SQLite

April 19, 2013

http://www.sqlite.org/
http://www.sqlite.org/docs.html

What is it?

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain.

from http://www.sqlite.org/

- What does all that mean?

Queriable with SQL.

Why do we need this?

http://www.sqlite.org/whentouse.html

- Databases embedded in an application
  - file format
  - temporary data
  - Google Chrome
  - Firefox
- Databases embedded on a device
  - Android
- Small datasets

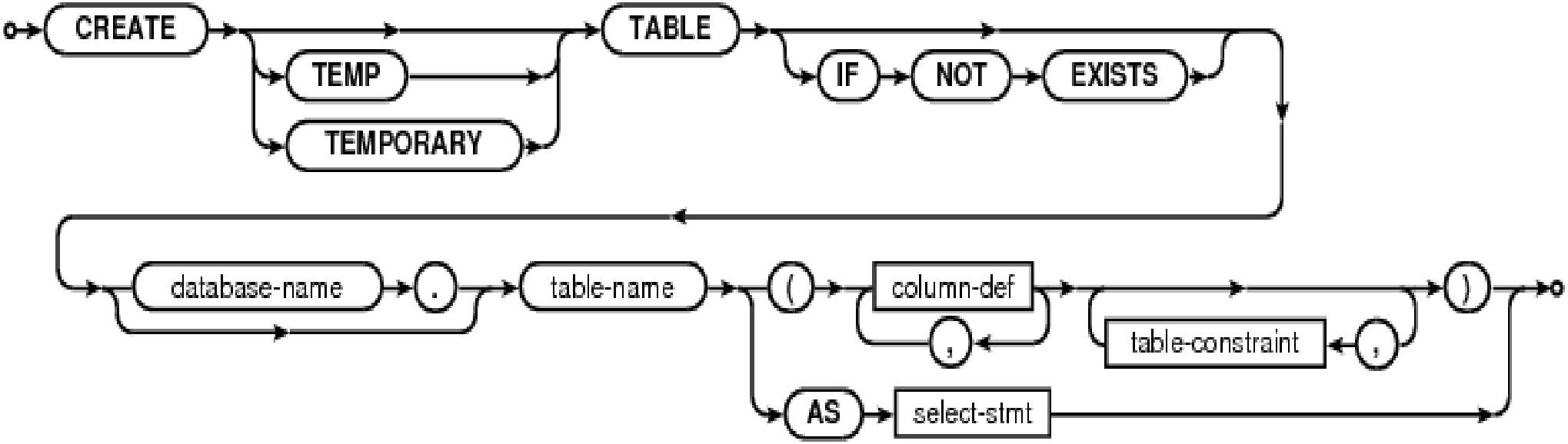
MySQL vs SQLite
Server vs Serverless

- When to use each?
- Strengths of MySQL
- Strengths of SQLite

Auto Increment
http://www.sqlite.org/autoinc.html

SQL

- http://www.sqlite.org/syntaxdiagrams.html
- create-table-stmt



- column-def:

Command Line

```
chadd@bart:~> sqlite3 movies.db

SQLite version 3.7.8 2011-09-19 14:49:19
Enter ".help" for instructions
Enter SQL statements terminated with a ";"

sqlite> CREATE TABLE Movies
   ...> (MovieID INTEGER NOT NULL,
   ...> TITLE TEXT(512) NOT NULL,
   ...> PRIMARY KEY (MovieID));

sqlite> INSERT INTO Movies VALUES (null, "Star Wars");
sqlite> INSERT INTO Movies VALUES (null, "Empire");

sqlite> SELECT * FROM Movies;
1|Star Wars
2|Empire
sqlite> .quit
```

Not Implemented

- http://www.sqlite.org/omitted.html
  - As compared to SQL92

- Right & Full Outer Join
- Complete Alter Table
- Complete Trigger
- Writing to Views
- Grant & Revoke

C/C++ interface

```
#include <sqlite3.h>

sqlite3 *psDB;
sqlite3_stmt *psStmt;

sqlite3_open_v2(..., &psDB, ...);                  // open the DB

sqlite3_prepare_v2(..., &psStmt, ...);             // build a Query

sqlite3_bind_int(psStmt, ...);                     // set a parameter

while ( SQLITE_ROW == sqlite3_step(psStmt) )       // retrieve one row
{
   sqlite3_column_int(psStmt, ...);                //retrieve one column
   sqlite3_column_text(psStmt, ...);               //retrieve one column

   // do something useful
}

sqlite3_finalize(psStmt);
sqlite3_close(psDB);
```

PHP!

```
<?php


  $dbh = new PDO
  ('sqlite:/space/sqlite/chadd/movies.db',
    null,  // user
    null   // password
   );


   $sth = $dbh->prepare(
      "SELECT * From Movies where MovieID = :movieid");

   $sth->bindValue(":movieid", 1);
   $sth->execute();
   $row = $sth->fetch();

   print $row[0] . " " . $row['Title'];

   $dbh = null;
?>
```

http://www.php.net/manual/en/book.sqlite.php
http://php.net/manual/en/ref.pdo-sqlite.php

Data Types http://www.sqlite.org/datatype3.html

- Dynamic Typing
- INTEGER (signed, at most 8 bytes)
- REAL (8-byte IEEE floating point)
- TEXT (UTF-8, UTF-16BE or UTF-16LE; Big/Little Endian)
- BLOB
- NULL
- Date & Time
  - no date and time data types, but data and time functions!
  - store as
    - TEXT “YYYY-MM-DD HH:MM:SS.SSSS” - ISO8601
    - REAL (number days since noon in Greenwich on November 24, 4714 B.C. proleptic Gregorian calendar)
    - INTEGER (Unix Time: the number of seconds since 1970-01-01 00:00:00 UTC)

Type Affinity

- What is the recommended data type for a column?
  - TEXT
    - null, text, blob
  - NUMERIC
    - any of the 5 types
  - INTEGER
    - same as numeric
  - REAL
    - like numeric, but forces ints to floats
  - NONE
    - wildcard

http://www.sqlite.org/datatype3.html 2.1 Determination of Column Affinity

Example

```
CREATE TABLE t1(
    t  TEXT,     -- text affinity by rule 2
    nu NUMERIC,  -- numeric affinity by rule 5
    i  INTEGER,  -- integer affinity by rule 1
    r  REAL,     -- real affinity by rule 4
    no BLOB      -- no affinity by rule 3
);

-- Values stored as TEXT, INTEGER, INTEGER, REAL, TEXT.
INSERT INTO t1 VALUES('500.0', '500.0', '500.0', '500.0', '500.0');
SELECT typeof(t), typeof(nu), typeof(i), typeof(r), typeof(no) FROM t1;
text|integer|integer|real|text

-- Values stored as TEXT, INTEGER, INTEGER, REAL, REAL.
DELETE FROM t1;
INSERT INTO t1 VALUES(500.0, 500.0, 500.0, 500.0, 500.0);
SELECT typeof(t), typeof(nu), typeof(i), typeof(r), typeof(no) FROM t1;
text|integer|integer|real|real

-- Values stored as TEXT, INTEGER, INTEGER, REAL, INTEGER.
DELETE FROM t1;
INSERT INTO t1 VALUES(500, 500, 500, 500, 500);
SELECT typeof(t), typeof(nu), typeof(i), typeof(r), typeof(no) FROM t1;
text|integer|integer|real|integer

-- BLOBs are always stored as BLOBs regardless of column affinity.
DELETE FROM t1;
INSERT INTO t1 VALUES(x'0500', x'0500', x'0500', x'0500', x'0500');
SELECT typeof(t), typeof(nu), typeof(i), typeof(r), typeof(no) FROM t1;
blob|blob|blob|blob|blob
```

Comparison Example

```
CREATE TABLE t1(
    a TEXT,       -- text affinity
    b NUMERIC,    -- numeric affinity
    c BLOB,       -- no affinity
    d             -- no affinity
);

-- Values will be stored as TEXT, INTEGER, TEXT, and INTEGER respectively
INSERT INTO t1 VALUES('500', '500', '500', 500);
SELECT typeof(a), typeof(b), typeof(c), typeof(d) FROM t1;
text|integer|text|integer

-- Because column "a" has text affinity, numeric values on the
-- right-hand side of the comparisons are converted to text before
-- the comparison occurs.
SELECT a < 40,   a < 60,   a < 600 FROM t1;
0|1|1

-- Text affinity is applied to the right-hand operands but since
-- they are already TEXT this is a no-op; no conversions occur.
SELECT a < '40', a < '60', a < '600' FROM t1;
0|1|1

-- Column "b" has numeric affinity and so numeric affinity is applied
-- to the operands on the right.  Since the operands are already numeric,
-- the application of affinity is a no-op; no conversions occur.  All
-- values are compared numerically.
SELECT b < 40,   b < 60,   b < 600 FROM t1;
0|0|1
```

Vacuum

http://www.sqlite.org/lang_vacuum.html

- Rebuild entire Database
  - No Server!
  - fragmented database
  - lots of empty space in the database file

Java/Android

http://developer.android.com/reference/android/database/sqlite/package-summary.html

package **android.database.sqlite** — Since: API Level 1

Interfaces

SQLiteCursorDriver	A driver for SQLiteCursors that is used to create them and gets notified by the cursors it creates on significant events in their lifetimes.
SQLiteDatabase.CursorFactory	Used to allow returning sub-classes of `Cursor` when calling query.
SQLiteTransactionListener	A listener for transaction events.

Classes

SQLiteClosable	An object created from a SQLiteDatabase that can be closed.
SQLiteCursor	A Cursor implementation that exposes results from a query on a `SQLiteDatabase`.
SQLiteDatabase	Exposes methods to manage a SQLite database.
SQLiteOpenHelper	A helper class to manage database creation and version management.
SQLiteProgram	A base class for compiled SQLite programs.
SQLiteQuery	A SQLite program that represents a query that reads the resulting rows into a CursorWindow.
SQLiteQueryBuilder	This is a convience class that helps build SQL queries to be sent to `SQLiteDatabase` objects.
SQLiteStatement	A pre-compiled statement against a `SQLiteDatabase` that can be reused.

Other DB connection techniques

- Open Database Connectivity
  - ODBC
  - JDBC
  - Each database may have their own set of connectors
  - http://dev.mysql.com/downloads/connector/odbc/5.1.html
  - http://dev.mysql.com/downloads/connector/j/5.1.html