Python & SQLite

Aaron Robinson

Chemical & Biological Signature Science
National Security Directorate

August 15, 2012

Background

Outline:

- SQLite
- Data Definition Language
- SQLite Datatypes

SQLite

Pacific Northwest
NATIONAL LABORATORY

What is it?

It is a...

- Public domain
- SQL database engine
- C library

Features:

- self-contained
- serverless
- zero-configuration
- transactional SQL

Arguably the most widely deployed SQL database engine in the world – due to embeded nature versus competing systems.

Ref: www.sqlite.org/mostdeployed.html

Data Definition Language

A data definition language (DDL) is a syntax for defining data structures, often database schemas.

SQLite's uses a dynamic type system which...

- is often compatible with other statically typed databases
- allows data manipulations not possible in traditional rigidly typed databases

SQLite supports the following statements for defining data definitions and maintaining them:

CREATE	INDEX, TABLE, TRIGGER, and VIEW
DROP	INDEX, TABLE, TRIGGER, and VIEW
ALTER	TABLE

SQLite Datatypes

INTEGER The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.

TEXT The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).

BLOB The value is a blob of data, stored exactly as it was input.

REAL The value is a floating point value, stored as an 8-byte IEEE floating point number.

NULL The value is a NULL value.

NUMERIC A column with NUMERIC affinity may contain values using all above five storage classes.

INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT
BLOB *no datatype specified*	NONE
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC

Let's build it!

Pacific Northwest
NATIONAL LABORATORY

Let's make a database for Michael Phelps' olympic records.

Outline:

- Phelps' Excel Data
- SQL Create Table
- Example 1: Building & Populating

Phelps' Excel Data

Pacific Northwest NATIONAL LABORATORY

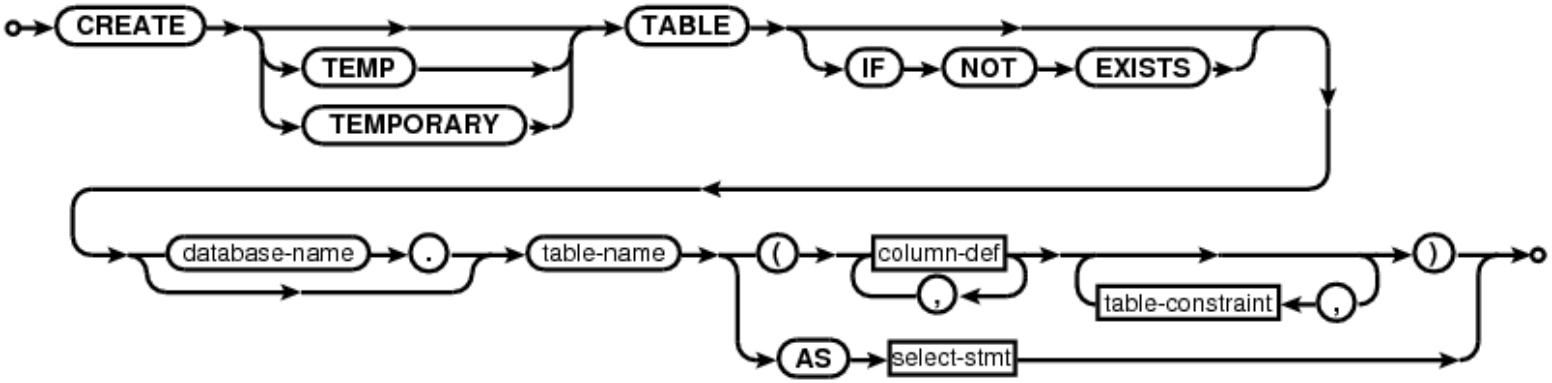
Phelps

Championship	Medal	Event	Time	Record
2004 Summer Olympics	Gold	100 m butterfly	51.25	OR
2004 Summer Olympics	Gold	200 m butterfly	01:54.04	OR
2004 Summer Olympics	Bronze	200 m freestyle	01:45.32	NR
2004 Summer Olympics	Gold	200 m individual medley	01:57.14	OR
2004 Summer Olympics	Bronze	4×100 m freestyle relay	03:14.62	NR
2004 Summer Olympics	Gold	4×100 m medley relay	03:30.68	WR
2004 Summer Olympics	Gold	4×200 m freestyle relay	07:07.33	NR
2004 Summer Olympics	Gold	400 m individual medley	04:08.26	WR
2008 Summer Olympics	Gold	100 m butterfly	50.58	OR
2008 Summer Olympics	Gold	200 m butterfly	01:52.03	WR
2008 Summer Olympics	Gold	200 m freestyle	01:42.96	WR
2008 Summer Olympics	Gold	200 m individual medley	01:54.23	WR
2008 Summer Olympics	Gold	4×100 m freestyle relay	03:08.24	WR
2008 Summer Olympics	Gold	4×100 m medley relay	03:29.34	WR
2008 Summer Olympics	Gold	4×200 m freestyle relay	06:58.56	WR
2008 Summer Olympics	Gold	400 m individual medley	04:03.84	WR
2012 Summer Olympics	Gold	100 m butterfly	51.21	NR
2012 Summer Olympics	Silver	200 m butterfly	01:53.01	NR
2012 Summer Olympics	Gold	200 m individual medley	01:54.27	NR
2012 Summer Olympics	Silver	4×100 m freestyle relay	03:10.38	NR
2012 Summer Olympics	Gold	4×100 m medley relay	03:29.35	NR
2012 Summer Olympics	Gold	4×200 m freestyle relay	06:59.70	NR

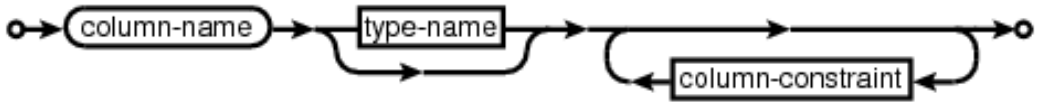
SQL Create Table

Pacific Northwest NATIONAL LABORATORY

CREATE TABLE



column-def:



Ref: www.sqlite.org/lang_createtable.html

Example 1: Building & Populating

Pacific Northwest NATIONAL LABORATORY

Exemplifies:

- Python
  - *sqlite3.connect*()
  - *Connection.execute*()
  - *Connection.commit*()
- SQL
  - *CREATE TABLE*
  - *INSERT INTO*

connect() will create the file if it doesn't exist. If we need to change this behavior we can use python's *os.path.exist*().

```
1  import sqlite3, csv
2
3  data = []
4  with open('phelps.csv') as input:
5      csvReader = csv.reader( input ,
6          delimiter = ',', quotechar = '"')
7      for row in csvReader:
8          data.append(row)
9  data = data[1:] # skip header line
10
11 # create connection object
12 con = sqlite3.connect('db.sqlite')
13
14 # create table
15 con.execute("""
16 CREATE TABLE phelps (
17     championship VARCHAR(25),
18     medal VARCHAR,
19     event TEXT,
20     time DATETIME,
21     record CHARACTER(2) )
22 """)
23
24 con.text_factory = str
25 for row in data:
26     con.execute( """INSERT INTO phelps
27         VALUES (?,?,?,?,?)""", row)
28 con.commit()
```

And query it!

"Well how many Gold's did that Michael Phelps get?"

Their's a query for that.

Outline:

- SQL Select Query
- Example 2: Select Queries

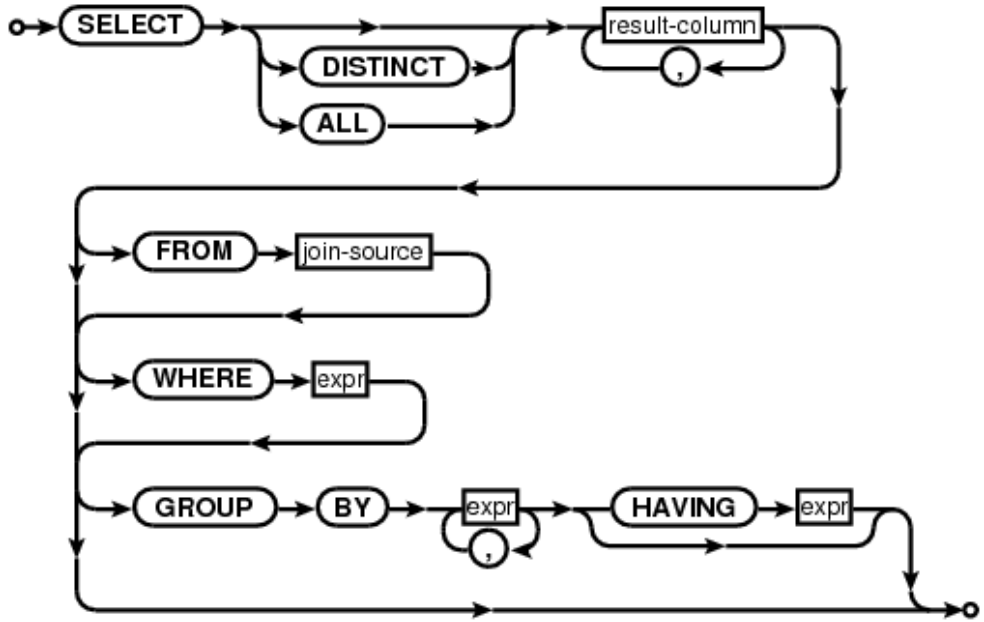
SQL Select Query

The 'SELECT' statement returns a result set from one or more tables.

SQL is not case sensitive.

Most programmers use capitals to denote SQL keywords but this is by practice only.

SELECT



Ref: www.sqlite.org/lang_select.html

Example 2: Select Queries

Pacific Northwest NATIONAL LABORATORY

Exemplifies:

- Python
  - *Cursor.fetchone()*
  - *Connection.text_factory*
- SQL
  - *SELECT*
  - *WHERE*

The '*Connection.text_factory*' property changes what type of object is returned from a SQLite TEXT datatype.

```
1  import sqlite3
2
3  # create connection object
4  con = sqlite3.connect('db.sqlite')
5  con.text_factory = str
6
7  from_where = """
8  FROM phelps
9  WHERE medal = 'Gold'
10 """
11
12 # lets count how many golds phelp has
13 select = "SELECT count(*)" + from_where
14 result = con.execute(select).fetchone()
15 golds = int(result[0])
16 print "The number of golds Michael Phelp's
       has won in his olympic career is: %i
      " % golds
17 print type(result) # returns a tuple
18
19 # lets see the events (NOTE: duplicates)
20 print "The events Phelps was in:"
21 select = "SELECT event" + from_where
22 for rec in con.execute(select):
23     print rec[0]
```

And schema changes?

Pacific Northwest
NATIONAL LABORATORY

"What if we want to change the schema?"

"What if we want to alter a column in our table?"

Let's make the medal field case-insensitive.

Outline:

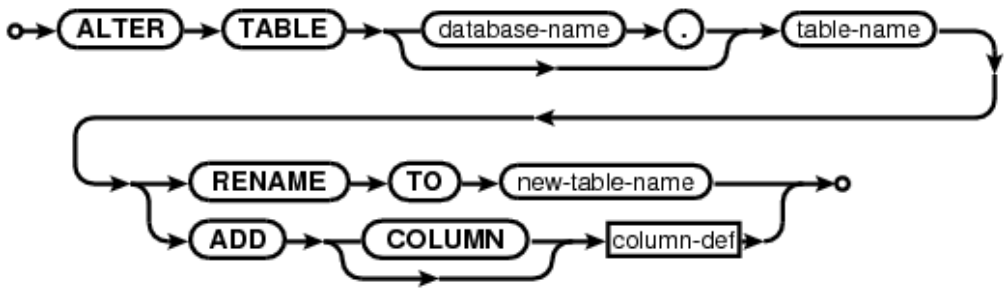
- SQL Alter Table
- SQL Alter Column
- Example 3: Table Updates

SQL Alter Table

SQLite supports a limited 'ALTER TABLE' statement for:

- Renaming tables
- Adding columns

ALTER TABLE



Ref: www.sqlite.org/lang_altertable.html

SQL Alter Column

SQLite supports no ‘ALTER COLUMN’ statement.

To do the behavior of an ‘ALTER COLUMN’ we can:

1. Rename the table to a temporary name
2. Create a new table with the corrected field
3. Insert data from the old table into the new table
4. Drop the old table

An alternative method exists via use of ’PRAGMA’ statement but is unsafe and may corrupt the database.

Example 3: Table Updates

Pacific Northwest NATIONAL LABORATORY

Exemplifies:

- Python
  - *Connection.executescript*()
- SQL
  - *ALTER TABLE*
  - *DROP TABLE*

We check that the medal field is now case-insensitive *via* a query counting bronze and silver medals

```
1  import sqlite3
2
3  # create connection object
4  con = sqlite3.connect('db.sqlite')
5  con.text_factory = str
6
7  con.executescript("""
8  ALTER TABLE phelps RENAME TO junk;
9  CREATE TABLE phelps (
10     championship VARCHAR(25),
11     medal VARCHAR COLLATE NOCASE,
12     event TEXT,
13     time DATETIME,
14     record CHARACTER(2) );
15 INSERT INTO phelps SELECT * FROM junk;
16 DROP TABLE junk;
17 """)
18
19 # lets count bronzes and silvers
20 select = """
21 SELECT count(*)
22 FROM phelps
23 WHERE medal = 'SiLvEr' OR medal = 'Bronze'
24 """
25 result = con.execute(select).fetchone()
26 print "The number of bronze and silvers
       Phelps received:", result[0]
```

Helper objects?

"Are there helper objects for mapping column names and datatypes to results?"

Yes.

Outline:

- Example 4: *sqlite*3.*Row*

Example 4: *sqlite3.Row*

Pacific Northwest NATIONAL LABORATORY

Exemplifies:

- Python
  - *Connection.row_factory*
  - *sqlite3.Row*

sqlite3.Row

- Allows access *via* column name and index
- The *keys*() method returns a tuple of column names
- Using slice notation generates "Not implemented yet" exception

```
1   import sqlite3
2
3   # create connection object
4   con = sqlite3.connect('db.sqlite')
5   con.row_factory = sqlite3.Row
6
7   select = """
8   SELECT event, MIN(time), MAX(time)
9   FROM phelps
10  GROUP BY event
11  """
12
13  print "Below are min and max times for
        Phelp's grouped by race:"
14  for row in con.execute(select):
15      print "Event:", row['event']
16      print "Min:", row[1]
17      print "Max:", row['MAX(time)']
18      print
```

We can rebuild it

Pacific Northwest
NATIONAL LABORATORY

"What if we want to change the schema?"

... and we are willing to be *risky*

PRAGMA statements allow us to edit database features and data access control.

Outline:

- Example 5: SQLite Master Table
- Example 6: Edit Schema

Example 5: SQLite Master Table

Pacific Northwest NATIONAL LABORATORY

Exemplifies:

- Python
  - *Connection.cursor*()
  - *Cursor.execute*()
  - *Cursor.description*

The 'sqlite_master' table stores the SQLite database's schema.

'sqlite_master' fields:

- Type
- Name
- Tbl_Name
- Rootpage
- SQL

```
1  import sqlite3
2
3  # create connection object
4  con = sqlite3.connect('db.sqlite')
5  con.row_factory = sqlite3.Row
6
7  cur = con.cursor()
8
9  # sqlite_master stores the schema
10 cur.execute("SELECT * FROM sqlite_master")
11
12 # description is a 7-tuple w/ [Nones]*6
13 headers = zip(*cur.description)[0]
14
15 # print out data in sqlite_master
16 for record in cur:
17     print "New Record:"
18     for field in headers:
19         print "%s: %s" %(field, record[
               field])
```

Example 6: Edit Schema

Pacific Northwest NATIONAL LABORATORY

Exemplifies:

- SQL
  - *PRAGMA*
  - *UPDATE*

"Warning: misuse of this pragma can *easily* result in a corrupt database file."

This potential exists because changes are made directly to the schema without any validation.

```
1  import sqlite3
2
3  # create connection object
4  con = sqlite3.connect('db.sqlite')
5  con.row_factory = sqlite3.Row
6  cur = con.cursor()
7
8  # sqlite_master stores the schema
9  cur.execute("SELECT sql FROM sqlite_master
         WHERE type='table' and name='phelps
        '")
10
11 sql = cur.fetchone()['sql']
12
13 caseless = "TEXT COLLATE NOCASE"
14 sql = sql.replace("TEXT",caseless)
15 sql = sql.replace("VARCHAR(25)",caseless)
16 sql = sql.replace("CHARACTER(2)",caseless)
17 sql = sql.replace("VARCHAR", "TEXT")
18
19 cur.execute("PRAGMA writable_schema = 1")
20 cur.execute("UPDATE sqlite_master SET sql
       = ? WHERE type='table' and name='
       phelps'", (sql,))
21 cur.execute("PRAGMA writable_schema = 0")
```

Game changing data

Since Michael Phelps is retiring, we've decided to add Ryan Lochte for tracking him in future olympic seasons.

Outline:

- Lochte's Excel Data
- Example 7: New Architecture
- Example 8: New Data

Lochte's Excel Data

Pacific Northwest NATIONAL LABORATORY

Lochte

Championship	Medal	Race	Time	Record
2008 Summer Olympics	**Gold**	200 m backstroke	01:53.94	WR
2008 Summer Olympics	**Gold**	4×200 m freestyle relay	06:58.56	WR
2008 Summer Olympics	**Bronze**	200 m individual medley	01:56.53	
2008 Summer Olympics	**Bronze**	400 m individual medley	04:08.09	
2012 Summer Olympics	**Gold**	400 m individual medley	04:05.18	NR
2012 Summer Olympics	**Gold**	4×200 m freestyle relay	06:59.70	NR
2012 Summer Olympics	**Silver**	200 m individual medley	01:54.90	NR
2012 Summer Olympics	**Silver**	4×100 m freestyle relay	03:10.38	NR
2012 Summer Olympics	**Bronze**	200 m backstroke	01:53.94	NR

Example 7: New Architecture


```
1   import sqlite3
2
3   # create connection object
4   con = sqlite3.connect('db.sqlite')
5
6   # create table
7   con.execute("""
8   CREATE TABLE champs (
9       id INTEGER PRIMARY KEY,
10      name TEXT COLLATE NOCASE,
11      championship TEXT COLLATE NOCASE,
12      medal VARCHAR,
13      event TEXT COLLATE NOCASE,
14      time DATETIME,
15      record TEXT COLLATE NOCASE )
16  """)
17
18  # insert values from old table (phelps) into new table (champs)
19  con.execute("""
20  INSERT INTO champs (name, championship, medal, event, time, record)
21  SELECT 'Michael Phelps', championship, medal, event, time, record
22  FROM phelps
23  """)
24
25  con.execute("DROP TABLE phelps")
```

Example 8: New Data


```
1   import sqlite3, csv
2
3   data = []
4   with open('lochte.csv') as input:
5       csvReader = csv.reader( input ,
6           delimiter = ',', quotechar = '"')
7       for row in csvReader:
8           data.append(row)
9   data = data[1:] # skip header line
10
11  # create connection object
12  con = sqlite3.connect('db.sqlite')
13
14  insert = """
15  INSERT INTO champs (championship, name, medal, event, time, record)
16  VALUES (?,?,?,?,?,?)
17  """
18
19  con.text_factory = str
20  con.executemany(insert, data)
21  con.commit()
```

As complexity grows

The major gains to relational databases are found when data is seperated into related tables.

The process of optimizing databases by minimizing redudancy and dependency is known as *database normalizaton*.

Outline:

- Example 9: Normalization
- Example 10: Making reports

Example 9: Normalization


```
1   import sqlite3
2
3   # create connection object
4   con = sqlite3.connect('db.sqlite')
5   con.row_factory = sqlite3.Row
6
7   con.execute("""
8   CREATE TABLE medals (
9       id INTEGER PRIMARY KEY,
10      name TEXT)
11  """)
12  # make name case insensitive unique
13  con.execute("CREATE UNIQUE INDEX uidxName ON medals (name COLLATE NOCASE)")
14
15  datafix = [ (3,'Gold'), (2,'Silver'), (1,'Bronze') ]
16  con.executemany("INSERT INTO medals VALUES (?,?)", datafix)
17  con.executemany("UPDATE champs SET medal = ? WHERE medal = ?", datafix)
18
19  where_clause = "WHERE type='table' and name='champs'"
20  sql = con.execute("SELECT sql FROM sqlite_master " + where_clause).fetchone()['sql']
21  sql = sql.replace("medal TEXT COLLATE NOCASE,","medal INTEGER")
22
23  con.execute("PRAGMA writable_schema = 1")
24  con.execute("UPDATE sqlite_master SET sql = ? " + where_clause, (sql,))
25  con.execute("PRAGMA writable_schema = 0")
```

Example 10: Making Reports


```
1   import sqlite3
2
3   # create connection object
4   con = sqlite3.connect('db.sqlite')
5   cur = con.cursor()
6
7   con.execute("""
8   CREATE VIEW report AS
9   SELECT c.id, c.name, c.championship, m.name AS medal, c.event, c.time, c.record
10  FROM champs AS c
11  JOIN medals AS m ON c.medal = m.id
12  """)
13
14  cur.execute("""
15  SELECT championship, event
16  FROM (SELECT championship, event, COUNT(*) AS cnt FROM champs GROUP BY championship,
          event)
17  WHERE cnt > 1
18  """)
19
20  s = "SELECT name,time,medal,record FROM report WHERE championship=? AND event=?"
21  for championship, event in cur:
22      print "Championship:", championship
23      print "Event:", event
24      for row in con.execute(s, (championship, event) ):
25          print "Name: %s\nTime:%s\tMedal:%s\tRecord:%s\n" % row
26      print "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
```

Code packing

"What if we want to pack our code inside an sqlite database?"

Yes.

Outline:

- Example 11: BLOBs

Example 11: BLOBs


```
1   import sqlite3, zlib, glob, os
2
3   # create connection object
4   con = sqlite3.connect('db.sqlite')
5   con.row_factory = sqlite3.Row
6
7   # create table
8   con.execute("CREATE TABLE code (id INT PRIMARY KEY, filename TEXT, data BLOB)")
9
10  def compressFile(infile):
11      with open(infile) as f:
12          name = os.path.basename(infile)
13          compressed_data = sqlite3.Binary(zlib.compress(f.read()))
14      return (name, compressed_data)
15
16  for f in glob.glob('*.py'):
17      con.execute("INSERT INTO code ('filename','data') VALUES(?,?)", compressFile(f))
18  con.commit()
19
20  for row in con.execute("SELECT * FROM code"):
21      print "Filename:", row['filename']
22      print "Compressed data lenght:", len(row['data'])
23      decompress = zlib.decompress(row['data'])
24      print "Decompressed data lenght:", len(decompress)
25      print
```

Mapping & Diagraming

As databases get more complex or multiple database dialets become involved, the need for more tools arises.

Object-relational mapping (ORM)

- Maps incompatible type systems
- Provides object-oriented access
- Often supports multiple dialects

Ex. SQLAlchemy
www.sqlalchemy.org

Entity-relationship diagrams (ERD)

- Show relationships within a database
- Contains entities, relationships, attributes
- May show cardinality of relationships

Ex. Open System Architect
www.codebydesign.com