GeekBrains

Flask

Модели автора и статьи, связи one-to-one и one-to-many

На этом уроке

Научимся создавать различные связи в базе данных:

a. one-to-one;
b. one-to-many.

Оглавление

Теория

Типы связей в реляционных базах данных

Логику соединения таблиц в БД важно понять с самого начала изучения SQL, так как наверняка Вы не будете писать запросы только к одной таблице.

Всего существует 3 типа связей:

- «Один к одному»;
- «Один ко многим»;
- «Многие ко многим».

Связь «Один к одному»

Связь «Один к одному» образуется, когда ключевой столбец (идентификатор) присутствует в другой таблице, в которой тоже является ключом, либо свойствами столбца задана его уникальность (одно и то же значение не может повторяться в разных строках).

На практике связь «Один к одному» наблюдается нечасто. Например, она может возникнуть, когда требуется разделить данные одной таблицы на несколько отдельных таблиц с целью безопасности.

Пример:

Представьте, что базой данных пользуются несколько менеджеров и аналитиков, а таблица «Сотрудники» содержит столбцы с персональными данными. Следовательно, доступ к персональным данным может получить любой из упомянутых работников.

Чтобы устранить возможность утечки конфиденциальной информации, принимается решение о переносе информации о паспортных данных в отдельную таблицу, доступ к которой предоставляется ограниченному кругу лиц.

Связь «Один ко многим»

В типе связей «Один ко многим» одной записи первой таблицы соответствует несколько записей в другой таблице.

Рассмотрим связь данных между должностями и сотрудниками, которая относится к рассматриваемому типу.

Записи должностей в таблице «Должность» уникальны, так как нет смысла повторно создавать имеющуюся запись. Записи в таблице «Сотрудники» также уникальны, но несколько различных сотрудников могут находиться на одинаковой должностной позиции.

Таблица «Должность» находится на стороне «один» (связанный столбец является первичным ключом), а таблица «Сотрудники» находится на стороне «многие» (такой столбец является внешним ключом).

Связь «Многие ко многим»

Если нескольким записям из одной таблицы соответствует несколько записей из другой таблицы, то такая связь называется «Многие ко многим» и организовывается посредством связывающей таблицы. Например, у новостной статьи может быть несколько тематических тегов. И каждый из тегов может быть связан с несколькими статьями.

Практическое задание

Создаём модель Автора в blog/models/author.py, добавляем связь с пользователем:

```
from sqlalchemy import Column, Integer, ForeignKey
from sqlalchemy.orm import relationship

from blog.models.database import db


class Author(db.Model):
    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey("user.id"), nullable=False)

    user = relationship("User", back_populates="author")
```

Сразу же делаем импорт в blog/models/__init__.py:

```
from blog.models.author import Author
```

И добавляем обратную связь в модель пользователя в blog/models/user.py:

```
from sqlalchemy.orm import relationship


class User(db.Model, UserMixin):
    ...

    author = relationship("Author", uselist=False, back_populates="user")
```

Создаём и выполняем миграцию (будет создана таблица автора, модель пользователя не изменилась относительно БД, обратная связь работает на уровне SQLAlchemy).

Создаём страницу списка авторов, добавляем в навигацию

Создаём темплейт blog/templates/authors/list.html. Отрисовываем там список авторов:

```
{% extends 'base.html' %}

{% block title %}
  Authors list
{% endblock %}

{% block body %}
  <h1>Authors</h1>
  <div>
    {% if authors %}
      <ul>
        {% for author in authors %}
          <li>
            #{{ author.id }} ({{ author.user.username }})
          </li>
        {% endfor %}
      </ul>
    {% else %}
      <h3>No authors yet</h3>
    {% endif %}
  </div>
{% endblock %}
```

Создаём новый блупринт в blog/views/authors.py, добавляем view для отрисовки списка авторов:

```
from flask import Blueprint, render_template
from blog.models import Author

authors_app = Blueprint("authors_app", __name__)


@authors_app.route("/", endpoint="list")
def authors_list():
    authors = Author.query.all()
    return render_template("authors/list.html", authors=authors)
```

Подключаем блупринт авторов в blog/app.py, указываем префикс:

```
from blog.views.authors import authors_app

app.register_blueprint(authors_app, url_prefix="/authors")
```

Добавляем ссылку в навигационную панель в blog/templates/base.html:

```
{% for (endpoint, label) in [
                ('users_app.list', 'Users'),
                ('articles_app.list', 'Articles'),
                ('authors_app.list', 'Authors'),
              ] %}
```

Создаём модель статей, настраиваем связь

Описываем модель в blog/models/article.py:

```
from sqlalchemy import Column, Integer, ForeignKey
from sqlalchemy.orm import relationship

from blog.models.database import db


class Article(db.Model):
    id = Column(Integer, primary_key=True)
    author_id = Column(Integer, ForeignKey("author.id"))

    author = relationship("Author", back_populates="articles")
```

Сразу импортируем модель в blog/models/__init__.py и добавляем в __all__

```
from blog.models.article import Article

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

Добавляем связь в blog/models/author.py:

```
class Author(db.Model):
    ...

    articles = relationship("Article", back_populates="author")
```

Создаём и выполняем миграцию.

Дополняем модель Article в blog/models/article.py:

```
from datetime import datetime
from sqlalchemy import Column, Integer, ForeignKey, String, Text, DateTime, func

class Article(db.Model):
    ...

    title = Column(String(200), nullable=False, default="", server_default="")
    body = Column(Text, nullable=False, default="", server_default="")
    dt_created = Column(DateTime, default=datetime.utcnow, server_default=func.now())
    dt_updated = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)
```

И снова создаём и выполняем миграцию.

Создаём статьи, view для списка статей, деталей статьи, показываем автора

Начинаем с формы для создания статей `CreateArticleForm` в модуле blog/forms/article.py:

```
from flask_wtf import FlaskForm
from wtforms import StringField, TextAreaField, SubmitField, validators


class CreateArticleForm(FlaskForm):
    title = StringField(
        "Title",
        [validators.DataRequired()],
    )
    body = TextAreaField(
        "Body",
        [validators.DataRequired()],
    )
    submit = SubmitField("Publish")
```

Добавляем шаблон blog/templates/articles/create.html и выводим форму для создания новой статьи:

```
{% extends 'base.html' %}

 {% block title %}
   Create article
 {% endblock %}

 {% block body %}
   <h1>Create article</h1>
   <div>
     {% if error %}
       <div class="alert alert-danger" role="alert">
         {{ error }}
       </div>
     {% endif %}
     <form method="post">
       {{ form.csrf_token }}

       {% for field_name in ['title', 'body'] %}
       {% set field = form[field_name] %}
         <div class="my-3">
           {{ field.label(class="form-label") }}
           {{ field(class="form-control") }}
         </div>
       {% endfor %}
       {{ form.submit(class="btn btn-primary my-3") }}
     </form>
   </div>
 {% endblock %}
```

Также делаем шаблон для вывода статьи blog/templates/articles/details.html:

```
{% extends 'base.html' %}

 {% block title %}
   Article #{{ article.id }}: {{ article.title }}
 {% endblock %}

 {% block body %}
   <h1>{{ article.title }}</h1>
   <div>
     <p>{{ article.body }}</p>
     <div>Published {{ article.dt_created }}</div>
     <div>by {{ article.author.user.username }}</div>
   </div>
 {% endblock %}
```

Добавляем в список статей ссылку на создание новой (редактируем blog/templates/articles/list.html):

```
<div>
    <ul>
      {% for article in articles %}
        <li>
          <a href="{{ url_for('articles_app.details', article_id=article.id) }}">{{
article.title }}</a>
           by {{ article.author.user.username }}
        </li>
      {% endfor %}
    </ul>
    <hr>
    <a href="{{ url_for('articles_app.create') }}">Create new</a>
  </div>
```

Правим view в blog/views/articles.py: вывод статей, отображение выбранной статьи.

```
from flask import Blueprint, render_template, request, current_app, redirect, url_for
from flask_login import login_required, current_user
from sqlalchemy.exc import IntegrityError
from werkzeug.exceptions import NotFound

from blog.models.database import db
from blog.models import Author, Article
from blog.forms.article import CreateArticleForm

articles_app = Blueprint("articles_app", __name__)


@articles_app.route("/", endpoint="list")
def articles_list():
    articles = Article.query.all()
    return render_template("articles/list.html", articles=articles)


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

Добавляем view для создания статей. Тут необходима авторизация пользователя. При создании статьи к модели пользователя добавляется модель автора, если её ещё нет. Модель статьи привязывается к модели автора.

```
@articles_app.route("/create/", methods=["GET", "POST"], endpoint="create")
@login_required
def create_article():
    error = None
    form = CreateArticleForm(request.form)
    if request.method == "POST" and form.validate_on_submit():
        article = Article(title=form.title.data.strip(), body=form.body.data)
        db.session.add(article)
        if current_user.author:
            # use existing author if present
            article.author = current_user.author
        else:
            # otherwise create author record
            author = Author(user_id=current_user.id)
            db.session.add(author)
            db.session.flush()
            article.author = current_user.author

        try:
            db.session.commit()
        except IntegrityError:
            current_app.logger.exception("Could not create a new article!")
            error = "Could not create article!"
        else:
            return redirect(url_for("articles_app.details", article_id=article.id))

    return render_template("articles/create.html", form=form, error=error)
```

Запускаем и можем проверить возможность создания статей, переход к созданной статье, отображение новых статей в списке, отображение авторов.

Итоги

Мы создали модели автора и статей, создали связи: у автора «один к одному» с моделью пользователя, а также у автора «один ко многим» со статьями. Текст статьи хранится чистым текстом.

Также реализовали вывод доступных статей на главную страницу и возможность прочитать любую статью целиком.

Практическое задание

1. Добавить в свой проект на Flask модели автора, статьи. Создать связи.
2. Вывести на страницу список статей.
3. Добаить возможность перейти к просмотру полной статьи.

Дополнительные материалы

1. Связи между таблицами базы данных / Хабр.
2. Relational database — Wikipedia.
3. https://docs.sqlalchemy.org/en/13/orm/basic_relationships.html.