Introduction to pysqlite

A crash course to accessing SQLite from within your Python programs.

Based on pysqlite 2.0.

Gerhard Häring gh@ghaering.de - Python, Web- and database development.

SQLite basics

- SQLite is embedded, there is no server
- Each SQLite database is stored in one file
- SQLite supports in-memory databases, too

SQLite basics (2)

SQLite supports the following types

- ✓ TEXT
- ✓ INTEGER
- ✓ FLOAT
- ✓ BLOB
- ✓ NULL

SQLite basics (3)

Type conversions from SQLite to Python

TEXT →	unicode
INTEGER →	int
FLOAT →	float
BLOB →	buffer
NULL →	None

Import the module

```
from pysqlite2 import dbapi2 as sqlite
```

Open a connection

```
con = sqlite.connect(“mydb.db”)
```

If the file does not exist, an empy database is created.

```
con = sqlite.connect(“:memory:”)
```

In-memory databases are always empty when created.

Create a cursor

```
cur = con.cursor()
```

Execute a query ... and fetch one row

Returned rows are tuples!

```
cur.execute("select firstname, lastname from person")
row = cur.fetchone()
if row is None:
    # Error, no result
else:
    firstname, lastname = row[0], row[1]
```

Execute a query ... and process all rows

The cursor is iterable, just loop over the cursor!

```
cur.execute(“select firstname, lastname from person”)
for row in cur:
    print “firstname: %s, lastname: %s” % (row[0], row[1])
```

Queries with parameters (1)

`cur.execute(SQL, parameters)`

SQL:

Python string, must be encoded in UTF-8 if it contains non-ASCII characters. Or: Unicode string.

Parameters:

Sequence (list, tuple, ...) or mapping (dict).

Queries with parameters (2)

Use ? as placeholders
and
use a sequence for the parameters:

```
cur.execute(“””
      insert into person(firstname, lastname)
       values (?, ?)”,
   (“Gerhard”, “Haering”)
)
```

Queries with parameters (3)

Use :name as placeholders
and
use a mapping for the parameters:

```
item = {“firstname”: “Gerhard”, “lastname”: “Haering”}
cur.execute(“””
      insert into person(firstname, lastname)
       values (:firstname, :lastname)”,
   item
)
```

Queries with parameters (4)

Neat hack - use the fact that locals() is a mapping, too:

```
firstname = “Gerhard”
lastname = “Haering”

cur.execute(“””
      insert into person(firstname, lastname)
       values (:firstname, :lastname)”,
  locals()
)
```

Oops? Where's my data???

pysqlite uses transactions so that your database is always in a consistent state.

To make changes permanent you must commit the changes!

Committing changes

```
cur = con.cursor()
cur.execute("insert into table1 ...")
cur.execute("insert into table2 ...")
con.commit()
```

After database modifications that belong together logically, commit your changes so that this consistent state is stored permanently.

Roll back changes

```
cur = con.cursor()
try:
    cur.execute("delete from ...")
    cur.execute("delete from ...")
    con.commit()
except sqlite.DatabaseError:
    con.rollback()
```

Roll back changes when an error occured, in order to keep your database consistent!

Be nice and clean up

```
cur.close()
con.close()
```

You should close cursors and connections that you no longer use. Only close the connection when you've closed all cursors you created from it!

Conclusion

That's it – I hope you've learnt something about using SQLite from Python using pysqlite 2.0!

Resources

The Wiki on http://pysqlite.org/

The pysqlite mailing list!

For how SQLite works and the SQL it supports:
http://sqlite.org/