SQLite Database

**Python SQLite**

**Python SQLite3** module is used to integrate the SQLite database with Python. It is a standardized Python DBI API 2.0 and provides a straightforward and simple-to-use interface for interacting with SQLite databases. There is no need to install this module separately as it comes along with Python after the 2.5x version.

This Python SQLite tutorial will help to learn how to use SQLite3 with Python from basics to advance with the help of good and well-explained examples and also contains Exercises for honing your skills.

**Introduction**

* [Introduction to SQLite in Python](https://www.geeksforgeeks.org/introduction-to-sqlite-in-python/)

SQLite is a lightweight, fast and embedded SQL database engine. SQLite is perfect for small to medium-sized applications, prototyping, embedded systems and local data storage in [Python](https://www.geeksforgeeks.org/python-programming-language/)applications because it doesn't require a separate server process like other relational database management systems (RDBMS) like **MySQL**or **PostgreSQL**.

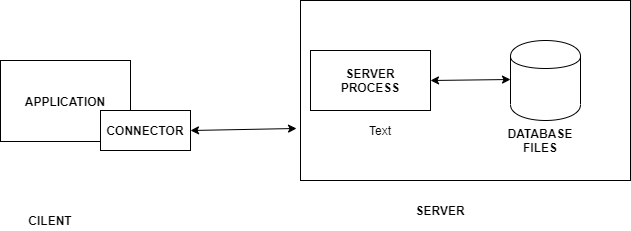
**Features of SQLite:**

1. Serverless
2. Self-Contained
3. Zero-Configuration
4. Transactional
5. Single-Database

We don't need to install anything additional to get started because Python has built-in support for SQLite through the **sqlite3 module**. Let's understance each of the features in detail.

**Serverless**

Generally, an [RDBMS](https://www.geeksforgeeks.org/rdbms-benefits-and-limitations/)such as **MySQL**, **PostgreSQL**, etc., needs a separate server process to operate. The applications that want to access the database server use**TCP/IP**protocol to send and receive requests and it is called **client/server**architecture. The diagram below illustrates the working of relational databases:

Working of Relational Databases

SQLite does not require a **server**to run. SQLite database read and write directly from the database files stored on disk and applications interact with that SQLite database. It is one of SQLite's biggest advantages is that it is **serverless**. Here's what that means:

* There is no separate server process to manage. The SQLite engine is**embedded directly** into the **application**.
* All you need is the **SQLite database file**, your program can **read**from and **write**to it without connecting to a remote service.
* This reduces **overhead**, **setup complexity** and **dependencies**, making SQLite perfect for desktop apps, mobile apps, IoT devices or lightweight data-driven Python scripts.

Working of SQLite

**Self-Contained**

SQLite is self-contained, here's what it means:

* It has **no external dependencies**. Everything it needs is included in a**single library**.
* The entire database (**schema + data**) is stored in a single **.sqlite** or **.db** file.
* This file can be **copied**,**backed up**, **shared** or moved like any other document.

**Key benefits of being Self-Contained:**

* **Portability**: Move the database file across machines or platforms without worrying about data loss or corruption.
* **Integration**: Easily bundle SQLite databases within applications, especially Python packages or tools.

**Zero-Configuration**

Unlike other databases, SQLite requires zero setup:

* No configuration files or startup services are needed.
* You can start using SQLite as soon as you import the sqlite3 module in Python.
* We can simply connect to a database file and if it doesn’t exist, SQLite automatically creates it.

**Example:** Connecting to a SQLite Database in Python

import sqlite3

conn = sqlite3.connect('example.db') # Creates a new database file if it doesn’t exist

cursor = conn.cursor()

**Transactional**

* SQLite supports full **ACID**(Atomicity, Consistency, Isolation, Durability) transactions:
* Every operation in **SQLite is atomic**, i.e changes are either fully applied or not applied at all.
* By default, SQLite wraps commands like INSERT, UPDATE and DELETE inside implicit transactions, ensuring data integrity.

We can also manage transactions explicitly using:

* BEGIN
* COMMIT
* ROLLBACKs.

**Transaction Control Example:**

conn.execute("BEGIN")

# perform database operations

conn.commit() # or conn.rollback() if something fails

**Single-Database**

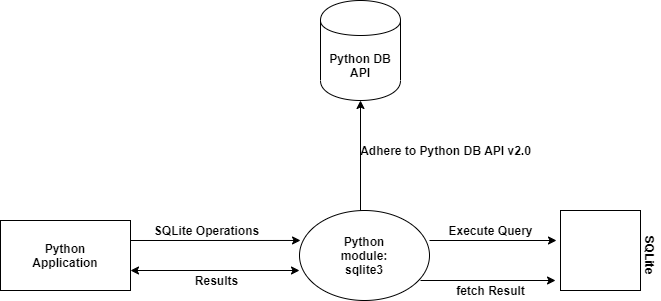
SQLite uses a single-file database architecture, meaning:

* The entire database, i.e. tables, indexes, triggers and data, lives in a single file only.
* This simplifies database management, as there's no need to manage multiple **configuration**or **log files**.
* The file can be used across different platforms, tools and programming languages.

Benefits of Single-File Storage:

* Easy to deploy and backup.
* Makes testing and debugging easier.
* Supports concurrent read operations (though writes are serialized).

**Working of SQLite in Python**

**Python includes built-in support for SQLite via the sqlite3 module, which conforms to the Python Database API Specification v2.0 (PEP 249).**

**Working:**

* We write Python code.
* The sqlite3 module handles connections, queries, transactions, etc.
* It interacts directly with the **.db** file (no server needed).

**Advanced Features in SQLite with Python**

While SQLite is simple to use, it offers several advanced features:

**Parameterized Queries**

To prevent SQL injection:

*cursor.execute("SELECT \* FROM users WHERE id = ?", (user\_id,))*

**Row Factory**

Fetch rows as dictionaries:

*conn.row\_factory = sqlite3.Row*

**In-Memory Databases**

For temporary, fast operations:

*conn = sqlite3.connect(':memory:')*

**Using with Statement for Safe Resource Handling:**

*with sqlite3.connect('example.db') as conn:   
cursor = conn.cursor()   
cursor.execute("SELECT \* FROM users")*

**Explanation:**

**import sqlite3:** imports Python’s built-in SQLite module, which allows interaction with SQLite databases.

**sqlite3.connect('example.db'):** establishes a connection to a SQLite database file named example.db.

* If the file does not exist, SQLite will automatically create it.
* This method returns a connection object that lets you interact with the database.

**conn.cursor():** creates a cursor object from the connection, which is used to execute SQL queries and fetch results.

* [Python SQLite – Connecting to Database](https://www.geeksforgeeks.org/python-sqlite-connecting-to-database/)

**Python SQLite - Connecting to Database**

Last Updated : 03 Feb, 2023

In this article, we'll discuss how to connect to an SQLite Database using the sqlite3 module in Python.

**Connecting to the Database**

Connecting to the SQLite Database can be established using the **connect()** method, passing the name of the database to be accessed as a parameter. If that database does not exist, then it'll be created.

sqliteConnection = sqlite3.connect('sql.db')

But what if you want to execute some queries after the connection is being made. For that, a cursor has to be created using the**cursor()** method on the connection instance, which will execute our SQL queries.

cursor = sqliteConnection.cursor()

print('DB Init')

The SQL query to be executed can be written in form of a string, and then executed by calling the **execute()**method on the cursor object. Then, the result can be fetched from the server by using the **fetchall()** method, which in this case, is the SQLite Version Number.

query = 'SQL query;'

cursor.execute(query)

result = cursor.fetchall()

print('SQLite Version is {}'.format(result))

Consider the below example where we will connect to an SQLite database and will run a simple query select sqlite\_version(); to find the version of the SQLite we are using

import sqlite3

try:

# Connect to DB and create a cursor

sqliteConnection = sqlite3.connect('sql.db')

cursor = sqliteConnection.cursor()

print('DB Init')

# Write a query and execute it with cursor

query = 'select sqlite\_version();'

cursor.execute(query)

# Fetch and output result

result = cursor.fetchall()

print('SQLite Version is {}'.format(result))

# Close the cursor

cursor.close()

# Handle errors

except sqlite3.Error as error:

print('Error occurred - ', error)

# Close DB Connection irrespective of success

# or failure

finally:

if sqliteConnection:

sqliteConnection.close()

print('SQLite Connection closed')

* [SQLite Datatypes and its Corresponding Python Types](https://www.geeksforgeeks.org/sqlite-datatypes-and-its-corresponding-python-types/)

**Storage Class in SQLite**

A storage class can be called a collection of similar DataTypes. SQLite provides the following storage classes:

| **Storage Class** | **Value Stored** |
| --- | --- |
| NULL | NULL |
| INTEGER | Signed Integer (1, 2, 3, 4, 5, or 8 bytes depending on magnitude) |
| REAL | Floating point value (8 byte IEEE floating-point numbers) |
| TEXT | TEXT string (encoded in UTF-8, UTF-16BE or UTF-16LE |
| BLOB (Binary Large Object) | Data stored exactly the way it was input, generally in binary format |

The term **Storage Class** can be used mutually with **DataType**.

**Corresponding Python Datatypes**

The SQLite DataTypes and their corresponding Python types are as follows

| **Storage Class** | **Python Datatype** |
| --- | --- |
| NULL | None |
| INTEGER | int |
| REAL | float |
| TEXT | str |
| BLOB | bytes |

The type() function can be used in python to get the class of an argument. In the program below, the type() function is used to print the classes of every value we store in a database.

***Note:*** *The program below uses the GeeksforGeeks logo as logo.png from* [*this*](https://www.geeksforgeeks.org/create-geeksforgeeks-logo-using-html-and-css/) *page for a demonstration.*

Let's take an example in which we are creating a database with the name 'gfg' and then create a table named exam\_hall having some columns which are as follows:

1. NAME (TEXT),
2. PIN (INTEGER),
3. OCCUPANCY (REAL),
4. LOGO (BLOB).

Then we insert some rows in it and check the data types of the values fetched with the help of SQL queries in Python.

*# Python3 program to demonstrate SQLite3 datatypes*

*# and corresponding Python3 types*

*# import the sqlite3 package*

**import** **sqlite3**

*# create connection to database*

cnt = sqlite3.connect('gfg.db')

*# Create a exam\_hall relation*

cnt.execute('''CREATE TABLE exam\_hall(

NAME TEXT,

PIN INTEGER,

OCCUPANCY REAL,

LOGO BLOB);''')

*# Open the logo file in read, binary mode*

*# read the image as binary data into a variable*

fileh = open('/content/JSBinCollaborativeJavaScriptDebugging6-300x160.png', 'rb')

img = fileh.read()

*# Insert tuples for the relation*

cnt.execute('''INSERT INTO exam\_hall VALUES(

'centre-a',1125,98.6,?)''', (img,))

cnt.execute('''INSERT INTO exam\_hall VALUES(

NULL,1158,80.5,?)''', (img,))

*# Query the data, print the data and its type*

*# note: Printing the image binary data is impractical due to its huge size*

*# instead number of bytes are being printed using len()*

cursor = cnt.execute('''SELECT \* FROM exam\_hall;''')

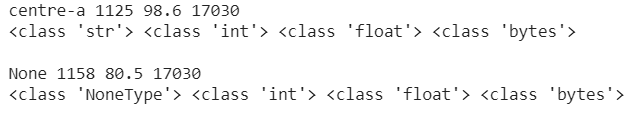
**for** i **in** cursor:

print(str(i[0])+" "+str(i[1])+" "+str(i[2])+" "+str(len(i[3])))

print(str(type(i[0]))+" "+str(type(i[1]))+" " +

str(type(i[2]))+" "+str(type(i[3]))+"**\n**")

**Output:**



From the output of this program, the following observations can be made:

1. 'centre-a' that was inserted as **TEXT** has been interpreted by python as **str**
2. 1125, 1158 that were inserted as **INTEGER** have been interpreted by python as **int**
3. 98.6, 80.5 that were inserted as **REAL**have been interpreted by python as **float**
4. **NULL** was interpreted by python as **NoneType**
5. The logo image which was inserted in binary format as **BLOB** has been interpreted by python as **bytes**.

**SQLite Queries**

* [Python SQLite – Cursor Object](https://www.geeksforgeeks.org/python-sqlite-cursor-object/)

A **Cursor** is an object used to execute SQL queries on an SQLite database. It acts as a middleware between the SQLite database connection and the SQL commands. It is created after establishing a connection to the SQLite database. **Example:**

**import** **sqlite3**

conn = sqlite3.connect('example.db')

c = conn.cursor() *# cursor*

print(c)

**Output**

<sqlite3.Cursor object at 0x000001AB6D9F7C00>

**Explanation:** In this code, we establish a connection to an SQLite database called example.db using **sqlite3.connect('example.db')**, creating it if it doesn't exist. Then, we create a cursor object **conn.cursor()** to execute SQL commands on the database.

**Cursor object syntax**

*cursor\_object = connection\_object.cursor()*

*cursor\_object.execute("SQL QUERY")*

**Parameters:**

* **connection\_object:** An active connection to the SQLite database, created with sqlite3.connect('your\_database.db').
* **cursor\_object.execute("SQL QUERY", parameters\_tuple):**Executes an SQL query.

**Returns: cursor\_object.execute()** returns the cursor object itself. After executing a SELECT query, results can be fetched using fetchone(), fetchall() or fetchmany(size).

**Important Methods of cursor**

Let's explore key cursor methods to understand how they interact with the SQLite database, making it easier to execute queries and fetch data efficiently.

| **Method** | **Description** |
| --- | --- |
| execute(sql\_query) | Executes a single SQL query |
| executemany(sql\_query, seq\_of\_parameters) | Executes SQL query against all parameter sequences |
| fetchone() | Fetches the next row of a query result set |
| fetchall() | Fetches all (remaining) rows of a query result set |
| fetchmany(size) | Fetches the next set of rows of a result set |
| close() | Closes the cursor |

**Cursor object Examples**

**Example 1:**In this example, we are creating a new table in the SQLite database called hotel (if it doesn't already exist) and inserting the data into the table.

**import** **sqlite3**

conn = sqlite3.connect('hotel\_data.db')

c = conn.cursor()

*# Create a new table*

c.execute('''

CREATE TABLE IF NOT EXISTS hotel (

FIND INTEGER PRIMARY KEY NOT NULL,

FNAME TEXT NOT NULL,

COST INTEGER NOT NULL,

WEIGHT INTEGER

)

''')

*# Insert records*

c.execute("INSERT INTO hotel (FIND, FNAME, COST, WEIGHT) VALUES (1, 'Cakes', 800, 10)")

c.execute("INSERT INTO hotel (FIND, FNAME, COST, WEIGHT) VALUES (2, 'Biscuits', 100, 20)")

c.execute("INSERT INTO hotel (FIND, FNAME, COST, WEIGHT) VALUES (3, 'Chocos', 1000, 30)")

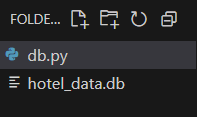
conn.commit()

print("Data inserted successfully.")

conn.close()

**Output**

Data inserted successfully**.**

hotel\_data.db created

**Explanation:**We connect to**hotel\_data.db**, create a cursor and use CREATE TABLE IF NOT EXISTS to create the hotel table. Three records are inserted, changes are committed with **conn.commit()** and the connection is closed.

**Example 2:** In this example, we are retrieving data from the hotel table in the SQLite database using the SELECT statement to fetch all the records.

**import** **sqlite3**

conn = sqlite3.connect('hotel\_data.db')

c = conn.cursor()

c.execute("SELECT \* FROM hotel")

*# Fetch all records*

rows = c.fetchall()

print("All Food Items:**\n**")

**for** row **in** rows:

print(f"Food ID: **{**row[0]**}**, Name: **{**row[1]**}**, Cost: **{**row[2]**}**, Weight: **{**row[3]**}**")

c.close()

**Output**

Retrieving data

**Explanation:**We connect to **hotel\_data.db**, create a cursor and execute a**SELECT \* FROM hotel**query to fetch all records. The records are printed and the connection is closed.

**Example 3:**In this example, we are retrieving specific columns (FIND and FNAME) from the hotel table in the SQLite database using the SELECT statement.

**import** **sqlite3**

conn = sqlite3.connect('hotel\_data.db')

c = conn.cursor()

c.execute("SELECT FIND, FNAME FROM hotel")

*# Fetch and print each row*

print("Food ID and Name:**\n**")

**for** row **in** c.fetchall():

print(f"Food ID: **{**row[0]**}**, Name: **{**row[1]**}**")

conn.close()

**Output**

Retrieving specific columns

**Explanation**: We connect to **hotel\_data.db**, create a cursor and execute a SELECT FIND, FNAME FROM hotel query to fetch the food ID and name. The results are printed and the connection is closed

**✅ 1. Create Table**

python

CopyEdit

import sqlite3

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute('''

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY,

name TEXT NOT NULL,

age INTEGER,

email TEXT

)

''')

conn.commit()

conn.close()

**✅ 2. Insert Data**

python

CopyEdit

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute("INSERT INTO users (name, age, email) VALUES (?, ?, ?)",

('Alice', 30, 'alice@example.com'))

# Insert multiple records

cursor.executemany("INSERT INTO users (name, age, email) VALUES (?, ?, ?)", [

('Bob', 25, 'bob@example.com'),

('Charlie', 35, 'charlie@example.com')

])

conn.commit()

conn.close()

**✅ 3. Select Data from Table**

python

CopyEdit

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM users")

rows = cursor.fetchall()

for row in rows:

print(row)

conn.close()

**✅ 4. WHERE Clause**

python

CopyEdit

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM users WHERE age > 30")

for row in cursor.fetchall():

print(row)

conn.close()

**✅ 5. ORDER BY Clause**

python

CopyEdit

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM users ORDER BY age DESC")

print(cursor.fetchall())

conn.close()

**✅ 6. LIMIT Clause**

python

CopyEdit

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM users LIMIT 2")

print(cursor.fetchall())

conn.close()

**✅ 7. JOIN Clause**

python

CopyEdit

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

# Example: Create another table

cursor.execute('''

CREATE TABLE IF NOT EXISTS orders (

order\_id INTEGER PRIMARY KEY,

user\_id INTEGER,

product TEXT,

FOREIGN KEY(user\_id) REFERENCES users(id)

)

''')

cursor.execute("INSERT INTO orders (user\_id, product) VALUES (?, ?)", (1, 'Laptop'))

# Join users and orders

cursor.execute('''

SELECT users.name, orders.product

FROM users

JOIN orders ON users.id = orders.user\_id

''')

print(cursor.fetchall())

conn.commit()

conn.close()

**✅ 8. Delete Data**

python

CopyEdit

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute("DELETE FROM users WHERE name = 'Charlie'")

conn.commit()

conn.close()

**✅ 9. DROP Table**

python

CopyEdit

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute("DROP TABLE IF EXISTS orders")

conn.commit()

conn.close()

**✅ 10. Update Data**

python

CopyEdit

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute("UPDATE users SET age = 28 WHERE name = 'Bob'")

conn.commit()

conn.close()

**✅ 11. Update Specific Column**

python

CopyEdit

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute("UPDATE users SET email = 'bob@newdomain.com' WHERE name = 'Bob'")

conn.commit()

conn.close()

**1. Check if Table Exists in SQLite using Python**

python

CopyEdit

import sqlite3

conn = sqlite3.connect("example.db")

cursor = conn.cursor()

table\_name = 'users'

cursor.execute("""

SELECT name FROM sqlite\_master

WHERE type='table' AND name=?;

""", (table\_name,))

exists = cursor.fetchone() is not None

print(f"Table '{table\_name}' exists:", exists)

conn.close()

**✅ 2. List All Tables in SQLite**

python

CopyEdit

conn = sqlite3.connect("example.db")

cursor = conn.cursor()

cursor.execute("SELECT name FROM sqlite\_master WHERE type='table';")

tables = cursor.fetchall()

print("Tables in DB:")

for t in tables:

print(t[0])

conn.close()

**✅ 3. Alter a SQLite Table (Add New Column)**

⚠️ SQLite has **limited support for ALTER TABLE** (can only rename table or add a column)

python

CopyEdit

conn = sqlite3.connect("example.db")

cursor = conn.cursor()

cursor.execute("ALTER TABLE users ADD COLUMN phone TEXT;")

conn.commit()

conn.close()

❌ You **cannot drop columns** or **change types** directly in SQLite without recreating the table.

**✅ 4. Delete a Specific Row from SQLite Table**

python

CopyEdit

conn = sqlite3.connect("example.db")

cursor = conn.cursor()

cursor.execute("DELETE FROM users WHERE id = ?", (2,)) # Example: delete id=2

conn.commit()

conn.close()

**✅ 5. Update All Values of a Specific Column**

python

CopyEdit

conn = sqlite3.connect("example.db")

cursor = conn.cursor()

# Example: Set all phone numbers to a default value

cursor.execute("UPDATE users SET phone = '+91-0000000000'")

conn.commit()

conn.close()

Let me know if you want:

* Ways to **drop columns** (manual workaround)
* **SQLite to Pandas DataFrame** bridge
* CRUD operations wrapped in reusable Python classes or functions

**1. Insert Image in SQLite using Python**

**🔸 Step 1: Store an image as binary (BLOB)**

python

CopyEdit

import sqlite3

# Convert image to binary

def convert\_to\_binary(filename):

with open(filename, 'rb') as file:

return file.read()

# Insert image into database

def insert\_image(image\_path):

conn = sqlite3.connect('image\_store.db')

cursor = conn.cursor()

cursor.execute('''

CREATE TABLE IF NOT EXISTS images (

id INTEGER PRIMARY KEY,

name TEXT,

data BLOB

)

''')

image\_data = convert\_to\_binary(image\_path)

cursor.execute("INSERT INTO images (name, data) VALUES (?, ?)",

(image\_path, image\_data))

conn.commit()

conn.close()

# Usage

insert\_image("cat.jpg")

**✅ 2. Read and Display Image from SQLite**

python

CopyEdit

import sqlite3

def read\_image(image\_id, output\_filename):

conn = sqlite3.connect('image\_store.db')

cursor = conn.cursor()

cursor.execute("SELECT data FROM images WHERE id = ?", (image\_id,))

record = cursor.fetchone()

with open(output\_filename, 'wb') as file:

file.write(record[0])

conn.close()

# Usage

read\_image(1, "retrieved\_cat.jpg")

**✅ 3. Storing and Retrieving OpenCV Images**

python

CopyEdit

import cv2

import sqlite3

import numpy as np

# Store OpenCV image

def store\_cv\_image(cv\_image, name='opencv\_img'):

conn = sqlite3.connect('image\_store.db')

cursor = conn.cursor()

cursor.execute('''

CREATE TABLE IF NOT EXISTS opencv\_images (

id INTEGER PRIMARY KEY,

name TEXT,

data BLOB

)

''')

\_, buffer = cv2.imencode('.png', cv\_image)

img\_bytes = buffer.tobytes()

cursor.execute("INSERT INTO opencv\_images (name, data) VALUES (?, ?)",

(name, img\_bytes))

conn.commit()

conn.close()

# Read OpenCV image

def read\_cv\_image(image\_id):

conn = sqlite3.connect('image\_store.db')

cursor = conn.cursor()

cursor.execute("SELECT data FROM opencv\_images WHERE id = ?", (image\_id,))

record = cursor.fetchone()

conn.close()

img\_array = np.frombuffer(record[0], dtype=np.uint8)

return cv2.imdecode(img\_array, cv2.IMREAD\_COLOR)

# Example usage

img = cv2.imread("cat.jpg")

store\_cv\_image(img)

retrieved\_img = read\_cv\_image(1)

cv2.imshow("Retrieved Image", retrieved\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**💡 Notes:**

* BLOB is used in SQLite for binary data like images.
* Use cv2.imencode() / cv2.imdecode() for OpenCV image storage.
* Always handle images as binary when reading/writing to SQLite.

Would you like to batch process multiple images or connect this with a web app or API?

4o

**Top of Form**

**fetchAll vs fetchMany**

**ChatGPT said:**

Great question! In Python’s sqlite3 module (and other DB-API compliant libraries), you often use fetchall() or fetchmany() with a cursor object to retrieve query results — but they work differently.

**✅ fetchall()**

* Retrieves **all** remaining rows of a query result.
* Returns a **list of tuples**.
* Be careful: If the result set is **very large**, this can consume a lot of memory.

**🔹 Example:**

python

CopyEdit

cursor.execute("SELECT \* FROM employee")

rows = cursor.fetchall()

for row in rows:

print(row)

**✅ fetchmany(size=N)**

* Retrieves **at most N rows** from the result set.
* Returns a **list of tuples**, just like fetchall().
* Good for handling large datasets in chunks (memory-efficient).

**🔹 Example:**

python

CopyEdit

cursor.execute("SELECT \* FROM employee")

while True:

rows = cursor.fetchmany(size=2)

if not rows:

break

for row in rows:

print(row)

**🧠 Summary Table:**

| **Method** | **Retrieves** | **Use Case** |
| --- | --- | --- |
| fetchall() | All rows at once | Small result sets |
| fetchmany(N) | N rows at a time | Large datasets (better for memory) |

Let me know if you'd like an example using pandas with fetchmany() as well!

import numpy as np

import pandas as pd

import sqlite3

# Create a connection to the SQLite database

connection = sqlite3.connect('practice.db')

# Create a cursor object to execute SQL commands

cursor = connection.cursor()

cursor.execute('''DROP TABLE IF EXISTS employee''')

# Create list of  Employee data json sample DataFrame

data = {

    'name': ['John', 'Jane', 'Doe', 'Alice', 'Bob'],

    'age': [28, 34, 29, 42, 25],

    'salary': [70000, 80000, 60000, 90000, 50000]

}

df = pd.DataFrame(data)

# Create a table in the SQLite database

print(df)

print('=====================================================')

print()

sql = ''' CREATE TABLE IF NOT EXISTS employee (

    name TEXT,

    age INTEGER,

    salary REAL

) '''

# Execute the SQL command to create the table

cursor.execute(sql)

# Insert the DataFrame into the SQLite database

cursor.executemany('INSERT INTO employee (name, age, salary) VALUES (?, ?, ?)', df.values.tolist())

connection.commit()

# Read the data from the SQLite database into a DataFrame

cursor.execute('SELECT \* FROM Employee')

# Fetch all rows from the result of the query

result = cursor.fetchall()

# Convert the result to a DataFrame

df\_result = pd.DataFrame(result, columns=['name', 'age', 'salary'])

# Print the DataFrame

print('=====================================================')

print('Data from SQLite database:')

print(df\_result)

# Close the cursor and connection

cursor.close()

connection.close()

Bottom of Form

* [Python SQLite – Create Table](https://www.geeksforgeeks.org/python-sqlite-create-table/)
* [Python SQLite – Insert Data](https://www.geeksforgeeks.org/python-sqlite-insert-data/)
* [Python SQLite – Select Data from Table](https://www.geeksforgeeks.org/python-sqlite-select-data-from-table/)
* [Python SQLite – WHERE Clause](https://www.geeksforgeeks.org/python-sqlite-where-clause/)
* [Python SQLite – ORDER BY Clause](https://www.geeksforgeeks.org/python-sqlite-order-by-clause/)
* [Python SQLite – LIMIT Clause](https://www.geeksforgeeks.org/python-sqlite-limit-clause/)
* [Python SQLite – JOIN Clause](https://www.geeksforgeeks.org/python-sqlite-join-clause/)
* [Python SQLite – Deleting Data in Table](https://www.geeksforgeeks.org/python-sqlite-deleting-data-in-table/)
* [Python SQLite – DROP Table](https://www.geeksforgeeks.org/python-sqlite-drop-table/)
* [Python SQLite – Update Data](https://www.geeksforgeeks.org/python-sqlite-update-data/)
* [Python SQLite – Update Specific Column](https://www.geeksforgeeks.org/python-sqlite-update-specific-column/)

**Working with Tables**

* [Check if Table Exists in SQLite using Python](https://www.geeksforgeeks.org/check-if-table-exists-in-sqlite-using-python/)
* [How to list tables using SQLite3 in Python ?](https://www.geeksforgeeks.org/how-to-list-tables-using-sqlite3-in-python/)
* [How to Alter a SQLite Table using Python ?](https://www.geeksforgeeks.org/how-to-alter-a-sqlite-table-using-python/)
* [How to Delete a Specific Row from SQLite Table using Python ?](https://www.geeksforgeeks.org/how-to-delete-a-specific-row-from-sqlite-table-using-python/)
* [How to Update all the Values of a Specific Column of SQLite Table using Python ?](https://www.geeksforgeeks.org/how-to-update-all-the-values-of-a-specific-column-of-sqlite-table-using-python/)

**Working with Images**

* [How to Insert Image in SQLite using Python?](https://www.geeksforgeeks.org/how-to-insert-image-in-sqlite-using-python/)
* [How to Read Image in SQLite using Python?](https://www.geeksforgeeks.org/how-to-read-image-in-sqlite-using-python/)
* [Storing OpenCV Image in SQLite3 with Python](https://www.geeksforgeeks.org/storing-opencv-image-in-sqlite3-with-python/)

**Exercises**

* [Count total number of changes made after connecting SQLite to Python](https://www.geeksforgeeks.org/count-total-number-of-changes-made-after-connecting-sqlite-to-python/)
* [How to Show all Columns in the SQLite Database using Python ?](https://www.geeksforgeeks.org/how-to-show-all-columns-in-the-sqlite-database-using-python/)
* [How to Count the Number of Rows of a Given SQLite Table using Python?](https://www.geeksforgeeks.org/how-to-count-the-number-of-rows-of-a-given-sqlite-table-using-python/)
* [How to import CSV file in SQLite database using Python ?](https://www.geeksforgeeks.org/how-to-import-csv-file-in-sqlite-database-using-python/)
* [How to Execute a Script in SQLite using Python?](https://www.geeksforgeeks.org/how-to-execute-a-script-in-sqlite-using-python/)
* [How to store Python functions in a Sqlite table?](https://www.geeksforgeeks.org/how-to-store-python-functions-in-a-sqlite-table/)
* [How to Create a Backup of a SQLite Database using Python?](https://www.geeksforgeeks.org/how-to-create-a-backup-of-a-sqlite-database-using-python/)
* [How to connect to SQLite database that resides in the memory using Python ?](https://www.geeksforgeeks.org/how-to-connect-to-sqlite-database-that-resides-in-the-memory-using-python/)
* [Change SQLite Connection Timeout using Python](https://www.geeksforgeeks.org/change-sqlite-connection-timeout-using-python/)