The Python InformixDB API

Carsten Haese
Unique Systems, Inc.

Atlanta, Georgia December 8-9, 2005

Overview

- Python Features
- InformixDB Features
- Installing InformixDB
- Interactive Tour of InformixDB
- Integration Examples
  - Dynamic Web Page with Apache/mod_python
  - OpenOffice.org Macro in Python
- Conclusion

Python Features

- Easy to learn
- Interactive command prompt
- Emphasizes code clarity
- OOP possible, but not required
- Garbage Collection
- Exception handling
- Rich library of standard and third-party modules
- Useful data types (lists, dictionaries, etc)
- Rapid Application Development
- Large, helpful user community
- Cross-platform support
- Extensible and Embeddable
- Standardized database API

InformixDB Features

- DB-API 2.0 compliant
- Portable (POSIX and Win32 Platforms)
- Multiple connections
- Parametrized queries
- Statement caching
- Access to sqlcode and sqlerrd
- Update, delete, and insert cursors
- Scroll cursors and cursors with hold
- Simple Large Objects, Smart Large Objects, and UDTs

Installing InformixDB

- Prerequisites:
  - Python 2.2 or better
  - Informix Client SDK
- Get InformixDB from informixdb.sourceforge.net
- POSIX Installation: Compile from source
  - **`python setup.py build_ext`**
  - **`python setup.py install`** (as root)
- Windows: Installer for Python 2.4

Establishing a Connection

```
>>> import informixdb
>>> conn = informixdb.connect(”test”, ”informix”, ”pw”)
```

Static SQL Query

```
>>> cur = conn.cursor()
>>> cur.execute(“create table test1(a integer, b integer)”)
```

SQL Query with Parameters

```
>>> for i in range(1,25):
...   cur.execute("insert into test1 values(?,?)", (i, i**2) )
```

Note: You can also use positional parameters (:1, :2, :3, etc.), which is useful when the same argument is used multiple times in the query.

Transactions

By default, a connection is always in a transaction. Committing or rolling back the current transaction begins a new transaction.

```
>>> conn.commit()
>>> conn.rollback()
```

Fetching Data Rows

```
>>> cur.execute("select * from test1")
>>> cur.fetchone()
>>> cur.fetchmany(5)
>>> cur.fetchall()
```

Looping through Data Rows

```
>>> cur.execute("select * from test1")
>>> for row in cur:
...   print "The square of %d is %d." % (row[0], row[1])
```

Data Rows as Dictionaries

```
>>> dictcur = conn.cursor(rowformat=informixdb.ROW_AS_DICT)
>>> dictcur.execute("select * from test1")
>>> for row in dictcur:
...   print "The square of %(a)d is %(b)d." % row
```

Data Rows as Objects

```
>>> objcur = conn.cursor(rowformat=informixdb.ROW_AS_OBJECT)
>>> objcur.execute("select * from test1")
>>> for row in objcur:
...   print "The square of %d is %d." % (row.a, row.b)
```

Update Cursor

```
>>> updcur = conn.cursor(name="updcur",
   rowformat=informixdb.ROW_AS_OBJECT)
>>> updcur.execute("select a from test1 for update of b")
>>> for row in updcur:
...   cur.execute("update test1 set b=? where current of updcur",
                  (row.a**3,) )
```

Delete Cursor

```
>>> delcur = conn.cursor(name="delcur")
>>> delcur.execute("select * from test1 for update")
>>> for row in delcur:
...   cur.execute("delete from test1 where current of delcur")
```

Insert Cursor

```
>>> param_list = [ (i,i**2) for i in range(1,25) ]
>>> cur.executemany("insert into test1 values(?,?)", param_list)
```

Scroll Cursor

```
>>> scrollcurs = conn.cursor(scroll=True)
>>> scrollcurs.execute(“select * from test1 order by a”)
>>> scrollcurs.scroll(10,”absolute”)
>>> scrollcurs.fetchone()
>>> scrollcurs.scroll(-5,”relative”)
>>> scrollcurs.fetchone()
```

Date and Datetime Columns

InformixDB uses Python's datetime module for handling date and datetime values.

```
>>> cur.execute("create table test2(d date)")
>>> dateval = informixdb.Date(2005,12,31)
>>> cur.execute("insert into test2 values(?)", (dateval,) )
```

BYTE and TEXT Columns

```
>>> cur.execute("create table test3(b byte)")
>>> binval = informixdb.Binary( file("binary.dat").read() )
>>> cur.execute("insert into test3 values(?)", (binval,) )
>>> cur.execute("select * from test3")
>>> row = cur.fetchone()
>>> readval = str(row[0])
>>> file("read.dat","w").write(readval)
```

Error Handling

By default, error conditions raise Python exceptions:

```
>>> my_sqlcode = 0
>>> try:
...   cur.execute("sleect * from systables")
... except informixdb.Error, e:
...   my_sqlcode = e.sqlcode
>>> print my_sqlcode
```

An errorhandler callback can be set up to customize error handling.

Accessing SQLERRD

```
>>> cur.execute("create table test4(s serial)")
>>> for i in range(5):
...   cur.execute("insert into test4 values(0)")
...   print cur.sqlerrd
```

Describing a Query

If a query produces a result set (e.g. SELECT), the cursor's `description` attribute describes the result columns. If a query doesn't produce a result set, `description` is `None`.

```
>>> cur.execute("select * from systables")
>>> print cur.description
```

Dynamic Web Page with Apache/mod_python

```
<%
import mod_python.util
fieldstorage = mod_python.util.FieldStorage(req)
query = ""
result = ""
if fieldstorage.has_key('query'): query = fieldstorage['query']

if fieldstorage.has_key('submit'):
  try:
     import informixdb
     conn = informixdb.connect("test")
     cur = conn.cursor()
     cur.execute(query)
     if cur.description == None:
        result += '<p>Query executed successfully.</p>\n'
        if cur.rowcount != None and cur.rowcount > -1:
          result += '<p>%s row(s) affected.</p>' % cur.rowcount
     else:
        result += '<table border="1">\n'
        result += '<tr>\n'
        for col in cur.description:
           result += '<th>%s</th>\n' % col[0]
        result += '</tr>\n'
        count = 0
        for row in cur:
           count += 1
           result += '<tr>\n'
           for col in row:
              result += '<td>%s</td>\n' % str(col)
           result += '</tr>\n'
        result += '</table>'
        result += '<p>%s row(s) selected.</p>' % count
  except informixdb.Error, e:
    result += '<pre>%s</pre>' % e
  conn.commit()
  conn.close()
#
%>
```

```
<html>
  <body>
    <form method="POST" action="dbaccess.psp">
    <p>Query:<br>
      <textarea name="query" cols="80"
      rows="10"><%=query%></textarea><br>
      <input type="submit" name="submit" value="Run"></p>
    </form>
    <hr><%=result%>
  </body>
</html>
```

OpenOffice.org Macro in Python

```
def InformixDBDemo( *args, **kwargs ):
    """Prints all tables in 'test' database into the current document"""
# get the doc from the scripting context which is made available to all scripts
    model = XSCRIPTCONTEXT.getDocument()
# get the XText interface
    text = model.Text
# create an XTextCursor
    cursor = text.createTextCursor()
# make the string
    import os
    os.environ['INFORMIXDIR'] = '/opt/informix'
    os.environ['INFORMIXSERVER'] = 'localhost_tcp'
    import informixdb
    conn = informixdb.connect("test","informix","pw")
    cur = conn.cursor()
    cur.execute("select tabname from systables order by tabname")
    tabnames = [ row[0] for row in cur ]
    tables = ", ".join(tabnames)
# and insert the string
    text.insertString( cursor, tables, 0 )
    return None
```

Conclusion

Thanks to Python's ubiquity and interoperability, InformixDB allows you to integrate your Informix database with a broad spectrum of applications.

Links

- Python's Home Page
  http://www.python.org/
- Information on DB Programming in Python
  http://www.python.org/topics/database/
- Python DB-API specification, version 2.0
  http://www.python.org/peps/pep-0249.html
- InformixDB on Sourceforge
  http://informixdb.sourceforge.net/

Acknowledgments

- Greg Stein and Michael Lorton
  Initial authors of InformixDB
- Bertil Reinhammar and Stephen J. Turner
  Previous maintainers of InformixDB
- Daniel Smertnig
  Contributed major improvements to InformixDB

The Python InformixDB API

Carsten Haese
carsten@uniqsys.com

Atlanta, Georgia December 8-9, 2005