SQLite in Python: Store your scraped Data Fast

Need persistent storage? Skip the setup. Use SQLite.

Why SQLite?

Built into Python. No installation. No server. Just a file. Perfect for web scrapers data.

Setup

```
import sqlite3

# Connect (or create) database
con = sqlite3.connect('example.db')

# Create cursor to execute commands
cur = con.cursor()
```

If `example.db` doesn't exist, it gets created. If it does, you connect to it.

Create a Table

```
cur.execute('''
    CREATE TABLE IF NOT EXISTS tshirts (
        sku TEXT,
        name TEXT,
        size TEXT,
        price REAL
    )
''')
```

`TEXT` for strings. `REAL` for floats/decimals. `IF NOT EXISTS` prevents errors on reruns.

Insert Data

```
cur.execute('''
    INSERT INTO tshirts VALUES
    ('SKU123', 'Cool Shirt', 'Large', 19.99)
''')

# CRITICAL: Commit or nothing saves
con.commit()
```

Without `con.commit()`, your data vanishes. Always commit.

Query Data

```
for row in cur.execute('SELECT * FROM tshirts'):
    print(row)
```

`SELECT *` grabs everything. In production, specify columns instead.

The Problem: Duplicates

Run your insert code twice? Duplicate entries. Fix it with primary keys.

```
cur.execute('''
    CREATE TABLE IF NOT EXISTS tshirts (
        sku TEXT PRIMARY KEY,
        name TEXT,
        size TEXT,
        price REAL
    )
''')
```

Primary keys must be unique. Try inserting the same SKU twice? Error.

INSERT OR IGNORE

```
cur.execute('''
    INSERT OR IGNORE INTO tshirts VALUES
    ('SKU123', 'Cool Shirt', 'Large', 19.99)
''')

con.commit()
```

If the primary key exists, skip it. No errors. No duplicates. Run it 100 times, still one entry.

Real Example: Web Scraper

```
import sqlite3
import requests
from bs4 import BeautifulSoup

def scrape_books():
    url = 'http://books.toscrape.com/'
    r = requests.get(url)
    soup = BeautifulSoup(r.text, 'html.parser')

    books = []
```

```
    for book in soup.find_all('article', class_='product_pod'):
        title = book.h3.a['title']
        price = book.find('p', class_='price_color').text[1:]  #
        Remove £
        stock = book.find('p', class_='instock
        availability').text.strip()

        books.append((title, price, stock))

    return books

# Setup database
con = sqlite3.connect('books.db')
cur = con.cursor()

cur.execute('''
    CREATE TABLE IF NOT EXISTS books (
        title TEXT PRIMARY KEY,
        price TEXT,
        stock TEXT
    )
''')

# Get data
book_list = scrape_books()

# Insert all at once
cur.executemany('''
    INSERT OR IGNORE INTO books VALUES (?, ?, ?)
''', book_list)

con.commit()

# View results
for row in cur.execute('SELECT * FROM books'):
    print(row)
```

execute vs executemany

execute: One insert at a time

```
cur.execute('INSERT INTO books VALUES (?, ?, ?)', ('Title',
        '19.99', 'In stock'))
```

executemany: Insert a list of tuples

```
books = [
    ('Book 1', '19.99', 'In stock'),
    ('Book 2', '24.99', 'Out of stock')
```

```
]
cur.executemany('INSERT INTO books VALUES (?, ?, ?)', books)
```

Use `executemany` for bulk inserts. Way faster.

Why Tuples Over Dicts?

Databases have column headers. Dictionaries have keys. They're the same thing. Don't duplicate.

```
# This works but redundant
{'title': 'Book', 'price': '19.99'}

# Better - matches column order
('Book', '19.99')
```

Tuples are cleaner for database inserts.

View Your Data

Use DB Browser for SQLite. GUI for browsing, editing, and querying. Works on Windows, Mac, Linux.

The Pattern

1. Connect to database
2. Create cursor
3. Create table with `IF NOT EXISTS`
4. Insert with `INSERT OR IGNORE` (prevent duplicates)
5. Use primary keys
6. Always commit
7. Query when needed

Run your scraper daily. Only new data gets added. Old data stays safe. That's the power of `INSERT OR IGNORE` with primary keys.