РАЗРАБОТКА ПРОГРАММНЫХ ПРИЛОЖЕНИЙ

Лекция 3. Работа с данными через ORM. Введение в Django Forms.

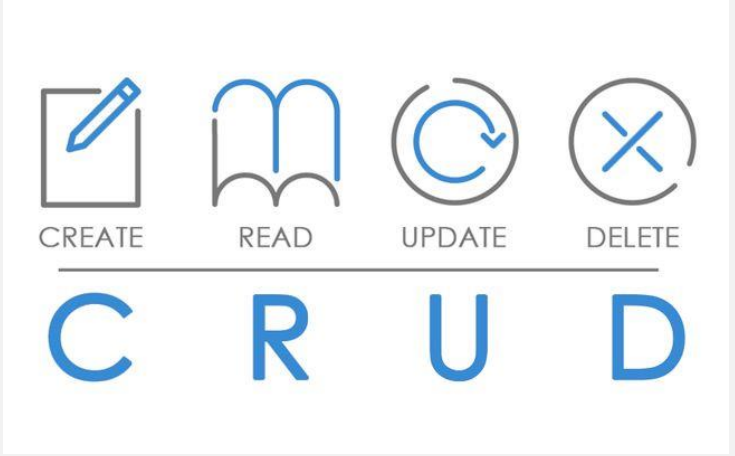
- Запросы через ORM. Манипуляции с данными в Django на основе CRUD.
- Введение в Django Forms

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

Для работы с базой данных в Django существует понятие ORM (объектно-реляционное отображение). Само понятие ORM обозначает возможность преобразования и взаимодействия прикладного языка (в нашем случае Python) и базы данных (обычно это SQL).

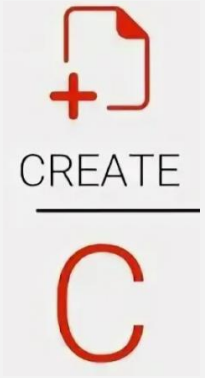
В любой реализации ORM есть 4 ключевых понятия, которые складываются в аббревиатуру **CRUD**:

- **Create** - создание или добавление данных в БД.
- **Read** - получение и чтение данных из БД.
- **Update** - обновление данных в БД.
- **Delete** - удаление данных из БД.



ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

Эти операции широко используются при работе с базами данных, и в Django ORM есть встроенные методы для реализации этих операций. Рассмотрим, как каждую из этих операций можно выполнить в Django.



1. Создание данных (Create)

Создание новой записи в базе данных в Django выполняется с использованием модели и метода **save()** для сохранения объекта. Django автоматически создает SQL-запрос для вставки данных в таблицу.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

Способы создания данных:

- **Программное создание объекта и вызов .save():**

Пример создания нового студента:

python

```
from .models import Student

student = Student(name = 'Иван', surname = 'Иванов', group = '410')
student.save()
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

Способы создания данных:

- **Использование метода .create():**

Это сокращенный способ создать и сохранить объект в одной строке:

python

```
from .models import Student

student = Student.objects.create(name='Иван', surname='Иванов', group = '410')
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

Способы создания данных:

- **Создание данных через Django формы:**

Форма позволяет пользователю вводить данные, которые потом сохраняются в базе. Пример:

python

```
if request.method == 'POST':
    form = StudentForm(request.POST)
    if form.is_valid():
        form.save()
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

Способы создания данных:

Также существует метод **get_or_create()**, он сначала ищет объект с заданными параметрами и создает его, если он не найден.

Как использовать данный метод – изучаем самостоятельно!!!

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

2. Чтение данных (Read)

Чтение данных означает выборку данных из базы данных. Django ORM предоставляет множество методов для выборки объектов.

Способы чтения данных:

- **all()** - для всех объектов;
- **get()** - для одного объекта;
- **filter()** - для группы объектов по фильтру;
- **exclude()** - для группы объектов с исключением некоторых.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

Способы чтения данных:

Методы all(), filter() и exclude() возвращают объект **QuerySet**. Это, по сути, некое промежуточное хранилище, в котором содержится информация, полученная из БД.

Объект QuerySet может быть создан, отфильтрован и затем использован фактически без выполнения дополнительных запросов к базе данных.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

QuerySets — это фундаментальная концепция Django ORM (Object Relational Mapper), которая используется для извлечения данных из базы данных. Наборы запросов позволяют получить список объектов из таблицы базы данных, отвечающих определенным критериям

Django QuerySet — это список объектов из таблицы базы данных, соответствующих определенным критериям. Наборы запросов позволяют фильтровать, упорядочивать и нарезать данные различными способами. Они также поддерживают объединение нескольких методов для создания сложных запросов.

QuerySet оцениваются лениво, что означает, что запрос к базе данных не выполняется до тех пор, пока не будет вызван QuerySet. Это позволяет создавать сложные запросы без ненужного обращения к базе данных.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

Методы возвращающие QuerySet

- **filter()**: Фильтр по заданным параметрам поиска. Несколько параметров объединяются операторами SQL AND.
- **exclude()**: Фильтр по объектам, которые не соответствуют заданным параметрам поиска.
- **annotate()**: Аннотирует каждый объект в QuerySet. Аннотации могут быть простыми значениями, ссылкой на поле или агрегированным выражением.
- **alias()**: То же, что annotate(), но вместо аннотирования объектов в QuerySet, сохраняет выражение для последующего повторного использования с другими QuerySet методами.
- **order_by()**: Изменение сортировки QuerySet по умолчанию.
- **reverse()**: Сортирует QuerySet по умолчанию в обратном порядке.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- **distinct()**: Выполнение SQL запроса SELECT DISTINCT для устранения дублирующихся строк.
- **values()**: Возвращает словари вместо экземпляров моделей.
- **values_list()**: Возвращает кортежи вместо экземпляров модели.
- **dates()**: Возвращает QuerySet, содержащий все доступные даты в указанном диапазоне дат.
- **datetimes()**: Возвращает QuerySet, содержащий все доступные даты в указанном диапазоне дат и времени.
- **none()**: Создает пустой QuerySet.
- **all()**: Возвращает объект QuerySet, который содержит все объекты из таблицы.
- **union()**: Использует SQL оператор UNION для объединения двух или более QuerySet.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- **intersection()**: Использует SQL оператор INTERSECT для возврата общих элементов двух или более QuerySet.
- **difference()**: Использует SQL оператор EXCEPT для возврата элементов первичного QuerySet которых нет в других.
- **select_related()**: Выборка всех связанных данных при выполнении запроса (за исключением отношений многие-ко-многим).
- **prefetch_related()**: Выборка всех связанных данных при выполнении запроса related()(включая отношения многие-ко-многим).
- **extra()**: Метод используется для переименования столбцов в ORM.
- **defer()**: Не извлекает указанные поля из БД. Используется для улучшения производительности запросов со сложными наборами данных.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- **only()**: Противоположность метода defer() - возвращает только указанные поля.
- **using()**: Выбирает, в какой базе данных будет оцениваться QuerySet(при использовании нескольких баз данных).
- **select_for_update()**: Возвращает QuerySet и блокирует строки таблицы до завершения транзакции.
- **raw()**: Выполняет исходный SQL оператор.
- **&**: Комбинирует два QuerySet с помощью SQL оператора AND.
- **|**: Комбинирует два QuerySet с помощью SQL оператора OR.
- **^**: Комбинирует два QuerySet с помощью SQL оператора XOR.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

Методы которые не возвращают QuerySet

- **get()**: Возвращает один объект. Вызывает ошибку, если поиск возвращает несколько объектов.
- **create()**: Быстрый метод создания и сохранения объекта за один шаг.
- **get_or_create()**: Возвращает один объект. Если объект не существует, то он его создает.
- **update_or_create()**: Обновляет один объект. Если объект не существует, то он его создает.
- **bulk_create()**: Вставляет список объектов в базу данных.
- **bulk_update()**: Обновляет указанные поля в списке экземпляров модели.
- **count()**: Подсчитывает количество объектов в возвращаемом QuerySet. Возвращает целое число.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- **in_bulk()**: Возвращает словарь, содержащий все объекты с указанными ID.
- **iterator()**: Выполнение QuerySet и возвращение итератора по результатам. Может улучшить производительность и потребление памяти при запросах, возвращающих большое количество объектов.
- **latest()**: Возвращает последний объект в таблице базы данных на основе заданного поля (полей).
- **earliest()**: Возвращает самый ранний объект в таблице базы данных на основе заданного поля (полей).
- **first()**: Возвращает первый объект в соответствующем QuerySet.
- **last()**: Возвращает последний объект в соответствующем QuerySet.
- **aggregate()**: Возвращает словарь агрегированных значений (средние, суммы и т.д.), рассчитанных по QuerySet.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- **exists()**: Возвращает True, если QuerySet содержит какие либо результаты.
- **contains()**: Метод возвращает True, если набор QuerySet содержит определенный объект, и False - если не содержит.
- **update()**: Выполняет SQL оператор UPDATE в указанном поле (полях).
- **delete()**: Выполняет SQL оператор DELETE, удаляющий все строки в QuerySet.
- **as_manager()**: Возвращает экземпляр класса Manager, содержащий копию методов QuerySet.
- **explain()**: Возвращает строку плана выполнения QuerySet. Используется для анализа выполнения запросов.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- **Получение всех объектов: all()**

Данный метод возвращает объект типа QuerySet.

✓ код Python:

```
groups = Group.objects.all() # Получаем все группы
for group in groups:
    print(group) # Выводим название каждой группы
```

✓ Для того, чтобы вывести данную информацию на страницу сайта:

1. Создаем шаблон в директории: students/templates/students, например temp.html:

```
{% extends 'students/base.html' %}
{% block title %}Разные примеры получения объектов из бд{% endblock %}
{% block header %}Разные примеры получения объектов из бд{% endblock %}
```

2. В файле models.py приложения students:

```
# Модель для групп студентов
class Group(models.Model):
    name = models.CharField(max_length=10)

    objects = models.Manager()  # Явное определение менеджера

    def __str__(self):
        return str(self.name)
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

3. В файле **views.py** приложения **students**:

```
from .models import Group


#Получение разных объектов
@login_required
def temp(request):
    groups = Group.objects.all()  # Получаем все группы
   return render(request, 'students/temp.html', {'groups': groups})
```

4. В файле urls.py приложения students:

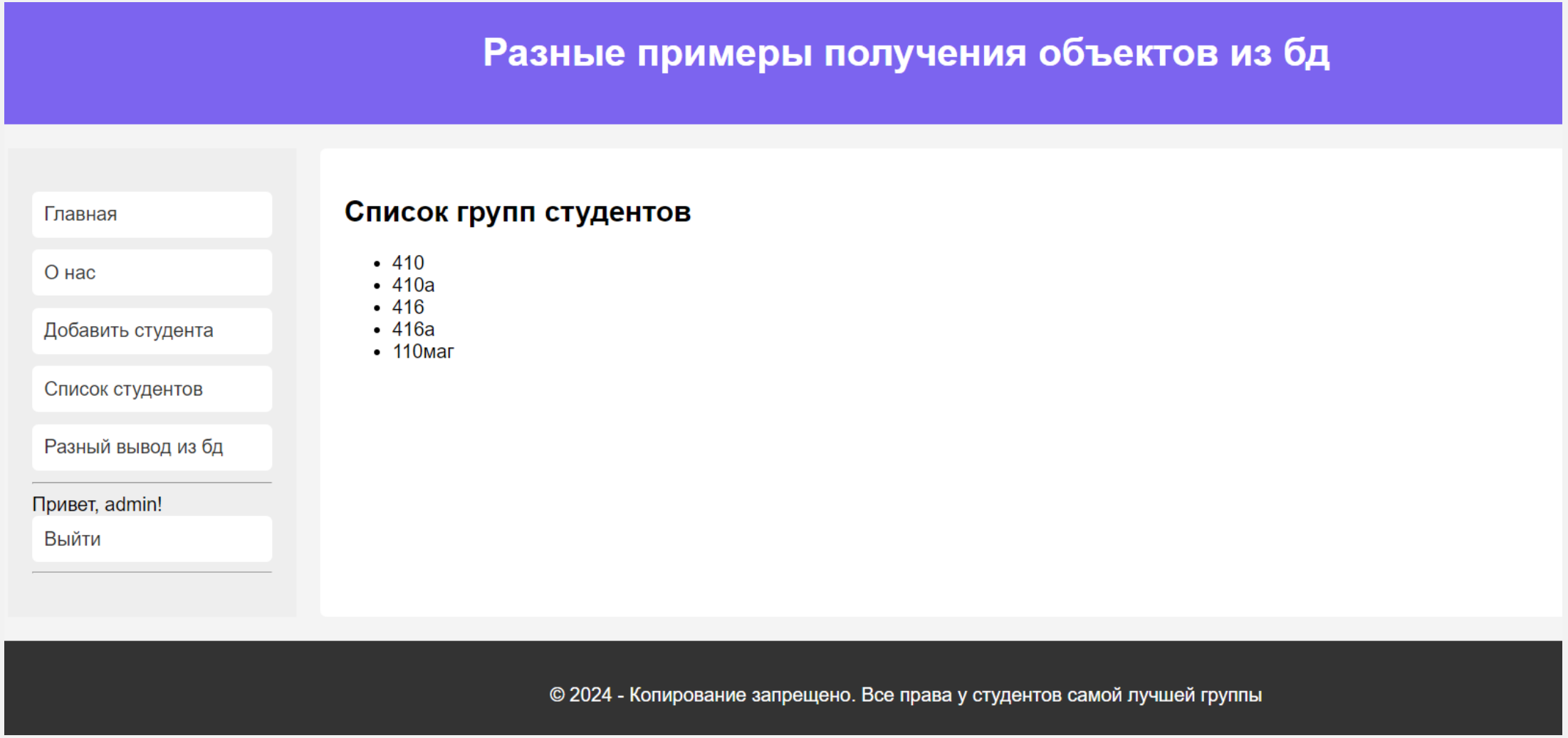
Добавляем следующий путь:

```
path('temp/', views.temp, name='temp'),
```

5. Добавляем в базовый шаблон students/templates/students/base.html:

В блок навигации следующий пункт:

```
{% block nav %}
    <li><a href="{% url 'index' %}">Главная</a></li>
    <li><a href="{% url 'about' %}">О нас</a></li>
    ....
    <li><a href="{% url 'temp' %}">Разный вывод из бд</a></li>
    ....
```


Разные примеры получения объектов из бд
Главная
О нас
Добавить студента
Список студентов
Разный вывод из бд
Привет, admin!
Выйти
Список групп студентов
410
410а
416
416а
110маг
© 2024 - Копирование запрещено. Все права у студентов самой лучшей группы

- **Получение одного объекта get().**

Возвращает одну запись, соответствующую условию

код Python:

```
# Получаем студента по id
student = Student.objects.get(id=1)
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- **Получение одного объекта get().**

При использовании этого метода надо учитывать, что он предназначен для выборки таких объектов, которые имеются в единичном числе в базе данных.

Если в таблице не окажется подобного объекта, то мы получим исключение имя_модели.DoesNotExist.

Если же в таблице будет несколько объектов, которые соответствуют условию, то будет сгенерировано исключение имя_модели.MultipleObjectsReturned.

Поэтому следует применять данный метод с осторожностью, либо применять обработку соответствующих исключений.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- **Фильтрация объектов filter(): возвращает объекты, которые соответствуют определённым условиям, переданным в виде списка параметров.**

Получение студентов определенной группы

код Python:

```
# Получаем группу по имени
group = Group.objects.get(name="410")
students_in_group = Student.objects.filter(group=group)  # Фильтруем студентов по группе
for student in students_in_group:
    print(student)  # Выводим студентов в группе
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

Фильтрация в Django ORM является мощным инструментом для работы с запросами. Django предоставляет множество различных фильтров, которые можно использовать для построения запросов к базе данных, учитывая различные условия.

- Точный поиск: **exact**

Ищет точное совпадение значения поля.

python

```
groups = Group.objects.filter(name__exact='410')
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Поиск без учета регистра: **iexact**

  Ищет точное совпадение значения поля, но без учета регистра.

python

```
groups = Group.objects.filter(name__iexact='410A')
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Частичное совпадение: **contains**

Ищет строки, которые содержат указанное значение (учитывая регистр).

python

```
groups = Group.objects.filter(name__contains='410')
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Частичное совпадение без учета регистра: **icontains**

Ищет строки, которые содержат указанное значение, без учета регистра.

python

```
groups = Group.objects.filter(name__icontains='A')
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Меньше чем, больше чем: **lt, lte, gt, gte**

Используются для числовых и датированных полей.

lt: меньше чем.

lte: меньше или равно.

gt: больше чем.

gte: больше или равно.

python

```
students = Student.objects.filter(scholarship__gte=5000)
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Диапазон значений: **range**

Проверяет, находится ли значение в пределах указанного диапазона.

python

```
students = Student.objects.filter(current_semester__range=(1, 3))
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Проверка на пустые значения: isnull

Проверяет, является ли значение поля NULL в базе данных.

python

```
students_without_group = Student.objects.filter(group__isnull=True)
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Поиск по списку значений: **in**

Используется для фильтрации по нескольким значениям.

python

```
groups = Group.objects.filter(name__in=['410', '416'])
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Регулярные выражения: regex и iregex

regex — поиск по регулярным выражениям с учетом регистра.

iregex — поиск по регулярным выражениям без учета регистра.

python

```
students = Student.objects.filter(name__regex=r'^[А-Яа-я]')
students = Student.objects.filter(name__iregex=r'^[а-я]')
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Комбинирование фильтров: **Q объекты**

Для более сложных запросов можно использовать Q объекты для построения запросов с логическими операторами **AND**, **OR**, **NOT**.

- **Логическое И** (по умолчанию):

  Фильтры применяются последовательно и соответствуют логическому И.

python

```
students = Student.objects.filter(name='Иван', surname='Иванов')
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- **Логическое ИЛИ (|)**:

Используются Q объекты для создания логических запросов с ИЛИ.

python

```
from django.db.models import Q

students = Student.objects.filter(Q(name='Иван') | Q(surname='Петров'))
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- **Логическое НЕ (~):**

Для отрицания фильтров используется символ ~.

python

```
from django.db.models import Q

students = Student.objects.filter(~Q(name='Иван'))
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- **Комбинация условий:**

Можно комбинировать несколько Q объектов для более сложных запросов.

python

```
from django.db.models import Q

students = Student.objects.filter(Q(name='Иван') | Q(surname='Петров') & ~Q(group__name='410'))
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- **Ограничение числа возвращаемых объектов**

Ограничение количества результатов: [:N]: и пропуск первых объектов: [N:]:

python

```
students = Student.objects.all()[:5]  # Возвращает первых 5 студентов
```

python

```
students = Student.objects.all()[5:]  # Пропустит первых 5 студентов
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

3. Обновление данных (Update)

Обновление данных в Django выполняется путем изменения атрибутов модели и вызова метода **.save()** для обновления записи в базе данных. Также существует метод **.update()**, который применяется к наборам запросов.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Изменение значений атрибутов и вызов **.save()**: Чтобы обновить запись, нужно сначала получить объект, изменить его поля и сохранить:

python

```
student = Student.objects.get(id=1)
student.name = 'Александр'
student.save()
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Использование метода **.update()** для массового обновления: Метод **.update()** используется для изменения значений нескольких объектов одновременно:

python

```
Student.objects.filter(group__name='410').update(scholarship=15000.0)
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Обновление через Django формы: Если у вас есть форма для редактирования данных, можно обновить запись следующим образом:

python

```
if request.method == 'POST':
    form = StudentForm(request.POST, instance=student)
    if form.is_valid():
        form.save()
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

4. Удаление данных (Delete)

Удаление данных из базы данных в Django осуществляется с помощью метода **.delete()**. Важно отметить, что удаление данных необратимо, и удаленные записи нельзя восстановить без резервной копии.

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

Способы удаления данных:

- Удаление одного объекта: Получив объект, можно вызвать метод **.delete()**, чтобы удалить его из базы данных:

python

```
student = Student.objects.get(id=1)
student.delete()
```

ЗАПРОСЫ ЧЕРЕЗ ORM. МАНИПУЛЯЦИИ С ДАННЫМИ В DJANGO НА ОСНОВЕ CRUD

- Массовое удаление с использованием **.filter()**: Можно удалить несколько объектов одновременно с помощью .filter():

python

```
Student.objects.filter(group__name='410').delete()
```

ВВЕДЕНИЕ В DJANGO FORMS

Django Forms — это инструмент Django для работы с HTML-формами. Они помогают автоматизировать процесс создания, рендеринга, валидации и обработки данных, отправленных пользователями через веб-формы. Формы позволяют легко собирать информацию от пользователей, валидировать её и сохранять в базу данных.

Формы несут целый набор виджетов для ввода различных типов данных: текстовые поля, флажки, переключатели, установщики дат и пр. Формы являются относительно безопасным способом взаимодействия пользовательского клиента и сервера, поскольку позволяют отправлять данные в POST-запросах, применяя защиту от межсайтовой подделки запроса (Cross-Site Request Forgery, CSRF)

ВВЕДЕНИЕ В DJANGO FORMS

Django предлагает два основных подхода для работы с формами:

- **Формы на основе классов (forms.Form)** — используются для создания пользовательских форм, не связанных напрямую с моделями.
- **Формы на основе моделей (forms.ModelForm)** — позволяют автоматически создавать форму на основе полей модели.

ВВЕДЕНИЕ В DJANGO FORMS

- Пример простой формы на основе класса **Form**, которая позволяет собирать информацию о студенте:

python

```
from django import forms

class SimpleStudentForm(forms.Form):
    name = forms.CharField(label="Имя", max_length=100)
    surname = forms.CharField(label="Фамилия", max_length=100)
    current_semester = forms.IntegerField(label="Текущий семестр", min_value=1)

# В представлении:
def student_form_view(request):
    form = SimpleStudentForm()
    return render(request, 'student_form.html', {'form': form})
```

Формы на основе **forms.Form** полезны, когда данные формы не обязательно должны быть сохранены в базе данных, или когда требуется создать форму с полностью кастомной логикой.

ВВЕДЕНИЕ В DJANGO FORMS

- ModelForm

Когда данные формы должны напрямую взаимодействовать с моделями (например, чтобы создать или обновить запись в базе данных), мы используем ModelForm. ModelForm упрощает создание формы, так как она автоматически генерирует поля на основе модели, и позволяет легко сохранять данные формы в связанную модель.

Django Forms (включая ModelForm) предоставляют следующие возможности:

- Автоматическая генерация HTML-форм на основе модели или указанных полей.
- Валидация данных — проверка того, что введенные пользователем данные соответствуют правилам валидации.
- Упрощенная обработка данных — получение и сохранение данных в базу.

ВВЕДЕНИЕ В DJANGO FORMS

- Пример создания ModelForm для модели Student

Наша модель Student имеет следующие поля: name, surname, patronimus, group, marks (через through-модель), current_semester, и scholarship. Можно создать форму для студента, которая будет генерироваться на основе модели:

python

```
from django import forms
from .models import Student


class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ['name', 'surname', 'patronimus', 'group', 'current_semester']
```

ВВЕДЕНИЕ В DJANGO FORMS

- Рендеринг формы в шаблоне: для отображения формы в шаблоне можно использовать такие подходы:

html

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }} <!-- Рендерит форму как абзацы -->
    <button type="submit">Отправить</button>
</form>
```

Вместо {{ form.as_p }} можно использовать:

- {{ form.as_table }} — для вывода формы в виде таблицы.
- {{ form.as_ul }} — для вывода формы в виде списка.

ВВЕДЕНИЕ В DJANGO FORMS

- Обработка данных формы

После того как пользователь заполняет и отправляет форму, нам нужно обработать введенные данные в представлении. Для этого мы можем проверить валидность формы и сохранить данные в модель:

ВВЕДЕНИЕ В DJANGO FORMS

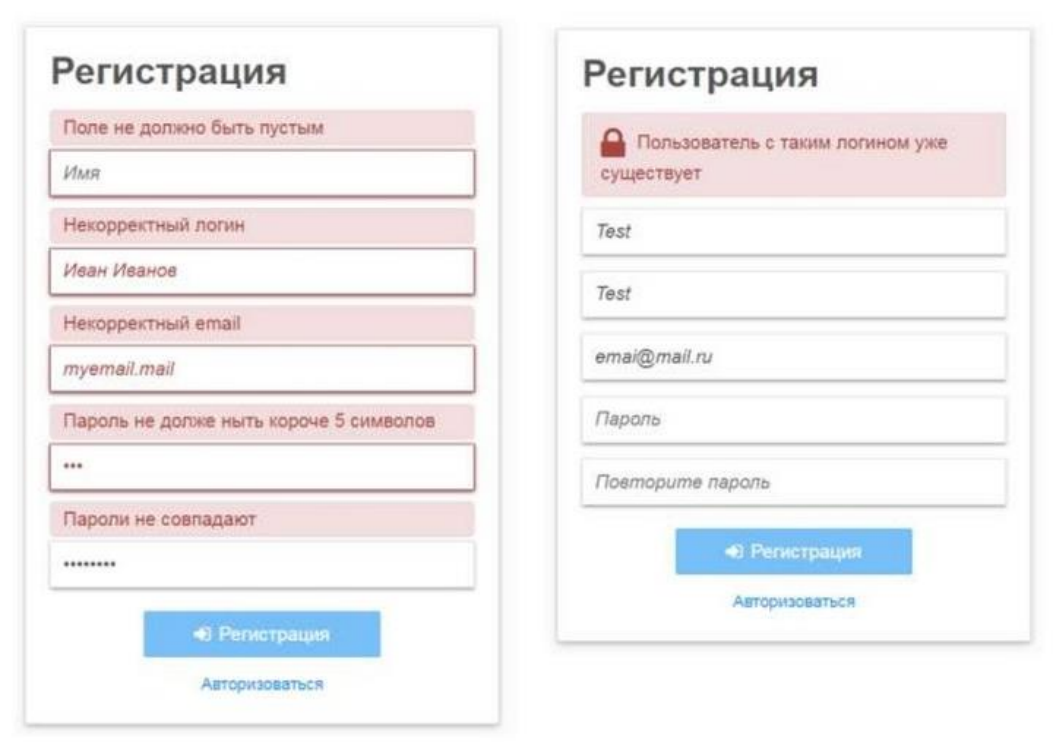
python

```
from django.shortcuts import render, redirect
from .forms import StudentForm


def add_student(request):
    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            form.save()  # Сохраняем данные в базу
            return redirect('success')  # Перенаправляем на другую страницу после успешного сохранения
    else:
        form = StudentForm()

    return render(request, 'add_student.html', {'form': form})
```

ВВЕДЕНИЕ В DJANGO FORMS

- Валидация данных формы

Django предоставляет встроенные механизмы для валидации данных формы. Помимо базовой проверки полей (например, что обязательные поля заполнены), можно добавлять свою логику валидации.



ВВЕДЕНИЕ В DJANGO FORMS

Пример пользовательской валидации для поля current_semester:

python

```
class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ['name', 'surname', 'patronimus', 'group', 'current_semester']

    def clean_current_semester(self):
        current_semester = self.cleaned_data['current_semester']
        if current_semester < 1:
            raise forms.ValidationError("Семестр не может быть меньше 1")
      return current_semester
```

ВОПРОСЫ ПО ТЕМЕ

Для чего используется Post.objects.all()?

```
from post.models import Post

m = Post.objects.all()
for i in m:
    print(i)
```

1. Удалит все объекты из базы данных.
2. Ни один вариант из перечисленных.
3. Он используется для создания нового объекта Post m.
4. Он возвращает все объекты, хранящихся в таблице Post.

ВОПРОСЫ ПО ТЕМЕ

Имеется модель:

```
class Post(models.Model):
    title   = models.CharField(max_length=200)
    content = models.TextField()
```

Что будет содержать переменная m?

```
from post.models import *
m = Post.objects.filter(title="post1")
```

1. Ни один вариант из перечисленных.
2. Она содержит все объекты, соответствующие заголовку «post1».
3. Она содержит все объекты Post.
4. Она содержит первый объект, который соответствует заголовку «post1» в аргументе.

ВОПРОСЫ ПО ТЕМЕ

В метод filter() вы можете передать несколько условий, чтобы сузить результаты?

1. Да.
2. Нет.

ВОПРОСЫ ПО ТЕМЕ

Какой файл по умолчанию не создается в директории приложения?

1. urls.py
2. models.py
3. views.py
4. __init__.py

ВОПРОСЫ ПО ТЕМЕ

Какой путь мы должны добавить в urlpatterns, чтобы отобразить данный текст по адресу http://127.0.0.1:8000/ ?

Файл my_app/views.py содержит следующее представление:

```
def index(request):
    return HttpResponse("Hello World!")
```

А файл my_app/urls.py содержит следующий код:

```
from my_app import views
from django.urls import path

urlpatterns = [
    ....
]
```

1. path('/index', views.home, name='index')
2. path('/index', index, name='index')
3. path(' ', views.index, name='index')
4. path('/index', views)

ВОПРОСЫ ПО ТЕМЕ

Что включают в себя статические файлы? (несколько вариантов ответа)

1. Файлы CSS
2. Файлы изображений
3. Файлы Javascript
4. Файлы html

ВОПРОСЫ ПО ТЕМЕ

Какой обязательный параметр принимает функция представления?