# Базовые методы обработки данных с использованием Python

*Лекция 7. Форматы файлов, CSV, JSON. Работа с файлами в Python. Модуль OS. Файловые СУБД, SQLite. Операции CRUD в SQLite*

Киреев Василий Сергеевич

к.т.н., доцент

**Москва, 2024**

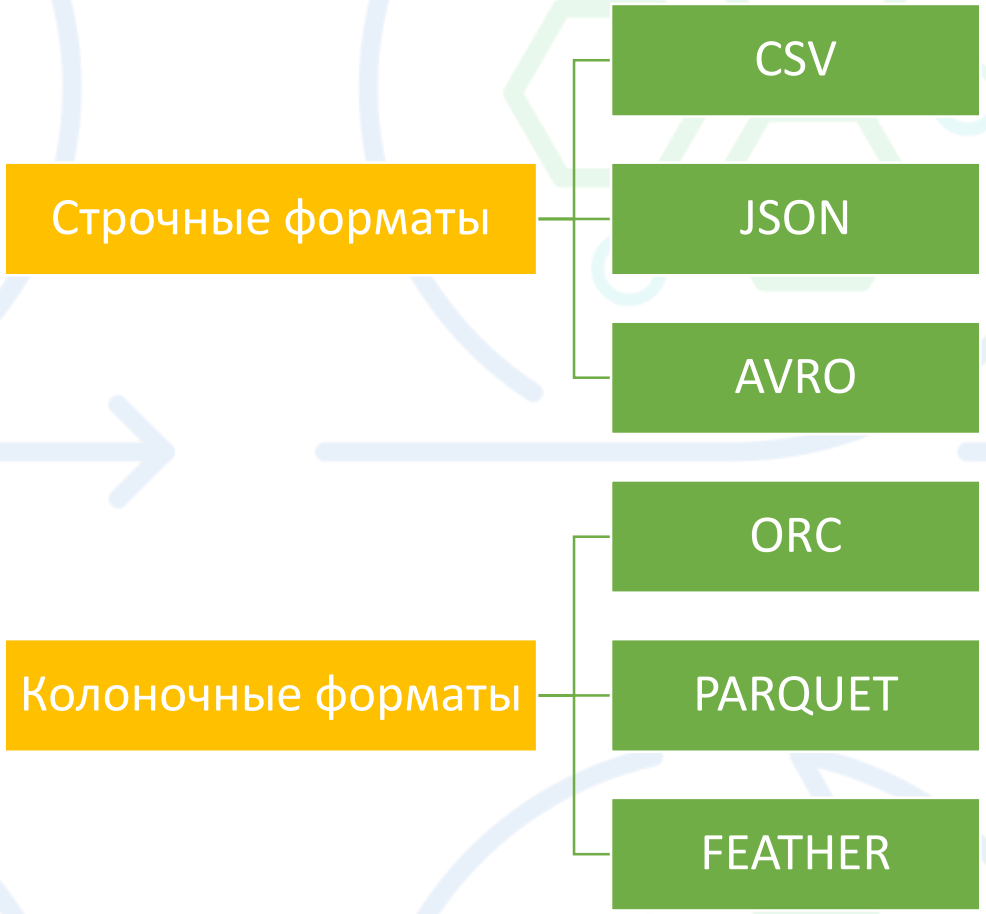
# Колоночные vs строчные форматы файлов

Существует два основных способа организации данных: строки и столбцы. Выбор одного из них в значительной степени определяет эффективность хранения и запроса данных.

Строка - данные организованы по записям. Это более "традиционный" способ организации и управления данными. Все данные, относящиеся к конкретной записи, хранятся рядом. Другими словами, данные каждой строки расположены таким образом, что последний столбец строки хранится рядом с первой записью столбца последующей строки данных.

Колоночный - значения каждого столбца (поля) таблицы хранятся рядом друг с другом. Это означает, что подобные элементы группируются и хранятся рядом друг с другом. Внутри полей сохраняется порядок чтения - это сохраняет возможность связывать данные с записями.

# Колоночные vs строчные форматы файлов



# Форматы файлов. CSV

CSV - это текстовый формат, в котором в качестве разделителя записей используется новая строка и необязательный заголовок. Он легко редактируется вручную и очень понятен с человеческой точки зрения.

CSV не поддерживает эволюцию схемы, хотя иногда заголовок опционально рассматривается как схема данных. В своем сложном виде он не поддается разбиению на части, поскольку не содержит определенного символа, на который можно было бы опереться, чтобы разделить текст, сохранив его актуальность.

# Форматы файлов. JSON

JSON (JavaScript object notation) - это текстовый формат, содержащий запись, которая может быть представлена в нескольких формах (строка, целое число, булевы символы, массив, объект, ключ-значение, вложенные данные...). Будучи человеко- и машиночитаемым, JSON является относительно легким форматом и широко используется в веб-приложениях.

Он широко поддерживается многими программами и сравнительно прост для реализации в различных языках. JSON изначально поддерживает процессы сериализации и десериализации. Его основное неудобство связано с тем, что он не может быть разделен на части.

# Операции ввода-вывода в Python

Для выполнения стандартных операций ввода-вывода в Python используются встроенные функции input() и print(). Программа Python взаимодействует с этими устройствами ввода-вывода через стандартные потоковые объекты stdin и stdout, определенные в модуле sys.

Функция input() считывает байты со стандартного устройства потока ввода, т.е. с клавиатуры.

# Операции ввода-вывода в Python. Методы работы с файлами

| | |
|---|---|
| **file.close()** | **Закрывает файл. Закрытый файл больше не может быть прочитан или записан.** |
| **file.flush()** | Промыть внутренний буфер, подобно fflush в stdio. Для некоторых файлоподобных объектов это может быть неактуально. |
| **file_fileno()** | Возвращает целочисленный дескриптор файла, который используется базовой реализацией для запроса операций ввода-вывода от операционной системы. |
| **file.isatty()** | Возвращает True, если файл подключен к tty(-подобному) устройству, иначе False. |
| **next(file)** | Возвращает следующую строку из файла при каждом вызове. |
| **file.read([size])** | Читает из файла не более size байт (меньше, если чтение достигает EOF до получения size байт). |

# Операции ввода-вывода в Python. Режимы открытия файлов

| | |
|---|---|
| **file.readlines([sizehint])** | **Считывает файл до EOF с помощью readline() и возвращает список, содержащий строки. Если присутствует необязательный аргумент sizehint, то вместо чтения до EOF считываются целые строки общим размером примерно sizehint байт (возможно, после округления до размера внутреннего буфера).** |
| **file.seek(offset[, whence])** | Устанавливает текущую позицию файла |
| **file.tell()** | Возвращает текущую позицию файла |
| **file.truncate([size])** | Усекает размер файла. Если присутствует необязательный аргумент size, то файл усекается до (не более) этого размера. |
| **file.write(str)** | Записывает в файл строку. Возвращаемое значение отсутствует. |
| **file.writelines(sequence)** | Записывает в файл последовательность строк. Последовательность может быть любым итерируемым объектом, производящим строки, обычно это список строк. |

# Операции ввода-вывода в Python. Запись в файл

Чтобы записать данные в файл в Python, необходимо открыть файл. Любой объект, взаимодействующий с входной и выходной парой, называется объектом File. Встроенная функция Python open() возвращает файловый объект.

После получения объекта file с помощью функции open() можно использовать метод write() для записи любой строки в файл, представленный объектом file.

Важно отметить, что строки Python могут содержать не только текст, но и двоичные данные. Метод write() не добавляет символ новой строки ('\n') в конец строки.

# Операции ввода-вывода в Python. Чтение из файла

Чтобы программно прочитать данные из файла с помощью Python, его необходимо сначала открыть. Для этого можно воспользоваться встроенной функцией open().

Ниже приведены сведения о параметрах:

- file_name - Аргумент имя_файла представляет собой строковое значение, содержащее имя файла, к которому требуется получить доступ.
- access_mode - Access_mode определяет режим, в котором должен быть открыт файл, т.е. чтение, запись, добавление и т.д. Это необязательный параметр, и по умолчанию используется режим доступа к файлу read (r).

# Операции ввода-вывода в Python. Режимы открытия файлов

| | |
|---|---|
| **r** | **Открывает файл только для чтения. Указатель файла помещается в начало файла. Это режим по умолчанию.** |
| **rb** | Открывает файл только для чтения в двоичном формате. Указатель файла помещается в начало файла. Это режим по умолчанию. |
| **r+** | Открывает файл как для чтения, так и для записи. Указатель файла помещается в начало файла. |
| **rb+** | Открывает файл для чтения и записи в двоичном формате. Указатель файла помещается в начало файла. |
| **w** | Открывает файл только для записи. Перезаписывает файл, если он существует. Если файл не существует, создается новый файл для записи. |
| **b** | Открывает файл в двоичном режиме |

# Операции ввода-вывода в Python. Режимы открытия файлов

| | |
|---|---|
| **t** | **Открывает файл в текстовом режиме (по умолчанию)** |
| **+** | открыть файл для обновления (чтения и записи) |
| **wb** | Открывает файл для записи только в двоичном формате. Перезаписывает файл, если он существует. Если файл не существует, создается новый файл для записи. |
| **w+** | Открывает файл как для записи, так и для чтения. Перезаписывает существующий файл, если он существует. Если файл не существует, создает новый файл для чтения и записи. |
| **wb+** | Открывает файл для записи и чтения в двоичном формате. Перезаписывает существующий файл, если он существует. Если файл не существует, создается новый файл для чтения и записи. |
| **a** | Открывает файл для добавления. Указатель файла находится в конце файла, если файл существует. То есть файл находится в режиме добавления. Если файл не существует, то создается новый файл для записи. |

# Одновременное чтение/запись в Python

Когда файл открыт для записи (с помощью 'w' или 'a'), чтение из него невозможно, и наоборот. Это приводит к ошибке UnSupportedOperation. Перед выполнением другой операции необходимо закрыть файл.

Для того чтобы выполнить обе операции одновременно, необходимо добавить в параметр mode символ '+'. Таким образом, режим 'w+' или 'r+' позволяет использовать как методы write(), так и read() без закрытия файла.

Объект File также поддерживает функцию seek(), позволяющую перемотать поток на любую нужную позицию байта.

# Одновременное чтение/запись в Python. Метод seek()

Метод seek() устанавливает текущую позицию файла по смещению. Аргумент whence необязателен и по умолчанию равен 0, что означает абсолютное позиционирование файла, другие значения - 1, что означает поиск относительно текущей позиции, и 2 - поиск относительно конца файла. Возвращаемое значение отсутствует. Обратите внимание, что если файл открыт для добавления с помощью 'a' или 'a+', то все операции seek() будут отменены при следующей записи.

Если файл открыт только для записи в режиме добавления с использованием 'a', то этот метод, по сути, не работает, но он остается полезным для файлов, открытых в режиме добавления с включенным чтением (режим 'a+').

Если файл открыт в текстовом режиме с помощью команды 't', то допустимы только смещения, возвращаемые функцией tell(). Использование других смещений приводит к неопределенному поведению.

# Модуль OS

Модуль OS Python предоставляет возможность установить взаимодействие между пользователем и операционной системой. Методы, включенные в heuj позволяют определять тип операционной системы, получать доступ к переменным окружения, управлять директориями и файлами , управлять процессами. OS входит в состав стандартных служебных модулей.

# Модуль OS. Работа с текущим каталогом

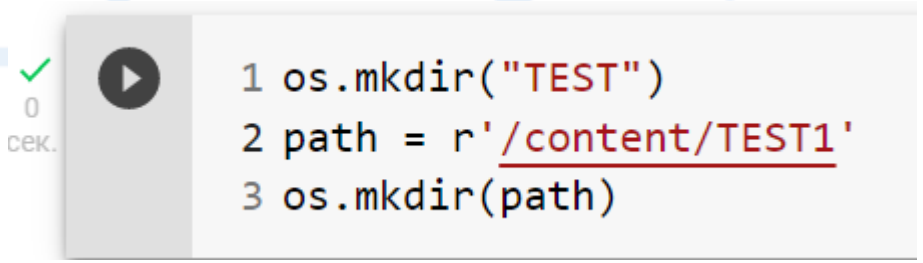
Функция ***os.chdir*** – меняет каталог, который в данный момент используется в сессии. ***os.getcwd()*** показывает текущий каталог.

```
1 os.chdir(r"/content/")
2 print(os.getcwd())
3 os.chdir(r"/content/sample_data")
4 print(os.getcwd())
```

```
/content
/content/sample_data
```

# Модуль OS. Создание папок

***os.mkdir()*** позволяет создать одну папку. ***os.makedirs()*** создает промежуточные папки в пути, если их там нет



# Модуль OS. Удаление файлов и папок

Функции ***os.remove()*** и ***os.rmdir()*** используются для удаления файлов и каталогов соответственно.

```
1 path = r"/content/"
2 os.chdir(path)
3 !echo 'dfsdf' > test.txt
4 print(list(os.walk(path))[0])
5 os.remove("test.txt")
6 print(list(os.walk(path))[0])
```

```
('/content/', ['.config', 'sample_data', 'TEST1'], ['test.txt'])
('/content/', ['.config', 'sample_data', 'TEST1'], [])
```

# Модуль OS. Итерация по каталогу

Метод ***os.walk()*** дает нам получить доступ ко всем подкаталогам и файлам для выбранного пути.

```
1 path = r'/content/sample_data'
2 list(os.walk(path))
```

```
[('/content/sample_data',
  ['TEST'],
  ['README.md',
   'anscombe.json',
   'california_housing_train.csv',
   'mnist_train_small.csv',
   'mnist_test.csv',
   'california_housing_test.csv']),
 ('/content/sample_data/TEST', [], [])]
```

# Модуль OS. Переименование файлов и папок

Функция ***os.rename()*** применяется тогда, когда нужно переименовать файл или папку.

```
1 os.chdir(r"/content/")
2 print(os.getcwd())
3 os.chdir(r"/content/sample_data")
4 print(os.getcwd())
```

```
/content
/content/sample_data
```

# Модуль OS.PATH

***os.path*** - модуль, реализующий некоторые полезные функции на работы с путями. Функция ***basename*** вернет название файла пути. Или последней вложенной папки.

```
1 path = r'/content/sample_data'
2 os.path.basename(path)
```

```
'sample_data'
```

# Модуль OS. Часть каталога пути

***os.path.dirname*** - возвращает только часть каталога пути.

```
1 os.path.dirname(r'/content/sample_data/california_housing_train.csv')
```

```
'/content/sample_data'
```

# Модуль OS. Проверка существования файла

Функция **exists** говорит, существует ли файл, или нет.

```
1 os.path.exists(r'/content/sample_data/california_housing_train.csv')
```

```
True
```

# Модуль OS. Проверка существования

Методы ***isdir*** и ***isfile*** тесно связаны с методом exists, так как они также тестируют присутствие или отсутствие файлов или папок на тех или иных путях. Однако, ***isdir*** проверяет только пути к папкам, а ***isfile***, соответственно, к файлам.

```
1 print(os.path.isfile(r'/content/sample_data/california_housing_train.csv'))
2 print(os.path.isdir(r'/content/sample_data/california_housing_train.csv'))
```

```
True
False
```

# Модуль OS. Создание строк с путями

Метод ***join*** позволяет совместить несколько путей при помощи присвоенного разделителя.

```
1 print(os.path.join(r'/content/sample_data/', 'california_housing_train.csv') )
```

```
/content/sample_data/california_housing_train.csv
```

# Модуль OS. Разделение строки пути

Метод ***split*** разъединяет путь на кортеж, который содержит и файл и каталог.

```
1 os.path.split(r'/content/sample_data/california_housing_train.csv')
```

```
('/content/sample_data', 'california_housing_train.csv')
```

# Модуль OS. Просмотр текущего каталога

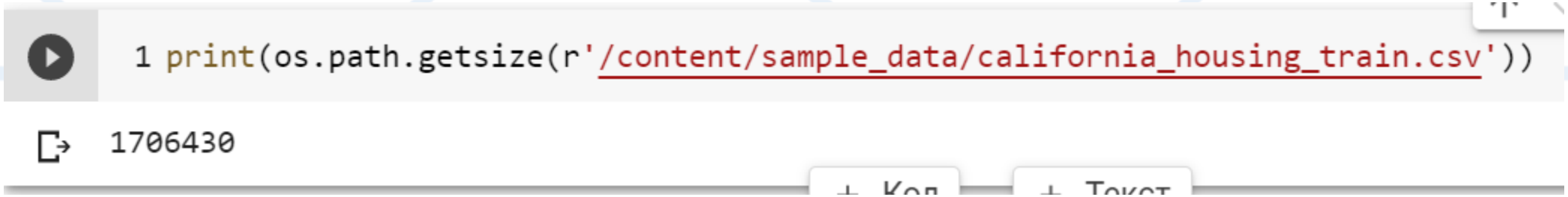
Функция ***os.listdir*** – показывает содержимое текущего каталога.

```
1 list(os.listdir(r'/content/sample_data/'))
```

```
['README.md',
 'anscombe.json',
 'test.txt',
 'TEST',
 'california_housing_train.csv',
 'mnist_train_small.csv',
 'mnist_test.csv',
 'california_housing_test.csv']
```

# Модуль OS. Размер файла или папки

Функция ***getsize*** используется для измерения объема директорий и файлов.



# Модуль OS. Свойства файла или папки

Функция ***os.stat*** – показывает свойства файла или папки.

```
1 stat_info = os.stat(r'/content/sample_data/')
2
3 print('  Size:', stat_info.st_size)
4 print('  Permissions:', oct(stat_info.st_mode))
5 print('  Owner:', stat_info.st_uid)
6 print('  Device:', stat_info.st_dev)
```

свойства файла или папки

```
Size: 4096
Permissions: 0o40755
Owner: 0
Device: 38
```

# Модуль OS. Подпроцессы в ОС

Функция ***os.fork*** – позволяет выделять подпроцессы в ОС.

```python
import os

pid = os.fork()

if pid:
    print('Child process id:', pid)
else:
    print('I am the child')
```

```
Child process id: 367
I am the child
```

# Модуль OS. Представление пути в ФС

Функция ***os.fspath*** – возвращает представление пути в файловой системе.

```
1 os.fspath(r'/content/sample_data/california_housing_train.csv')
```

```
'/content/sample_data/california_housing_train.csv'
```

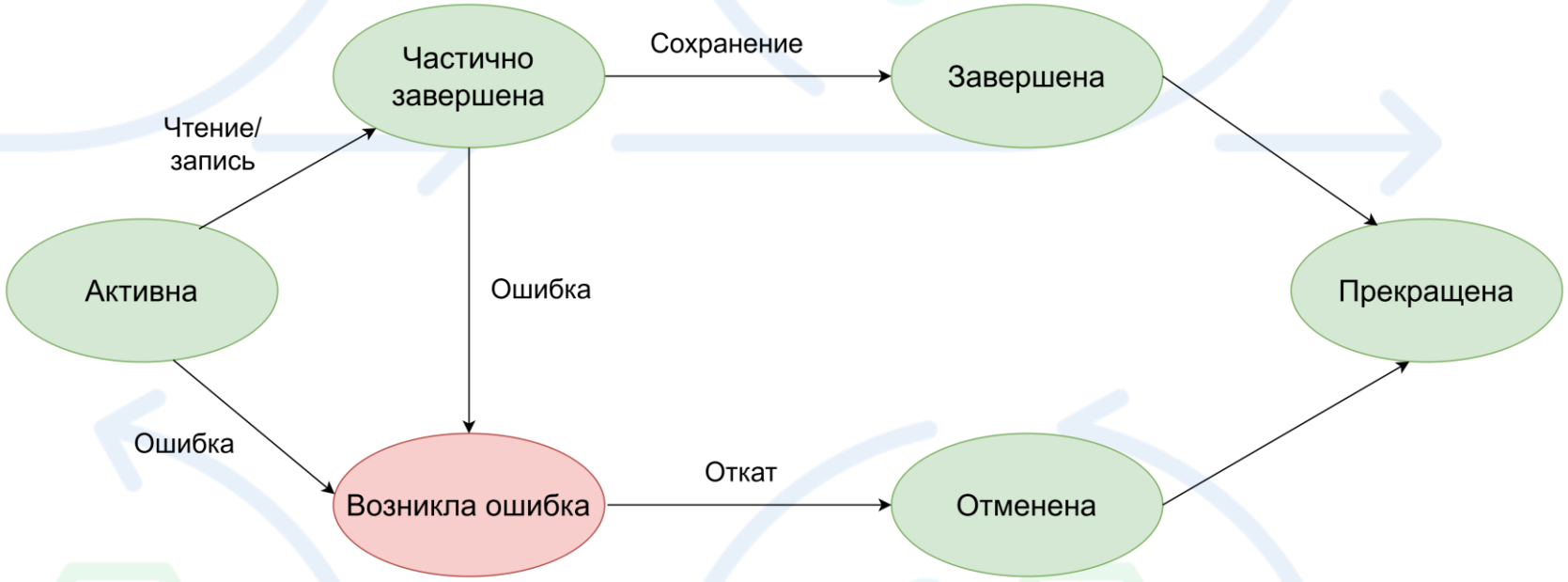
# Файловая СУБД SQLite

SQLite — это высокоэффективный , бессерверный и автономный движок базы данных SQL, который выделяется своей простотой и легкостью интеграции. Разработанный для встраивания в приложения, SQLite устраняет необходимость в отдельных процессах сервера базы данных и сложных конфигурациях.

SQLite была разработана в 2000 году Д. Ричардом Хиппом, как легкая и простая СУБД, которую можно было бы легко встраивать в другие приложения. Она была создана как альтернатива более сложным и тяжеловесным системам баз данных, таким как MySQL и PostgreSQL. SQLite поддерживает транзакции ACID.

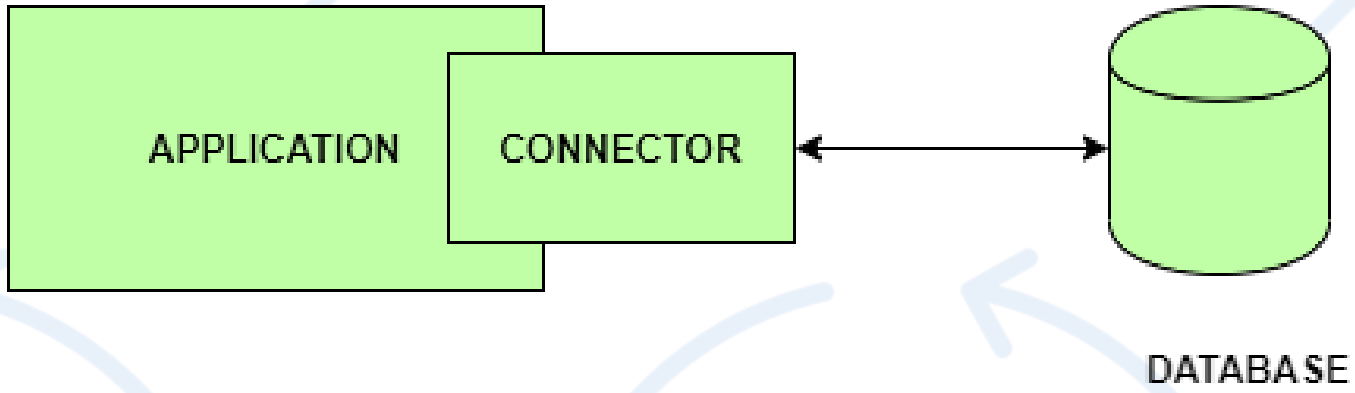
# Транзакции ACID

Транзакция — это набор операций, которые выполняются как единое, атомарное целое. Транзакции обеспечивают согласованность данных, гарантируя, что либо все операции в рамках транзакции фиксируются в базе данных, либо ни одна из них. Термин ACID означает атомарность, согласованность, изоляцию и долговечность.



# Файловая СУБД SQLite. Бессерверность

SQLite не требует сервера для работы. База данных SQLite объединена с приложением, которое обращается к базе данных. База данных SQLite считывает и записывает данные непосредственно из файлов базы данных, хранящихся на диске, и приложения взаимодействуют с этой базой данных SQLite.



# Достоинства SQLite

- Простота использования: SQLite очень легко начать использовать, так как он не требует установки или настройки. Вы можете просто включить библиотеку в свой проект и начать ее использовать.
- Встраиваемость: SQLite разработан для встраивания в другие приложения. Это автономный, бессерверный движок базы данных, что означает, что вы можете включить его в свое приложение без необходимости в отдельном сервере базы данных.
- Легковесность: SQLite — очень легкая СУБД с небольшим размером библиотеки (обычно менее 1 МБ). Это делает ее хорошо подходящей для использования в приложениях, где база данных встроена непосредственно в двоичный файл приложения, например, в мобильных приложениях .
- Бессерверность: SQLite — это серверный движок базы данных, что означает, что нет необходимости настраивать и поддерживать отдельный процесс сервера базы данных. Это упрощает развертывание и управление, поскольку нет дополнительных зависимостей, о которых нужно беспокоиться.
- Кроссплатформенность: SQLite доступен на многих платформах, включая Linux, macOS и Windows , что делает его хорошим выбором для кроссплатформенной разработки.
- Автономность: SQLite хранит все данные в одном файле в файловой системе, что упрощает копирование или резервное копирование базы данных.
- Высокая надежность: SQLite широко тестировалась и использовалась в производственных системах на протяжении многих лет и имеет репутацию надежного и производительного ядра базы данных.

# Ограничения SQLite

- Ограниченный параллелизм: SQLite использует файловую блокировку для управления доступом к базе данных, что может привести к проблемам с производительностью, когда несколько клиентов пытаются одновременно читать и писать в базу данных. Это делает его менее подходящим для использования в системах с высокой степенью параллелизма.
- Нет поддержки хранимых процедур: SQLite не поддерживает хранимые процедуры, которые являются предварительно скомпилированными операторами SQL, которые могут быть выполнены на сервере. Это означает, что весь код SQL должен быть отправлен на сервер и скомпилирован во время выполнения, что может быть менее эффективно, чем использование хранимых процедур.
- Нет поддержки триггеров: SQLite не поддерживает триггеры, которые являются действиями базы данных, которые автоматически запускаются указанными событиями (например, вставкой строки в таблицу). Это означает, что вам придется вручную реализовать любую логику, которая должна запускаться определенными событиями.
- Ограниченная поддержка типов данных: SQLite имеет относительно небольшой набор типов данных по сравнению с другими движками баз данных. Он не поддерживает многие из более продвинутых типов данных, таких как массивы и JSON, которые доступны в других базах данных.
- Ограниченная масштабируемость: SQLite не предназначена для работы в качестве высококонкурентной, высокоскоростной СУБД. Она больше подходит для использования в системах меньшего масштаба с низким уровнем параллелизма и может не иметь возможности масштабироваться для обработки очень больших объемов данных или очень высоких уровней параллелизма.

# Типы данных в SQLite

| Тип | Описание | Соответствие другим типам |
|---|---|---|
| **NULL** | Значение равно NULL. | |
| **INTEGER** | Значение представляет собой целое число со знаком, хранящееся в 1, 2, 3, 4, 6 или 8 байтах в зависимости от величины значения. | INT, INTEGER, TINYINT, SMALLINT, MEDIUMINT, BIGINT, UNSIGNED BIG INT, INT2, INT8 |
| **REAL** | Значение представляет собой значение с плавающей запятой, хранящееся как 8-байтовое число с плавающей запятой IEEE. | REAL, DOUBLE, DOUBLE PRECISION, FLOAT |
| **TEXT** | Значение представляет собой текстовую строку, сохраненную с использованием кодировки базы данных (UTF-8, UTF-16BE или UTF-16LE). | CHARACTER(20), VARCHAR(255), VARYING CHARACTER(255), NCHAR(55), NATIVE CHARACTER(70), NVARCHAR(100), TEXT, CLOB |
| **BLOB** | Значение представляет собой блок данных, хранящийся точно в том виде, в котором он был введен. | |

# Язык структурированных запросов SQL

Язык структурированных запросов (SQL) – это язык программирования для хранения и обработки информации в реляционной базе данных. Реляционная база данных хранит информацию в табличной форме со строками и столбцами, представляющими различные атрибуты данных и различные связи между значениями данных.

Инструкции SQL можно использовать для хранения, обновления, удаления, поиска и извлечения информации из базы данных. Можно также использовать SQL для поддержания и оптимизации производительности базы данных.

# Языки DDL, DQL, DML

DDL или язык определения данных,  состоит из команд SQL, которые могут использоваться для определения, изменения и удаления структур базы данных, таких как таблицы , индексы и схемы . Он имеет дело с описаниями схемы базы данных и используется для создания и изменения структуры объектов базы данных в базе данных

Операторы DQL используются для выполнения запросов к данным в объектах схемы . Целью команды DQL является получение некоторого отношения схемы на основе переданного ей запроса. Эта команда позволяет извлекать данные из базы данных для выполнения операций с ними. Когда SELECT запускается для таблицы или таблиц, результат компилируется в дополнительную временную таблицу, которая отображается или, возможно, принимается программой.

Команды SQL , которые имеют дело с манипулированием данными, присутствующими в базе данных, принадлежат к DML или языку манипулирования данными, и это включает в себя большинство операторов SQL.  Это компонент оператора SQL, который управляет доступом к данным и к базе данных . В основном операторы DCL группируются с операторами DML.

# DDL, DQL, DML в SQLite

| Язык | Команда | Описание |
|---|---|---|
| DDL | CREATE TABLE | создает новую таблицу в базе данных. |
| | ALTER TABLE | изменяет существующую таблицу в базе данных. |
| | DROP TABLE | удаляет таблицу из базы данных. |
| | CREATE INDEX | создает новый индекс для таблицы. |
| | DROP INDEX | удаляет индекс из таблицы. |
| DQL | INSERT INTO | вставляет новую строку в таблицу. |
| | UPDATE | обновляет данные в одной или нескольких строках таблицы. |
| | DELETE FROM | удаляет одну или несколько строк из таблицы. |
| DML | SELECT | извлекает данные из одной или нескольких таблиц в базе данных. |
| | JOIN | извлекает данные из нескольких таблиц на основе общего поля. |
| | GROUP BY | группирует результаты запроса по одному или нескольким полям. |
| | HAVING | фильтрует результаты запроса на основе условия |

# Работа с SQLite в Python. Метод connect.

Подключение к базе данных SQLite можно установить с помощью метода connect() , передавая имя базы данных, к которой необходимо получить доступ, в качестве параметра. Если эта база данных не существует, то она будет создана.

```
import sqlite3
try:
    sqliteConnection = sqlite3.connect('sql.db')
    cursor = sqliteConnection.cursor()
    print('Инициализация БД')
    query = 'select sqlite_version();'
    cursor.execute(query)
    result = cursor.fetchall()
    print('Версия SQLite: {}'.format(result))
    cursor.close()
except sqlite3.Error as error:
    print('Возникла ошибка - ', error)
finally:
    if sqliteConnection:
        sqliteConnection.close()
        print('Соединение с SQLite закрыто')
```

```
Инициализация БД
Версия SQLite: [('3.37.2',)]
Соединение с SQLite закрыто
```

# Работа с SQLite в Python. Метод connect.

Подключение к базе данных SQLite можно установить с помощью метода connect() , передавая имя базы данных, к которой необходимо получить доступ, в качестве параметра. Если эта база данных не существует, то она будет создана.

```
import sqlite3
connection = sqlite3.connect('sql.db')
connection.execute('''CREATE TABLE
customer_address
        (ID INT PRIMARY KEY     NOT NULL,
        NAME           TEXT    NOT NULL,
        AGE            INT     NOT NULL,
        ADDRESS        CHAR(50)); ''')
connection.close()
```

# Шаблон запроса SELECT

<SELECT statement> ::= [WITH <common_table_expression> [,...n]] <query_expression> [ ORDER BY { order_by_expression | column_position [ ASC | DESC ] } [ ,...n ] ] [ COMPUTE { { AVG | COUNT | MAX | MIN | SUM } ( expression ) } [ ,...n ] [ BY expression [ ,...n ] ] ] [ <FOR Clause>] [ OPTION ( <query_hint> [ ,...n ] ) ] <query_expression> ::= { <query_specification> | ( <query_expression> ) } [ { UNION [ ALL ] | EXCEPT | INTERSECT } <query_specification> | ( <query_expression> ) [...n ] ] <query_specification> ::= SELECT [ ALL | DISTINCT ] [TOP ( expression ) [PERCENT] [ WITH TIES ] ] < select_list > [ INTO new_table ] [ FROM { <table_source> } [ ,...n ] ] [ WHERE <search_condition> ] [ <GROUP BY> ] [ HAVING < search_condition > ]

# Работа с SQLite в Python. Метод insert into.

Оператор SQL INSERT INTO из SQL используется для вставки новой строки в таблицу. Существует два способа использования оператора INSERT INTO для вставки строк: Только значения: указание только значений данных, которые необходимо вставить, без имен столбцов; имена столбцов и значения.

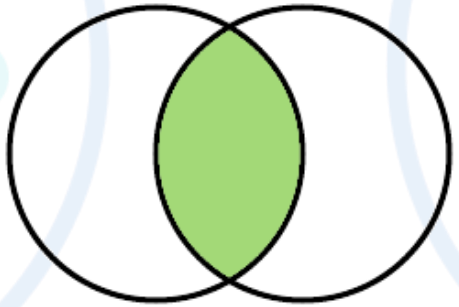
```
import sqlite3
conn = sqlite3.connect('sql.db')
cursor = conn.cursor()
table ="""CREATE TABLE STUDENT(NAME VARCHAR(255), CLASS
VARCHAR(255),
SECTION VARCHAR(255));"""
cursor.execute(table)
cursor.execute('''INSERT INTO STUDENT VALUES ('Иванов', '7th', 'A')''')
cursor.execute('''INSERT INTO STUDENT VALUES ('Петров', '8th', 'B')''')
cursor.execute('''INSERT INTO STUDENT VALUES ('Сидоров', '9th', 'C')''')
data=cursor.execute('''SELECT * FROM STUDENT''')
for row in data:
    print(row)
conn.commit()
conn.close()
```

```
('Иванов', '7th', 'A')
('Петров', '8th', 'B')
('Сидоров', '9th', 'C')
```

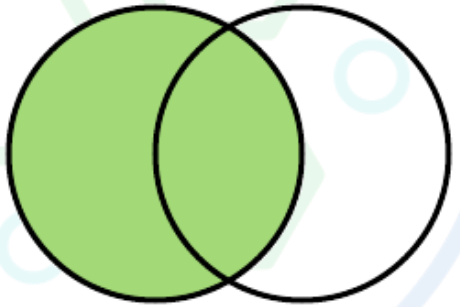
# Виды соединений в SQLite

| Команда | Описание |
| --- | --- |
| **INNER JOIN (JOIN)** | выдает записи, имеющие общие атрибуты в обеих таблицах. |
| **LEFT JOIN** | возвращает все записи из левой таблицы и только общие записи из правой таблицы. |
| **RIGHT JOIN** | возвращает все записи из правой таблицы и только общие записи из левой таблицы. |
| **ПОЛНОЕ ВНЕШНЕЕ ОБЪЕДИНЕНИЕ** | возвращает все записи, если в левой или правой таблице есть общий атрибут. |
| **CROSS JOIN** | объединяет записи одной таблицы со всеми остальными записями другой таблицы. |

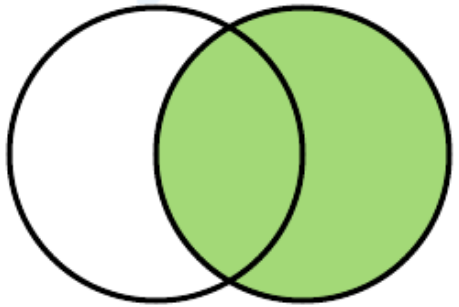
# Виды соединений в SQLite



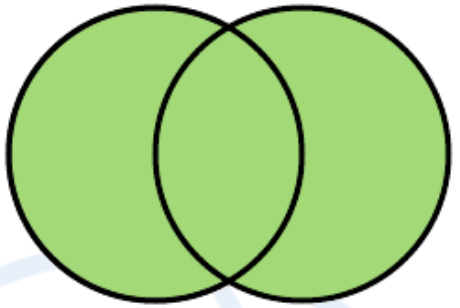
INNER JOIN



LEFT JOIN



RIGHT JOIN



FULL OUTER JOIN

# Аналитические функции в SQLite

| Команда | Описание |
|---|---|
| **MAX()** | Возвращает максимальное значение в выражении. Возвращает такое же значение, как и expression.MAX ( [ ALL \| DISTINCT ] expression )<br>Выражение может быть константой, именем столбца или функцией, а также любым сочетанием арифметических, побитовых и строковых операторов. Функцию MAX можно использовать с колонками типа numeric, character и datetime, но не с колонками типа bit. |
| **MIN()** | Возвращает минимальное значение выражения. Возвращает такое же значение, как и expression.MIN ( [ ALL \| DISTINCT ] expression )<br>Выражение может быть константой, именем столбца или функцией, а также любым сочетанием арифметических, побитовых и строковых операторов. Функцию MIN можно использовать с колонками типа numeric, character и datetime, но не с колонками типа bit. |
| **SUM()** | Возвращает сумму всех (либо только уникальных) значений в выражении. Функция SUM может быть использована только для числовых колонок. Сумма всех значений выражения expression представлена в наиболее точном формате данных, используемом в выражении expression.SUM ( [ ALL \| DISTINCT ] expression ) |

# Спасибо за внимание!