T- BASES DE DATOS

Contenido

1. Introducción

Las bases de datos se han vuelto tan comunes que existen implementaciones de distintos sistemas gestores personales que se encuentran instalados por defecto en los sistemas operativos o en los lenguajes. Este es el caso de Python que incorpora el sistema gestor de bases de datos Sqlite dentro de los paquetes estándar desde la version 2.5 del intérprete.

Independientemente de la base de datos incorporada, aparecen librerías para acceder a cualquier tipo de sistema gestor simplemente importando el modulo correspondiente y haciendo uso de las funciones.

OpenERP raramente utiliza el acceso directo a PostgreSql, en su defecto usa un mecanismo ORM (mapeo a objetos de la base de datos) de gestión de los datos integrando todas las tablas en objetos y simplificando su uso.

2. Pasos a dar:

1- Importar el módulo correspondiente
2- Abrir una conexión con ella facilitándole los datos necesarios para validar el acceso.
3- Crear las tablas en memoria en vez de hacerlas físicas. Por supuesto cuando termine la aplicación los datos se perderán.
4- Con el objeto recogido si no ha habido ningún error tendremos acceso a todas las funciones.

3. Tipos de datos en sqlite:

- **NULL**. Valor nulo.
- **INTEGER**. Entero con signo que se almacena en 1, 2, 3, 4, 6, o 8 bytes dependiendo su magnitud
- **REAL**. Numero en coma flotante almacenado en 8 bytes.
- **TEXT**. Para almacenar datos, strings.
- **BLOB**. Para almacenar datos BLOB, se almacena exactamente igual que la entrada. Para guardar objetos cuyo tamaño es o puede ser muy grande (imágenes, archivos, canciones,…)
- **NUMERIC**: Números decimales.

Casos ESPECIALES:

- Boolean: No existe. Es un integer con valor 0 o 1
- Fecha y Hora: No existen, pero sus funciones son accesibles a través de datos de tipo TEXT, REAL, o INTEGER.

  Ejemplos Fecha y Hora:

  - TEXT como ISO8601 strings ("YYYY-MM-DD HH:MM:SS.SSS").
  - REAL número de días julianos
  - INTEGER: número de segundos desde el 1 de enero de 1970 (como la fecha en unix)

4. AFINIDAD en SQLite- Reglas

1. Si se declara como INT → INTEGER
2. Si se declara como "CHAR", "VARCHAR","CLOB", o "TEXT" → TEXT
3. Si el tipo contiene la palabra BLOB →BLOB
4. Si el tipo contiene cualquiera de los siguientes strings: "REAL", "FLOA" o "DOUB" → REAL
5. En cualquier otro caso→ NUMERIC

El orden es importante a la hora de determinar la afinidad. Si un campo se declara como "CHARINT" cumplirá las reglas 1 and 2 pero la primera de ellas tiene preferencias, por lo tanto → INTEGER.

Example Typenames From The CREATE TABLE Statement or CAST Expression	**Resulting Affinity**	**Rule Used To Determine Affinity**
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER	1
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT	2
BLOB *no datatype specified*	BLOB	3
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL	4
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC	5

"FLOATING PO*INT*" tiene afinidad con integer y no con REAL porque en el string del tipo se encuentra el substring *INT*

5. *Ejemplo:*

```
#-*- coding: utf-8 -*-
'''
@author: Amaia
'''
import sqlite3 as dbapi


if __name__ == '__main__':
    bbdd= dbapi.connect(":memory:")     # Guardar en memoria  en vez de en disco
    bd2=dbapi.connect("TESTDB") #Crea la bd (si no existe. Si existe, la abre) en el gestor sqlite. Se crea dentro del directorio del proyecto la BD.


    c = bbdd.cursor() #Crear el cursor
    c2=bd2.cursor()
    #Ejecutar las sentencias a través del cursor
    c.execute("create table empleados (dni text,nombre text,departamento text)")
    c.execute("insert into empleados values ('a1','b1','c1')")
    c.execute("insert into  empleados values ('a2','b2','c2')")
    c.execute("insert into empleados values ('a3','b3','c3')")


    #En bd2 también tengo creada una tabla llamada empleados con datos. La utilizo en main2
    c2.execute("create table empleados2 (dni text,nombre text,departamento text)")
    c2.execute("insert into empleados2 values ('a1','b1','c1')")
    c2.execute("insert into  empleados2 values ('a2','b2','c2')")
    c2.execute("insert into empleados2 values ('a3','b3','c3')")


    bbdd.commit()
    bd2.commit()
```

```
    c.execute("select   * from empleados")

    print "Imprimir TODO lo recogido en el cursor"

    for t in c.fetchall(): #Recoge todas las filas del cursor en t que es una
lista de listas

        print t #Imprimirá registro a registro con todos sus campos. La primera
lista corresponderá al primer registro de BD y así sucesivamente.



    c.execute("select * from empleados")

    t = c.fetchone()

    i=1

    print "Imprimir uno a uno los registros"

    while t:

        print "Registro:",i

        print t

        t = c.fetchone()

        i=i+1

    c.execute("create table alumnado (dni text,nombre text,edad
integer)")

    c.execute("insert into alumnado values ('a1','b1',23)")

    c.execute("insert into  alumnado values ('a2','b2',32)")

    c.execute("insert into alumnado values ('a3','b3',22)")

    bbdd.commit()



    print "Tras update- resultado tabla"

    c.execute ("update alumnado set nombre='cambio1' where dni='a1' ")

    bbdd.commit()

    c.execute("select * from alumnado")

    print "Imprimir TODO lo recogido en el cursor"

    for t in c.fetchall():

        print t
```

```
c.execute("delete from alumnado where dni='a3'")
c.execute("select * from alumnado")
print "Tras borrar:"
for t in c.fetchall():
    print t[0] #Primera columna del cursor, del registro--> Dni
    print t[1] #Segunda columna --> Nombre
    print t[2] #Tercera columna --> Edad
bbdd.commit()

print "\n\nEjecutar script SQL"
c.executescript("""
create table person(
    firstname,
    lastname,
    age
);

create table book(
    title,
    author,
    published
);

insert into book(title, author, published)
values (
    'Dirk Gently''s Holistic Detective Agency',
    'Douglas Adams',
    1987
);
```

```
    """)



    print "\n\nEjecutar insert con parámetros"

    c.execute("create table people (nombre, edad)") #Crear la tabla con 2 columnas sin especificar el tipo


    #Variables que voy a utilizar como parámetros en las sentencias de BD

    quien = "Amaia"

    anos = 72


    # Forma1: utilizar ? para indicar parámetro. Este parámetro puede ser un literal o una variable

    print "FORMA1: "

    c.execute("insert into people values (?, ?)", ("Manolito",43))

    c.execute("insert into people values (?, ?)", (quien,anos))


    c.execute("select * from people")

    print c.fetchone()

    print c.fetchone()




    # Forma2: se utiliza una variable en la sentencia que después recibirá un valor (literal o variable)

    print "FORMA2: "

    c.execute("select * from people where nombre=:persona and edad=:cuantos", {"persona": quien, "cuantos": anos})

    print c.fetchone()

    c.execute("select * from people where nombre=:persona and edad=:cuantos", {"persona": "Manolito", "cuantos": 43})

    print c.fetchone()


    bbdd.close()
```

```
    c2.execute ("drop table empleados2") #Elimino esta tabla de bd2 para
evitar errores la proxima vez que ejecuto este main ya que se queda creada la tabla.

    bd2.close()
```

MAIN2.PY

```
#-*- coding: utf-8 -*-

'''

@author: Amaia

Este programa comprueba que la bd2 creada en el main.py se guarda y se
puede acceder posteriormente a ella.

La bd bbdd como se crea en memoria no es accesible tras finalizar
main.py

'''

import sqlite3 as dbapi

if __name__ == '__main__':

    bd2=dbapi.connect("TESTDB" )

    c2=bd2.cursor()

    c2.execute("select * from empleados")

    print "Imprimir TODO lo recogido en el cursor"

    for t in c2.fetchall(): #Recoge todas las filas del cursor en t que es una
lista de listas

        print t
```

Fuente:

https://docs.python.org/2/library/sqlite3.html

http://rafinguer.blogspot.com.es/2010/10/conceptos-basicos-de-python-y-sqlite.html