GeekBrains

Flask

Модель статьи: категории, теги, связи в БД

На этом уроке

1. Создадим модель тегов.
2. Сделаем связь «многие ко многим» со статьёй.
3. Добавим команду для миграции данных (создание стандартных тегов).
4. Добавим в форму создания статьи возможность выбрать теги.
5. Сохраним выбранные теги для этой статьи (через таблицу ассоциации).
6. Выведем на страницу просмотра статьи связанные теги.
7. Используем опцию joinedload.

Оглавление

Теория

joinedload

Функция SQLAlchemy. Указывает, что данный (переданный) атрибут должен быть загружен с помощью объединённой активной загрузки. Эта функция является частью интерфейса загрузки и поддерживает как связанные методы, так и автономные операции. При помощи этой функции можно в том же запросе подгрузить из БД связанные записи.

Практическое задание

Создаём модель тегов

Добавляем в blog/models/tag.py описание модели:

```
from sqlalchemy import Column, Integer, String
from blog.models.database import db

class Tag(db.Model):
    id = Column(Integer, primary_key=True)
    name = Column(String(32), nullable=False, default="", server_default="")
```

И добавляем импорт в blog/models/__init__.py:

```
from blog.models.tag import Tag

__all__ = [
    "User",
    "Author",
    "Article",
    "Tag",
]
```

Создаём и выполняем миграцию.

Создаём промежуточную таблицу для связи статей с тегами

Нам необходимо создать простую таблицу, мы не будем работать с этими объектами связи отдельно, они нужны нам только для привязывания тегов к статьям. Наполняем модуль blog/models/article_tag.py:

```
from sqlalchemy import Table, Column, Integer, ForeignKey

from blog.models.database import db


article_tag_association_table = Table(
    "article_tag_association",
    db.metadata,
    Column("article_id", Integer, ForeignKey("article.id"), nullable=False),
    Column("tag_id", Integer, ForeignKey("tag.id"), nullable=False),
)
```

В blog/models/article.py добавляем связь статей с тегами через промежуточную таблицу (для этого в связи указываем `secondary`):

```
from blog.models.article_tag import article_tag_association_table


class Article(db.Model):
    ...

    tags = relationship(
         "Tag",
         secondary=article_tag_association_table,
         back_populates="articles",
     )
```

Также добавляем связь тегов со статьями в blog/models/tag.py:

```
from sqlalchemy.orm import relationship
from blog.models.article_tag import article_tag_association_table

class Tag(db.Model):
    ...

    articles = relationship(
        "Article",
        secondary=article_tag_association_table,
        back_populates="tags",
    )
```

Создаём и выполняем миграцию. Будет создана отдельная таблица для хранения связей.

Создаём теги

Создаём новую команду в blog/app.py. Создаём стандартные теги (чтобы не прописывать их вручную в базе данных).

```
@app.cli.command("create-tags")
def create_tags():
    """
    Run in your terminal:
    ➜ flask create-tags
    """
    from blog.models import Tag
    for name in [
        "flask",
        "django",
        "python",
        "sqlalchemy",
        "news",
    ]:
        tag = Tag(name=name)
        db.session.add(tag)
    db.session.commit()
    print("created tags")
```

Выполняем в терминале `flask create-tags`.

Добавляем теги при создании статьи

Начнём с формы для создания статьи blog/forms/article.py. Указываем поле SelectMultipleField:

```
from wtforms import StringField, TextAreaField, SubmitField, SelectMultipleField, validators

class CreateArticleForm(FlaskForm):
    ...

    tags = SelectMultipleField("Tags", coerce=int)
```

В темплейте blog/templates/articles/create.html выводим новое поле

```
{% for field_name in ['title', 'body', 'tags'] %}
```

Далее меняем blog/templates/articles/details.html, показывая все теги, что есть у статьи:

```
{% if article.tags %}
  <div>Tags:</div>
  <ul>
    {% for tag in article.tags %}
      <li>{{ tag.name }}</li>
    {% endfor %}
  </ul>
{% endif %}
```

И самое главное — меняем view в blog/views/articles.py:

- подгружаем связанные теги (чтобы не отправлять новый запрос в БД при отрисовке тегов);
- устанавливаем выбранные теги при создании статьи.

```
@articles_app.route("/<int:article_id>/", endpoint="details")
def article_details(article_id):
    article = Article.query.filter_by(id=article_id).options(
        joinedload(Article.tags)  # подгружаем связанные теги!
    ).one_or_none()
    if article is None:
        raise NotFound
    return render_template("articles/details.html", article=article)

@articles_app.route("/create/", methods=["GET", "POST"], endpoint="create")
@login_required
def create_article():
    ...
    # добавляем доступные теги в форму
    form.tags.choices = [(tag.id, tag.name) for tag in Tag.query.order_by("name")]
    ...
    if request.method == "POST" and form.validate_on_submit():  # при создании
статьи
        if form.tags.data:  # если в форму были переданы теги (были выбраны)
           selected_tags = Tag.query.filter(Tag.id.in_(form.tags.data))
           for tag in selected_tags:
              article.tags.append(tag)  # добавляем выбранные теги к статье
    ...
```

Итоги

На занятии мы создали модель тега, создали связь с моделью статьи, научились работать со связью «многие ко многим» на примере статьи и тегов, познакомились с функцией `joinedload`.

Практическое задание

1. Добавить в свой проект на Flask модель тега. Создать связь с моделью статьи.
2. Обновить форму создания статьи — добавить туда выбор тегов.
3. Вывести на страницу просмотра статьи связанные теги.

Дополнительные материалы

1. https://docs.sqlalchemy.org/en/13/orm/basic_relationships.html.
2. https://docs.sqlalchemy.org/en/13/orm/loading_relationships.html.