Fairfield University
DOLAN
School of Business

SQLite

Databases for Analytics

SQLite Quirks

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

Learning Objectives

- **Skills:** You should know how to ...
  - Use SQLite for for lightweight RDBMS needs
  - Work around some of the functional omissions needed to keep SQLite so lightweight
- **Theory:** You should be able to explain ...
  - The limitations of SQLite compared with full-featured SQL server

SQL Support … is barely enough

- 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 1, 2, 3, ...
- To set a proper **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!

Connection Strings (again)

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:///`*filepath*

Note that that's ***3 slashes*** before the *filepath*. If the file is in the same folder as the notebook then the *filepath* is just the *filename*.

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.

So why use SQLite again?

SQLite is great for offline data analysis:

1. Extract data you need from a live, online database directly into normalized tables in a local SQLite database with *no loss of structure*.
2. Analyze the data without worrying about it changing out from under you. Pandas supports SQLite natively, converting to/from DataFrames as needed. You don't even have to `import sqlite`.
3. Publish your work to a repository (e.g., Git or Kaggle) for further analysis. No data import is needed.

Fairfield University
DOLAN
School of Business

SQLite

Databases for Analytics

SQLite Quirks

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