**Gerenciando banco de dados SQLite3 com Python - Parte 1**

// Tags [Python](http://pythonclub.com.br/tag/python.html) [Banco de dados](http://pythonclub.com.br/tag/banco-de-dados.html)

Eu separei este post em duas partes: a **Parte 1** é bem elementar e objetiva, visando apresentar o básico sobre a realização do CRUD num banco de dados SQLite3 em Python usando o terminal.

A [parte 2](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte2.html) , num nível intermediário, usa classes e métodos mais elaborados para gerenciar o CRUD, e algumas coisinhas a mais.

Nota: Para entender o uso de classes e métodos leia o post [Introdução a Classes e Métodos em Python](http://pythonclub.com.br/introducao-classes-metodos-python-basico.html). E para entender os comandos SQL e a manipulação de registros no SQLite3 leia [Guia rápido de comandos SQLite3](http://pythonclub.com.br/guia-rapido-comandos-sqlite3.html).

Para os exemplos considere a tabela clientes e seus campos:

| **Campo** | **Tipo** | **Requerido** |
| --- | --- | --- |
| id | inteiro | sim |
| nome | texto | sim |
| idade | inteiro | não |
| cpf | texto (11) | sim |
| email | texto | sim |
| fone | texto | não |
| cidade | texto | não |
| uf | texto (2) | sim |
| criado\_em | data | sim |
| bloqueado | boleano | não |

Obs: O campo bloqueado nós vamos inserir depois com o comando ALTER TABLE.

Veja os exemplos em [github](https://github.com/rg3915/python-sqlite).

Como mencionado antes, esta parte será **básica e objetiva**. A intenção é realizar o CRUD da forma mais simples e objetiva possível.

PS: *Considere a sintaxe para Python 3*.

[Conectando e desconectando do banco](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#conectando-e-desconectando-do-banco)

[Criando um banco de dados](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#criando-um-banco-de-dados)

[Criando uma tabela](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#criando-uma-tabela)

[Create - Inserindo um registro com comando SQL](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#create-inserindo-um-registro-com-comando-sql)

[Inserindo n registros com uma tupla de dados](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#inserindo-n-registros-com-uma-tupla-de-dados)

[Inserindo um registro com parâmetros de entrada definido pelo usuário](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#inserindo-um-registro-com-parametros-de-entrada-definido-pelo-usuario)

[Read - Lendo os dados](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#read-lendo-os-dados)

[Update - Alterando os dados](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#update-alterando-os-dados)

[Delete - Deletando os dados](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#delete-deletando-os-dados)

[Adicionando uma nova coluna](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#adicionando-uma-nova-coluna)

[Lendo as informações do banco de dados](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#lendo-as-informacoes-do-banco-de-dados)

[Fazendo backup do banco de dados (exportando dados)](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#fazendo-backup-do-banco-de-dados-exportando-dados)

[Recuperando o banco de dados (importando dados)](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#recuperando-o-banco-de-dados-importando-dados)

[Exemplos](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#exemplos)

[Referências](http://pythonclub.com.br/gerenciando-banco-dados-sqlite3-python-parte1.html#referencias)

**Conectando e desconectando do banco**

Podemos criar o banco de dados de duas formas: na **memória RAM**

# conectando...

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

ou persistindo em um **banco de dados**, vamos usar sempre este caso.

# conectando...

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

Uma sintaxe mínima para se conectar a um banco de dados é:

# connect\_db.py

# 01\_create\_db.py

import sqlite3

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

conn.close()

O último método desconecta do banco.

Considere um arquivo para cada operação.

Nota: Os arquivos estão numerados apenas para sugerir uma sequência.

**Criando um banco de dados**

O código para criar um banco de dados é o mesmo mencionado anteriormente.

Para rodar este programa abra o **terminal** e digite:

$ python3 01\_create\_db.py

$ ls \*.db

Digitando ls você verá que o banco foi criado.

**Criando uma tabela**

Para criar uma tabela no banco de dados usamos dois métodos fundamentais:

* **cursor**: é um interador que permite navegar e manipular os registros do bd.
* **execute**: lê e executa comandos SQL puro diretamente no bd.

# 02\_create\_schema.py

import sqlite3

# conectando...

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

# definindo um cursor

cursor = conn.cursor()

# criando a tabela (schema)

cursor.execute("""

CREATE TABLE clientes (

id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

nome TEXT NOT NULL,

idade INTEGER,

cpf VARCHAR(11) NOT NULL,

email TEXT NOT NULL,

fone TEXT,

cidade TEXT,

uf VARCHAR(2) NOT NULL,

criado\_em DATE NOT NULL

);

""")

print('Tabela criada com sucesso.')

# desconectando...

conn.close()

Para executar digite no terminal:

$ python3 02\_create\_schema.py

$ sqlite3 clientes.db '.tables'

$ sqlite3 clientes.db 'PRAGMA table\_info(clientes)'

Digitando sqlite3 clientes.db '.tables' você verá que a tabela foi criada.

E o comando sqlite3 clientes.db 'PRAGMA table\_info(clientes)' retorna os campos da tabela.

**Nota**: A única diferença, caso você use *Python 2* é no print, onde você deve tirar os parênteses. E no início do arquivo é recomendável que se defina a codificação utf-8, que no caso do Python 3 já é padrão.

# 02\_create\_schema.py

# -\*- coding: utf-8 -\*-

# usando Python 2

import sqlite3

...

print 'Tabela criada com sucesso.'

Agora vamos fazer o CRUD. Começando com a letra

**Create - Inserindo um registro com comando SQL**

A única novidade aqui é o método **commit()**. É ele que grava de fato as alterações na tabela. *Lembrando que uma tabela é alterada com as instruções SQL ``INSERT, UPDATE`` e ``DELETE``.*

# 03\_create\_data\_sql.py

import sqlite3

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

cursor = conn.cursor()

# inserindo dados na tabela

cursor.execute("""

INSERT INTO clientes (nome, idade, cpf, email, fone, cidade, uf, criado\_em)

VALUES ('Regis', 35, '00000000000', 'regis@email.com', '11-98765-4321', 'Sao Paulo', 'SP', '2014-06-08')

""")

cursor.execute("""

INSERT INTO clientes (nome, idade, cpf, email, fone, cidade, uf, criado\_em)

VALUES ('Aloisio', 87, '11111111111', 'aloisio@email.com', '98765-4322', 'Porto Alegre', 'RS', '2014-06-09')

""")

cursor.execute("""

INSERT INTO clientes (nome, idade, cpf, email, fone, cidade, uf, criado\_em)

VALUES ('Bruna', 21, '22222222222', 'bruna@email.com', '21-98765-4323', 'Rio de Janeiro', 'RJ', '2014-06-09')

""")

cursor.execute("""

INSERT INTO clientes (nome, idade, cpf, email, fone, cidade, uf, criado\_em)

VALUES ('Matheus', 19, '33333333333', 'matheus@email.com', '11-98765-4324', 'Campinas', 'SP', '2014-06-08')

""")

# gravando no bd

conn.commit()

print('Dados inseridos com sucesso.')

conn.close()

Para executar digite no terminal:

$ python3 03\_create\_data\_sql.py

**Inserindo n registros com uma tupla de dados**

Usando uma *lista* podemos inserir vários registros de uma vez, e o método executemany faz essa ação.

# 04\_create\_data\_nrecords.py

import sqlite3

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

cursor = conn.cursor()

# criando uma lista de dados

lista = [(

'Fabio', 23, '44444444444', 'fabio@email.com', '1234-5678', 'Belo Horizonte', 'MG', '2014-06-09'),

('Joao', 21, '55555555555', 'joao@email.com',

'11-1234-5600', 'Sao Paulo', 'SP', '2014-06-09'),

('Xavier', 24, '66666666666', 'xavier@email.com', '12-1234-5601', 'Campinas', 'SP', '2014-06-10')]

# inserindo dados na tabela

cursor.executemany("""

INSERT INTO clientes (nome, idade, cpf, email, fone, cidade, uf, criado\_em)

VALUES (?,?,?,?,?,?,?,?)

""", lista)

conn.commit()

print('Dados inseridos com sucesso.')

conn.close()

Observe o uso de **?** isto significa que no lugar de cada **?** entrará os valores da lista na sua posição respectiva. É o que nós chamamos de *parâmetros de entrada*.

Para executar digite no terminal:

$ python3 04\_create\_data\_nrecords.py

**Inserindo um registro com parâmetros de entrada definido pelo usuário**

Neste exemplo usaremos parâmetros de entrada, que deverá ser digitado pelo usuário. Esta é a forma mais desejável de entrada de dados porque o usuário pode digitar os dados em tempo de execução.

# 05\_create\_data\_param.py

import sqlite3

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

cursor = conn.cursor()

# solicitando os dados ao usuário

p\_nome = input('Nome: ')

p\_idade = input('Idade: ')

p\_cpf = input('CPF: ')

p\_email = input('Email: ')

p\_fone = input('Fone: ')

p\_cidade = input('Cidade: ')

p\_uf = input('UF: ')

p\_criado\_em = input('Criado em (yyyy-mm-dd): ')

# inserindo dados na tabela

cursor.execute("""

INSERT INTO clientes (nome, idade, cpf, email, fone, cidade, uf, criado\_em)

VALUES (?,?,?,?,?,?,?,?)

""", (p\_nome, p\_idade, p\_cpf, p\_email, p\_fone, p\_cidade, p\_uf, p\_criado\_em))

conn.commit()

print('Dados inseridos com sucesso.')

conn.close()

**Nota**: Caso use *Python 2* use o método raw\_input() em

# python 2

p\_nome = raw\_input('Nome: ')

...

print 'Dados inseridos com sucesso.'

Para executar digite no terminal:

$ python3 05\_create\_data\_param.py

Veja a interação do programa:

Nome: Regis

Idade: 35

CPF: 30020030011

Email: regis@email.com

Fone: 11 9537-0000

Cidade: Sao Paulo

UF: SP

Criado em (yyyy-mm-dd): 2014-06-15

Dados inseridos com sucesso.

**Read - Lendo os dados**

Aqui nós usamos o famoso SELECT. O método fetchall() retorna o resultado do SELECT.

# 06\_read\_data.py

import sqlite3

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

cursor = conn.cursor()

# lendo os dados

cursor.execute("""

SELECT \* FROM clientes;

""")

for linha in cursor.fetchall():

print(linha)

conn.close()

Para executar digite no terminal:

$ python3 06\_read\_data.py

Eis o resultado:

(1, 'Regis', 35, '00000000000', 'regis@email.com', '11-98765-4321', 'Sao Paulo', 'SP', '2014-06-08')

(2, 'Aloisio', 87, '11111111111', 'aloisio@email.com', '98765-4322', 'Porto Alegre', 'RS', '2014-06-09')

(3, 'Bruna', 21, '22222222222', 'bruna@email.com', '21-98765-4323', 'Rio de Janeiro', 'RJ', '2014-06-09')

(4, 'Matheus', 19, '33333333333', 'matheus@email.com', '11-98765-4324', 'Campinas', 'SP', '2014-06-08')

(5, 'Fabio', 23, '44444444444', 'fabio@email.com', '1234-5678', 'Belo Horizonte', 'MG', '2014-06-09')

(6, 'Joao', 21, '55555555555', 'joao@email.com', '11-1234-5600', 'Sao Paulo', 'SP', '2014-06-09')

(7, 'Xavier', 24, '66666666666', 'xavier@email.com', '12-1234-5601', 'Campinas', 'SP', '2014-06-10')

(8, 'Regis', 35, '30020030011', 'regis@email.com', '11 9750-0000', 'Sao Paulo', 'SP', '2014-06-15')

**Update - Alterando os dados**

Observe o uso das variáveis id\_cliente onde definimos o id a ser alterado, novo\_fone e novo\_criado\_em usados como parâmetro para alterar os dados. Neste caso, salvamos as alterações com o método commit().

# 07\_update\_data.py

import sqlite3

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

cursor = conn.cursor()

id\_cliente = 1

novo\_fone = '11-1000-2014'

novo\_criado\_em = '2014-06-11'

# alterando os dados da tabela

cursor.execute("""

UPDATE clientes

SET fone = ?, criado\_em = ?

WHERE id = ?

""", (novo\_fone, novo\_criado\_em, id\_cliente))

conn.commit()

print('Dados atualizados com sucesso.')

conn.close()

Para executar digite no terminal:

$ python3 07\_update\_data.py

**Delete - Deletando os dados**

Vamos excluir um registro pelo seu id.

# 08\_delete\_data.py

import sqlite3

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

cursor = conn.cursor()

id\_cliente = 8

# excluindo um registro da tabela

cursor.execute("""

DELETE FROM clientes

WHERE id = ?

""", (id\_cliente,))

conn.commit()

print('Registro excluido com sucesso.')

conn.close()

Para executar digite no terminal:

$ python3 08\_delete\_data.py

**Adicionando uma nova coluna**

Para inserir uma nova coluna na tabela usamos o comando SQL ALTER TABLE.

# 09\_alter\_table.py

import sqlite3

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

cursor = conn.cursor()

# adicionando uma nova coluna na tabela clientes

cursor.execute("""

ALTER TABLE clientes

ADD COLUMN bloqueado BOOLEAN;

""")

conn.commit()

print('Novo campo adicionado com sucesso.')

conn.close()

Para executar digite no terminal:

$ python3 09\_alter\_table.py

**Lendo as informações do banco de dados**

Para ler as informações da tabela usamos o comando PRAGMA.

Para listar as tabelas do banco usamos o comando SELECT name FROM sqlite\_master ....

Para ler o schema da tabela usamos o comando SELECT sql FROM sqlite\_master ....

# 10\_view\_table\_info.py

import sqlite3

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

cursor = conn.cursor()

nome\_tabela = 'clientes'

# obtendo informações da tabela

cursor.execute('PRAGMA table\_info({})'.format(nome\_tabela))

colunas = [tupla[1] for tupla in cursor.fetchall()]

print('Colunas:', colunas)

# ou

# for coluna in colunas:

# print(coluna)

# listando as tabelas do bd

cursor.execute("""

SELECT name FROM sqlite\_master WHERE type='table' ORDER BY name

""")

print('Tabelas:')

for tabela in cursor.fetchall():

print("%s" % (tabela))

# obtendo o schema da tabela

cursor.execute("""

SELECT sql FROM sqlite\_master WHERE type='table' AND name=?

""", (nome\_tabela,))

print('Schema:')

for schema in cursor.fetchall():

print("%s" % (schema))

conn.close()

Para executar digite no terminal:

$ python3 10\_view\_table\_info.py

Eis o resultado:

Colunas: ['id', 'nome', 'idade', 'cpf', 'email', 'fone', 'cidade', 'uf', 'criado\_em', 'bloqueado']

Tabelas:

clientes

sqlite\_sequence

Schema:

CREATE TABLE clientes (

id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

nome TEXT NOT NULL,

idade INTEGER,

cpf VARCHAR(11) NOT NULL,

email TEXT NOT NULL,

fone TEXT,

cidade TEXT,

uf VARCHAR(2) NOT NULL,

criado\_em DATE NOT NULL

, bloqueado BOOLEAN)

**Fazendo backup do banco de dados (exportando dados)**

Talvez seja este o item mais importante: **backup**. Observe o uso da biblioteca **io** que salva os dados num arquivo externo através do método write, e o método iterdump() que exporta a estrutura e dados da tabela para o arquivo externo.

# 11\_backup.py

import sqlite3

import io

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

with io.open('clientes\_dump.sql', 'w') as f:

for linha in conn.iterdump():

f.write('%s\n' % linha)

print('Backup realizado com sucesso.')

print('Salvo como clientes\_dump.sql')

conn.close()

Para executar digite no terminal:

$ python3 11\_backup.py

$ cat clientes\_dump.sql

Com o comando cat você poderá ler a estrutura da tabela salva.

**Recuperando o banco de dados (importando dados)**

Criaremos um novo banco de dados e iremos reconstruir a tabela e os dados com o arquivo *clientes\_dump.sql*. O método read() lê o conteúdo do arquivo *clientes\_dump.sql* e o método executescript() executa as instruções SQL escritas neste arquivo.

# 12\_read\_sql.py

import sqlite3

import io

conn = sqlite3.connect('clientes\_recuperado.db')

cursor = conn.cursor()

f = io.open('clientes\_dump.sql', 'r')

sql = f.read()

cursor.executescript(sql)

print('Banco de dados recuperado com sucesso.')

print('Salvo como clientes\_recuperado.db')

conn.close()

Para executar digite no terminal:

$ python3 12\_read\_sql.py

Banco de dados recuperado com sucesso.

Salvo como clientes\_recuperado.db

$ sqlite3 clientes\_recuperado.db 'SELECT \* FROM clientes;'