Python – DB Access

Contents

Database Access

SQL Interface Overview

- You can access SQL databases from Python, through a straight-forward and portable model.
- The Python database API specification defines an interface for communicating with underlying database systems from Python scripts.
- Vendor- specific database interfaces for Python may or may not conform to this API completely, but all database extensions for Python in common use are minor variations on a theme.
- Under the database API, SQL databases in Python are grounded on three core concepts:

Connection objects

- Represent a connection to a database, are the interface to rollback and commit operations, provide package implementation details, and generate cursor objects.

SQL Interface Overview

Cursor objects

- Represent an SQL statement submitted as a string and can be used to access and step through SQL statement results.

Query results of SQL select *statements*

- Are returned to scripts as Python sequences of sequences (e.g., a list of tuples), representing database tables of rows. Within these row sequences, column field values are normal Python objects such as strings, integers, and floats (e.g., [('bob', 48), ('emily',47)]). Column values may also be special types that encapsulate things such as date and time, and database NULL values are returned as the Python None object.

SQL Interface Overview

- For instance, to establish a database connection under the Python API-compliant Oracle interface, install the commonly used Python Oracle extension module as well as Oracle itself, and then run a statement of this form:

connobj = connect("user/password@system")

- This call's arguments may vary per database and vendor (e.g., some may require network details or a local file's name), but they generally contain what you provide to log in to your database system.
- Once you have a connection object, there a variety of things you can do with it, including:
- connobj.close() – close connection now
- connobj.commit() – commit any pending transactions to the database
- connobj.rollback() – rollback database to start of pending transactions

SQL Interface Overview

- But one of the most useful things to do with a connection object is to generate a cursor object:

cursobj = connobj.cursor()   *return a new cursor object for running SQL*

- Cursor objects have a set of methods, too (e.g., close to close the cursor before its destructor runs, and callproc to call a stored procedure), but the most important may be this one:

cursobj.execute(*sqlstring* [, *parameters*])   *run SQL query or command string*

- Parameters are passed in as a sequence or mapping of values, and are substituted into the SQL statement string according to the interface module's replacement target conventions.

SQL Interface Overview

- The execute method can be used to run a variety of SQL statement strings:
- DDL definition statements (e.g., CREATE TABLE)
- DML modification statements (e.g., UPDATE or INSERT)
- DQL query statements (e.g., SELECT)

- After running an SQL statement, the cursor's rowcount attribute gives the number of rows changed (for DML changes) or fetched (for DQL queries), and the cursor's description attribute gives column names and types after a query; execute also returns the number of rows affected or fetched in the most vendor interfaces.

SQL Interface Overview

- For DQL query statements, you must call one of the fetch methods to complete the operation:

tuple = cursobj.fetchone() -- *fetch next row of a query result*

listoftuple = cursobj.fetchmany([*size*]) -- *fetch next set of rows of query result*

listoftuple = cursobj.fetchall() -- *fetch all remaining rows of the result*

- And once you've received fetch method results, table information is processed using normal Python sequence operations; for example, you can step through the tuples in a fetchall result list with a simple for loop or comprehension expression.
- Most Python database interfaces also allow you to provide values to be passed to SQL statement strings, by providing targets and a tuple of parameters.

SQL Interface Overview

- For instance :

```
query = 'SELECT name, shoesize FROM spam WHERE job = ? AND age = ?'
cursobj.execute(query, (value1, value2))
results = cursobj.fetchall()
for row in results: ...
```

- In this event, the database interface utilizes prepared statements (an optimization and convenience) and correctly passes the parameters to the database regardless of their Python types.
- The notation used to code targets in the query string may vary in some database interfaces (e.g., :p1 and :p2 or two %s, rather than the two ?s used by the Oracle interface); in any event, this is not the same as Python's % string formatting operator, as it sidesteps security issues along the way.

SQL Interface Overview

- Finally, if your database supports stored procedures, you can call them with the callproc method or by passing an SQL CALL or EXEC statement string to the execute method.
- callproc may generate a result table retrieved with a fetch variant, and returns a modified copy of the input sequence—input parameters are left untouched, and output and input/output parameters are replaced with possibly new values.
- Additional API features, including support for database blobs (roughly, with sized results), is described in the API's documentation.
- For now, let's move on to do some real SQL processing in Python.
- First of all, we will show you 'sqlite3', which is part of Python distribution

SQL Database API with SQLite

- You can run the Python in interactive mode as :

```
$ python
>>> import sqlite3
>>> conn = sqlite3.connect('dbase1') # if not present, it creates a database
>>> curs = conn.cursor()
>>> tblcmd = 'create table people (name char(30), job char(10), pay int(4))'
>>> curs.execute(tblcmd)
<sqlite3.Cursor object at 0x101c26340>

>>> curs.execute('insert into people values (?, ?, ?)', ('Bob', 'dev', 5000))
<sqlite3.Cursor object at 0x101c26340>
>>> curs.rowcount
1
```

SQL Database API with SQLite

```
>>> curs.execute('insert into people values (?, ?, ?)', ('Sue', 'mus', 4000))
<sqlite3.Cursor object at 0x101c26340>
>>> curs.rowcount
1
>>> conn.commit()

>>> curs.execute('select * from people')
<sqlite3.Cursor object at 0x101c26340>
>>> curs.fetchall()
[('Bob', 'dev', 5000), ('Sue', 'mus', 4000)]
>>>
```

SQL Database API with SQLite

```
>>> curs.execute(‘update people set pay=? Where pay <= ?’, [5000, 4500])
<sqlite3.Cursor object at 0x101c26340>
>>> curs.rowcount()
1
>>> curs.execute(‘select * from people’)
<sqlite3.Cursor object at 0x101c26340>
>>> curs.fetchall()
[('Bob', 'dev', 5000), ('Sue', 'mus', 5000)]
>>> con.commit()
>>>
```

- Similarly [almost like ‘update’] , you can delete record(s) as well.
- Finally, remember to commit your changes to the database before exiting Python, assuming you wish to keep them.

MySQL®
python™

Contents

Python Database Interface

- The Python standard for database interfaces is the Python DB-API.
- Most Python database interfaces adhere to this standard.
- You can choose the right database for your application.
- Python Database API supports a wide range of database servers such as −
  - mSQL
  - MySQL
  - PostgreSQL
  - Microsoft SQL Server
  - Informix
  - Oracle
  - Sybase
- Please note : You must download a separate DB API module for each database you need to access.
- For example to use MySQL, you need to install either :
  - mysql-connector-python
  - mysqldb

Python Database Interface

- The basic difference between the two are :
  - MySQLdb is a C module that links against the MySQL protocol implementation in the libmysqlclient library. It is much faster, but requires the library in order to work.
  - mysql-connector is a Python module that reimplements the MySQL protocol in Python. It is slower, but does not require the C library and so is more portable.
- Here we will use 'mysql-connector-python' .

- Note :

1. In the target machine you need to install MySQL Database server
2. You need to install mysql-connector-python
3. You need to have proper access to mySQL and its database
4. You need to create a 'test' database in mySQL server

To connect to a Database

- Let us consider the following example :[this has to be in a file e.g. mysql1.py]

```
import mysql.connector
from mysql.connector import Error
""" Connect to MySQL database """”
try:
        conn = mysql.connector.connect(host='localhost’,  database='test',
                        user='prasun', password='cmckolkata@1')
        if conn.is_connected():
           print('Connected to MySQL database')
           curr = conn.cursor()
           curr.execute("SELECT VERSION()" )
           ver = curr.fetchone()
           print("Database version : %s " % ver)
except Error as e:
       print(e)
finally:
       conn.close()
```

To create a Table

```
## To create a Table – mysql2.py
import mysql.connector as db
from mysql.connector import Error
""" Connect to MySQL database """
try:
        conn = db.connect(host='localhost’,database='test’,user='prasun’,password='cmckolkata@1')
        if conn.is_connected():
                cursor = conn.cursor()
                cursor.execute("DROP TABLE IF EXISTS EMPLOYEE" )
                sql = """CREATE TABLE EMPLOYEE (
                  FIRST_NAME      CHAR(20) NOT NULL,
                  LAST_NAME       CHAR(20),
                  AGE           INT,
                  SEX           CHAR(1),
                  INCOME        FLOAT )""”
                cursor.execute(sql)
                print("Database Created ....!")
except Error as e:
                print(e)
finally:
                conn.close()
```

To insert a Record into a Table

```
## Insert a record into a Table in a Database – mysql3.py
import mysql.connector as conn
# Open database connection
db = conn.connect(host='localhost’, database='test’, user='prasun’,password='cmckolkata@1')
if db.is_connected():
    print('Connected to MySQL database')
    # prepare a cursor object using cursor() method
    cursor = db.cursor()
    # Prepare SQL query to INSERT a record into the database
    sql = """INSERT INTO  EMPLOYEE ( FIRST_NAME, LAST_NAME, AGE, SEX, INCOME )
            VALUES ('Raghu', 'Rai', 43, 'M', 15000 )"""”
    try :
        # Execute the SQL command
        cursor.execute(sql)
        # Commit your changes in the database
        db.commit()
        print("record created ......!")
    except:
        # Rollback in case there is any error
        db.rollback()
        print(“record could not be created......!”)
finally:
        db.close()
```

To insert multiple Records into a Table

```
## Insert a record into a Table in a Database
import mysql.connector as conn
# Open database connection
db = conn.connect(host='localhost', database='test', user='prasun',password='cmckolkata@1')
if db.is_connected():
        print('Connected to MySQL database')
        # prepare a cursor object using cursor() method
        cursor = db.cursor()
        # Prepare SQL query to INSERT a record into the database
        sql1 = """INSERT INTO  EMPLOYEE ( FIRST_NAME, LAST_NAME, AGE, SEX, INCOME )
                VALUES ('Raghu', 'Rai', 43, 'M', 15000 )"""
        sql2 = """INSERT INTO  EMPLOYEE ( FIRST_NAME, LAST_NAME, AGE, SEX, INCOME )
                VALUES ('Raghu', 'Roy', 23, 'M', 25000 )"""
        sql3 = """INSERT INTO  EMPLOYEE ( FIRST_NAME, LAST_NAME, AGE, SEX, INCOME )
                VALUES ('Sumita', 'Mondol', 43, 'F', 5000 )"""
        sql4 = """INSERT INTO  EMPLOYEE ( FIRST_NAME, LAST_NAME, AGE, SEX, INCOME )
                VALUES ('Anindita', 'Bose', 35, 'F', 11000 )"""
```

To insert multiple Records into a Table

```
 try :
     # Execute the SQL command
     cursor.execute(sql)
     # Commit your changes in the database
     db.commit()
     print("record created ......!")
except:
     # Rollback in case there is any error
     db.rollback()
     print("record could not be created......!")
 finally:
     db.close()
```

To Retrieve Records from a Table

```
import mysql.connector  as mcon  – mysql4.py
from mysql.connector import Error
try:
        conn = mcon.connect(host='localhost’,  database='test', user='prasun', password='cmckolkata@1')
        if conn.is_connected():
                print('Connected to MySQL database')
                curr = conn.cursor()
                curr.execute("SELECT * FROM EMPLOYEE" )
                results  = curr.fetchall()
                for row in results:
                        fname = row[0]
                        lname = row[1]
                        age = row[2]
                        income = row[4]
                        print("fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \
                          (fname, lname, age, sex, income) )
except :
      print(“Error : Unable to fetch data......”)
finally:
      conn.close()
```

To Retrieve Records from a Table

```
## to read records from a table – mysql5.py
import mysql.connector as mcon
# Open database connection
db = mcon.connect(host='localhost’,database='test’,user='prasun’, password='cmckolkata@1')
if db.is_connected():
        print('Connected to MySQL database')
        # prepare a cursor object using cursor() method
        cursor = db.cursor()
        # Prepare SQL query to fetch records from the table
        sql = """SELECT * FROM EMPLOYEE WHERE INCOME > '%d' """ % (2000)
        try :
                # Execute the SQL command
                cursor.execute(sql)
                results = cursor.fetchall()
                for row in results:
                              fname = row[0]
                              lname = row[1]
                               age = row[2]
                               sex = row[3]
                              income = row[4]
```

To Retrieve Records from a Table

```
            # Now print the fetched result
             print("fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \
                    (fname, lname, age, sex, income) )
except:
      print ("Error : Unable to fectch the data......!")
finally:
      db.close()
```

To Update a record from a Table

```
## Update a record in a Table – mysql6.py
import mysql.connector as mcon
# Open database connection
db = mcon.connect(host='localhost’, database='test’,user='prasun’,password='cmckolkata@1')
if db.is_connected():
    print('Connected to MySQL database')
    # prepare a cursor object using cursor() method
    cursor = db.cursor()
    # Prepare SQL query to INSERT a record into the database
    sql = """UPDATE  EMPLOYEE SET  AGE = AGE + 1 WHERE SEX = 'M' ""”
    try :
        # Execute the SQL command
        cursor.execute(sql)
        # Commit your changes in the database
        db.commit()
        print("record Updated ......!")
     except :
        # Rollback in case there is any error
        db.rollback()
        print(“Error : Record could not be Updated ….. !!”)
     finally:
        db.close()
```

To Delete a record from a Table

```
## Delete a record in a Table – mysql7.py
import mysql.connector as mcon
# Open database connection
db = mcon.connect(host='localhost', database='test',user='prasun',password='cmckolkata@1')
if db.is_connected():
        print('Connected to MySQL database')
        # prepare a cursor object using cursor() method
        cursor = db.cursor()
        # Prepare SQL query to INSERT a record into the database
        sql = """DELETE FROM  EMPLOYEE WHERE SEX = 'M' """
        try :
                # Execute the SQL command
                cursor.execute(sql)
                # Commit your changes in the database
                db.commit()
                print("record Deleted ......!")
         except :
                # Rollback in case there is any error
                db.rollback()
                print("Error : Record could not be Deleted..... !!")
         finally:
                db.close()
```

Reference

- Python for Informatics – C. Severance
- Think Python – A. B. Downey - [O'Reilly]
- Python Crash Course – Eric Matthes [No starch Press]
- A Byte of Python – Swaroop C H
- Introducing Python – Bill Lubanovic [O'Reilly]

- Learning Python – 5th Ed – Mark Lutz - [O'Reilly]

End of Presentation

Python – SQL