УРОК №11

Django
Запись данных

Ключевые темы

- Просмотр SQL-кода
- Методы для работы с моделью
- Метод save()
- Обработчики ошибок (4xx, 5xx)

SQL-команды

QuerySet.query позволяет получить доступ к сгенерированному SQL-запросу, который будет выполнен при выполнении данного **QuerySet**.

```
clients = Client.objects.all()


# SELECT "shop_client"."id", "shop_client"."username" FROM "shop_client"
print(clients.query)
```

get_or_create

Метод **get_or_create()** получает объект из БД, а если его там нет, то он будет добавлен в БД как новый объект.

```
product, create_status = Product.objects.get_or_create(title="title",
                                                       price=100500)


# Строковое представление объекта модели
print(product)
# True, если запись была успешна добавлена
# False, если такая запись уже есть
print(create_status)
```

update_or_create

Метод **update_or_create** пытается получить объект из базы данных на основе заданных аргументов. Если совпадение найдено, обновляются поля, переданные в словарь **defaults** по умолчанию.

```
product, create_status = Product.objects.update_or_create(title="Чайник",
                                                          defaults={"price": 1500})
# Цена изменилась с 1000 на 1500
print(product.price)
# True, если запись была успешна добавлена
# False, если такая запись уже есть
print(create_status)
```

in_bulk

Метод **in_bulk**() возвращает словарь, где ключами являются идентификаторы объектов, а значениями - сами объекты модели. Если какой-либо идентификатор не найден в базе данных, соответствующий ключ в словаре будет отсутствовать.

```
products = Product.objects.in_bulk()


for pk, product in products.items():
    print(pk, "-", product)
```

```
1 - Стиральная машина
2 - Холодильник
3 - Утюг
4 - Электрическая плита
5 - Чайник
```

update на +1 к полю

Иногда возникает необходимость изменить значение столбца в БД на основании уже имеющегося там значения. Для таких случаев существует функция **F()**.

Основная цель функции **F()** – позволить использовать значения полей модели в выражениях БД.

```
from django.db.models import F


# После применения данной команды поле price
# во всех полях модели Product увеличится на 1000
products = Product.objects.update(price=F('price') + 1000)
```

Метод save()

save([update_fields=None] [force_insert=False] [force_update=False]

- **update_fileds** – позволяет указать список полей, которые должны быть обновлены при сохранении объекта (для оптимизации).
- **force_insert** – принудительное добавление объекта в БД.
- **force_update** – принудительное обновление объекта в БД.

```
product.price = 50000
product.save(update_fields=['price'], force_update=True)
```

404 error

Многие ресурсы имеют оформленные страницы ошибок, если происходит сбой в обработке запроса от клиента. Django предоставляет функционал для индивидуальной обработки каждой из ошибок.

При обнаружении **ошибки 404** (страница не найдена) Django ищет соответствующее представление, которое обрабатывает эту ошибку. По умолчанию, Django использует представление **django.views.defaults.page_not_found**, которое генерирует страницу с сообщением об ошибке.

404 error

Для обработки ошибки 404 требуется:

1) Перевести сервер Django в режим реальной работы (в **settings.py debug**=False, **allowed_hosts**=[127.0.0.1])
2) Написать функцию-обработчик для ошибки 404
3) В **urls.py** задать значение для переменной handler404 в виде строки (путь до функции-обработчика)

```
def handler404func(request, exception):
    return HttpResponse(f"<h1>404 ERROR</h1>")
```

```
DEBUG = False
ALLOWED_HOSTS = ['127.0.0.1']
```

```
handler404 = "shop.views.handler404func"
```

Конец