Лабораторная работа №14
SQLite3

1 Цель работы

Изучить SQLite3 и научиться применять полученные знания на практике.

2 Краткая теория

SQLite – это C библиотека, реализующая легковесную дисковую базу данных (БД), не требующую отдельного серверного процесса и позволяющую получить доступ к БД с использованием языка запросов SQL. Некоторые приложения могут использовать SQLite для внутреннего хранения данных. Также возможно создать прототип приложения с использованием SQLite, а затем перенести код в более многофункциональную БД, такую как PostgreSQL или Oracle.

Руководство по SQLite: https://metanit.com/sql/sqlite/.

2.1 Создание соединения

Чтобы воспользоваться SQLite3 в Python необходимо импортировать модуль sqlite3, а затем создать объект подключения к БД.

Объект подключения создается с помощью метода connect():

```
import sqlite3

con = sqlite3.connect('mydatabase.db')
```

2.2 Курсор

Для выполнения операторов SQL, нужен объект курсора, создаваемый методом cursor().

Курсор SQLite3 – это метод объекта соединения. Для выполнения операторов SQLite3 сначала устанавливается соединение, а затем создается объект курсора с использованием объекта соединения следующим образом:

```
con = sqlite3.connect('mydatabase.db')
cursorObj = con.cursor()
```

Теперь можно использовать объект курсора для вызова метода execute() для выполнения любых запросов SQL.

2.3 Создание базы данных

После создания соединения с SQLite, файл БД создается автоматически, при условии его отсутствия. Данный файл создаётся на диске, но также можно создать базу данных в оперативной памяти, используя параметр «:memory:» в методе connect. При этом база данных будет называться инмемори.

Рассмотрим приведенный ниже код, в котором создается БД с блоками try, except и finally для обработки любых исключений:

```
import sqlite3
from sqlite3 import Error


def sql_connection():
    try:
        con = sqlite3.connect(':memory:')
```

```
        print("Connection is established: Database is created in memory")
    except Error:
        print(Error)
    finally:
        con.close()


sql_connection()
```

Сначала импортируется модуль sqlite3, затем определяется функция с именем sql_connection. Внутри функции определен блок try, где метод connect() возвращает объект соединения после установления соединения.

Затем определен блок исключений, который в случае каких-либо исключений печатает сообщение об ошибке. Если ошибок нет, соединение будет установлено, тогда скрипт распечатает текст «Connection is established: Database is created in memory».

Далее производится закрытие соединения в блоке finally. Закрытие соединения необязательно, но это хорошая практика программирования, позволяющая освободить память от любых неиспользуемых ресурсов.

2.4 Создание таблицы

Чтобы создать таблицу в SQLite3, выполним запрос Create Table в методе execute(). Для этого выполним следующую последовательность шагов:

1. Создание объекта подключения.
2. Объект Cursor создаётся с использованием объекта подключения.
3. Используя объект курсора, вызывается метод execute с SQL запросом create table в качестве параметра.

Давайте создадим таблицу Employees с колонками id, name, salary, department, position, hireDate.

Код будет таким:

```
import sqlite3
from sqlite3 import Error


def sql_connection():
    try:
        con = sqlite3.connect('mydatabase.db')
        return con
    except Error:
        print(Error)


def sql_table(con):
    cursorObj = con.cursor()
    cursorObj.execute(
        "CREATE TABLE employees(id integer PRIMARY KEY, name text, salary
real, department text, position text, hireDate text)")
    con.commit()


con = sql_connection()
sql_table(con)
```

В приведенном выше коде определено две функции: первая устанавливает соединение; а вторая - используя объект курсора выполняет SQL оператор create table.

Метод commit() сохраняет все сделанные изменения. В конце скрипта производится вызов обеих функций.

Для проверки существования таблицы воспользуемся браузером БД для sqlite.

Если интересно, то о графическом клиенте DB Browser for SQLite можно почитать тут: https://metanit.com/sql/sqlite/1.3.php. Если нет, то нет.

2.5 Вставка данных в таблицу

Чтобы вставить данные в таблицу воспользуемся оператором INSERT INTO. Рассмотрим следующую строку кода:

```
cursorObj.execute("INSERT INTO employees VALUES(1, 'John', 700, 'HR',
'Manager', '2017-01-04')")
```

Также можем передать значения / аргументы в оператор INSERT в методе execute (). Также можно использовать знак вопроса (?) в качестве заполнителя для каждого значения. Синтаксис INSERT будет выглядеть следующим образом:

```
cursorObj.execute('''INSERT INTO employees(id, name, salary, department,
position, hireDate) VALUES(?, ?, ?, ?, ?, ?)''', entities)
```

Где картеж entities содержат значения для заполнения одной строки в таблице:

```
entity = (2, 'Andrew', 800, 'IT', 'Tech', '2018-02-06')
```

Код выглядит следующим образом:

```
import sqlite3

con = sqlite3.connect('mydatabase.db')


def sql_insert(con, entities):
    cursorObj = con.cursor()
    cursorObj.execute(
        'INSERT INTO employees(id, name, salary, department, position,
hireDate) VALUES(?, ?, ?, ?, ?, ?)', entities)
    con.commit()


entities = (2, 'Andrew', 800, 'IT', 'Tech', '2018-02-06')
sql_insert(con, entities)
```

2.6 Обновление таблицы

Предположим, что нужно обновить имя сотрудника, чей идентификатор равен 2. Для обновления будем использовать инструкцию UPDATE. Также воспользуемся предикатом WHERE в качестве условия для выбора нужного сотрудника.

Рассмотрим следующий код:

```
import sqlite3

con = sqlite3.connect('mydatabase.db')
```

```
def sql_update(con):
    cursorObj = con.cursor()
    cursorObj.execute('UPDATE employees SET name = "Rogers" where id = 2')
    con.commit()


sql_update(con)
```

Это изменит имя Andrew на Rogers.

2.7 Оператор SELECT

Оператор SELECT используется для выборки данных из одной или более таблиц. Если нужно выбрать все столбцы данных из таблицы, можете использовать звёздочку (*). SQL синтаксис для этого будет следующим:

```
select * from table_name
```

В SQLite3 инструкция SELECT выполняется в методе execute объекта курсора. Например, выберем все стрики и столбцы таблицы employee:

```
cursorObj.execute('SELECT * FROM employees')
```

Если нужно выбрать несколько столбцов из таблицы, укажем их, как показано ниже:

```
select column1, column2 from tables_name
```

Например,

```
cursorObj.execute('SELECT id, name FROM employees')
```

Оператор SELECT выбирает все данные из таблицы employees БД.

2.8 Выборка всех данных

Чтобы извлечь данные из БД выполним инструкцию SELECT, а затем воспользуемся методом fetchall() объекта курсора для сохранения значений в переменной. При этом переменная будет являться списком, где каждая строка из БД будет отдельным элементом списка. Далее будет выполняться перебор значений переменной и печатать значений.

Код будет таким:

```
import sqlite3

con = sqlite3.connect('mydatabase.db')


def sql_fetch(con):
    cursorObj = con.cursor()
    cursorObj.execute('SELECT * FROM employees')
    rows = cursorObj.fetchall()
    for row in rows:
        print(row)


sql_fetch(con)
```

Также можно использовать fetchall() в одну строку:

```
[print(row) for row in cursorObj.fetchall()]
```

Если нужно извлечь конкретные данные из БД, воспользуйтесь предикатом WHERE. Например, выберем идентификаторы и имена тех сотрудников, чья зарплата превышает 800. Для этого заполним нашу таблицу большим количеством строк, а затем выполним запрос.

Можете использовать оператор INSERT для заполнения данных или ввести их вручную в программе браузера БД.

Теперь, выберем имена и идентификаторы тех сотрудников, у кого зарплата больше 800:

```
import sqlite3

con = sqlite3.connect('mydatabase.db')


def sql_fetch(con):
    cursorObj = con.cursor()
    cursorObj.execute('SELECT id, name FROM employees WHERE salary > 800.0')
    rows = cursorObj.fetchall()

    for row in rows:
        print(row)


sql_fetch(con)
```

В приведенном выше операторе SELECT вместо звездочки (*) были указаны атрибуты id и name.

2.9 rowcount

Счётчик строк SQLite3 используется для возврата количества строк, которые были затронуты или выбраны последним выполненным запросом SQL.

Когда вызывается rowcount с оператором SELECT, будет возвращено -1, поскольку количество выбранных строк неизвестно до тех пор, пока все они не будут выбраны. Рассмотрим пример:

```
print(cursorObj.execute('SELECT * FROM employees').rowcount)
```

Поэтому, чтобы получить количество строк, нужно получить все данные, а затем получить длину результата:

```
rows = cursorObj.fetchall()
print(len(rows))
```

Когда оператор DELETE используется без каких-либо условий (предложение where), все строки в таблице будут удалены, а общее количество удаленных строк будет возвращено rowcount.

```
print(cursorObj.execute('DELETE FROM employees').rowcount)
```

Если ни одна строка не удалена, будет возвращено 0.

2.10 Список таблиц

Чтобы вывести список всех таблиц в базе данных SQLite3, нужно обратиться к таблице sqlite_master, а затем использовать fetchall() для получения результатов из оператора SELECT.

Sqlite_master – это главная таблица в SQLite3, в которой хранятся все таблицы.

```
import sqlite3

con = sqlite3.connect('mydatabase.db')
```

```
def sql_fetch(con):
    cursorObj = con.cursor()
    cursorObj.execute('SELECT name from sqlite_master where type= "table"')
    print(cursorObj.fetchall())


sql_fetch(con)
```

2.11 Проверка существования таблицы

При создании таблицы необходимо убедиться, что таблица еще не существует. Аналогично, при удалении таблицы она должна существовать.

Чтобы проверить, если таблица еще не существует, используем «if not exists» с оператором CREATE TABLE следующим образом:

```
import sqlite3

con = sqlite3.connect('mydatabase.db')


def sql_fetch(con):
    cursorObj = con.cursor()
    cursorObj.execute('create table if not exists projects(id integer, name
text)')
    con.commit()


sql_fetch(con)
```

Точно так же, чтобы проверить, существует ли таблица при удалении, мы используем «if not exists» с инструкцией DROP TABLE следующим образом:

```
cursorObj.execute('drop table if exists projects')
```

Также проверим, существует ли таблица, к которой нужно получить доступ, выполнив следующий запрос:

```
cursorObj.execute('SELECT name from sqlite_master WHERE type = "table" AND
name = "employees"')
print(cursorObj.fetchall())
```

Если указанное имя таблицы не существует, будет возвращен пустой массив.

2.12 Удаление таблицы

Удаление таблицы выполняется с помощью оператора DROP. Синтаксис оператора DROP выглядит следующим образом:

```
drop table table_name
```

Чтобы удалить таблицу, таблица должна существовать в БД. Поэтому рекомендуется использовать «if exists» с оператором DROP. Например, удалим таблицу employees:

```
import sqlite3

con = sqlite3.connect('mydatabase.db')


def sql_fetch(con):
    cursorObj = con.cursor()
    cursorObj.execute('DROP table if exists employees')
```

```
    con.commit()


sql_fetch(con)
```

2.13 Исключения

Исключением являются ошибки времени выполнения скрипта. При программировании на Python все исключения являются экземплярами класса производного от BaseException.

В SQLite3 у есть следующие основные исключения Python:

DatabaseError

Любая ошибка, связанная с базой данных, вызывает ошибку DatabaseError.

IntegrityError

IntegrityError является подклассом DatabaseError и возникает, когда возникает проблема целостности данных, например, когда внешние данные не обновляются во всех таблицах, что приводит к несогласованности данных.

ProgrammingError

Исключение ProgrammingError возникает, когда есть синтаксические ошибки или таблица не найдена или функция вызывается с неправильным количеством параметров / аргументов.

OperationalError

Это исключение возникает при сбое операций базы данных, например, при необычном отключении. Не по вине программиста.

NotSupportedError

При использовании некоторых методов, которые не определены или не поддерживаются базой данных, возникает исключение NotSupportedError.

2.14 Массовая вставка строк

Для вставки нескольких строк одновременно использовать оператор executemany.

Рассмотрим следующий код:

```
import sqlite3

con = sqlite3.connect('mydatabase.db')
cursorObj = con.cursor()
cursorObj.execute('create table if not exists projects(id integer, name
text)')
data = [(1, "Ridesharing"), (2, "Water Purifying"), (3, "Forensics"), (4,
"Botany")]
cursorObj.executemany("INSERT INTO projects VALUES(?, ?)", data)
con.commit()
```

Здесь создали таблицу с двумя столбцами, тогда у «данных» есть четыре значения для каждого столбца. Эта переменная передается методу executemany() вместе с запросом.

Обратите внимание, что использовался заполнитель для передачи значений.

2.15 Закрытие соединения

Когда работа с БД завершена, рекомендуется закрыть соединение. Соединение может быть закрыто с помощью метода close().

Чтобы закрыть соединение, используйте объект соединения с вызовом метода close() следующим образом:

```
con = sqlite3.connect('mydatabase.db')
# program statements
con.close()
```

2.16 datetime

В базе данных Python SQLite3 можно легко сохранять дату или время, импортируя Python модуль datetime. Следующие форматы являются наиболее часто используемыми форматами для даты и времени:

```
YYYY-MM-DD
YYYY-MM-DD HH:MM
YYYY-MM-DD HH:MM:SS
YYYY-MM-DD HH:MM:SS.SSS
HH:MM
HH:MM:SS
HH:MM:SS.SSS
now
```

Рассмотрим следующий код:

```
import sqlite3
import datetime

con = sqlite3.connect('mydatabase.db')
cursorObj = con.cursor()
cursorObj.execute('create table if not exists assignments(id integer, name
text, date date)')
data = [(1, "Ridesharing", datetime.date(2017, 1, 2)), (2, "Water Purifying",
datetime.date(2018, 3, 4))]
cursorObj.executemany("INSERT INTO assignments VALUES(?, ?, ?)", data)
con.commit()
```

В этом коде модуль datetime импортируется первым, далее создали таблицу с именем assignments с тремя столбцами.

Тип данных третьего столбца – дата. Чтобы вставить дату в столбец, воспользовались datetime.date. Точно так же можно использовать datetime.time для обработки времени.

2.17 Внешние ключи FOREIGN KEY

Внешние ключи позволяют установить связи между таблицами. Внешний ключ устанавливается для столбцов из зависимой, подчиненной таблицы, и указывает на один из столбцов из главной таблицы. Как правило, внешний ключ указывает на первичный ключ из связанной главной таблицы.

Общий синтаксис установки внешнего ключа на уровне таблицы:

```
[CONSTRAINT имя_ограничения]
FOREIGN KEY (столбец1, столбец2, ..., столбецN)
REFERENCES главная_таблица (столбец_главной_таблицы1,
столбец_главной_таблицы2, ..., столбец_главной_таблицыN)
[ON DELETE действие]
[ON UPDATE действие]
```

Для создания ограничения внешнего ключа после FOREIGN KEY указывается столбец таблицы, который будет представляет внешний ключ. А после ключевого слова REFERENCES указывается имя связанной таблицы, а затем в скобках имя связанного столбца, на который будет указывать внешний ключ. После выражения REFERENCES идут выражения ON DELETE и ON UPDATE, которые задают действие при удалении и обновлении строки из главной таблицы соответственно.

Например, определим две таблицы и свяжем их посредством внешнего ключа:

```
CREATE TABLE companies
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL
);
CREATE TABLE users
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER,
    company_id INTEGER,
    FOREIGN KEY (company_id)  REFERENCES companies (id)
);
```

В данном случае определены таблицы companies и users. companies является главной и представляет компании, где может работать пользователь. users является зависимой и представляет пользователей. Таблица users через столбец company_id связана с таблицей companies и ее столбцом id. То есть столбец company_id является внешним ключом, который указывает на столбец id из таблицы companies.

С помощью оператора CONSTRAINT можно задать имя для ограничения внешнего ключа:

```
CREATE TABLE users
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER,
    company_id INTEGER,
    CONSTRAINT users_companies_fk
    FOREIGN KEY (company_id)  REFERENCES companies (id)
);
```

ON DELETE и ON UPDATE

С помощью выражений ON DELETE и ON UPDATE можно установить действия, которые выполняются соответственно при удалении и изменении связанной строки из главной таблицы. В качестве действия могут использоваться следующие опции:

CASCADE: автоматически удаляет или изменяет строки из зависимой таблицы при удалении или изменении связанных строк в главной таблице.

SET NULL: при удалении или обновлении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение NULL. (В этом случае столбец внешнего ключа должен поддерживать установку NULL)

RESTRICT: отклоняет удаление или изменение строк в главной таблице при наличии связанных строк в зависимой таблице.

NO ACTION: то же самое, что и RESTRICT.

SET DEFAULT: при удалении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение по умолчанию, которое задается с помощью атрибуты DEFAULT.

Каскадное удаление

Каскадное удаление позволяет при удалении строки из главной таблицы автоматически удалить все связанные строки из зависимой таблицы. Для этого применяется опция CASCADE:

```
CREATE TABLE users
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER,
    company_id INTEGER,
    FOREIGN KEY (company_id)  REFERENCES companies (id) ON DELETE CASCADE
);
```

Подобным образом работает и выражение ON UPDATE CASCADE. При изменении значения первичного ключа автоматически изменится значение связанного с ним внешнего ключа. Однако поскольку первичные ключи изменяются очень редко, да и с принципе не рекомендуется использовать в качестве первичных ключей столбцы с изменяемыми значениями, то на практике выражение ON UPDATE используется редко.

Установка NULL

При установки для внешнего ключа опции SET NULL необходимо, чтобы столбец внешнего ключа допускал значение NULL:

```
CREATE TABLE users
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER,
    company_id INTEGER,
    FOREIGN KEY (company_id)  REFERENCES companies (id) ON DELETE SET NULL
);
```

3 Порядок выполнения работы

Получить задание для выполнения лабораторной работы (раздел 4) согласно своему варианту (по журналу). Разработать программу.

4 Задания для выполнения работы

На основе данных примеров разработайте ER-диаграмму и собственную базу данных по выбранной теме. В созданной базе данных создайте как минимум 10 таблиц. Хотя бы в 1 таблицу вставьте как минимум 1000 записей. Реализуйте как минимум 50 запросов к базе данных (запросы на выборку, обновление и удаление данных).

Темы:

1. Обменный пункт: сотрудники пункта, виды валют, курсы валют, операции обмена.
2. Ювелирный магазин: названия изделий, комитенты (кто сдал изделия на комиссию), журнал сдачи изделий на продажу, журнал покупки изделий.
3. Поликлиника: врачи, пациенты, виды болезней, журнал приёма пациентов.
4. Кондитерский магазин: виды конфет, поставщики, торговые точки, журнал поступления и отпуска товара.
5. Автобаза: автомашины, водители, рейсы, журнал выезда машин на рейсы.
6. Парикмахерская: клиенты, прайс услуг, сотрудники, кассовый журнал.
7. Школа: учителя, предметы, ученики, журнал успеваемости.
8. Оплата услуг на дачных участках: виды услуг, список владельцев, сотрудники управления, журнал регистрации оплат.
9. Гостиница: проживающие, сотрудники гостиницы, номера, журнал регистрации проживающих.
10. Книжный магазин: авторы, книги, продавцы, покупатели, регистрация поступлений и продаж.
11. Ремонтная мастерская: виды работ, исполнители, заказы на ремонт, заказчики.
12. Аптечный киоск: номенклатура лекарств, работники аптеки, журнал регистрации продаж.
13. Выставка: стенды, стендисты, экскурсии, посетители.
14. Охранная служба: список постов охраны, список охранников, журнал выхода на дежурство, журнал учета замечаний.
15. Столовая: продукты, блюда, меню, журнал заказов.
16. Фото мастерская: заказчики работ, прайс работ, журнал поступления заказов, исполнители.
17. Ветеринарная лечебница: список животных, список болезней, список хозяев, журнал посещений.
18. Сельское хозяйство: список растений, список угодий, список работников, журнал посевной.
19. Холдинг: список регионов, список предприятий, список показателей, журнал учета отчетных данных.
20. Фонды предприятия: список основных средств, список категорий основных средств, список материально ответственных лиц, журнал учета состояния основных средств.
21. Учет расхода материалов в компании: список статей затрат, список сотрудников, журнал учета расхода канцтоваров, список департаментов.
22. Фильмотека: список фильмов, список клиентов, список библиотекарей, журнал выдачи фильмов.
23. Цирк: список категорий артистов, список артистов, журнал выхода артистов на работу, список цирковых площадок.

24. Спортивные мероприятия: список спортсменов, список видов спорта, список стадионов, журнал учета выступлений спортсменов.

25. Компьютерные занятия: список слушателей курсов, список предметов, список преподавателей, журнал учета успеваемости.

26. Сбор урожая: список видов продукции, список сборщиков, список бригад, журнал учета сбора урожая.

27. Фирма по обслуживанию населения: список заказчиков, список товаров, список разносчиков, журнал заказов.

28. Партийная работа: список членов партии, список мероприятий, журнал учета выхода на мероприятие, список городов

29. Экономическая база данных: список регионов, список показателей, список отраслей, отчетные статистические данные.

30. Журнальные статьи: список тем, список авторов, список названия статей, список журналов.

31. Анализ причин заболеваемости: список больных, список болезней, список районов, журнал учета заболевших.

32. Отдел кадров: список сотрудников, штатное расписание, список отделов, журнал перемещения сотрудников по службе.

33. Расчет нагрузки на преподавателя: список преподавателей, список кафедр, предметов, журнал нагрузки.

34. Проектные работы: список проектов, список специалистов, список должностей, журнал учета работ.

35. Учет компьютерного оборудования: список типов оборудования, список материально ответственных лиц, список департаментов, журнал регистрации выдачи оборудования.

36. Прививки детям: список прививок, список детей, список родителей, журнал учета сделанных прививок.

37. Начисление налогов в бюджет: виды налогов, список отраслей, список предприятий, журнал учета поступления налогов.

38. Экспертная система: список оцениваемых объектов, список экспертов, список регионов, журнал учета оценок.

39. Ремонтная мастерская электронного оборудования: список работ, список мастеров, список запасных частей, журнал учета выполненных работ, список поступившего оборудования.

40. Магазин по продаже автомобилей: список фирм производителей, список автомобилей, журнал поступления автомобилей; водители, пригнавшие машину.

41. Автомобильный гараж: список владельцев, список автомобилей, список сторожей, журнал прихода и ухода автомобилей.

42. Учет криминогенной ситуации в городе: список районов, список типов преступлений, список дежурных, журнал регистрации преступлений.

43. Система здравоохранения: список регионов, список санаториев, список пенсионеров, журнал регистрации выдачи путевок в санатории.

44. Туристические агентства: список туров, список стран, список клиентов, журнал регистрации продаж туров.

45. Продажа билетов на рейсы: список рейсов, прайс билетов, список компаний, журнал продаж билетов.

46. Продажа пиломатериалов: виды пиломатериалов, регионы поставщики, список заказчиков, журнал учета продаж пиломатериалов.

47. Склад металлоконструкций: прайс товара металлоконструкций, список поставщиков, список продавцов, журнал учета продаж.

48. Система поддержки решений: список экспертов, список тем обсуждений, список департаментов, журнал учета предложений.

49. Детский сад: список родителей, список детей, список групп, журнал посещения детского сада.

50. Дом творчества молодежи: список кружков, список руководителей, список детей, журнал регистрации посещения кружков.