Fairfield University
DOLAN
School of Business

SQLite

Databases for Analytics

Basic SQLite Usage

Learning Objectives

- **Skills:** You should know how to ...
  - Use the `sqlite3` command line interpretter (CLI)
  - Create a SQLite database from scratch from the CLI and from Python (Jupyter)
  - Write SQLite-flavored SQL scripts
  - Populate the database from MySQL or CSV files
- **Theory:** You should be able to explain ...
  - SQLite's in-memory database mode
  - The pros and cons of SQLite for analytics

SQLite from the Command Line

The old-school way to use SQLite

The `sqlite3` Interpreter

While SQLite is not a full-featured DBMS, the `sqlite3` interpreter does many of the same functions.

- Create a new database / Open an existing one
- Issue SQL DDL and DML
- Inspect database schema (tables, indexes, etc.)
- Dump/load to/from SQL files

CLI Docs: https://www.sqlite.org/cli.html

Starting `sqlite3`

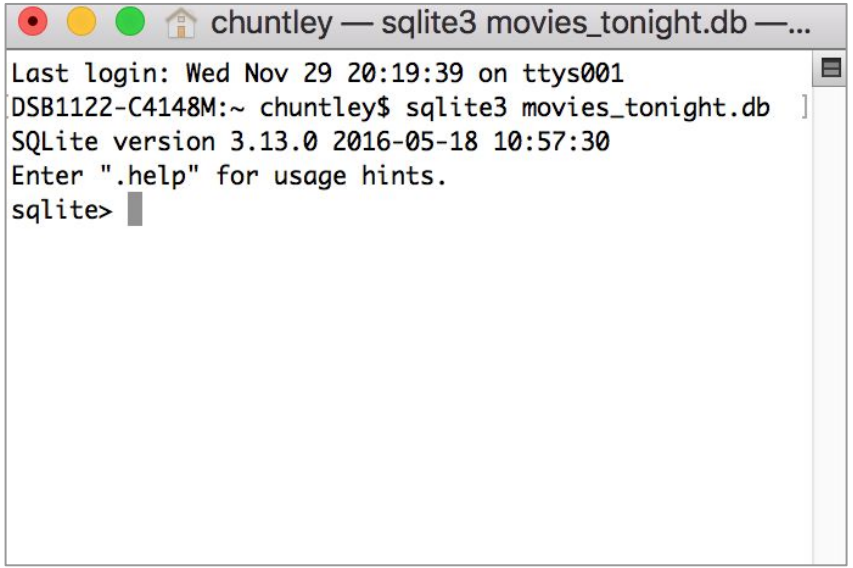
`sqlite3` is included in Anaconda!

Step 1. Open Anaconda from the command line:

- Macs: just open up Terminal
- Windows: open Anaconda Prompt

Step 2. Type `sqlite3` ***database.db***

- ***database.db*** is just a filename
- Creates the file if needed
- If no filename, then the started in memory mode



In-Memory vs File Storage

SQLite is designed to be very lightweight, able to work on even the smallest devices.

In fact, it does not need need a proper file system! We can use SQLite in memory, with any permanent storage at all.

Just fire up `sqlite3` without specifying a filename.

Dot Commands

Admin functions are available using dot commands. To get a list just use the **`.help`** command.

Note that the **`.exit`** command is used to close `sqlite3`.

chuntley — sqlite3 movies_tonight.db — 82×35

```
sqlite> .help
.auth ON|OFF            Show authorizer callbacks
.backup ?DB? FILE       Backup DB (default "main") to FILE
.bail on|off            Stop after hitting an error.  Default OFF
.binary on|off          Turn binary output on or off.  Default OFF
.changes on|off         Show number of rows changed by SQL
.clone NEWDB            Clone data into NEWDB from the existing database
.databases              List names and files of attached databases
.dbinfo ?DB?            Show status information about the database
.dump ?TABLE? ...       Dump the database in an SQL text format
                          If TABLE specified, only dump tables matching
                          LIKE pattern TABLE.
.echo on|off            Turn command echo on or off
.eqp on|off|full        Enable or disable automatic EXPLAIN QUERY PLAN
.exit                   Exit this program
.explain ?on|off|auto?  Turn EXPLAIN output mode on or off or to automatic
.fullschema ?--indent?  Show schema and the content of sqlite_stat tables
.headers on|off         Turn display of headers on or off
.help                   Show this message
.import FILE TABLE      Import data from FILE into TABLE
.indexes ?TABLE?        Show names of all indexes
                          If TABLE specified, only show indexes for tables
                          matching LIKE pattern TABLE.
.limit ?LIMIT? ?VAL?    Display or change the value of an SQLITE_LIMIT
.load FILE ?ENTRY?      Load an extension library
.log FILE|off           Turn logging on or off.  FILE can be stderr/stdout
.mode MODE ?TABLE?      Set output mode where MODE is one of:
                          ascii    Columns/rows delimited by 0x1F and 0x1E
                          csv      Comma-separated values
                          column   Left-aligned columns.  (See .width)
                          html     HTML <table> code
                          insert   SQL insert statements for TABLE
                          line     One value per line
                          list     Values delimited by .separator strings
                          tabs     Tab-separated values
```

Inspecting Database Schema

`.tables`

- Lists all the tables in the database
- Equivalent to the MySQL show tables command

`.indexes`

`.indexes` *tablename*

- Can be used to see every PK, FK, or other index in a table

`.schema`

`.schema` *tablename*

- Shows the CREATE TABLE DDL for one or more tables

PRAGMA

If you need to inspect tables or other SQLite metadata programmatically (e.g., in a Jupyter Notebook), use the special `PRAGMA` pseudo-SQL statement.

```
PRAGMA table_info(tablename);
PRAGMA foreign_key_list(tablename);
```

Docs: https://www.sqlite.org/pragma.html

CSV Import/Export

Importing from a CSV file:

```
sqlite> .mode csv
sqlite> .import filename.csv tablename
```

If the table already exists then the columns headers are treated as data. It's usually best to make sure the table is empty.

Exporting to a CSV file:

```
sqlite> .header on
sqlite> .mode csv
sqlite> .once filename.csv
sqlite> SELECT * FROM tablename;
```

You'll likely want to set `.mode list` afterwards.

`.once` is used to direct query output to the file one time and then switch output back to the screen.

Dumping Loading the Database

`.dump`

- Need to call **`.once`** first to direct the dump to a file.
- The SQL DDL is just a direct copy of whatever was typed in, capitalization and spacing included.

`.read` *filename*

- Loads and executes the SQL in the file against the current database.
- Works best with file produced by the **.dump** command

SQLite Quirks

Generally standard SQL but just not as full-featured

Source: https://www.sqlite.org/lang.html

SQL Support

- Core SQL syntax (CREATE TABLE, DROP TABLE, INSERT, UPDATE DELETE, SELECT, etc.) is pretty much standard
- Data types are **very** limited (to native Python types)
  - TEXT, INTEGER, FLOAT, BLOB, NULL
  - Functions are required to implement more complex types
- ALTER TABLE is also very limited
  - Can only rename tables and add columns
- JOINS don't include RIGHT JOIN

Primary Keys

SQLite has a fairly unique approach to primary keys that works a lot like indexes in Pandas DataFrames

- Every table has a unique index called **`rowid`**.
- The **`rowid`** index works like we'd expect, numbering the rows starts with 1, 2, 3.
- To set a **surrogate key** we then just set the column definition to **INTEGER PRIMARY KEY**.
  - The column then becomes an alias for **`rowid`**
- Non-integer PKs are defined as usual, however.

SELECT and rowid

Take care when using `*` in **`SELECT`** queries. The **`rowid`** index is not considered a column unless you explicitly select it by name.

```
SELECT * FROM …
SELECT rowid, * FROM …
```

No GRANT, REVOKE, etc.

"Since SQLite reads and writes an ordinary disk file, the only access permissions that can be applied are the normal file access permissions of the underlying operating system."

So, we don't need a username or password to access the database!

Very Poor Multiuser Support

From the SQLAlchemy docs:

> "SQLite is not designed for a high level of write concurrency. The database itself, being a file, is locked completely during write operations within transactions, meaning exactly one 'connection' (in reality a file handle) has exclusive access to the database during this period - all other “connections” will be blocked during this time."

Basically, it operates as a single-user database.

SQLite, SQLAlchemy, and Jupyter

Since they are all Python-native, they play very nicely together

Stacked Architecture

Each layer is built on top of the ones below it, providing greater convenience and ease of use …

But you can always go deeper if needed.

IPython-SQL (**`%sql`**)

SqlAlchemy

Python 3 (includes **`sqlite3`**)

Connection Strings

Since SQLite does not have username or passwords and the database file is always on the local computer, the usual SQLAlchemy connection string reduces to

`sqlite:///`*filename*

Note that that's ***3 slashes*** before the *filename*.

Expanded Data Type Support

SQLAlchemy expands the list of supported data types:

> **BLOB**, BOOLEAN, CHAR, DATE, DATETIME, DECIMAL, **FLOAT**, **INTEGER**, **NULL**, NUMERIC, SMALLINT, **TEXT**, TIME, TIMESTAMP, VARCHAR

In some cases it uses SQLite's own type conversion functions. In others (`DATE, DATETIME,TIME`) it uses it's own custom conversion functions.

All storage is still in the five native types, however.

TL;DR: Use `%sql` where you can

As long as we abide by the minor SQL dialect differences, SQLite queries in Jupyter work pretty much the same as they did with MySQL, only simpler.

```
%load_ext sql
%sql sqlite:///deals.db
c = %sql SELECT * FROM COMPANIES
companies = c.DataFrame()
```

Fairfield University
DOLAN
School of Business

SQLite

Databases for Analytics

Basic SQLite Usage