Python e SQLite

Introduzione

Questo manuale copre l'integrazione tra **Python e SQLite** per la gestione di database. Include esempi pratici dal mondo reale (ristorante, inventario, playlist musicale) con il formato "Si scrive così" + esempio funzionante.

---

1. Cos'è SQLite

SQLite è un database **leggero** integrato in Python. Non richiede server separato - il database è un semplice file `.db`.

Caratteristiche:

- Zero configurazione - nessun server da installare
- Database in un singolo file (esempio: `database.db`)
- Perfetto per applicazioni piccole e prototipi
- Già incluso in Python (basta `import sqlite3`)

---

2. Architettura a 3 Livelli

La Struttura dell'Applicazione

Livello	Descrizione
FRONTEND	Interfaccia utente (CLI/Terminale) - Ciò che l'utente vede
BACKEND (Python)	Logica dell'applicazione - Elabora richieste e collega tutto
DATABASE (SQL)	Memoria dati - Salva e recupera informazioni

La Metafora del Ristorante

CLIENTE (Frontend)	CAMERIERE (Python)	DISPENSA (Database)

Si siede al tavolo	Prende l'ordine	Conserva ingredienti
Consulta il menu	Porta in cucina	Fornisce materiali
Fa un ordine	Verifica disponibilità	Su richiesta specifica
Non entra in cucina	Riporta il piatto	Usa linguaggio SQL

Python è il collante - Senza il cameriere (Python), il cliente (utente) non mangia e la cucina (database) resta isolata.

---

3. Connessione al Database

Come si Scrive - Si scrive così:

```
import sqlite3

conn = sqlite3.connect("nome_database.db")
cursor = conn.cursor()
```

Spiegazione componenti:

- `import sqlite3` - Importa la libreria per database
- `sqlite3.connect("file.db")` - Crea o apre il database
- `cursor()` - Crea un cursore per eseguire comandi SQL

Esempio Pratico:

```
import sqlite3

# Connessione al database (lo crea se non esiste)
conn = sqlite3.connect("ristorante.db")
cursor = conn.cursor()

print("Connessione al database riuscita!")
```

Output:
Connessione al database riuscita!

Cosa succede:

- Se `ristorante.db` non esiste → viene creato
- Se esiste già → viene aperto
- Il cursore è pronto per eseguire comandi SQL

---

4. Creare una Tabella (CREATE TABLE)

Come si Scrive - Si scrive così:

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS nome_tabella (
id INTEGER PRIMARY KEY,
colonna1 TEXT,
colonna2 INTEGER,
colonna3 REAL
)
""")
```

Tipi di Dato SQL

Tipo SQL	Equivalente Python	Esempio
INTEGER	int	42, -10, 0
TEXT	str	"Marco", "Ciao"
REAL	float	3.14, 12.50
PRIMARY KEY	-	Chiave unica auto-incrementale

Esempio - Tabella Menu Ristorante:

```
import sqlite3

conn = sqlite3.connect("ristorante.db")
cursor = conn.cursor()

# Creazione tabella menu

cursor.execute("""
CREATE TABLE IF NOT EXISTS menu (
id INTEGER PRIMARY KEY,
nomepiatto TEXT,
prezzo REAL
)
""")

print("Tabella 'menu' creata con successo!")
conn.close()
```

Output:
Tabella 'menu' creata con successo!

Spiegazione:

- `IF NOT EXISTS` → Crea solo se non esiste già (evita errori)
- `id INTEGER PRIMARY KEY` → Chiave univoca, si auto-incrementa
- `nomepiatto TEXT` → Colonna per il nome del piatto
- `prezzo REAL` → Colonna per il prezzo (numero decimale)

---

5. Inserire Dati (INSERT)

Inserimento Singolo - Si scrive così:

```
cursor.execute("INSERT INTO tabella (colonna1, colonna2) VALUES (?, ?)",
(valore1, valore2))
conn.commit()
```

⚠️ **REGOLA FONDAMENTALE:** Usa sempre `?` per i valori (protezione da SQL injection)

Esempio - Inserire un Piatto:

```
import sqlite3

conn = sqlite3.connect("ristorante.db")
cursor = conn.cursor()
```

Creazione tabella

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS menu (
id INTEGER PRIMARY KEY,
nomepiatto TEXT,
prezzo REAL
)
""")
```

Inserimento singolo piatto

```
cursor.execute("INSERT INTO menu (nomepiatto, prezzo) VALUES (?, ?)",
("Spaghetti alla Carbonara", 12.50))

conn.commit() # FONDAMENTALE per salvare!
print("Piatto inserito con successo!")
conn.close()
```

Output:
Piatto inserito con successo!

Inserimento Multiplo con Ciclo - Si scrive così:

```
lista_dati = [
{"campo1": valore1, "campo2": valore2},
{"campo1": valore3, "campo2": valore4}
]

for elemento in lista_dati:
cursor.execute("INSERT INTO tabella (campo1, campo2) VALUES (?, ?)",
(elemento["campo1"], elemento["campo2"]))

conn.commit()
```

Esempio - Inserire Ordinazioni Multiple:

```
import sqlite3

conn = sqlite3.connect("ordini.db")
cursor = conn.cursor()
```

Creazione tabella comande

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS comande (
id INTEGER PRIMARY KEY,
cliente TEXT,
piatto TEXT
)
""")
```

Lista delle ordinazioni

```
ordinazioni = [
{"cliente": "Silvio", "piatto": "Spaghetti alla puttanesca"},
{"cliente": "Matteo", "piatto": "Wurstel"},
{"cliente": "Andrea", "piatto": "Piccione"}
]
```

Inserimento con ciclo

```
for ordinazione in ordinazioni:
cursor.execute("INSERT INTO comande (cliente, piatto) VALUES (?, ?)",
(ordinazione["cliente"], ordinazione["piatto"]))

conn.commit()
print(f"Inserite {len(ordinazioni)} ordinazioni!")
conn.close()
```

Output:
Inserite 3 ordinazioni!

---

6. Recuperare Dati (SELECT)

SELECT Base - Si scrive così:

```
cursor.execute("SELECT * FROM tabella")
risultati = cursor.fetchall()
```

Metodi per recuperare risultati:

- `fetchall()` - Restituisce TUTTE le righe come lista di tuple
- `fetchone()` - Restituisce UNA SOLA riga (la prima)

Esempio - Mostrare Tutte le Ordinazioni:

```
import sqlite3

conn = sqlite3.connect("ordini.db")
cursor = conn.cursor()
```

SELECT di tutte le comande

```
cursor.execute("SELECT * FROM comande")
risultati = cursor.fetchall()

print("=== ORDINAZIONI ===")
for risultato in risultati:
print(f"Il cliente {risultato[1]} attende il piatto {risultato[2]}")

conn.close()
```

Output:
=== ORDINAZIONI ===
Il cliente Silvio attende il piatto Spaghetti alla puttanesca
Il cliente Matteo attende il piatto Wurstel
Il cliente Andrea attende il piatto Piccione

Spiegazione accesso tuple:

- `risultato[0]` → id
- `risultato[1]` → cliente
- `risultato[2]` → piatto

SELECT con Filtro (WHERE) - Si scrive così:

```
cursor.execute("SELECT * FROM tabella WHERE colonna = ?", (valore,))
risultati = cursor.fetchall()
```

Esempio - Filtrare Ingredienti Urgenti:

```
import sqlite3

conn = sqlite3.connect("inventario.db")
cursor = conn.cursor()
```

Creazione tabella ingredienti

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS ingredienti (
id INTEGER PRIMARY KEY,
nome TEXT,
quantita_kg REAL,
urgente TEXT
)
""")
```

Inserimento ingredienti

```
ingredienti = [
{"nome": "Farina", "quantita_kg": 2.5, "urgente": "SI"},
{"nome": "Zucchero", "quantita_kg": 1.0, "urgente": "NO"},
{"nome": "Uova", "quantita_kg": 0.5, "urgente": "SI"},
{"nome": "Latte", "quantita_kg": 2.0, "urgente": "NO"}
]

for ingrediente in ingredienti:
cursor.execute("INSERT INTO ingredienti (nome, quantita_kg, urgente) VALUES (?, ?, ?)",
(ingrediente["nome"], ingrediente["quantita_kg"], ingrediente["urgente"]))

conn.commit()
```

SELECT solo ingredienti urgenti

```
cursor.execute("SELECT * FROM ingredienti WHERE urgente = ?", ("SI",))
risultati = cursor.fetchall()

print("=== INGREDIENTI URGENTI ===")
for risultato in risultati:
print(f"ALLARME DISPENSA! Rifornire subito {risultato[1]}! Restano solo {risultato[2]}
kg.")

conn.close()
```

Output:
```
=== INGREDIENTI URGENTI ===
ALLARME DISPENSA! Rifornire subito Farina! Restano solo 2.5 kg.
ALLARME DISPENSA! Rifornire subito Uova! Restano solo 0.5 kg.
```

---

7. Salvare e Chiudere Connessione

Come si Scrive - Si scrive così:

conn.commit() # Salva modifiche (OBBLIGATORIO per INSERT/UPDATE/DELETE)
conn.close() # Chiude connessione

Quando Usare commit() e close()

Metodo	Quando Usarlo
`commit()`	Dopo INSERT, UPDATE, DELETE per rendere permanenti le modifiche
`close()`	Sempre alla fine, per liberare risorse

⚠️ **IMPORTANTE:** Senza `commit()` le modifiche vanno perse!

Esempio - Workflow Completo:

import sqlite3

1. CONNESSIONE

conn = sqlite3.connect("test.db")
cursor = conn.cursor()

2. CREAZIONE TABELLA

cursor.execute("""
CREATE TABLE IF NOT EXISTS utenti (
id INTEGER PRIMARY KEY,
nome TEXT,
eta INTEGER
)
""")

3. INSERIMENTO

cursor.execute("INSERT INTO utenti (nome, eta) VALUES (?, ?)", ("Marco", 25))

4. COMMIT (salvataggio permanente)

conn.commit()
print("Dati salvati!")

5. LETTURA

cursor.execute("SELECT * FROM utenti")
risultati = cursor.fetchall()
print(f"Utenti nel database: {risultati}")

6. CHIUSURA

```
conn.close()
print("Connessione chiusa!")
```

Output:
Dati salvati!
Utenti nel database: [(1, 'Marco', 25)]
Connessione chiusa!

---

8. SQL Injection - Come Evitarlo

❌ PERICOLOSO (SQL Injection):

```
nome = input("Nome: ")
query = f"INSERT INTO utenti VALUES ('{nome}')"
cursor.execute(query) # VULNERABILE!
```

Problema: Se l'utente inserisce `'); DROP TABLE utenti; --` il database viene distrutto!

✅ SICURO (Parametri con ?):

```
nome = input("Nome: ")
cursor.execute("INSERT INTO utenti (nome) VALUES (?)", (nome,)) # SICURO!
```

Confronto Visivo

❌ INSICURO	✅ SICURO
`f"...VALUES('{x}')"`	`"...VALUES (?)", (x,)`
`f"...WHERE id={id}"`	`"...WHERE id=?", (id,)`
Vulnerabile a injection	Protetto automaticamente

REGOLA D'ORO: Usa sempre `?` per valori dinamici e passa una tupla come secondo parametro.

---

9. Workflow Completo Python + SQLite

Schema Passo-Passo - Si scrive così:

```
import sqlite3
```

1. CONNESSIONE

```
conn = sqlite3.connect("database.db")
cursor = conn.cursor()
```

2. CREAZIONE TABELLA (se non esiste)

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS tabella (
id INTEGER PRIMARY KEY,
campo TEXT
)
""")
```

3. INSERIMENTO DATI

```
cursor.execute("INSERT INTO tabella (campo) VALUES (?)", ("valore",))
```

4. COMMIT (salvataggio)

```
conn.commit()
```

5. LETTURA DATI

```
cursor.execute("SELECT * FROM tabella")
risultati = cursor.fetchall()

for riga in risultati:
print(riga)
```

6. CHIUSURA

```
conn.close()
```

---

10. Esempio Completo: Gestione Playlist Musicale

Codice Completo Funzionante:

```
import sqlite3
```

1. CONNESSIONE

```
conn = sqlite3.connect("musica.db")
cursor = conn.cursor()
```

2. CREAZIONE TABELLA

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS canzoni (
id INTEGER PRIMARY KEY,
titolo TEXT,
artista TEXT,
anno INTEGER
)
""")
```

3. LISTA CANZONI DA INSERIRE

```
playlist = [
{"t": "Imagine", "a": "John Lennon", "anno": 1971},
{"t": "Billie Jean", "a": "Michael Jackson", "anno": 1982},
{"t": "Bohemian Rhapsody", "a": "Queen", "anno": 1975},
{"t": "Hotel California", "a": "Eagles", "anno": 1977}
]
```

4. INSERIMENTO MULTIPLO

```
for brano in playlist:
cursor.execute("INSERT INTO canzoni (titolo, artista, anno) VALUES (?, ?, ?)",
(brano["t"], brano["a"], brano["anno"]))

conn.commit()
print(f"✓ Inserite {len(playlist)} canzoni!\n")
```

5. RECUPERO TUTTE LE CANZONI

```
cursor.execute("SELECT * FROM canzoni")
risultati = cursor.fetchall()

print("=== PLAYLIST COMPLETA ===")
for riga in risultati:
print(f"{riga[0]}. {riga[1]} - {riga[2]} ({riga[3]})")
```

6. FILTRO: CANZONI DEGLI ANNI '70

```
print("\n=== CANZONI ANNI '70 ===")
cursor.execute("SELECT * FROM canzoni WHERE anno >= 1970 AND anno < 1980")
anni70 = cursor.fetchall()

for riga in anni70:
print(f"♫ {riga[1]} - {riga[2]}")
```

7. CONTARE CANZONI PER ARTISTA

```
print("\n=== STATISTICHE ===")
cursor.execute("SELECT COUNT(*) FROM canzoni")
totale = cursor.fetchone()[0]
print(f"Totale canzoni: {totale}")
```

8. CHIUSURA

```
conn.close()
print("\n✓ Database chiuso correttamente!")
```

Output Completo:

```
✓ Inserite 4 canzoni!

=== PLAYLIST COMPLETA ===
```

1. Imagine - John Lennon (1971)
2. Billie Jean - Michael Jackson (1982)
3. Bohemian Rhapsody - Queen (1975)
4. Hotel California - Eagles (1977)

```
=== CANZONI ANNI '70 ===
♫ Imagine - John Lennon
♫ Bohemian Rhapsody - Queen
♫ Hotel California - Eagles

=== STATISTICHE ===
Totale canzoni: 4

✓ Database chiuso correttamente!
```

---

11. Comandi SQL Principali

Tabella di Riferimento Rapido

Comando	Esempio
`CREATE TABLE`	`CREATE TABLE utenti (id INTEGER PRIMARY KEY, nome TEXT)`
`INSERT INTO`	`INSERT INTO utenti (nome) VALUES (?)`
`SELECT`	`SELECT * FROM utenti`
`WHERE`	`SELECT * FROM utenti WHERE eta > 18`
`UPDATE`	`UPDATE utenti SET nome=? WHERE id=?`
`DELETE`	`DELETE FROM utenti WHERE id=?`
`COUNT`	`SELECT COUNT(*) FROM utenti`

`ORDER BY`	`SELECT * FROM utenti ORDER BY nome`

---

12. Metodi Python per SQLite

Tabella Completa Metodi

Metodo	Descrizione
`sqlite3.connect(file)`	Connette o crea database
`conn.cursor()`	Crea cursore per eseguire comandi
`cursor.execute(sql, params)`	Esegue comando SQL
`cursor.fetchall()`	Recupera tutte le righe (lista di tuple)
`cursor.fetchone()`	Recupera una sola riga (tupla)
`conn.commit()`	Salva modifiche permanentemente
`conn.close()`	Chiude connessione

---

13. Differenze SQLite vs MySQL

Confronto Principale

Aspetto	SQLite	MySQL
Tipo	File locale (.db)	Server separato
Setup	Zero configurazione	Richiede installazione server
Auto-increment	`PRIMARY KEY` automatico	`AUTO_INCREMENT` esplicito
Connessione	`sqlite3.connect()`	Credenziali richieste
Uso ideale	App piccole, prototipi	Produzione, multi-utente
Concorrenza	Limitata	Alta

Sintassi Auto-incremento Confronto

SQLite:

```
CREATE TABLE canzoni (
id INTEGER PRIMARY KEY, -- Auto-incrementale automatico
titolo TEXT
)
```

MySQL:

```
CREATE TABLE canzoni (
id INT AUTO_INCREMENT,
titolo VARCHAR(255),
PRIMARY KEY (id)
)
```