficulty = 4

*# Сохраняем изменения*

concept\_to\_update.save()

print(f"Новая сложность '{concept\_to\_update.title}': {concept\_to\_update.difficulty}")

*# Выбираем концепцию, которую хотим удалить (например, "Базы данных")*

print("\n--- УДАЛЕНИЕ ЗАПИСИ ---")

concept\_to\_delete = ComputerScienceConcept.objects.get(*slug*="bazy-dannyh")

print(f"Удаляем концепцию: {concept\_to\_delete.title}")

*# Удаляем ее*

concept\_to\_delete.delete()

print(f"Концепция '{concept\_to\_delete.title}' успешно удалена.")

print("\n--- ФИЛЬТРАЦИЯ И СОРТИРОВКА ---")

*# Получаем все опубликованные концепции со сложностью больше или равной 3, отсортированные по сложности*

published\_and\_complex\_concepts = ComputerScienceConcept.published.filter(*difficulty\_\_gte*=3).order\_by('difficulty')

print("Опубликованные и сложные концепции (сложность >= 3, отсортированы по сложности):")

*for* concept *in* published\_and\_complex\_concepts:

    print(f"- {concept.title} (Сложность: {concept.difficulty}, Опубликовано: {concept.get\_is\_published\_display()})")

*# Получаем все концепции (включая черновики), отсортированные по времени создания*

all\_concepts = ComputerScienceConcept.objects.all().order\_by('-time\_create')

print("\nВсе концепции (отсортированы по времени создания):")

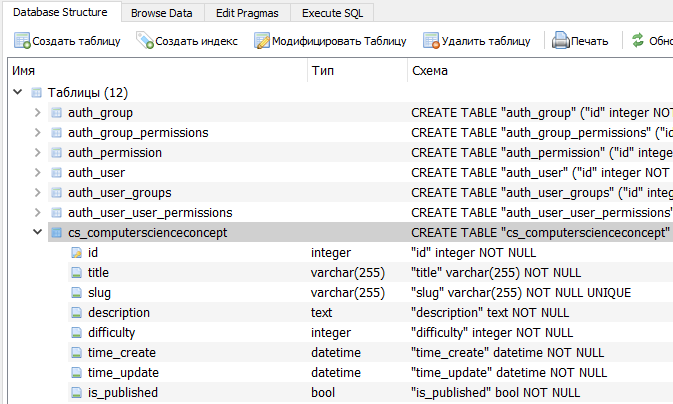
*for* concept *in* all\_concepts:

    print(f"- {concept.title} (Сложность: {concept.difficulty}, Опубликовано: {concept.get\_is\_published\_display()})")

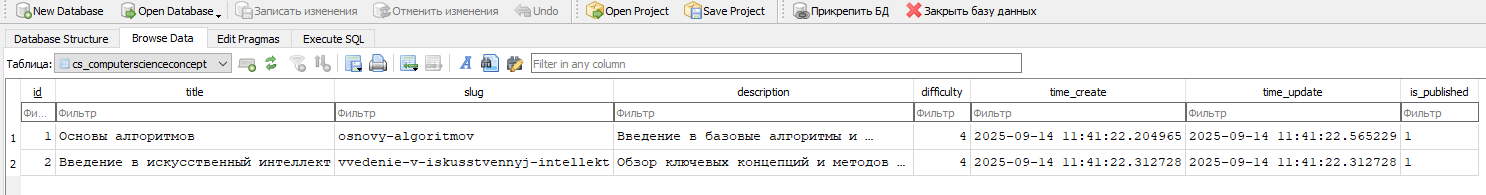
2. Зайти в БД с помощью менеджера DB Browser for SQLite и убедиться, что данные в таблице действительно появились

С помощью DB Browser for SQLite открыт файл db.sqlite3, расположенный в корневой папке проекта.

Проверена структура таблицы cs\_computerscienceconcept – убедились, что все поля (id, title, slug, description, difficulty, time\_create, time\_update, is\_published) созданы согласно модели.



Теперь убедимся что наполнение данными прошло успешно.



3. Обеспечить наполнение страницы информацией из базы данных.

В файле Cs\_ty/cs/views.py обновлена функция index, чтобы выбирать данные из базы с использованием менеджера published:

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)

В шаблоне Cs\_ty/cs/templates/cs/index.html добавлен код для динамического вывода данных с применением фильтров:

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

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

{% block heading %}Добро пожаловать в мир компьютерных наук!{% endblock %}

{% block content %}

  <h2>Наши концепции</h2>

  <ul class="list-articles">

      {% for concept in concepts %}

        <li>

          <h3>{{ concept.title }}</h3>

          <p>{{ concept.description|linebreaks|truncatewords:30 }}</p>

          <p>Сложность: {{ concept.difficulty }}</p>

          <a href="{{ concept.get\_absolute\_url }}">Подробнее</a>

        </li>

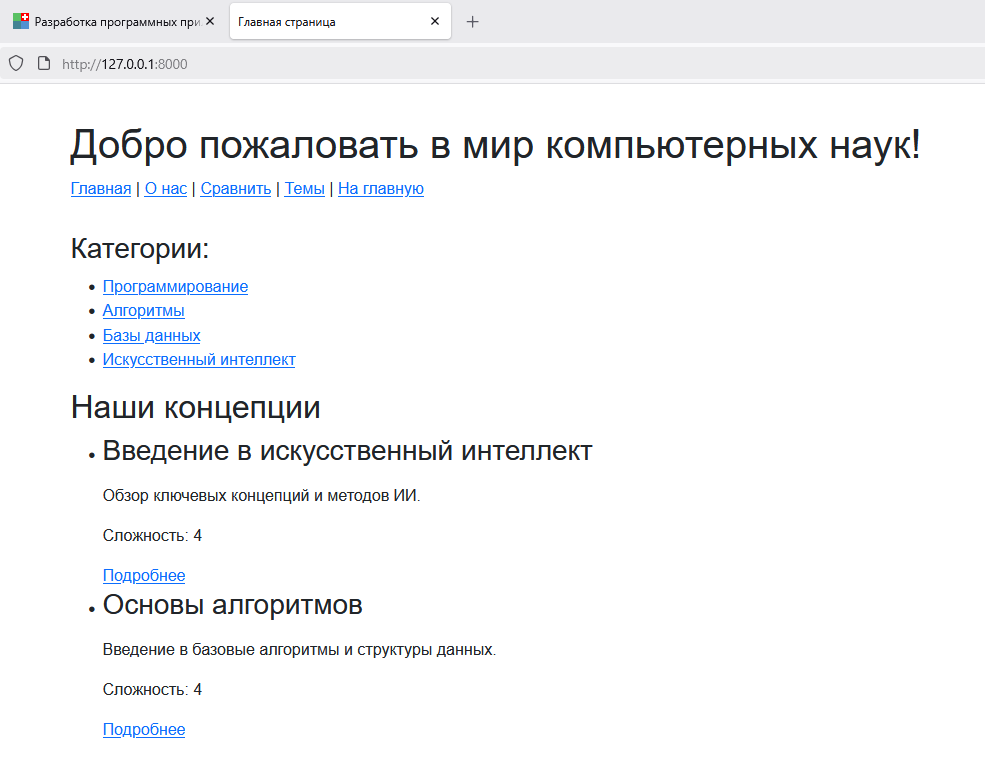
      {% endfor %}

  </ul>

{% endblock %}

* Передача данных через контекст позволяет динамически формировать страницу, отображая актуальные записи из БД.
* Фильтры (например, truncatewords и linebreaks) применяются для улучшения читаемости текста и корректного отображения форматирования.

**Результат:** Главная страница теперь отображает список опубликованных концепций, полученных из базы данных.

****

4. Реализовать работу с записями БД из своей программы: создание,

изменение, удаление, выборку, фильтрацию, сортировку. Проверить,

что все эти операции работают корректно.

В Django Shell проверены операции создания, изменения, удаления, выборки, фильтрации и сортировки:

* **Создание:** создание с помощью cs.objects.create(...)
* **Изменение:** получение записи через get(), изменение полей, вызов save()
* **Удаление:** получение записи и вызов метода delete()
* **Фильтрация и сортировка:** использование методов filter() и order\_by()

*from* cs.models *import* ComputerScienceConcept

*# Создание*

concept\_honda = ComputerScienceConcept.objects.create(

*title*="Основы криптографии",

*slug*="osnovy-kriptografii",

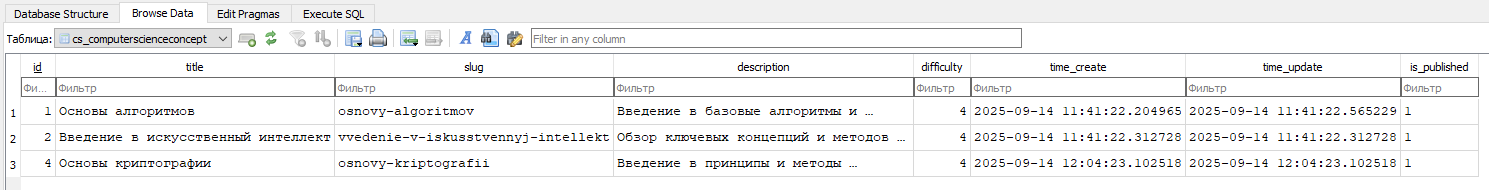
*description*="Введение в принципы и методы шифрования данных.",

*difficulty*=4,

*is\_published*=ComputerScienceConcept.Status.PUBLISHED

)

print(f"Создана концепция: {concept\_honda.title}")



*# Изменение*

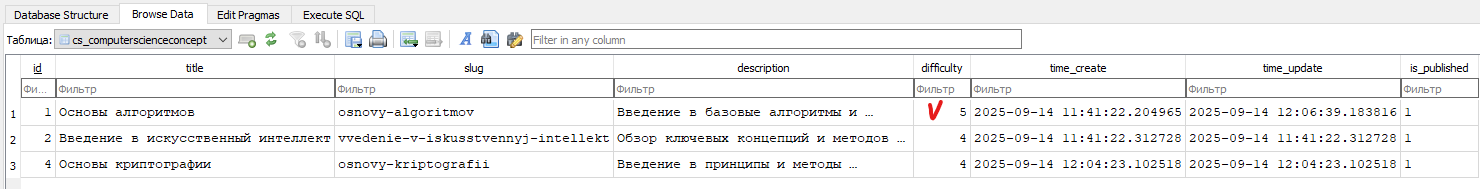
concept\_to\_update = ComputerScienceConcept.objects.get(*slug*="osnovy-algoritmov")

print(f"Исходная сложность '{concept\_to\_update.title}': {concept\_to\_update.difficulty}")

concept\_to\_update.difficulty = 5

concept\_to\_update.save()

print(f"Новая сложность '{concept\_to\_update.title}': {concept\_to\_update.difficulty}")

****

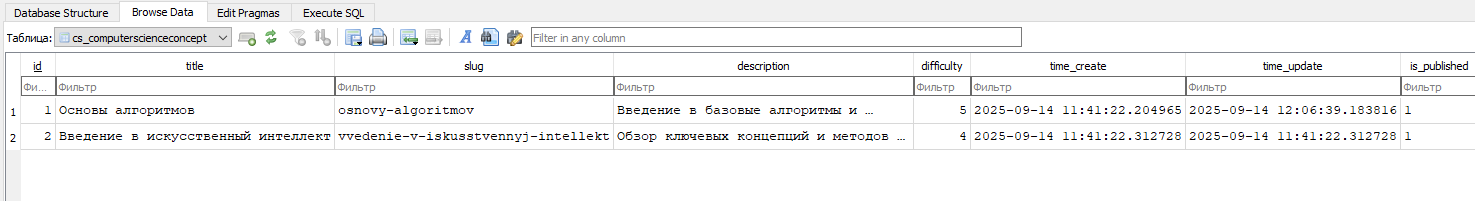
*# Удаление*

concept\_to\_delete = ComputerScienceConcept.objects.get(*slug*="osnovy-kriptografii")

print(f"Удаляем концепцию: {concept\_to\_delete.title}")

concept\_to\_delete.delete()

print(f"Концепция '{concept\_to\_delete.title}' успешно удалена.")

****

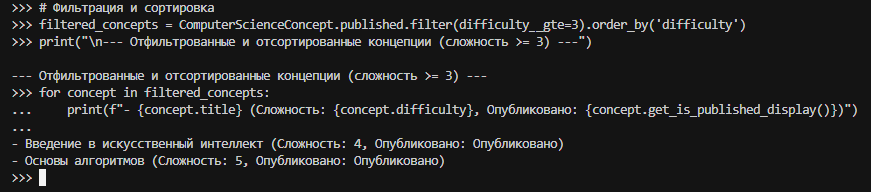
*# Фильтрация и сортировка*

filtered\_concepts = ComputerScienceConcept.published.filter(*difficulty\_\_gte*=3).order\_by('difficulty')

print("\n--- Отфильтрованные и отсортированные концепции (сложность >= 3) ---")

*for* concept *in* filtered\_concepts:

    print(f"- {concept.title} (Сложность: {concept.difficulty}, Опубликовано: {concept.get\_is\_published\_display()})")

****

Все операции работают корректно, что подтверждается результатами команд в Django Shell и отображением данных на страницах.

5. Добавить в БД слаги и обеспечить отображение записей по их слагу.

В модели ComputerScienceConcept уже добавлено поле slug для формирования читаемого URL.

Обновлена URL-конфигурация в файле Cs\_ty/cs/urls.py:

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

Представление для страницы деталей изменено в Cs\_ty/cs/views.py:

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)

Создан шаблон Cs\_ty/cs/templates/cs/concept\_detail.html:

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

{% block title %}{{ concept.title }}{% endblock %}

{% block content %}

  <h2>{{ concept.title }}</h2>

  <p>Сложность: {{ concept.difficulty }}</p>

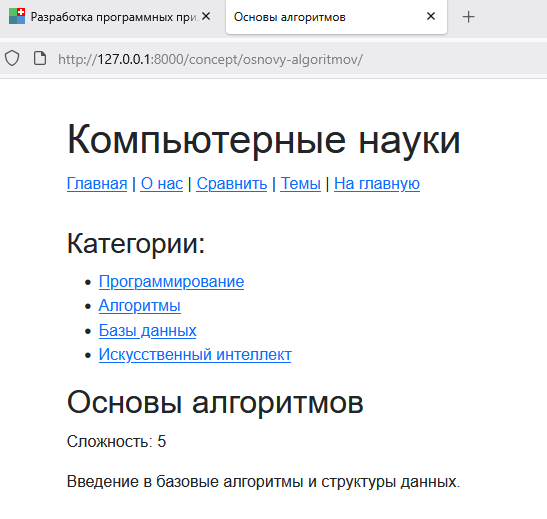
  <div>

      {{ concept.description|linebreaks }}

  </div>

{% endblock %}

Теперь записи можно просматривать по слагу, например, страница с деталями концепции будет доступна по адресу типа http://127.0.0.1:8000/concept/osnovy-algoritmov/.



### 7. Использование пользовательского менеджера и класса перечисления

В модели ComputerScienceConcept был добавлен вложенный класс Status, который наследуется от models.IntegerChoices. Этот класс задаёт понятные имена для статусов публикации:

Файл cs\_ty/cs\_ty/cs/models.py

class Status(models.IntegerChoices):

        DRAFT = 0, 'Черновик'

        PUBLISHED = 1, 'Опубликовано'

Это позволяет использовать именованные константы вместо числовых значений. Например, вместо того чтобы писать is\_published=1, мы можем использовать is\_published= ComputerScienceConcept.Status.PUBLISHED. Это делает код более читаемым и уменьшает вероятность ошибок.

**Пользовательский менеджер для выборки опубликованных записей**  
В том же файле мы создали пользовательский менеджер PublishedManager, который переопределяет метод get\_queryset(), чтобы возвращать только те записи, у которых поле is\_published установлено в значение ComputerScienceConcept.Status.PUBLISHED:

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)

Затем в модели ComputerScienceConcept мы добавили два менеджера:

Стандартный менеджер objects (если он нужен для работы с полными данными).

Пользовательский менеджер published, который позволяет получать только опубликованные записи:

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

published = PublishedManager()

**Использование метода get\_absolute\_url**  
Также в модели ComputerScienceConcept добавлен метод get\_absolute\_url(), который использует поле slug для формирования URL страницы с деталями концепции:

def get\_absolute\_url(*self*):

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

Этот метод позволяет в шаблонах использовать выражение

{{ computerscienceconcept.get\_absolute\_url }} для создания ссылок на страницу конкретного концепции.

**Использование в представлениях:**

* В файле **Cs\_ty/cs/views.py** мы применяем пользовательский менеджер для выборки только опубликованных концепций:

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)

Это гарантирует, что на главной странице будут отображаться только концепции, которые отмечены как "Опубликовано".

**Листинг программ**

models.py

*from* django.db *import* models

*from* django.urls *import* reverse

*# 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")

    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})

views.py

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

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

*from* .models *import* ComputerScienceConcept *# Импортируем нашу новую модель*

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 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>")

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'),

]

handler404 = 'cs.views.custom\_404'

index.html

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

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

{% block heading %}Добро пожаловать в мир компьютерных наук!{% endblock %}

{% block content %}

  <h2>Наши концепции</h2>

  <ul class="list-articles">

      {% for concept in concepts %}

        <li>

          <h3>{{ concept.title }}</h3>

          <p>{{ concept.description|linebreaks|truncatewords:30 }}</p>

          <p>Сложность: {{ concept.difficulty }}</p>

          <a href="{{ concept.get\_absolute\_url }}">Подробнее</a>

        </li>

      {% endfor %}

  </ul>

{% endblock %}

concept\_detail.html

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

{% block title %}{{ concept.title }}{% endblock %}

{% block content %}

  <h2>{{ concept.title }}</h2>

  <p>Сложность: {{ concept.difficulty }}</p>

  <div>

      {{ concept.description|linebreaks }}

  </div>

{% endblock %}

**Вывод**

В ходе лабораторной работы удалось выполнить:

* **Модель и таблица БД:**  
  Создана модель ComputerScienceConcept в cs/models.py с полями: title, slug, description, difficulty, time\_create, time\_update и is\_published. Миграции успешно создали таблицу в базе (db.sqlite3).
* **Наполнение БД:**  
  Через Django Shell добавлены тестовые записи с использованием метода create(). Проверка через DB Browser for SQlite показала, что записи появились в таблице.
* **Динамическое наполнение страниц:**  
  Представление index (в cs/views.py) выбирает опубликованные концепции через менеджер ComputerScienceConcept.published.all() и передаёт их в шаблон cs/index.html, где данные выводятся с использованием фильтров (truncatewords, linebreaks).
* **CRUD-операции:**  
  Операции создания, изменения, удаления, фильтрации и сортировки записей реализованы и протестированы через Django ORM.
* **Отображение по слагу:**  
  Добавлено поле slug и настроено отображение страницы деталей (в cs/views.py и cs/templates/cs/concept\_detail.html) по слагу, что обеспечивает удобочитаемые URL (например, /concept/osnovy-algoritmov/)
* **Пользовательский менеджер и класс перечисления:**  
  Пользовательский менеджер published позволяет получать только опубликованные записи, а вложенный класс Status (наследник models.IntegerChoices) задаёт понятные значения для публикации.

Таким образом, лабораторная работа выполнена: база данных подключена, данные динамически выводятся на страницах, реализованы все базовые операции с записями и обеспечено удобное отображение через слаги.