

Министерство цифрового развития, связи и массовых коммуникаций  
Российской Федерации  
Ордена Трудового Красного Знамени  
федеральное государственное бюджетное  
образовательное учреждение высшего образования  
«Московский технический университет связи и информатики»

Кафедра МКиИТ  
Проектирование клиент-серверных приложений

Лабораторная работа №3  
“Создание первой модели данных и её регистрация в административном  
приложении Django”

Выполнил:  
студент 3 курса,  
группы БФИ2001  
Лушин Е. А.

Москва 2023

**Цель работы:** научиться создавать модели данных и регистрировать их в Django, а также динамически генерировать шаблон для вывода данных из БД.

**Задание:**

Создание первой модели данных и её регистрация в административном приложении Django

- Когда модель зарегистрирована, можно создать новые записи, достаточно перейти в административную панель, находящуюся по адресу (<http://127.0.0.1:8000/admin/>). Перейдите по вкладке Articles и кликнув по кнопке “Add article” справа наверху. Заполнив поля Title, Author и Text значениями по своему усмотрению. Создать ещё две статьи подобным образом.
- С помощью программы управления базами данных sqlite3 (например, программы SQLiteManager) открыть файл базы данных текущего проекта, который хранится в папке проекта с именем, объявленным в настройках проекта в переменной «**DATABASES.NAME**». Найти созданные в предыдущем пункте задания экземпляры записей. Изменить текст одной записи и название статьи для другой.

Динамическое генерирование шаблона для вывода всех экземпляров этой модели

- Вновь открыть файл базы данных **sqlite3**, где хранятся экземпляры статей к текущему проекту. Добавить новую запись в блог непосредственно через менеджер базы данных (не забыть зафиксировать транзакцию, иными словами, сохранить все внесённые изменения).

## **Краткая теория**

### **Модели в Django**

Логика современных веб-приложений часто требует обращения к базе данных. Такой управляемый данными сайт подключается к серверу базы данных, получает от него данные и отображает их на странице.

Некоторые сайты позволяют посетителям пополнять базу данных. Многие развитые сайты сочетают обе возможности. Например, Amazon.com — прекрасный пример сайта, управляемого данными. Страница каждого продукта представляет собой результат запроса к базе данных Amazon, отформатированный в виде HTML, а отправленный вами отзыв сохраняется в базе данных.

Django отлично подходит для создания управляемых данными сайтов, поскольку включает простые и вместе с тем мощные средства для выполнения запросов к базе данных из программы на языке Python.

Чтобы лучше понимать, что из себя представляют модели, сначала попробуйте решить следующую задачу: у вас есть некое хранилище данных в виде таблицы (пусть это будет всем знакомый файл для программы Microsoft Excel) и вам нужно сначала на языке, подобном Pythonу, как-то определить структуру вашего файла, в котором есть несколько таблиц или листов, а затем программно как-то читать данные с минимальным количеством кода. Здесь немаловажной деталью является тот факт, что структура таблиц должна быть описана именно на языке программирования и внедрена самим скриптом в файл, а не наоборот, когда пользователь самостоятельно задает содержимое, а программа читает содержимое файла и распознаёт, какие строки есть в таблице. У последнего замечания (которое, естественно, имеет прямой аналог в реальной системе моделей Django) есть несколько оснований для существования:

- При работе с реальными таблицами баз данных скорость прочтения таблиц и разбор структуры всей БД будет занимать довольно много времени.
- Если сначала описывать структуру в таблице, а потом читать её из скрипта, программисту придется работать в двух реальностях, которые очень сильно отличаются друг от друга (как справедливо замечают создатели Django, «писать на Python вообще приятно, и если представлять все на этом языке, то сокращается количество мысленных

«переключений контекста». Чем дольше разработчику удастся оставаться в рамках одного программного окружения и менталитета, тем выше его продуктивность. Когда приходится писать код на SQL, потом на Python, а потом снова на SQL, продуктивность падает»).

- Высокоуровневые типы данных повышают продуктивность и степень повторной используемости кода. Так, в большинстве СУБД нет специального типа данных для представления адресов электронной почты или URL. А в моделях Django это возможно, потому что стоит лишь добавить пару строчек кода на Python, и обычное текстовое поле превращается в узконаправленное поле, в которое можно поместить только адрес URL, а если попытаться что-то некорректное, то в процессе сохранения просто напроосто возникнет исключение.

Итак, решением, возможно, стало бы создать словарь, где в качестве ключей шли бы названия столбцов, а по каждому ключу хранился бы массив данных в этих данных. Однако такой вариант не очень удобен, потому что постоянно бы пришлось синхронизировать количество элементов в массивах, которые расположены по разным ключам.

Чтобы создать по такому принципу таблицу 1 вам пришлось бы создать примерно такой словарь:

```
{  
    "name": ["George", "Mike", "Steve"],  
    "age": [32, 25, 43],  
    "occupation": ["doctor", "lawyer", "engineer"]  
}
```

Таблица 1 – Список пользователей

Имя	Возраст	Профессия
George	32	Doctor
Mike	25	Lawyer
Steve	43	Engineer

Можно было бы применить другой подход: сделать словарь, где каждый ключ — опять же название столбца, и по этому ключу хранится ещё один, вложенный, словарь, в котором каждый ключ — это уже название строки. Однако и такой вариант не очень удобен, ведь пришлось бы для каждого столбца заново прописывать название строки.

Другой пример словаря с данными:

```
{  
    "age": {"George": 32, "Mike": 25, "Steve": 43},  
    "occupation": {"George": "doctor", "Mike": "lawyer",  
"Steve": "engineer"},  
}
```

Тут из-за того, что имена и так хранятся во всех словарях, можно не описывать одно из полей.

И чтобы добавить нового пользователя, в любом случае пришлось бы по каждому из столбцов добавлять новое значение, не слабая такая задача!

Создатели Django пошли по немного другому пути. Они предложили каждую таблицу в БД описывать через класс данных. Вы уже знаете, что в языке Python существуют такие классы данных (класс и тип данных по сути одно и то же), как строка, число, список и другие. Однако в языке также существует возможность создавать свои собственные типы данных, которые могут быть как очень похожими, так и принципиально отличными от какого-либо встроенного типа. Для создания класса достаточно воспользоваться ключевым словом `class`, после которого нужно указать имя класса и в скобках отметить, от какого типа данных текущий класс должен наследоваться.

Пример создания модели для Django:

```
from django.db import models  
  
class Article(models.Model):  
    title = models.CharField(max length=200)  
    text = models.TextField()
```

С помощью таких трех строк кода была описана целая таблица в базе данных, которая имеет два столбца — title и text. Так как здесь производится наследование от уже написанного создателями Django класса models.Model, у вашего типа данных будет очень богатый функциональный набор по работе с базой данных. Например, чтобы создать новую запись вам будет достаточно выполнить следующую строку кода (пример создания нового экземпляра модели):

```
new article = Article.objects.create(title="Новая статья", text="С интересным текстом")
```

Теперь в вашей таблице появится новая строка, которая будет содержать данные о статье «Новая статья». Также для того, чтобы вы могли далее оперировать этой строкой, вы можете сохранить данные этой строки в переменной, которая в данном случае называется new article. Например, можно изменить значение столбца text таким образом:

Пример изменения существующего экземпляра модели:

```
new article.text = "Теперь у статьи будет новый текст"  
new article.save()
```

Чтобы данные изменились не только в переменной Python, но и в базе данных, нужно вызвать метод save().

Чтобы получить данные о всех записях в таблице, достаточно выполнить следующую строку кода (пример получения всех экземпляров модели Article):

```
all articles = Article.objects.all()
```

А если помимо получения всех записей нужно ещё и отсортировать данные (пример получения всех экземпляров модели Article и их сортировки по названию):

```
all articles by title = Article.objects.order by('title')
```

Также от models.Model ваш класс унаследует метод filter(), который позволяет выбрать только те данные, что вам нужны.

Пример получения экземпляров модели Article, у которых название «Новая статья»:

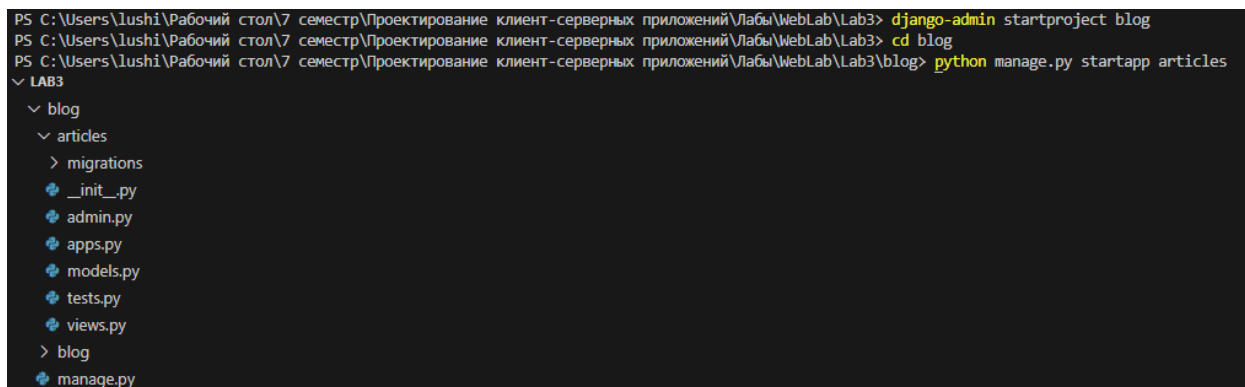
```
some articles = Article.objects.filter(title="Новая статья")
```

Если по заданному критерию в вашей базе данных есть несколько статей, то вернутся все они в формате QuerySet, который по поведению очень напоминает список, только он наделен многими дополнительными возможностями. Например, можно и сам объект QuerySet отфильтровать и отсортировать, причем таким же образом, как было показано ранее:

```
some articles = Article.objects.filter(title="Новая статья")  
# теперь some articles — объект QuerySet  
some articles = some articles.order by('text')  
# и у этого объекта есть такие же методы same articles = some  
articles.filter(text="Какой-то текст статьи")
```

## Выполнение

Создадим новую директорию для лабораторной работы №3 и создадим новый проект blog, выполнив команды как показано на рисунке 1.



```
PS C:\Users\lushi\Рабочий стол\7 семестр\Проектирование клиент-серверных приложений\Лабы\WebLab\Lab3> django-admin startproject blog  
PS C:\Users\lushi\Рабочий стол\7 семестр\Проектирование клиент-серверных приложений\Лабы\WebLab\Lab3> cd blog  
PS C:\Users\lushi\Рабочий стол\7 семестр\Проектирование клиент-серверных приложений\Лабы\WebLab\Lab3\blog> python manage.py startapp articles  
✓ LAB3  
  ✓ blog  
    ✓ articles  
      > migrations  
      + _init_.py  
      + admin.py  
      + apps.py  
      + models.py  
      + tests.py  
      + views.py  
      > blog  
      + manage.py
```

Рисунок 1 – Создание проекта

В начале работы над новым проектом зададим базовые настройки: имя базы данных и создадим её таблицы, как и в прошлых лабораторных.

Заходим в директорию articles и в файл models.py вставляем код из методички.

### Листинг 1. Содержимое файла models.py

```
blog > articles > models.py > ...
1  from django.db import models
2  from django.contrib.auth.models import User
3
4  # Create your models here.
5
6  class Article(models.Model):
7      title = models.CharField(max_length=200)
8      author = models.ForeignKey(User)
9      text = models.TextField()
10     created_date = models.DateField(auto_now_add=True)
11
12     def __unicode__(self):
13         return "%s: %s" % (self.author.username, self.title)
14
15     def get_excerpt(self):
16         return self.text[:140] + "..." if len(self.text) > 140 else self.text
```

Будущая модель статей будет иметь 4 поля: заголовок, автор, текст и время создания (в последнем значении будет устанавливаться автоматически). Метод `get_excerpt` позволяет в списке всех статей выводить текст статьи не целиком, а показывать первые 140 символов.

В этой же директории откроем файл `admin.py` (он ответственен за настройку страницы записей в административном приложении) и сохраним в нем следующий код:

### Листинг 2. Содержимое файла admin.py

```
blog > articles > admin.py > ...
1  from django.contrib import admin
2  from models import Article
3
4  # Register your models here.
5
6  class ArticleAdmin(admin.ModelAdmin):
7      list_display = ('title', 'author', 'get_excerpt', 'created_date')
8
9  admin.site.register(Article, ArticleAdmin)
```

Далее был запущен локальный сервер нашего сайта командой: **python manage.py runserver**. Результат работы представлен на рисунке 2 на странице 9.



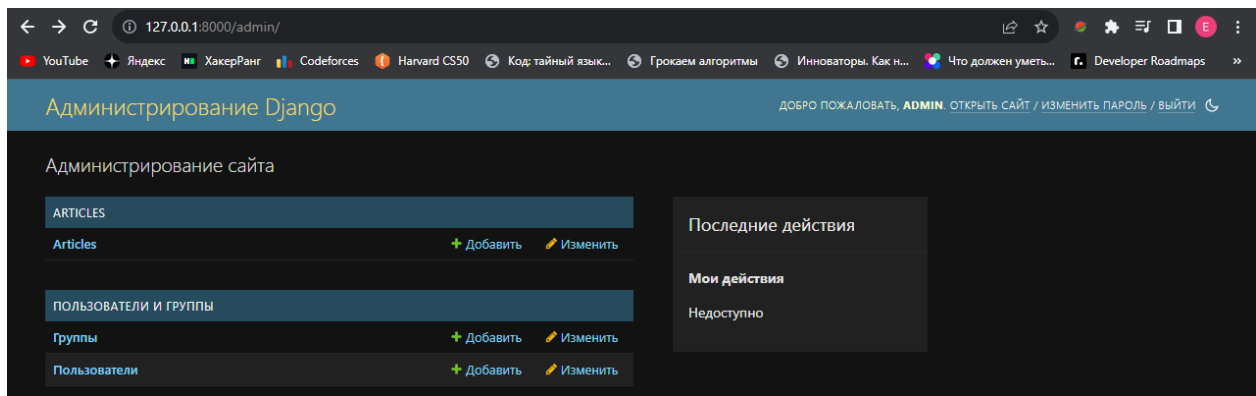


Рисунок 2 – Страница администрирования Django

По заданию необходимо создать 2 статьи (было создано 3) и затем посмотреть их в файле базы данных с помощью программы управления базами данных. В нашем случае это программа SQLiteManager. Результаты представлены на рисунках 3–4, на страницах 9 и 10 соответственно.

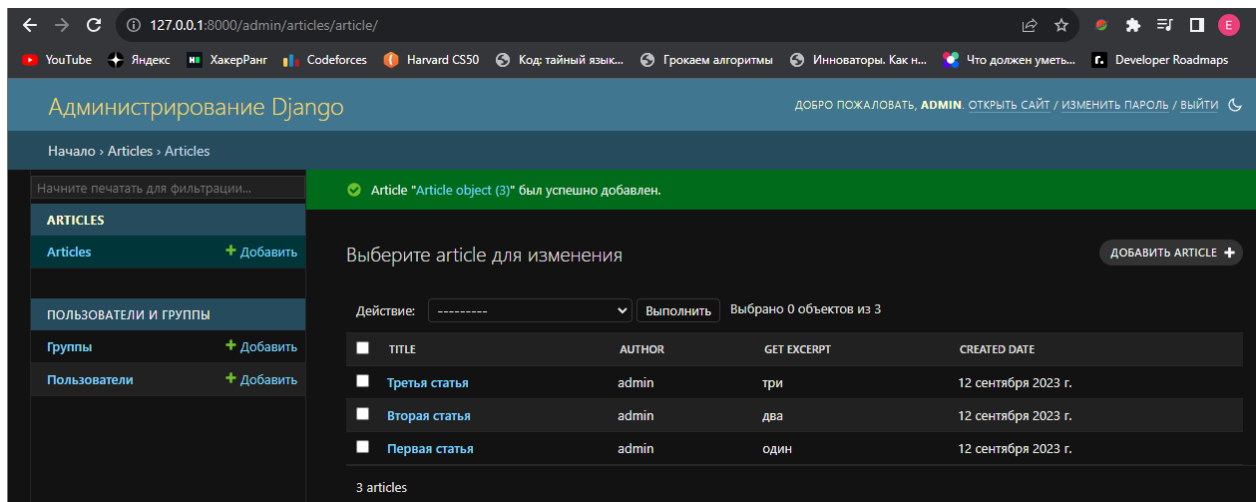


Рисунок 3 – Созданные статьи

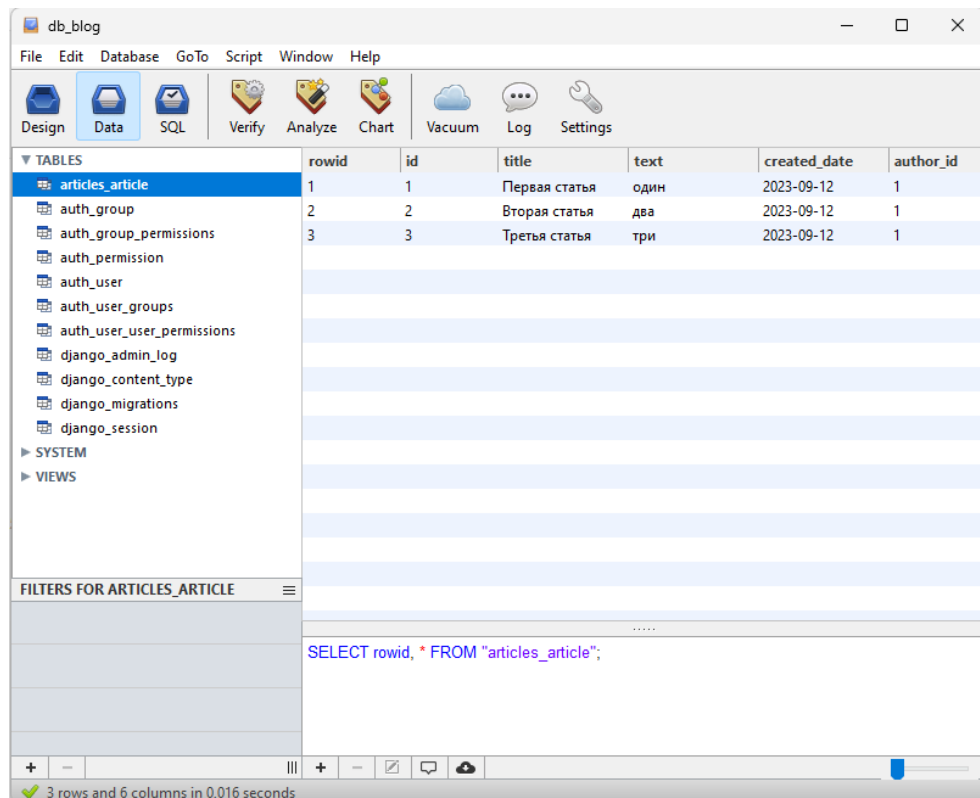


Рисунок 4 – Отображение таблицы в SQLiteManager

Далее отредактируем некоторые статьи через SQLiteManager. На странице со статьями также всё изменилось. Это продемонстрировано на рисунках 5 и 6, на страницах 10 и 11 соответственно.

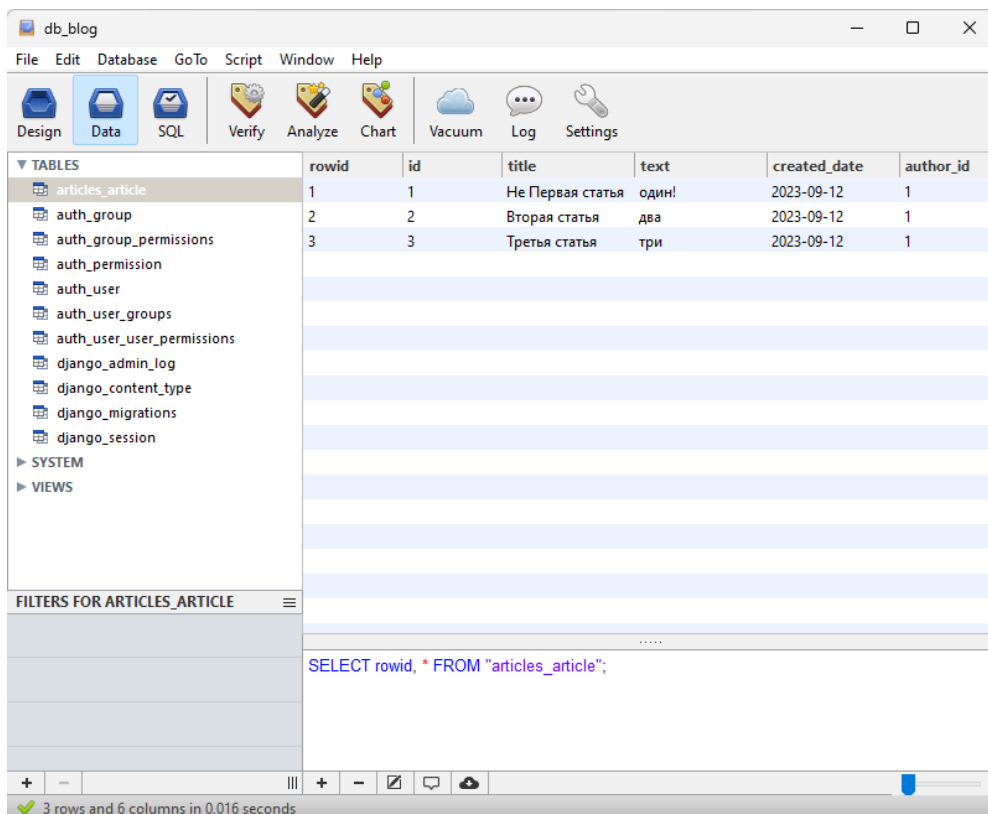


Рисунок 5 – Обновление таблицы в SQLiteManager

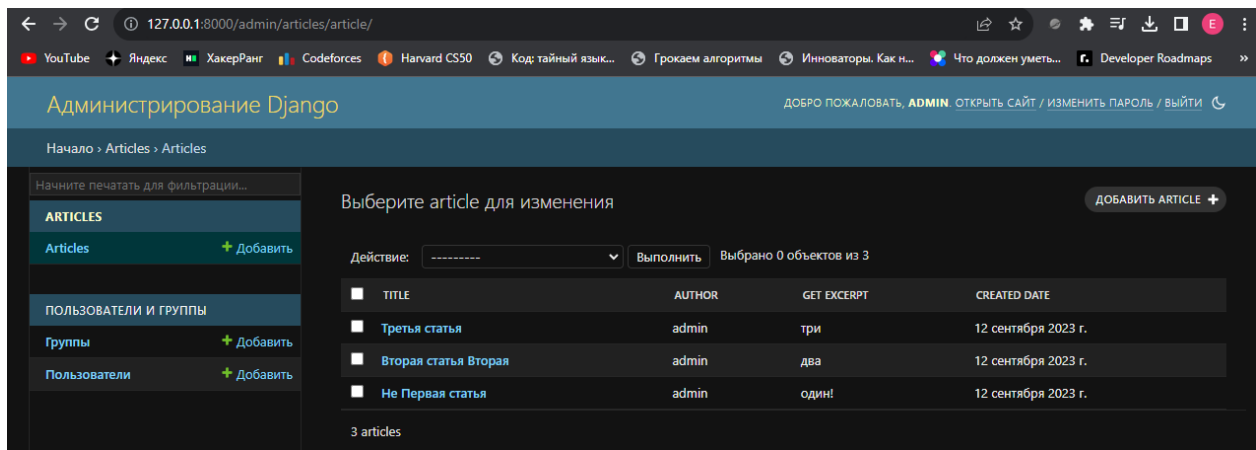


Рисунок 6 – Обновлённые статьи в административной панели

Далее перейдём в директорию articles, в ней создадим папку templates, внутри которой создаём файл archive.html.

Листинг 3. Содержимое файла archive.html

```
blog > articles > templates > <> archive.html > ...

1  {% load static %}
2
3  <!DOCTYPE html>
4  <html>
5
6  <head>
7  |   <title>Архив всех статей</title>
8  </head>
9
10 <body>
11 |
12 |   <body>
13 |     <div class="header">
14 |       
15 |     </div>
16 |     <div class="archive">
17 |       {% for post in posts %}
18 |         <div class="one-post">
19 |           <h2 class="post-title">{{ post.title }}</h2>
20 |           <div class="article-info">
21 |             <div class="article-author">{{ post.author.username }}</div>
22 |             <div class="article-createddate">{{
23 |               post.created_date }}</div>
24 |           </div>
25 |           <div>
26 |             <p class="article-text">{{ post.get_excerpt }}</p>
27 |           </div>
28 |         </div>
29 |       {% endfor %}
30 |     </body>
31 |   </body>
32 </html>
```

В файле `views.py` в директории `articles` создадим представление `archive`, которое будет возвращать `html`-страницу со всеми созданными постами в текущем проекте.

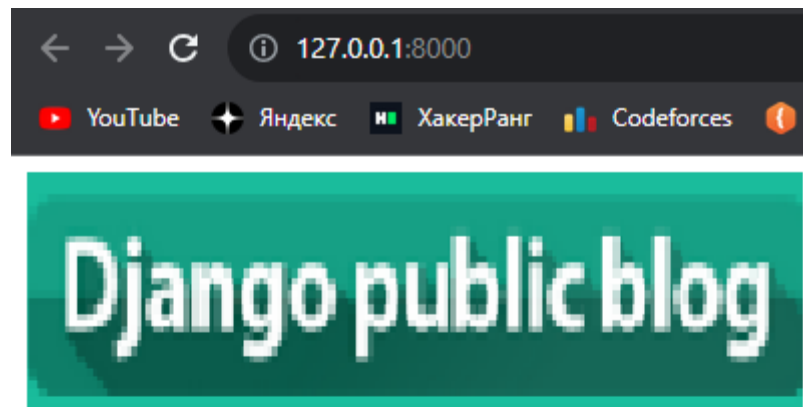
Листинг 4. Содержимое файла `views.py`

```
blog > articles > views.py > ...
1  from .models import Article
2  from django.shortcuts import render
3
4  # Create your views here.
5
6  def archive(request):
7      return render(request, 'archive.html', {"posts":
8
9      Article.objects.all()})
```

Настраиваем наш `url` по которому будут отображаться все наши статьи. По заданию это должна быть домашняя страница. Результат представлен на рисунке 7 на странице 13.

Листинг 5. Содержимое файла `urls.py`

```
blog > blog > urls.py > ...
1  from articles import views
2  from django.contrib import admin
3  from django.urls import path
4
5  urlpatterns = [
6      path('admin/', admin.site.urls),
7      path('', views.archive, name= 'Arcticle')
8  ]
```



## Не Первая статья

```
admin
{{ post.created_date }}
```

один!

## Вторая статья Вторая

```
admin
{{ post.created_date }}
```

два

## Третья статья

```
admin
{{ post.created_date }}
```

три

Рисунок 7 – Вывод информации о статьях на главную страницу

Далее необходимо открыть файл нашей базы данных с помощью программы SQLiteManager и добавить новую статью, после чего проверим, появилась ли она на странице. Результат представлен на рисунках 8–9 на странице 14.

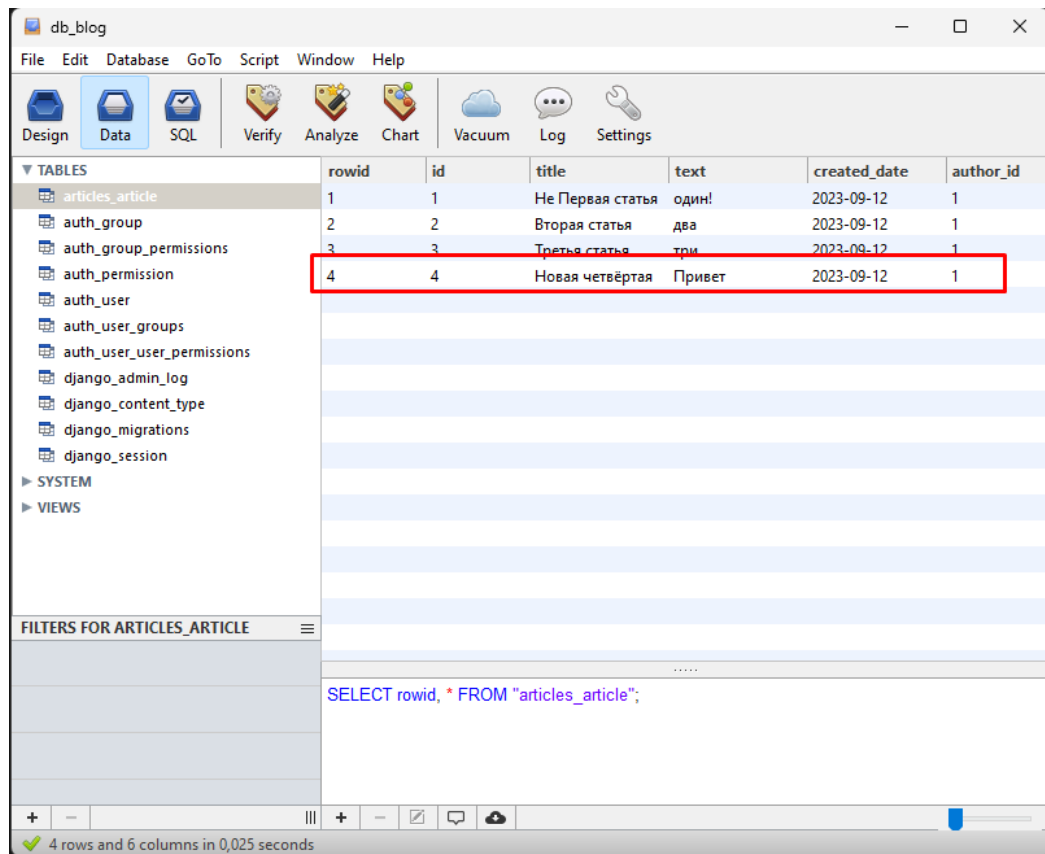
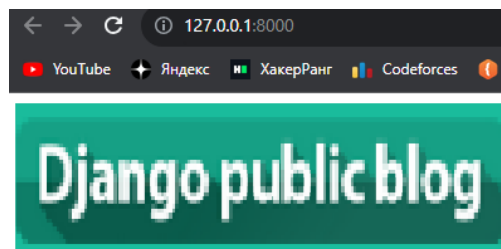


Рисунок 8 – Добавление новой статьи через СУБД



### Не Первая статья

admin  
 {{ post.created\_date }}  
 один!

### Вторая статья Вторая

admin  
 {{ post.created\_date }}  
 два

### Третья статья

admin  
 {{ post.created\_date }}  
 три

### Новая четвёртая статья (из БД)

admin  
 {{ post.created\_date }}  
 Привет

Рисунок 8 – Добавление новой статьи через СУБД

**Вывод:** В данной лабораторной работе я научился создавать модели данных и регистрировать их в Django, а также динамически генерировать шаблон для вывода данных из БД.