Database programming with Python

Magnus Lyckå
Thinkware AB
www.thinkware.se

EuroPython Conference 2004
Chalmers, Göteborg, Sweden

In the next 30 minutes you should...

- Learn about various database solutions for Python, and what they are good for.
- Get aquainted to the Python DB-API 2.0 and see some examples of how it is used.
- Get a brief introduction to an object oriented layer on top of the DB-API.
- Get a brief introduction to the Zope Object Database (ZODB).

What's a database?

- "...a logically coherent collection of data with some inherent meaning" –Fundamentals of Database Systems
- "..an information set with a regular structure" – Wikipedia
- Persistent data storage
- Structure, consistency, relationships
- Transactions (ACID, 2PC)
- Backup/Restore/Replication/Security
- Ad hoc queries – not just for one program...

What's it for?

- One user – Thousands of simultaneous users
- Handheld device – Cluster of servers
- Amusement – Life support system
- Mainly lookup – Lots of transactions
- Uniform data – Widely varied kinds of data
- Simple structures – Complex relationships
- etc...

Types of database systems

- Deployment
  - Embedded in application
  - Standalone server process
    - Networked / Distributed
- Database structure
  - (Hierachical/Network)
  - Simple table (flat)
  - Relational
  - Object-oriented
- Interface between program and database system
  - Dictionary-like
  - Special API
  - Special language (SQL)
  - Seamless

Standard modules

- Unix style dbm – `anydbm, dbhash (bsddb), gdbm, dbm, dumbdbm`
  - Embedded
  - Standard library
  - Simple table
  - Dictionary-like
  - Both keys and values are strings
  - File formats vary among implementations!
- Shelve module – `shelve`
  - Like above but values can be arbitrary Python objects.

Other embedded databases

- SQLite
  - Relational, SQL subset (typeless), DB-API
- Gadfly
  - Relational, SQL subset
- xsdb
  - Flat tables, multiple indices, arbitrary values, also server version
- mxBeeBase
  - Flat file, Dictionary-like, transaction support
- MetaKit
  - Special API, “It fills the gap between flat-file, relational, object-oriented, and tree-structured databases”

Python DB-API 2.0

- Fairly uniform access to (mainly) SQL databases.
- Balance the effort needed to develop applications and the effort needed to maintain the database interfaces.
- Implemented for most available SQL database systems.

Some DB-API implementations

- MySQL
  - MySQLdb
- PostgreSQL
  - psycopg
  - PyGresQL
  - pyPgSQL
- Oracle
  - dc_oracle2
  - cx_oracle
- Interbase/Firebird
  - Kinterbasdb
- SAP DB / MaxSQL
  - sapdbapi
- DB2
  - pydb2
- ODBC
  - mxODBC
  - adodbapi

Alternative SQL APIs

- ADO
  - http://www.ecp.cc/pyado.html
  - http://www.mayukhbose.com/python/ado/index.php
- DAO
  - http://starship.python.net/crew/bwilk/access.html
- SQLRelay
  - http://sqlrelay.sourceforge.net/
- PL/Python for PostgreSQL (stored procedures)
  - http://www.postgresql.org/docs/7.4/static/plpython.html

Higher level interfaces over DB-API

- Simple wrappers
  - providing more convenient access to result sets
    - db_row – http://opensource.theopalgroup.com/
    - dtuple – http://www.lyra.org/greg/python/
- Object-oriented wrappers
  - Hides DB-API completely
    - SQLObject – http://sqlobject.org/
    - PDO – http://pdo.neurokode.com/index.php
    - Modeling – http://modeling.sourceforge.net/

See http://www.thinkware.se/cgi-bin/thinki.cgi/ObjectRelationalMappersForPython

Object-oriented databases

- ZODB – Zope Object Database
  - http://zope.org/Wikis/ZODB/FrontPage
- Cog – Checkpointed Object Graph object database
  - http://www.itamarst.org/software/cog/
- ATOP – the Atomic Transactional Object Persistor
  - http://www.divmod.org/Home/Projects/Atop/
- PyVersant – wrapper for Versant OODBMS
  - http://www.amk.ca/python/unmaintained/versant.html

Introduction to Python's DB-API

- Connection objects
- Cursor objects
- Result sets
- Standard exceptions
- Other module contents

Connection object

- Represents a database session.
- Responsible for transactions.
- Each cursor belongs to a connection.
- Constructor:

```
cnx = dbi.connect('mydb', user='mly', password='secret')
```

The connect parameters vary between databases!

- Methods:

```
cur = cnx.cursor()
cnx.commit()   cnx.rollback()   cnx.close()
```

Cursor object

- cursor.execute('SELECT * FROM aTable')
- cursor.execute('"SELECT * FROM aTable WHERE name = ? AND age = ?"', ('Bill', 23))
  - The syntax varies...it might also be...
  - cursor.execute('"SELECT * FROM aTable WHERE name = %s AND age = %s"', ('Bill', 23))
- cursor.executemany('"SELECT * FROM aTable WHERE name = ? AND age = ?"', [('Bill', 23), ('Mary', 12), ('Anne', 87)])
- cursor.close() - Do this before commit!

Cursor object

- a_row = cursor.fetchone()
- a_sequence_of_rows = cursor.fetchall()
- a_sequence_of_rows = cursor.fetchmany(100)

- cursor.description
  - Read-only attribute with a sequence of tuples, one per column in the result set. Each tuple contains the following info about the column: name, type code, display size, internal size, precision, scale, nullable.
  - Could also be None... (I.e. after an insert.)

Result set

- One or more rows from a relation.
- .fetchone() returns a sequence (often a tuple).
- .fetchall() and .fetchmany() returns a sequence of sequences, for instance a list of tuples.
- Don't assume more about types than you need!
  - It might not be a list of tuples, but it will be a sequence of sequences.
- Some dbi's have more elaborate row objects.
  - There's always addons if yours hasn't...

DB-API Example

- dbapitest.py

DB-API Exceptions

- Warning
- Error
  - InterfaceError
  - DatabaseError
    - IntegrityError
    - InternalError
    - NotSupportedError
    - OperationalError
    - ProgrammingError

DB-API module attributes

- dbi.paramstyle
  - format – %s
  - named – :col_name
  - numeric – :1
  - pyformat – %(col_name)s
  - qmark – ?
- dbi.threadsafety – 0, 1...
- dbi.apilevel – e.g. '2.0'

DB-API variations

- Different module names (obviously)
  - import xxx as dbi
- Different parameters for connect()
  - conn = adodbapi.connect('dsn=db1;uid=tom;pwd=x')
  - conn = sqlite.connect('c:/dbs/accounting.db')
- Different parameter styles (?, %s, :1 etc)
- Differences in SQL dialects
- Optional features of the DB-API

db_row

- A wrapper for DB-API result sets (or other sequences of tuples).
- Exposes the “normal” sequence-like access
  - row[0], row[1]
- Provides dictionary-like access to columns
  - row['name'], row['email'], row.keys(), row.values()
- Provides attribute access to columns
  - row.?.name, row.?.email

SQLObject

- Object-oriented wrapper over DB-API.
- Insulates the programmer from SQL.
- Not so easy to use with legacy databases.
- Example (a whisky tasting database).

ZODB example

```
import ZODB, ZODB.FileStorage

class Counter(ZODB.Persistent):
  def __init__(self,name):self.name,self.value = name,0
  def __call__(self):self.value += 1
  def __str__(self):return "%s: %i"%(self.name, self.value)

storage = ZODB.FileStorage.FileStorage('zodbtest.fs')
db = ZODB.DB(storage)
conn = db.open(); root = conn.root()
name = raw_input('Counter? ')
if not root.has_key(name):
    root[name] = Counter(name)
obj = root[name]; obj()
ZODB.Transaction.get_transaction().commit()
print obj
```

Useful Links

- PEP 249 Python DB-API 2.0
  - http://www.python.org/peps/pep-0249.html
- Python DB topic guide
  - http://www.python.org/topics/database/
- ZODB
  - http://zope.org/Wikis/ZODB/FrontPage

Useful books...

- Python in a Nutshell by Alex Martelli
  - Concise reference DB-API
- Web Programming with Python by Steve Holden
  - DB-API tutorial
- SQL Visual Quickstart Guide by Chris Fehily
  - For learning SQL
- Agile Database Modeling by Scott Ambler
  - For modeling large database systems