Using database engines and unicode

Marcin Szewczyk
PhD student
msz@imm.dtu.dk

DTU Informatics
Technical University of Denmark

21 IX 2009 / web mining

Outline

Python database interface

Python database interface: DB-API 2.0 Many implementations:

- sqllite
- MySQL
- IBM DB2
- ...

Logical steps

Independent from actual implementation:

- connect to the database
- acquire cursor
- execute SQL statement
- fetch the results
- commit changes
- close connection

Sample usage

```
1 import sqllite3
2
3 test_connection= sqllite3.connect('/tmp/db') #
    or fx. ':memory'
4 test_cursor= test_connection.cursor()
5
6 test_cursor.execute('CREATE TABLE people (id
    INTEGER,name TEXT)')
7
8 test_cursor.execute('INSERT INTO people(id,
    name) VALUES(?,?)',(78,'Marcin'))
9 test_connection.commit()
```

Example continued

When changing the database remember:

```
1 id= 78 #our query data
2 # in MySQLdb use %s instead of ?
3 test_cursor.execute('SELECT_name_FROM_people_
    WHERE_id=?',(id,))
4 # OR in sqllite3
5 test_cursor.execute('SELECT_name_FROM_people_
    WHERE_id=:id',{'id': id})
6
7 print test_cursor.fetchone() #or .fetchall()
    or .fetchmany(n)
8
9 test_connection.commit()
```

Codepage chaos

- Pre-Unicode:
  - ASCII - 7bit: 127 symbols based on latin alphabet
  - Many regional 8-bit code pages
  - Central Europe: Latin2, cp852, windows1250, iso-8859-2
- Unicode:
  - $>$1000000 symbols - about 2^{20}
  - 16 planes each able to contain about 2^{16} symbols
  - most alphabets belong to plane 0 - BMP
  - symbols referenced as U+xxxx U+1xxxx ....
  - 'A' is U+0041

Unicode encodings

How to encode unicode

- UTF32(BE,LE), BOM
- UTF16(BE,LE), BOM
- UTF8

Byte Order Mark: U+FEFF to distinguish Big-Endian, Little-Endiax

Unicode in Python

In Python since version 2.2, two kinds of strings:

- Byte string (bytes <128 interpreted as ASCII)
- Unicode string

```
1 astring= 'Hello string!' #byte string, non
    ASCII displayed in hex as \xYY
2 unicode_string= u'Hello Unicode string!'
3 unicode_string= unicode(astring)
4
5 old_persian_letter= '\xF0\x90\x8E\xA0'    # OLD
    PERSIAN SIGN A
6 len(old_persian_letter)                   # =4
7 len(unicode(old_persian_letter,'utf8'))   # =1
```

String vs. Unicode String

- String:
  - Stores bytes
  - Can store any encoding
- Unicode String
  - Stores Unicode characters
  - Can be stored as bytes
  - Internal representation does not matter!

encode(), unicode() example

```
1 utf8_string='\xc5\x81\xc3\xb3d\xc5\xba'
2 print(utf8_string)
3 unicode_string=unicode(utf8_string,'utf8')
4 print(unicode_string)
5 latin2_string=unicode_string.encode('latin2')
6 print(latin2_string)
7 print(unicode(latin2_string,'latin2'))
```

encode(), unicode() example

screenshot

```
>>> utf8_string='Łódź'
>>> utf8_string
'\xc5\x81\xc3\xb3d\xc5\xba'
>>> print(utf8_string)
Łódź
>>> unicode_string=unicode(utf8_string,'utf8')
>>> unicode_string
u'\u0141\xf3d\u017a'
>>> print(unicode_string)
Łódź
>>> latin2_string=unicode_string.encode('latin2')
>>> latin2_string
'\xa3\xf3d\xbc'
>>> print(latin2_string)
??d?
>>> print(unicode(latin2_string,'latin2'))
Łódź
>>>
```

Bypassing characters

Dirty tricks:

```
1 s = unicode_string.encode("iso-8859-1") # fail
    if some character cannot be converted
2 s = unicode_string.encode("iso-8859-1", "
   replace") # instead of failing, replace
   with ?
3 s = unicode_string.encode("iso-8859-1", "
   ignore") # instead of failing, leave it out
```

References

- General python databases `http://wiki.python.org/moin/DatabaseProgramming/`
- Unicode info `http://www.tbray.org/ongoing/When/200x/2003/04/26/UTF`
- Unicode in Python `http://evanjones.ca/python-utf8.html`

Thank you.

- Thank you for your attention!