Python SQLite

SQLite

●SQLiteとは

SQLiteとは、オープンソースのRDBMS(リレーショナルデータベース管理システム)です。簡易的なデータベースシステムで、アプリケーションに組み込んで利用されます。SQLiteはサーバではなく、ライブラリです。そのためサーバの様な設定ファイルなどを用意せずに使用することができます。またマルチプラットフォームに対応しており、windows,mac,linuxやAndroid,iosなどのスマートフォンなどにも対応しています。また、本体ファイルが600KB程度と非常に軽量であることも特徴です。pythonは標準ライブラリでSQLiteを持っていますので、インポートするだけでSQLiteを使用することができます。

●データベースを操作できるデータベースオブジェクトの作成

import sqlite3

データベースオブジェクト名=sqlite3.connect(データベース名)

sqlite3をインポートし「connect」コマンドで、対象のデータベースに接続し、データベースを操作できるオブジェクトを作成します。この時、対象のデータベースが無い場合は空のデータベースが作成されます。

1	import sqlite3	
2	con=sqlite3.connect('test.db')	test.dbに接続(無ければ作成)

●カーソルの作成

カーソル名=データベースオブジェクト名.cursor()

データベースを操作するためには、カーソルを使用する必要があります。カーソルはデータベースの処理を一行づつ行うための仕組みです。

1	import sqlite3	
2	con=sqlite3.connect('test.db')	
3	c=con.cursor()	カーソルを作成

●SQL文実行

カーソル名.execute("SQL文")

データベースオブジェクト名.commit()

「execute」の引数にSQL文を記載することで、SQLiteを操作することができます。実行したSQL文を反映させるためには「commit」コマンドを使用します。詳細は後述します。

●終了処理

データベースオブジェクト名.close()

「close」コマンドを実行することにより、データベースとの接続を解除し、オブジェクトの破棄を行えます。

データ定義語(DDL)

●テーブルの作成(create)

create table テーブル名(カラム1,カラム2,,)

まずは、構文に従い商品名(p_name)、商品の価格(p_price)をカラムに持つ商品テーブル(p_table)を作成してみましょう。

1	import sqlite3	
2	con=sqlite3.connect("test.db")	データベースに接続
3	c=con.cursor()	カーソルを作成
4	c.execute("create table p_table(p_name, p_price)")	テーブルを作成
5	con.close()	データベースを閉じる

上記コードでテーブルが作成されています。MySQLで学んだような、データの型の指定や文字数の指定をしなくてもよいです。一度テーブルを作成した後に再度同じ名前のテーブルを作成しようとすると、エラーがでますので注意してください。

●テーブルの表示(sqlit_master)

select name from sqlite_master

SQLiteには、MYSQLのように「SHOW」コマンドがありません。データベースのテーブル一覧を見るときは、すべてのテーブル情報を管理している「sqlite_master」テーブルを確認します。

1	import sqlite3	
2	con=sqlite3.connect("test.db")	
3	c=con.cursor()	
4	data=c.execute("select name from sqlite_master").fetchone()	一つだけ表示
5	print(data)	
6	con.close()	

sqlite_master：上記4行目を下記構文に変えるとテーブルのカラムが確認できる
data=c.execute("pragma table_info('sqlite_master')").fetchall()

typy	name	tbl_name	root_page	sql
tableかindex	テーブル名かindex名	テーブル名	ー	テーブルを作るためのSQL文

●fetch

オプション	内容
fetchone()	カーソルの位置から１つのデータを取得
fetchall()	カーソルの位置からすべてのデータを取得
fetchmany(size)	カーソルの位置からsizeの数だけデータを取得

「fetch」はカーソルの位置からデータを抜き出します。データを取得する際に便利な「fetch」は３種類ありますので、それぞれの使用用途に合わせ活用してください。

●テーブルの編集(alter)

alter table テーブル名 rename to 新しいテーブル名

alterを使用し、テーブル名の変更ができます。

1	import sqlite3	
2	con=sqlite3.connect("test.db")	
3	c=con.cursor()	
4	c.execute("alter table p_table rename to product_table")	テーブル名変更
5	data=c.execute("select name from sqlite_master")	一覧取得
6	print(data.fetchall())	テーブル表示
7	con.close()	

sqliteの公式ドキュメントによると、alterはテーブル名のリネーム、カラムの削除、カラムの追加ができます。

カラムの追加：

alter table テーブル名 add column 追加するカラム名

カラムの削除：

alter table テーブル名 drop column カラム名

●テーブルの削除(drop)

drop table テーブル名

Dropを使用し、テーブルの削除ができます。指定のテーブルが無い場合はエラーが出ますので、削除する前に、存在確認「if exists」をします。

1	import sqlite3	
2	con=sqlite3.connect("test.db")	
3	c=con.cursor()	指定されたテーブルがあれば削除を実行
4	c.execute("drop table **if exists** product_table")	
5	data=c.execute("select name from sqlite_master")	一覧取得
6	print(data.fetchall())	テーブル表示
7	con.close()	

※if exists テーブル名　　：指定されたテーブルが有れば
　if not exists テーブル名：指定されたテーブルが無ければ

制約

●制約

テーブル作成時にカラムに対する制約を行えます。SQLiteには下記の制約があります。

制約	内容
primary key 制約	主キーの宣言
autoincrement 制約	自動的に数値を格納する
not null 制約	NULL値を許可しない
unique 制約	重複を排除する
default 制約	データを入力しない時の初期設定値を指定
check 制約	入力値の条件を指定する

●primary key　制約

create table テーブル名(カラム1　primary key, , ,)

複合キー：

create table テーブル名(カラム1,カラム2,,,
primary key(カラム1,カラム2))

カラムに「primary key」を設定した場合に、そのカラムが主キーになります。主キーは１つ、または複数のカラムの組み合わせに対して設定することができます。主キーが設定されたカラムは重複データが許されません。重複した値を入れようとするとエラーが出ます。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	c.execute("drop table if exists t_table")	t_tableがあれば削除
5	c.execute("create table t_table(t_no primary key, t_name,t_curriculum)")	主キー： t_no
6	c.execute("create table s_table(s_grade,s_no,s_name, s_tel, primary key(s_grade,s_no))")	複合キー： s_grade,s_no
7	data=c.execute("select name from sqlite_master")	
8	print(data.fetchall())	
9	con.close()	

●autoincrement　制約

create table テーブル名

(カラム1 integer primary key autoincrement,カラム2,,)

データ型がinteger型のカラムに対してpraimary keyとautoincrementを設定した場合、データを追加すると自動で最大値+1が追加されます。例えばカラム1の値が10の場合次にデータを入力したときにカラム1の値がnull値であった場合「11」が自動で入ります。
また、autoincrementを設定していない場合でもinteger型のカラムにpraimary keyを設定するだけで同じように最大値+1の値を自動で入力します。しかし、もしカラム1の最大値が10の場合、カラム1が10である行を削除し、再度データを入力した際に違いが出ます。autoincrementを入力してない場合はカラム1の値が「10」になり、autoincrementを設定した場合のカラム1の値は「11」になります。よってデータの削除などを行うデータベースの場合はautoincrementを設定したほうがよいでしょう。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	c.execute("drop table if exists t_table")	t_tableを削除
5	c.execute("create table t_table(t_no integer primary key autoincrement,t_name,t_curriculum)")	autoincrementを設定して再度t_tableを作成
6	c.execute("insert into t_table(t_name,t_curriculum) values('宮沢','国語')")	t_name,t_curriculumだけを入力
7	c.execute("insert into t_table(t_name,t_curriculum) values('ラマヌジャン','数学')")	
8	c.execute("insert into t_table(t_name,t_curriculum) values('湯川','物理')")	
9	c.execute("insert into t_table(t_name,t_curriculum) values('野口','科学')")	
10	c.execute("delete from t_table where t_name ='野口'")	
11	c.execute("insert into t_table(t_name,t_curriculum) values('保井','科学')")	
12	con.commit()	
13	data=c.execute("select * from t_table")	t_noが自動で入力される t_noの4が削除されたため欠番になっている
14	print(data.fetchall())	
15	con.close()	

t_noが自動で割り当てられ、削除された行のt_noは欠番となっている。この働きによりほかのテーブルとの整合性も確保されるため主キーにはautoincrementを設定するほうが良い。

●rowid

SQLiteでデータを入力すると、rowidという値が自動的に割り振られています。rowidは非表示のカラムです。rowidはこれまでのrowidの最大値に1を足したものが自動で割り振られます。
primary keyと同じ値をとります。
t_tableのデータで確認してみましょう。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	data=c.execute("select * ,rowid from t_table")	非表示のrowidも表示
5	print(data.fetchall())	
6	con.close()	

●not null　制約

create table テーブル名(カラム1 not null,,,,,)

カラムに格納する値にnull値を禁止したい場合に、カラムにnot null制約を設定します。必ず値を設定しないといけない項目にnot nullを設定します。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	c.execute("drop table if exists p_table")	
5	c.execute("create table p_table(p_name,p_price not null)")	null制約
6	c.execute("insert into p_table values('ミカン',150)")	
7	c.execute("insert into p_table values('リンゴ',250)")	
8	c.execute("insert into p_table values('バナナ',null)")	エラー
9	c.execute("insert into p_table(p_name) values('バナナ')")	
10	con.commit()	
11	data=c.execute("select * from p_table")	
12	print(data.fetchall())	
13	con.close()	

8行目、9行目で「NOT NULL constraint failed: p_table.p_price」エラーが表示される

●unique　制約

create table テーブル名(カラム1 unique,カラム2,,,,)

複数：

create table テーブル名(カラム1,カラム2,,,, unique(カラム1,カラム2))

カラムにunique制約を設定すると対象のカラムには重複した値を入れることができなくなります。また、複数のカラムに対しunique制約を設定すると、それぞれのカラムが重複してもよいのですが組み合わせの重複ができなくなります。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	c.execute("drop table if exists p_table")	
5	c.execute("create table p_table(p_name unique ,p_price)")	
6	c.execute("insert into p_table values('ミカン',150)")	ミカンを重複して入れようとしているため下記エラーが表示される UNIQUE constraint failed: p_table.p_name
7	c.execute("insert into p_table values('リンゴ',250)")	
8	c.execute("insert into p_table values('ミカン',200)")	
9	con.commit()	

複数：

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	c.execute("drop table if exists p_table")	
5	c.execute("create table p_table(p_name,p_price,unique(p_name,p_price))")	
6	c.execute("insert into p_table values('ミカン',150)")	
7	c.execute("insert into p_table values('リンゴ',250)")	
8	c.execute("insert into p_table values('ミカン',200)")	６行目とは組み合わせが違うためエラーは表示されない
9	con.commit()	
10	data=c.execute("select * from p_table")	
11	print(data.fetchall())	
12	con.close()	

●default　制約

create table テーブル名(カラム1 default 値, ,,,,)

データを追加する時にデータの入力値を省略した場合に、default値に設定した値が自動で入力されます。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	c.execute("drop table if exists p_table")	
5	c.execute("create table p_table(p_name ,p_price default 100)")	
6	c.execute("insert into p_table values('ミカン',150)")	
7	c.execute("insert into p_table(p_name) values('リンゴ')")	p_priceにdefault値の100が入力される
8	c.execute("insert into p_table(p_name) values('バナナ')")	
9	con.commit()	
10	data=c.execute("select * from p_table")	
11	for a in data:	
12	print(a)	
13	con.close()	

●check　制約

create table テーブル名(カラム1 check(条件式),,,,)

カラムにcheck制約を設定することで入力値に制限を設けることができます。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	c.execute("drop table if exists p_table")	
5	c.execute("create table p_table(p_name ,p_price check(p_price >=0))")	金額0以上
6	c.execute("insert into p_table values('ミカン',150)")	
7	c.execute("insert into p_table values('リンゴ',0)")	金額がマイナスなのでエラー
8	c.execute("insert into p_table values('バナナ',-100)")	

データ操作言語(DML)

●データの挿入(insert)

insert into テーブル名 values (データ1,データ2,,,,)

データの挿入もSQL文を使用することでできます。SQL文自体は変わりません。

1	import sqlite3	
2	con=sqlite3.connect("test.db")	
3	c=con.cursor()	
4	c.execute("drop table if exists p_table")	
5	c.execute("create table p_table(p_name ,p_price)")	
6	c.execute("insert into p_table values('ミカン',180)")	
7	c.execute("insert into p_table values('リンゴ',250)")	
8	c.execute("insert into p_table values('バナナ',280)")	
9	con.commit()	
10	data1=c.execute("select * from p_table")	
11	print(data1.fetchone())	1つのデータを表示する
12	data2=c.execute("select * from p_table")	
13	print(data2.fetchall())	すべてのデータを表示する
14	data3=c.execute("select * from p_table")	
15	print(data3.fetchmany(2))	指定した量のデータを表示する
16	con.close()	

●データの更新(update)

update テーブル名 set カラム名=更新値 where 条件

データの更新も同じSQL構文を使用することでデータの更新ができます。

1	import sqlite3	
2	con=sqlite3.connect("test.db")	
3	c=con.cursor()	
4	c.execute("update p_table set p_price=150 where p_name='ミカン'")	ミカンの値段を150に更新
5	data=c.execute("select * from p_table")	
6	print(data.fetchall())	
7	con.close()	

●データの削除(delete)

delete from テーブル名 where 条件

1	import sqlite3	
2	con=sqlite3.connect("test.db")	
3	c=con.cursor()	
4	c.execute("delete from p_table where p_name='ミカン'")	ミカンを削除
5	data=c.execute("select * from p_table")	
6	print(data.fetchall())	
7	con.close()	

データの取得(select)

●特定のカラムを取得(射影)

select カラム1,カラム2 from テーブル名

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	data=c.execute("select * from t_table")	すべて表示
5	for a in data:	
6	print(a)	
7	data=c.execute("select t_name from t_table")	名前の列だけ表示
8	for a in data:	
	print(a)	
	con.close()	

●条件に合った行を取得(選択)

select * from テーブル名 where 条件

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	data=c.execute("select * from t_table where t_name='湯川'")	
5	print(data.fetchall())	
6	con.close()	

上記例は「t_name」が湯川という値のデータを取得します。これを検索値を入力してもらうには下記のように書きます。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	name=input("検索名を入力してください")	
5	data=c.execute(f"select * from t_table where t_name='{name}'")	
6	print(data.fetchall())	
7	con.close()	

このように書くと検索名を入力し検索することができます。
それでは検索名に「**'a' or True**」と入力してみてください。
すると全てのデータが表示されます。これは、意図的にSQL文を検索値に紛らせ、不正にデータベースを操作する、いわゆるSQLインジェクション攻撃です。
このプログラムの作成方法では、SQLインジェクション攻撃に脆弱なプログラムとなってしまいます。そこで、プレースホルダという方法で対策を行います。プレースホルダとは実際の値を後から挿入するために、とりあえず仮に確保した場所という意味です。それをSQL文の中に適用します。

●?プレースホルダー

“select * from テーブル名 where カラム=?”,(検索値)

このようにSQL文の中に直接変数を入力するのではなく「?」を使用することで、「?」に入力される値は純粋に文字列として扱われるため、たとえSQL文を書いたとしても無効化されます。プレースホルダーはpythonだけではなく、PHPや他の言語でも使用しています。SQLを操作する場合はプレースホルダーを使用しましょう。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	name=input("検索名を入力してください")	
5	data=c.execute("select * from t_table where t_name=?",(name,))	プレースホルダーを使用
6	print(data.fetchall())	
7	con.close()	

※executeの第二引数はタプルなので、内容が１つの時は「,」をつけて明示的にタプルにする必要があります。
他にも「?プレースホルダー」を使用してセキュアなプログラムにしましょう。

・**insert**

c.execute(“insert into テーブル名 values (?,?,,,,)”,(データ1,データ2,,,)

・**update**

c.execute(“update テーブル名 set カラム名=? where 条件”,(更新値,))

・**delete**

c.execute(“delete from テーブル名 where カラム=?”,(カラム値))

●重複排除

“select distinct カラム1,カラム2 from テーブル名”

「distinct」は重複排除のコマンドです。あくまで表示(select)に対して重複を排除しているだけなので、データ自体を削除などはしていません。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	c.execute("drop table if exists p_table")	
5	c.execute("create table p_table(p_name, p_price)")	
6	c.execute("insert into p_table values(?,?)",("ミカン",150))	プレースホルダー使用
7	c.execute("insert into p_table values(?,?)",("リンゴ",250))	
8	c.execute("insert into p_table values(?,?)",("ミカン",100))	
9	c.execute("insert into p_table values(?,?)",("リンゴ",350))	
10	c.execute("insert into p_table values(?,?)",("バナナ",100))	
11	con.commit()	
12	data=c.execute("select distinct p_name from p_table")	重複を排除して表示
13	print(data.fetchall())	
14	con.close()	

・プレースホルダーを使用した、更新と削除

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	data1=c.execute("select *,rowid from p_table")	
5	print(data1.fetchall())	
6	c.execute("update p_table set p_price=? where p_name=?",(500,"バナナ"))	update
7	c.execute("delete from p_table where rowid=?",(3,))	delete
8	data2=c.execute("select *,rowid from p_table")	
9	print(data2.fetchall())	
10	con.close()	

●ソート(order by)

select * from テーブル名 order by カラム名[asc:昇順 | desc:降順]

Selectしたデータを、昇順、降順に並べ替えができます。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	data=c.execute("select * from p_table order by p_price desc")	値段で降順に表示
5	print(data.fetchall())	
6	con.close()	

●グループ化(group by)

select カラム名1,集合関数 from テーブル名 group by カラム名2

集合関数	内容
sum	グループの合計を求める
avg	グループの平均を求める
min	グループの最小値を求める
max	グループの最大を求める
count	グループの件数を求める

例として、p_tableの品名の価格の合計と件数を表示するプログラムを書いてみましょう。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	data=c.execute("select p_name,sum(p_price) from p_table group by p_name ")	各商品の合計価格
5	print(data.fetchall())	
6	data=c.execute("select p_name,count(p_price) from p_table group by p_name ")	各商品の種類の個数
7	print(data.fetchall())	
8	con.close()	

view

●viewの作成

create view ビュー名 as select文

ビューは、テーブルのカラムや計算結果など一時的なデータを持った仮想的なテーブルです。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	c.execute("create view if not exists test_view as select p_name,p_price*2 from p_table")	単価を2倍にしたビューを作成
5	data=c.execute("select * from test_view")	
6	print(data.fetchall())	
7	con.close()	

●viewの一覧

select name from sqlite_master where type='view'

上記コマンドでビューの一覧が確認できます。

1	import sqlite3	
2	con=sqlite3.connect("school.db")	
3	c=con.cursor()	
4	data=c.execute("select name from sqlite_master where type = 'view' ")	
5	print(data.fetchall())	
6	con.close()	

●viewの削除

drop view ビュー名

上記コマンドでビューの削除ができます。(1～3行目省略)

4	data=c.execute("select name from sqlite_master where type = 'view'")	
5	print(data.fetchall())	
6	c.execute("drop view test_view")	
7	data=c.execute("select name from sqlite_master where type = 'view'")	
8	print(data.fetchall())	
9	con.close()	

練習

●練習1(商品マスタ作成)

果物屋の商品マスタを作成します。
下記カラムを持つ商品マスタ「s_master」を作成しなさい。

カラム名	制約	型	内容
s_no	primary key,autoincrement	integer	商品ナンバー
s_name	unique	str	商品名(フジ、紅玉など)
s_type	not null	str	商品の種類(リンゴ、梨など)
s_price	check (0以上)	int	商品の価格

「s_master」が作成できたら、下記データを入力しなさい。

s_no	s_name	s_type	s_price
1	ツガル	リンゴ	300
2	フジ	リンゴ	350
3	ジョナゴールド	リンゴ	285
4	紅玉	リンゴ	270
5	幸水	ナシ	280
6	王秋	ナシ	650
7	豊水	ナシ	540
8	二十世紀	ナシ	325
9	新高	ナシ	310
10	巨峰	ブドウ	1000
11	ピオーネ	ブドウ	1400
12	シャインマスカット	ブドウ	2500

上記と同様に、顧客マスタ「k_master」を作成しなさい。

k_no	k_name
K01	田中
K02	斎藤
K03	武田
K04	川上
K05	井上
K06	御手洗

k_no　　：primary key
k_name ：unique

●練習2(売上テーブル作成)

果物屋の売り上げテーブルを作成します。

下記カラムを持つ売り上げテーブル「reception_t」を作成しなさい。

カラム名	制約	型	内容
r_no	primary key,autoincrement	integer	売上番号
r_date	not null	date	販売日
r_s_no	not null	str	商品番号
r_s_num	not null	int	販売個数
r_k_no	not null	str	顧客番号

「reception_t」テーブルができたら下記データを入力しなさい。

r_no	r_date	r_s_no	r_s_num	r_k_no
1	24/1/10	11	10	K01
2	24/1/11	11	15	K02
3	24/1/10	12	8	K01
4	24/1/10	11	7	K01
5	24/1/14	9	10	K02
6	24/1/11	11	5	K02
7	24/1/11	12	10	K02
8	24/1/11	11	3	K01
9	24/1/12	8	10	K03
10	24/1/12	7	5	K03
11	24/1/12	6	10	K04
12	24/1/13	11	15	K04
13	24/1/11	11	8	K03
14	24/1/14	5	10	K05
15	24/1/15	6	6	K05
16	24/1/14	3	10	K05
17	24/1/11	10	5	K06
18	24/1/15	2	5	K06
19	24/1/10	6	20	K06

●練習3(明細ビュー作成)

1.販売日、商品名、個数、合計金額を表示するビューを作成しなさい
2.顧客ごとの売り上げを表示するビューを作成しなさい
3.商品ごとの販売個数と合計金額を表示するビューを作成しなさい
4.日にちごとの販売個数と金額を表示するビューを作成しなさい

●練習4(CUIアプリ化)

練習1～3をふまえて下記プログラムを作成しなさい。
プログラムを起動したときに「s_master」「k_master」「reception_t」が無ければ作成する。
起動画面で「1.売上入力　2.マスタ登録　3.帳票印刷」を表示。

- 「1.売上入力」では、販売日、商品番号、販売個数、顧客番号を入力し情報を登録する。
- 「2.マスタ登録」では「1.商品マスタ登録」「2.顧客マスタ登録」の選択画面を表示。
    - 「1.商品マスタ登録」では、商品名、商品の種類、商品の価格を入力する。
    - 「2.顧客マスタ登録」では、顧客名を入力する。
- 「3.帳票印刷」では「1.売上明細」「2.顧客明細」「3.商品明細」「4.日別売上明細」を表示
    - 「1.売上明細」は販売日、商品名、個数、合計金額を表示
    - 「2.顧客明細」は顧客ごとの売り上げを表示
    - 「3.商品明細」は商品ごとの販売個数と合計金額を表示
    - 「4.日別売上明細」は日にちごとの販売個数と金額を表示

●練習5(GUIアプリ化)

練習4をGUIアプリ化します。