МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Национальный исследовательский университет «МЭИ»

Институт ИВТИ

Кафедра УИТ

Производственная практика: научно-исследовательская работа
ОСНОВЫ РАБОТЫ С РЕЛЯЦИОННЫМИ БАЗАМИ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА Python

Выполнила

Ишутина Е. И.,

А-01-22

Проверил

Полотнов М. М.,

доц. кафедры Управления и интеллектуальных технологий

Оценка

_____

Москва, 2024

ЭТАП 2 НИР. РАБОТА С ДАННЫМИ SQLite В ПРОГРАММАХ НА ЯЗЫКЕ Python

2.1. Подготовительные операции

Перед началом работы проверено наличие в каталоге Python необходимой библиотеки DDLS/sqlite3.dll и пакета Lib/sqlite3. Они есть, поэтому импортируем этот модуль:

```
>>> import sqlite3
```

Для работы создана отдельная директория и установлена как рабочая:

```
>>> import os
>>> os.chdir(“d:/STUDY/NIR”)
```

Дальнейшие действия будут содержать типовые операции с базами данных, применимые в программах на Python. А именно:

– создание базы данных и входящих в нее таблиц;
– занесение записей в таблицы;
– удаление записей;
– изменение структуры таблиц (удаление или добавление полей, изменение их параметров); – изменение значений в заданных полях и записях.

2.2. Создание БД bd1.sqlite с двумя таблицами: stud и sessija

Для начала работы создается БД с таблицей stud, где хранится информация о студентах, обучающихся по некоторому направлению, и с таблицей sessija, где хранятся данные об их успеваемости.

Подключимся к БД с помощью метода sqlite3.connect(). Функция sqlite3.connect() в Python при попытке подключения к несуществующей базе данных создаст в рабочей директории новую пустую базу данных с указанным именем, если указан файл, который не существует.

```
>>> con = sqlite3.connect("bd1.sqlite")
```

Функция возвращает объект типа объект типа sqlite3.Connection, выводя адрес этого объекта в памяти.

```
>>> con
<sqlite3.Connection object at 0x0000018FAB6B88A0>
```

Далее необходимо создать переменную-курсор. Она нужна, чтобы связать работу БД и Python. С помощью нее отправляются запросы. Аналогично, это объект класса sqlite3.Cursor.

```
>>> cur = con.cursor()
```

В строковую переменную запишем DDL-запросы, где укажем имя, атрибуты (поля) и типы полей для таблицы stud и для таблицы sessija:

```
>>> sql = """\
CREATE TABLE stud (id TEXT PRIMARY KEY,
fio TEXT, email TEXT);
CREATE TABLE sessija (id TEXT SECONDARY KEY,
Disz TEXT, Ozenka INTEGER, Prepod TEXT, Dat TEXT)
"""
```

Примечание: обратный слэш в начале убирает автоматический перенос строки, который обычно происходит в многострочных строках в Python.

Теперь эти запросы надо выполнить:

```
>>> cur.executescript(sql)
<sqlite3.Cursor object at 0x000001A1411D6D50>
```

Примечание: используется именно метод `executescript()`, потому что, в отличие от метода `execute()`, он может выполнить несколько запросов, разделенных точкой с запятой, за раз.

Этап закончен, так что нужно освободить ресурсы памяти, закрыв курсор и подключение к БД:

```
>>> cur.close()
>>> con.close()
```

2.3. Запись данных в таблицы stud и sessija

Необходимо установить подключение к ранее созданной БД и создать переменную-курсор:

```
>>> con = sqlite3.connect('bd1.sqlite')
>>> cur = con.cursor()
```

Далее создается переменная с запросом вставки записи в таблицу stud:

```
    sql = """\
    INSERT INTO stud (id, fio, email) VALUES
('0020223229', 'ИШУТИНА Е.И.', 'ishutinayi@mpei.ru')
    """
```

Поскольку запрос только один, ограничимся методом execute():

```
>>> cur.execute(sql)
<sqlite3.Cursor object at 0x000001A1411D6C70>
```

Важно понимать, что сейчас изменения произошли только в рамках транзакции (атомарной логической единицы работы с БД), то есть в рамках временной памяти, где их можно отменить. Чтобы изменения стали постоянными и применились к БД, необходимо сделать коммит:

```
>>> con.commit()
```

После этого можно закрыть курсор и подключение к БД:

```
>>> cur.close()
>>> con.close()
```

2.4. Множественная вставка данных в таблицы БД

Если нужно вставить сразу много данных, лучше записать их в список кортежей в нужном порядке, а затем передать этот список как параметр при применении запроса. Создадим такой список:

```
    >>> arr = [('0020223229', 'ТАУ', 'Сидорова Е.Ю. ',
4, '21.01.2025'),
    ( '0020223351', ' ПО АС', 'Козлюк Д.А. ',5,
'25.12.2024')]
```

Подключение к БД, создание курсора:

```
    >>> con=sqlite3.connect('bd1.sqlite')
    >>> cur=con.cursor()
```

Формирование запроса:

```
    >>> sql="""\
INSERT INTO sessija (id, Disz, Ozenka, Prepod, Dat)
VALUES (?,?,?,?,?)
"""
```

Здесь вместо конкретных значений проставлены вопросительные знаки по числу записываемых полей. Теперь при вставке можно использовать специальный метод для множественной вставки. Объект с данными передается как аргумент. В результате в таблицу sessija вставлены 2 строки с данными из кортежей в arr.

Примечание: в общем случае, может быть передан любой итерируемый объект. Чаще используются в силу удобства использовать кортеж кортежей или список кортежей.

Выполнение коммита, отключение от БД:

```
    >>> con.commit()
    >>> cur.close()
    >>> con.close()
```

2.5. Чтение и отображение содержимого одной из таблиц в БД

Теперь обращение к БД произойдет с помощью программы test3.py. В этом файле описана функция `select_cmd()`. Она не принимает никаких параметров, используя имя таблицы из глобальной области видимости переменных. Ее назначение – вывести все поля из заданной таблицы. Затем у пользователя запрашивается ввод имени файла и имени таблицы, и к нему

применяется вышеописанная функция. Затем закрывается курсор и БД. Ниже приведен код, пояснения по его работе добавлены курсивом.

```
    import os, sqlite3

    def select_cmd():
        sql = "SELECT * FROM {}".format(tblname)
        with con: # контекстный менеджер with
автоматически закрывает соединение с БД и освобождает
ресурсы
            data = cur.execute(sql).fetchall()
    # fetchall() возвращает список, где каждый элемент
— это кортеж, представляющий одну строку результата.
        return (data)

    dbname = ""

    while not os.path.isfile(dbname):
        dbname = input("Укажите имя файла SQlite: ")
        if os.path.isfile(dbname): break  # ввод
запрашивается до тех пор, пока пользователь не введёт
существующее имя файла
        print("Нет такого файла!")

    tblname = input("Укажите имя таблицы: ")
    con = sqlite3.connect(dbname)
    cur = con.cursor()
    dan = select_cmd()
    nzap = len(dan)

    print("Таблица:", tblname, "из БД", dbname)
    for i in range(nzap): #для вывода каждого кортежа с
новой строки
        print(dan[i])

    cur.close() # SELECT не изменяет БД, так что коммит
можно не делать
    con.close()
```

Тестирование получившейся программы:

- Указание существующего файла и существующей таблицы stud:

```
Укажите имя файла SQlite: bd1.sqlite
Укажите имя таблицы: stud
Таблица: stud из БД bd1.sqlite
('0020223229', 'ИШУТИНА Е.И.', 'ishutinayi@mpei.ru')
('0020223351', 'ЧУДОВ Б.И.', 'chudovbi@mpei.ru')
```

- Указание существующего файла и существующей таблицы sessija

```
Укажите имя файла SQlite: bd1.sqlite
Укажите имя таблицы: sessija
Таблица: sessija из БД bd1.sqlite
('0020223229', 'ТАУ', 'Сидорова Е.Ю. ', '4',
'21.01.2025')
('0020223351', ' ПО    АС', 'Козлюк    Д.А. ', '5',
'25.12.2024')
```

- Указание несуществующего файла:

```
Укажите имя файла SQlite: doesntexist.sqlite
Нет такого файла!
Укажите имя файла SQlite:
```

- Указание несуществующей таблицы:

```
Укажите имя файла SQlite: bd1.sqlite
Укажите имя таблицы: kurs
…
sqlite3.OperationalError: no such table: kurs
```

2.6. Чтение данных из таблицы БД

Попробуем получить все записи из таблицы stud базы данных bd1.sqlite. Как обычно, начнём с подключения к БД:

```
>>> con = sqlite3.connect("bd1.sqlite")
>>> cur = con.cursor()
```

Запрос к БД можно отправить, не создавая для этого отдельный строковый объект. Так удобнее делать, когда запрос занимает только одну строку.

```
>>> cur.execute("SELECT * FROM stud")
```

Запишем содержимое курсора в список кортежей:

```
>>> cur.execute("SELECT * FROM stud")
```

Закрытие курсора и соединения с БД. Коммит не требуется.

```
>>> cur.close()
>>> con.close()
```

Можно отобразить список кортежей в одну строку, а можно сделать отображение в цикле, чтобы каждый элемент-кортеж начинался с новой строки:

```
>>> print(ar)
[('0020223229', 'ИШУТИНА Е.И.', 'ishutinayi@mpei.ru'), ('0020223351', 'ЧУДОВ Б.И.', 'chudovbi@mpei.ru')]

>>> for i in ar: print(i)

('0020223229', 'ИШУТИНА Е.И.', 'ishutinayi@mpei.ru')
('0020223351', 'ЧУДОВ Б.И.', 'chudovbi@mpei.ru')
```

2.7. Получение списка таблиц в составе БД

Если, например, мы работаем с базой данных, которая была создана другим человеком, первым делом стоит узнать, какие таблицы в нее входят. Для этого выполняется подключение к БД VUZ.sqlite (файл с ней находится в рабочей директории):

```
>>> con = sqlite3.connect("vuz.sqlite")
>>> cur = con.cursor()
```

Далее формируется переменная с SQL-запросом:

```
>>> sql="""\
SELECT name FROM sqlite_master WHERE type IN
```

```
('table','view') AND name NOT LIKE 'sqlite_%' UNION ALL
SELECT
name FROM sqlite_temp_master WHERE type IN
('table','view') ORDER BY 1;"""
```

Рассмотрим подробнее этот запрос. Он состоит из двух частей, соединенных между собой с помощью оператора `UNION ALL`. Этот оператор объединяет результаты двух запросов `SELECT`.

Первая часть: `SELECT name FROM sqlite_master WHERE type IN ('table','view') AND name NOT LIKE 'sqlite_%'`. Эта часть возвращает имена всех пользовательских таблиц и представлений в основной базе данных.

- `sqlite_master` — это специальная системная таблица в SQLite, которая хранит метаданные обо всех таблицах, триггерах, представлениях и других объектах в базе данных. Отсюда получится вытянуть все названия таблиц.

- `WHERE type IN ('table', 'view')` — оператор фильтрации, который оставляет только те метаданные, которые имеют тип таблицы (table) или представления (view).

Примечание: представления (views) в базах данных — это виртуальные таблицы, которые отображают результаты SQL-запросов. Представление выглядит как обычная таблица, но данные в ней фактически не хранятся. Вместо этого она сохраняет SQL-запрос, который выполняется всякий раз, когда происходит обращение к этому представлению.

- `AND name NOT LIKE 'sqlite_%'` — еще одно условие, которое отсекает системные объекты (их имена начинаются с префикса ‘`sqlite_`’).

Вторая часть: `SELECT name FROM sqlite_temp_master WHERE type IN ('table','view')`. Этот запрос возвращает то же, но для временных таблиц, созданных в рамках одной транзакции.

- `sqlite_temp_master` — это системная таблица, которая хранит метаданные о временных таблицах и представлениях, которые были созданы в текущей сессии.

`ORDER BY 1;` — общий для двух подзапросов оператор сортировки. Результат сортируется по первому столбцу (по именам таблиц и представлений в библиографическом порядке).

Выполнение этого запроса:

```
>>> cur.execute(sql)
<sqlite3.Cursor object at 0x00000233B29FA3B0>
```

Примечание: вся конструкция, несмотря на наличие двух `SELECT`, является одним запросом, там что метод .executescripts() здесь не нужен.

Коммитить запросы `SELECT` необязательно, т.к. они не изменяют структуру таблицы, но, тем не менее, не запрещается. Сделаем это, отобразим результат и закроем подключение.

```
>>> con.commit()
>>> cur.fetchall()
[('vuz_rf',), ('vuzkart',), ('vuzstat',)]
>>> cur.close()
>>> con.close()
```

2.8. Считывание информации о структуре таблицы, имеющейся в БД

Для работы с неизвестной ранее БД мало иметь только имена таблиц. Надо знать, какие поля (атрибуты) у них есть. Получим список имен всех полей таблицы `vuzkart`.

Подключение к таблице:

```
>>> con = sqlite3.connect("vuz.sqlite")
```

Определим функцию `my_factory(c, r)`. Её параметры - это курсор, который содержит описание столбцов в запросе (c), а также строка данных, которую мы получили в результате SQL-запроса (r). Внутри этой функции

создается пустой словарь. Атрибут `c.description` содержит информацию о столбцах, а именно – их имена.

Примечание: метод `enumerate()` возвращает объект-итератор к заданной коллекции, который на каждой итерации возвращает кортеж из двух элементов: индекс и значение элемента из итерируемого объекта.

Далее функция для каждого столбца в описании курсора создаёт пару «ключ-значение», где ключ — это имя столбца, а значение — соответствующее значение из строки данных r. Одновременно функция добавляет второй ключ — это индекс столбца (порядковый номер), чтобы доступ к данным был возможен как по имени столбца, так и по номеру индекса.

```
>>> def my_factory(c,r):
    d = {}
    for i, name in enumerate(c.description):
        d[name[0]] = r[i]
        d[i] = r[i]
    return(d)
```

Мы создали эту функцию для того, чтобы теперь установить её как фабрику строк.

Примечание: фабрика строк – это специальный механизм, который позволяет изменять формат возвращаемых данных при выполнении SQL-запросов. По умолчанию они возвращаются как кортеж, но это не всегда удобно. В данном случае мы прописали функцию, которая возвращает словарь, задали её как фабрику строк, и теперь результат запроса будет возвращаться как словарь, а не как кортеж.

```
>>> con.row_factory = my_factory
```

Теперь можно выполнить привычние действия: отправить запрос, преобразовать его в читабельную коллекцию и вывести её, отключившись от БД и закрыв курсор.

```
>>> cur = con.cursor()
>>> cur.execute('SELECT * FROM vuzkart')
<sqlite3.Cursor object at 0x00000233B29FA3B0>
```

```
>>> ar=cur.fetchone()
>>> cur.close()
>>> con.close()
>>> fld_names=list(ar.keys())[::2]
```

`ar.keys()` включает в себя как имена столбцов, так и индексы (по одному ключу на каждый столбец). Поэтому используется срез [::2], который выбирает каждый второй элемент из списка ключей. В данном случае это будет список имён столбцов, потому что в словаре ключи расположены чередующимися: сначала имя столбца, потом его индекс.

```
>>> print(fld_names)
['codvuz', 'z1', 'z1full', 'z2', 'z2ustav',
'foundyear', 'z8', 'z9', 'z12', 'e_mail', 'www', 'z15',
'region', 'city', 'status', 'oblname', 'gr_ved',
'prof']
```

2.9. Ввод данных по запросу с клавиатуры и занесение их в таблицу

Подключение к таблице и создание курсора:

```
>>> con=sqlite3.connect('bd1.sqlite')
>>> cur = con.cursor()
```

Создание двух пустых объектов-списков. В vv будут добавляться строки, содежащие номер студенческого билета, ФИО и адрес электронной почты. Затем эти данные будут запакованы в кортеж и добавлены в список ar.

```
>>> ar = []
>>> vv = []
>>> vv.append(input("Номер студенческого билета "))
Номер студенческого билета 0020221095
>>> vv.append(input('ФИО студента = '))
ФИО студента = МАЧУЛИНА Д.В.
>>> vv.append(input('e-mail студента ='))
e-mail студента =MachulinaDV@mpei.ru
>>> ar.append(tuple(vv))
```

Перед отправкой запроса о вставке записи проконтролируем, что объект сформировался правильно:

```
>>> print(ar)
```

```
    [('0020221095', 'МАЧУЛИНА Д.В.',
'MachulinaDV@mpei.ru')]
```

Формируем SQL-запрос о добавлении записи в таблицу аналогично тому, как это было в пункте 2.4:

```
    >>> sql='INSERT INTO stud (id,fio,email) VALUES
(?,?,?)'
    >>> cur.executemany(sql,ar)
    <sqlite3.Cursor object at 0x00000233B29FA500>
```

Естественно, INSERT изменяет таблицу, так что изменения необходимо подтвердить коммитом:

```
    >>> con.commit()
```

Проверим, как выглядит таблица теперь:

```
    >>> cur.execute('SELECT * FROM stud')
    <sqlite3.Cursor object at 0x00000233B29FA500>
    >>> ar1=cur.fetchall()
    >>> for i in ar1: print(i)

('0020223229', 'ИШУТИНА Е.И.', 'ishutinayi@mpei.ru')
('0020223351', 'ЧУДОВ Б.И.', 'chudovbi@mpei.ru')
('0020221095', 'МАЧУЛИНА Д.В.', 'MachulinaDV@mpei.ru')
```

Закроем подключение и курсор:

```
    >>> cur.close()
    >>> con.close()
```