Week 6 - Databases in python

Содержание практической работы:

1. Схема базы данных
2. Подключение к базам данных
3. Создание таблиц
4. Вставка записей
5. Извлечение записей
6. Обновление содержания
7. Удаление записей таблицы

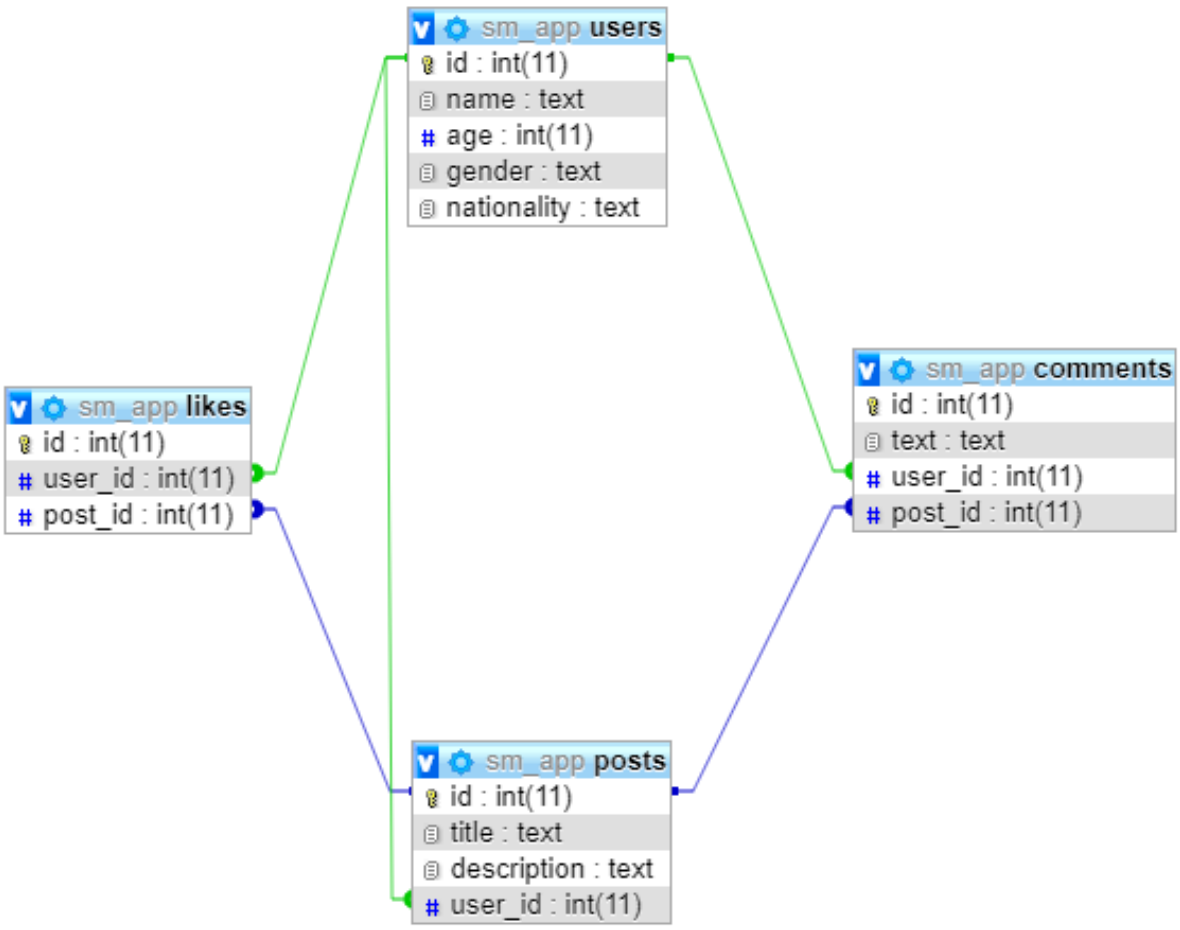
В каждом разделе по три подраздела: SQLite, MySQL и PostgreSQL.

1. Схема базы данных для обучения

В этом уроке мы разработаем очень маленькую базу данных приложения для социальных сетей. База данных будет состоять из четырех таблиц:

1. users
2. posts
3. comments
4. likes

Схема базы данных показана на рисунке ниже.



Пользователи (users) и публикации (posts) будут находиться иметь тип связи один-ко-многим: одному читателю может понравиться несколько постов. Точно так же один и тот

же юзер может оставлять много комментариев (comments), а один пост может иметь несколько комментариев. Таким образом, и users, и posts по отношению к commentsимеют тот же тип связи. А лайки (likes) в этом плане идентичны комментариям.

2. Подключение к базам данных

Прежде чем взаимодействовать с любой базой данных через SQL-библиотеку, с ней необходимо связаться. В этом разделе мы рассмотрим, как подключиться из приложения Python к базам данных SQLite , MySQL и PostgreSQL. Рекомендуем сделать собственный .py файл для каждой из трёх баз данных.
Примечание. Для выполнения разделов о MySQL и PostgreSQL необходимо самостоятельно запустить соответствующие серверы. Для быстрого ознакомления с тем, как запустить сервер MySQL, ознакомьтесь с разделом MySQL в публикации Запуск проекта Django (англ.). Чтобы узнать, как создать базу данных в PostgreSQL, перейдите к разделу Setting Up a Database в публикации Предотвращение атак SQL-инъекций с помощью Python (англ.).

PostgreSQL

Как и в случае MySQL, для PostgreSQL в стандартной библиотеке Python нет модуля для взаимодействия с базой данных. Но и для этой задачи есть решение – модуль psycopg2:

```
pip install psycopg2
```

Определим функцию create_connection() для подключения к базе данных PostgreSQL:

```
from psycopg2 import OperationalError

def create_connection(db_name, db_user, db_password, db_host, db_port):
    connection = None
    try:
        connection = psycopg2.connect(
            database=db_name,
            user=db_user,
            password=db_password,
            host=db_host,
            port=db_port,
        )
        print("Connection to PostgreSQL DB successful")
    except OperationalError as e:
        print(f"The error '{e}' occurred")
    return connection
```

Подключение осуществляется через интерфейс psycopg2.connect(). Далее используем написанную нами функцию:

```
connection = create_connection(
    "postgres", "postgres", "abc123", "127.0.0.1", "5432"
)
```

Теперь внутри дефолтной БД postgres нужно создать базу данных sm_app. Ниже определена соответствующая функция create_database():

```
def create_database(connection, query):
    connection.autocommit = True
    cursor = connection.cursor()
    try:
        cursor.execute(query)
```

```
        print("Query executed successfully")
    except OperationalError as e:
        print(f"The error '{e}' occurred")

create_database_query = "CREATE DATABASE sm_app"
create_database(connection, create_database_query)
```

Запустив вышеприведенный скрипт, мы увидим базу данных sm_app на своем сервере PostgreSQL. Подключимся к ней:

```
connection = create_connection(
    "sm_app", "postgres", "abc123", "127.0.0.1", "5432"
)
```

Здесь 127.0.0.1 и 5432 это соответственно IP-адресу и порт хоста сервера.

3. Создание таблиц

В предыдущем разделе мы увидели, как подключаться к серверам баз данных SQLite, MySQL и PostgreSQL, используя разные библиотеки Python. Мы создали базу данных sm_app на всех трех серверах БД. В данном разделе мы рассмотрим, как формировать таблицы внутри этих трех баз данных.

Как обсуждалось ранее, нам нужно получить и связать четыре таблицы:

1. users
2. posts
3. comments
4. likes

PostgreSQL

Применение библиотеки psycopg2 в execute_query() также подразумевает работу с cursor:

```
def execute_query(connection, query):
    connection.autocommit = True
    cursor = connection.cursor()
    try:
        cursor.execute(query)
        print("Query executed successfully")
    except OperationalError as e:
        print(f"The error '{e}' occurred")
```

Мы можем использовать эту функцию для организации таблиц, вставки, изменения и удаления записей в вашей базе данных PostgreSQL.

Создадим внутри базы данных sm_app таблицу users :

```
create_users_table = """
CREATE TABLE IF NOT EXISTS users (
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  age INTEGER,
  gender TEXT,
  nationality TEXT
```

```
)
"""

execute_query(connection, create_users_table)
```

Запрос на создание таблицы users в PostgreSQL немного отличается от SQLite и MySQL. Здесь для указания столбцов с автоматическим инкрементом используется ключевое слово SERIAL. Кроме того, отличается способ указания ссылок на внешние ключи:

```
create_posts_table = """
CREATE TABLE IF NOT EXISTS posts (
  id SERIAL PRIMARY KEY,
  title TEXT NOT NULL,
  description TEXT NOT NULL,
  user_id INTEGER REFERENCES users(id)
)
"""

execute_query(connection, create_posts_table)
```

4. Вставка записей

В предыдущем разделе мы разобрали, как развертывать таблицы в базах данных SQLite, MySQL и PostgreSQL с использованием различных модулей Python. В этом разделе узнаем, как вставлять записи.

PostgreSQL

В предыдущем подразделе мы познакомились с двумя подходами для вставки записей в таблицы баз данных MySQL. В psycopg2 используется второй подход: мы передаем SQL-запрос с заполнителями и списком записей методу execute(). Каждая запись в списке должна являться кортежем, значения которого соответствуют значениям столбца в таблице БД. Вот как мы можем вставить пользовательские записи в таблицу users:

```
users = [
    ("James", 25, "male", "USA"),
    ("Leila", 32, "female", "France"),
    ("Brigitte", 35, "female", "England"),
    ("Mike", 40, "male", "Denmark"),
    ("Elizabeth", 21, "female", "Canada"),
]

user_records = ", ".join(["%s"] * len(users))

insert_query = (
    f"INSERT INTO users (name, age, gender, nationality) VALUES {user_records}"
)

connection.autocommit = True
cursor = connection.cursor()
cursor.execute(insert_query, users)
```

Список users содержит пять пользовательских записей в виде кортежей. Затем мы создаём строку с пятью элементами-заполнителями (%s), соответствующими пяти пользовательским записям. Строка-заполнитель объединяется с запросом, который вставляет записи в таблицу users. Наконец, строка запроса и пользовательские записи передаются в метод execute().
Следующий скрипт вставляет записи в таблицу posts:

```
posts = [
    ("Happy", "I am feeling very happy today", 1),
    ("Hot Weather", "The weather is very hot today", 2),
    ("Help", "I need some help with my work", 2),
    ("Great News", "I am getting married", 1),
    ("Interesting Game", "It was a fantastic game of tennis", 5),
    ("Party", "Anyone up for a late-night party today?", 3),
]

post_records = ", ".join(["%s"] * len(posts))

insert_query = (
    f"INSERT INTO posts (title, description, user_id) VALUES {post_records}"
)

connection.autocommit = True
cursor = connection.cursor()
cursor.execute(insert_query, posts)
```

По той же методике можно вставить записи в таблицы comments и likes.

5. Извлечение данных из записей

PostgreSQL

Процесс выбора записей из таблицы PostgreSQL с помощью модуля psycopg2 тоже похож на SQLite и MySQL. Снова используем cursor.execute(), затем метод fetchall() для выбора записей из таблицы. Следующий скрипт выбирает все записи из таблицы users:

```
def execute_read_query(connection, query):
    cursor = connection.cursor()
    result = None
    try:
        cursor.execute(query)
        result = cursor.fetchall()
        return result
    except OperationalError as e:
        print(f"The error '{e}' occurred")

select_users = "SELECT * FROM users"
users = execute_read_query(connection, select_users)

for user in users:
    print(user)
```

6. Обновление записей таблицы

PostgreSQL

Запрос на обновление PostgreSQL аналогичен SQLite и MySQL.

7. Удаление записей таблицы

SQLite

В качестве примера удалим комментарий с id равным 5:

```
delete_comment = "DELETE FROM comments WHERE id = 5"
execute_query(connection, delete_comment)
```

Теперь, если мы извлечем все записи из таблицы comments, то увидим, что пятый комментарий был удален. Процесс удаления в MySQL и PostgreSQL идентичен SQLite:

Заключение

В этом руководстве мы разобрались, как применять три распространенные библиотеки Python для работы с реляционными базами данных. Научившись работать с одним из модулей sqlite3, mysql-connector-python и psycopg2, вы легко сможете перенести свои знания на другие модули и оперировать любой из баз данных SQLite, MySQL и PostgreSQL.

Существуют также библиотеки для работы с SQL и объектно-реляционными отображениями, такие как SQLAlchemy и Django ORM, которые автоматизируют задачи взаимодействия Python с базами данных.

Если вам интересна тематика работы с базами данных с помощью Python, напишите об этом в комментариях – мы подготовим дополнительные материалы.