Python PostgreSQL Programming Basics

Jim C. McDonald

Michigan Unix Users Group

April 11, 2006

Attributions

Most of the material in this presentation
is a repackaging of information in the following documents.
Any errors in the presentation are my own.

PEP 249 http://www.python.org/peps/pep-0249.html

- Book: PostgreSQL (2nd Edition) (Paperback) by Korry Douglas ISBN: 0672327562
- Book: Python Web Programming by Steve Holden

DB-API v2.0

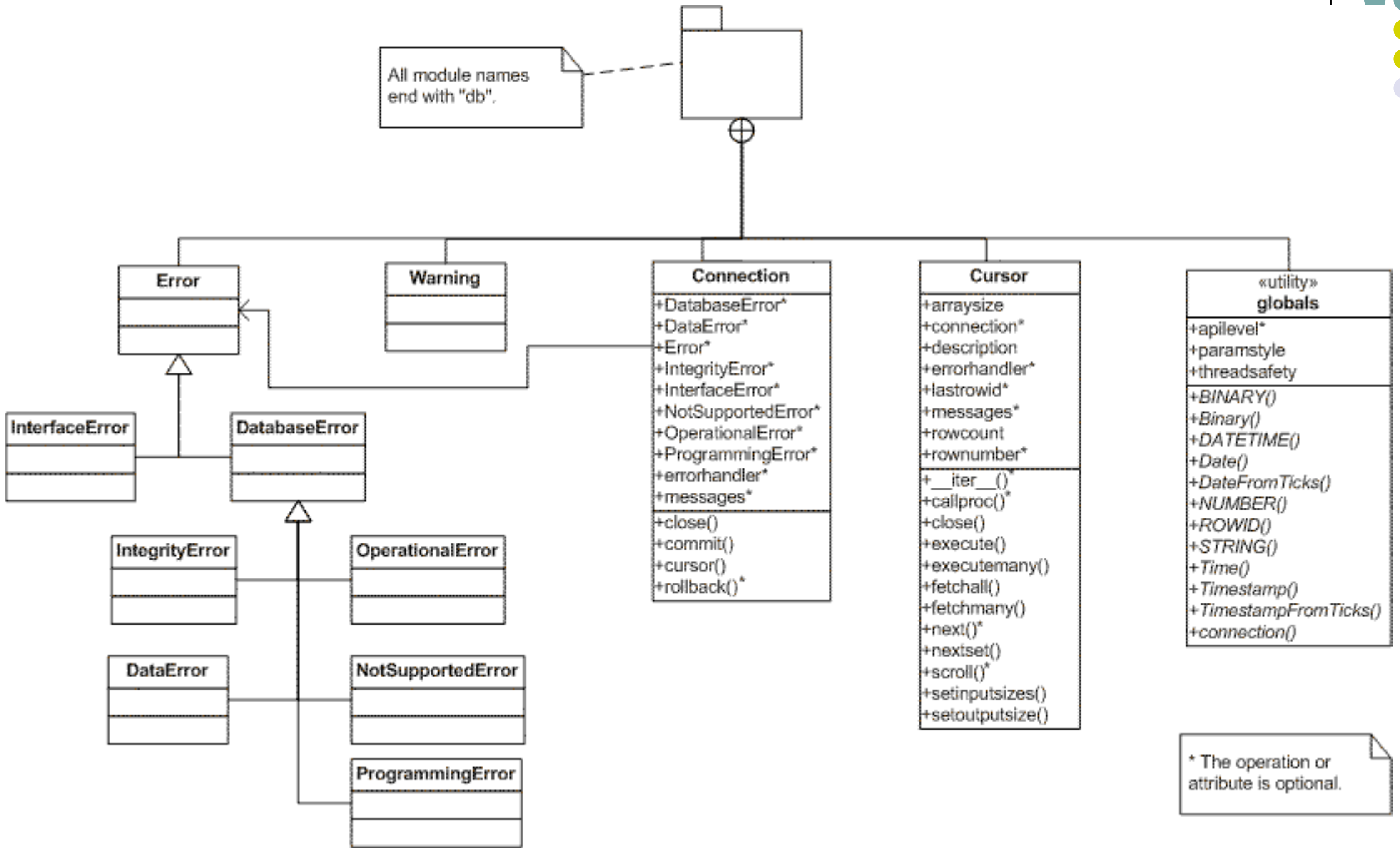
- The Python DB-API provides a database independent API to interact with SQL databases
- Officially documented in PEP 249 http://www.python.org/peps/pep-0249.html
- The python API described for PostgreSQL will also work for *any* database which has a DB-API compliant adapter available. [MySQL,DB2,Oracle,Sybase,Informix,ODBC, …]

Database Adapter Modules

- Database Adapter Modules are drivers for specific databases that implement the standard Python DB-API
- Popular modules for PostgreSQL include:
    - psycopg/psycopg2
      http://initd.org/projects/psycopg1
      http://initd.org/projects/psycopg2
    - PyGreSQL http://www.pygresql.org/
    - pyPySQL http://pypgsql.sourceforge.net/

Python DB-API UML Diagram

Created by Travis Spencer



Connection Object

Access to the database is available through connection objects.

- `conn = psycopg.connect(parameters…)` : Constructor for creating a connection. Some (small) changes to your parameters may be necessary for each specific adapter. For PostgreSQL `(host= dbname= user= password= )` or a DSN (Data Source Name)
- `cur=conn.cursor()` : use the connection to create a cursor
- `conn.commit() conn.rollback()` : commit/rollback
- `conn.close()` : to close the connection (with implicit rollback)
- `conn.apilevel conn.threadsafety conn.paramstyle` : dba module global variables that describe the capabilities of the adapter module

Importing the database adapter module and connecting

```
#!/usr/bin/env python2.4
# example_connect.py

# **
# create a psycopg namespace with the objects in the psycopg module
# **
import psycopg

print 'example_connect.py\n'
# **
# psycopg features
# **
print 'Features of the psycopg are:',
print ('API Level: %s, Thread Safety Level: %s, Parameter Style: %s.\n' % \
    (psycopg.apilevel,psycopg.threadsafety,psycopg.paramstyle))

# **
# create a connection to the database
# **
try:
    conn=psycopg.connect("host=127.0.0.1 dbname=test user=postgres")
    print 'Successfully connected to the database.\n'
except:
    print 'Error: Unable to connect to database!'
```

Cursor Object

These objects represent a database cursor, which is used to manage the context of fetch operations. Cursors created from the same connection are not isolated.

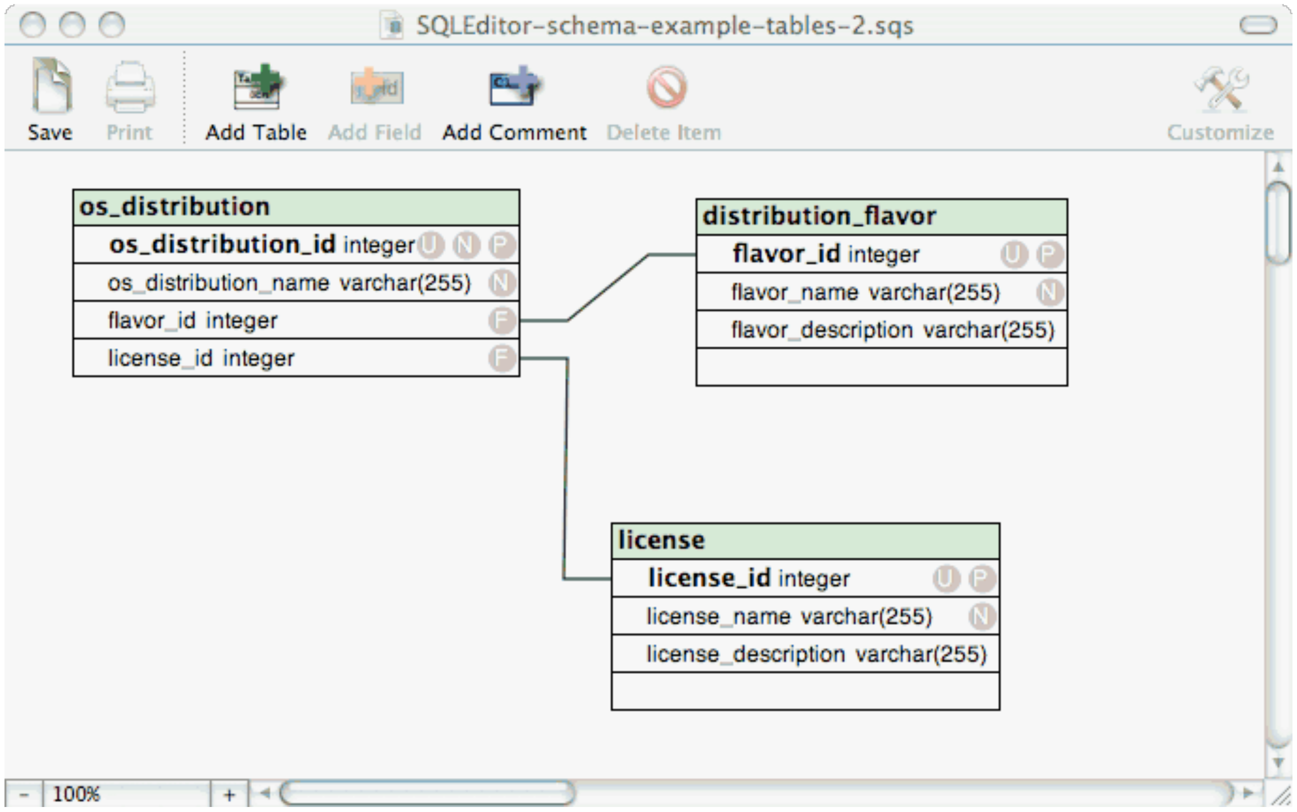
<u>Cursor methods</u>

- `cur.execute(operation,parameters)`, `cur.executemany(operation,sequence_of_parameters)`
- `cur.fetch(), .fetchone(), .fetchmany([N]), .fetchall()`
- `cur.close()`
- `cur.commit(), cur.rollback()` (only if module supports it)

Metadata is also available about the results of an operation.

- `cur.description (a sequence of 7 item sequences)`
    - `(name,type_code,display_size,internal_size,precision,scale,null_ok)`
- `cur.rowcount`

Schema for Example



os_distribution
- **os_distribution_id** integer U N P
- os_distribution_name varchar(255) N
- flavor_id integer F
- license_id integer F

distribution_flavor
- **flavor_id** integer U P
- flavor_name varchar(255) N
- flavor_description varchar(255)

license
- **license_id** integer U P
- license_name varchar(255) N
- license_description varchar(255)

Example Programs

example1.py - SELECT queries
example2.py - INSERT, UPDATE queries

Example1.py

```
#!/usr/bin/env python2.4
# example1.py
import sys
import psycopg

print "running example1.py...\n"

# create a connection to the db
try:
    connection=psycopg.connect("host=127.0.0.1 dbname=test user=postgres")
except StandardError, e:
    print "Unable to Connect!", e
    sys.exit()

# create a cursor
cursor=connection.cursor()

# get one row from a table
print "================= Selecting one records =================="
l_dict = {'license_id': 1}
cursor.execute("SELECT * FROM license WHERE license_id = %(license_id)s",l_dict)
rows = cursor.fetchall()
for row in rows:
    print row

# get multiple rows from a table
print "\n\n=============== Selecting multiple records ==============="
l_dict2 = {'license_id': 2}
cursor.execute("SELECT * FROM license WHERE license_id > %(license_id)i",l_dict2)
rows = cursor.fetchall()
for row in rows:
    print row
#
print "\n"
cursor.close()
connection.close()
sys.exit()
```

Example2.py

```
if __name__ == '__main__':
    '''example2.py - simple Python DB-API example
    '''
    print "running example2.py...\n"

    # create a connection to the db
    try:
        connection=psycopg.connect("host=127.0.0.1 dbname=test user=postgres")
    except StandardError, e:
        print "Unable to Connect!", e
        sys.exit()

    # create a cursor
    cursor=connection.cursor()
    # run the queries
    example2_insert(connection,cursor)
    example2_update(connection,cursor)
    example2_delete(connection,cursor)

    print ""
    print "== All Done ==================================================================="
    print ""

    cursor.close()
    connection.close()
    sys.exit()
```

Example2.py (continued)

```
def example2_insert(connection,cursor):
    """Insert some rows into the os_distributions
       table.
    """
    # the raw data tuples
    os_distributions = [
    ( 1,'Ubuntu',0,1),
    ( 2,'Solaris',3,4),
    ( 3,'Next',4,2),
    ( 4,'OpenBSD',1,0),
    ( 5,'Windows 2000',2,2)
    ]
    # build a list of the dictionaries for each row to be inserted
    insert_list = [create_item_dict(parm_col_name_list,item) for item in os_distributions]
    #

    # look at the records before we insert
    print "== Before the INSERT ================================================================"
    example2_select(cursor)

    # insert some OS Distributions
    print ""
    print "== Performing several INSERTs ======================================================="
    cursor.executemany("""
        INSERT INTO os_distribution
        (os_distribution_id,os_distribution_name,flavor_id,license_id)
        VALUES (%(od_id)s,%(od_name)s,%(od_flav)s,%(od_license)s) """,insert_list)
    connection.commit()

    # look at the records we just inserted
    print ""
    print "== After the INSERTs ================================================================"
    example2_select(cursor)
```

Transactions

- DB-API specifies that “auto-commit” mode is initially switched off
- Use explicit `conn.commit()` and `conn.rollback()` statements
- Pay attention to thread-safety level
- Different modules have different capabilities (e.g. psycopg can commit on cursor object with `psycopg.connect(DSN, serialize=0)`

Parameter Styles

Separate the SQL from your data by using parameter substitution (i.e. don’t just build fixed SQL statements as strings). The benefits include:

Improved security

(resistance to SQL injection attacks)

Improved performance

(may optimize queries that use the same template)

- 'qmark'   Question mark style, e.g. '...WHERE name=?'
- 'numeric' Numeric, positional style, e.g. '...WHERE name=:1'
- 'named'   Named style, e.g. '...WHERE name=:name'
- 'format'   ANSI C printf format codes, e.g. '...WHERE name=%s'
- 'pyformat'   Python extended format codes, e.g. '...WHERE name=%(name)s'

Error Handling

All error information is available through the following Exceptions:

```
StandardError
|__Warning
|__Error
   |__InterfaceError
   |__DatabaseError
      |__DataError
      |__OperationalError
      |__IntegrityError
      |__InternalError
      |__ProgrammingError
      |__NotSupportedError
```

Additional Topics

- Calling stored procedures via `cur.callproc()`
- Writing stored procedures in Python (PL/Python)
- Optimizing cursor usage (connection pools, multithreaded app, asynchronous event queues)
- Many helper recipes/classes available http://aspn.activestate.com/ASPN/Cookbook/Python
- Python Object Relational Mapping tools (SQLObject, SQL Alchemy, …)

Resources

- URL: http://www.python.org/doc/topics/database/
- Article: The Python DB-API http://www.linuxjournal.com/node/2605/
- Book: PostgreSQL (2nd Edition) (Paperback) by Korry Douglas ISBN: 0672327562
- Book: Python Web Programming by Steve Holden

Questions?