Theory & Practice of Data Cleaning

Introduction to SQL(ite)

Back to …
Using Databases and SQL(ite)!

- Why database research? Why SQL(ite)?
  - Vast amount of literature and results
    - Conceptual modeling, DB normalization, data integration, data clean(s)ing, logic, integrity constraints, repair, …
    - The world runs on databases
    - Intellectually satisfying and gratifying
      - YMMV (your mileage my vary); may take a while to appreciate
    - SQL(ite) rocks!
      - Unpopular advice?
        - » Stop goofing around with XML, RDF, JSON, …
      - … and get the darn job done with SQLite!
      - … just like these guys: https://www.sqlite.org/famous.html

… even after OpenRefine, Kurator, Python data cleaning workflows, "dirty data" can make it into our database tables …

Id	Name	DOB	Age	Sex	Phone	Zip	Email
43	Doe, Joe	1970-02-27	56	M	(999)-999-999	94102	
43	Jane Dunbar	1.1.1990	26	W	NULL	61820	jdunbar@foobar.com
27	Joe Doe	2/30/70	46	F	+1-530-777-1234	D-6951	joe.doe@gargle.edu

ADDRESS

ZIP	City	State
94102	San Franzisco	CA
61821	Champagne	IL
D-6951	Obrigheim	Deutschland

- Errors and IC Violations:
  - Uniqueness (primary key) violation
  - Different representations & formats
  - Contradictions
  - Incorrect values (typos, domain, …)
  - Duplicates
  - Referential Integrity (FK → PK)
    - PERSON.ZIP → ADDRESS.ZIP
  - Incompleteness
  - …

Databases are ubiquitous!

Databases are ubiquitous: SQL Error on a CTA Bus in Chicago

Photo by Prof. Boris Glavic, IIT Chicago

For more on the specific SQL error see here:
https://stackoverflow.com/questions/24225164/sqlstatehy000-1040-too-many-connections

About SQLite

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. Think of SQLite not as a replacement for Oracle but as a replacement for fopen()

SQLite is a compact library. With all features enabled, the library size can be less than 500KiB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger.) If optional features are omitted, the size of the SQLite library can be reduced below 300KiB. SQLite can also be made to run in minimal stack space (4KiB) and very little heap (100KiB), making SQLite a popular database engine choice on memory constrained gadgets such as cellphones, PDAs, and MP3 players. There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments.

See Also...

- Features
- When to use SQLite
- Frequently Asked Questions
- Well-known Users
- Books About SQLite
- Getting Started
- SQL Syntax
  - Pragmas
  - SQL functions
  - Date & time functions
  - Aggregate functions
- C/C++ Interface Spec
  - Introduction
  - List of C-language APIs
- The TCL Interface Spec
- Development Timeline
- Report a Bug

SQLite is very carefully tested prior to every release and has a reputation for being very reliable. Most of the SQLite source code is devoted purely to testing and verification. An automated test suite runs millions and millions of test cases involving hundreds of millions of individual SQL statements and achieves 100% branch test coverage. SQLite responds gracefully to memory allocation failures and disk I/O errors. Transactions are ACID even if interrupted by system crashes or power failures. All of this is verified by the automated tests using special test harnesses which simulate system failures. Of course, even with all this testing, there are still bugs. But unlike some similar projects (especially commercial competitors) SQLite is open and honest about all bugs and provides bugs lists and minute-by-minute chronologies of bug reports and code changes.

The SQLite code base is supported by an international team of developers who work on SQLite full-time. The developers continue to expand the capabilities of SQLite and enhance its reliability and performance while maintaining backwards compatibility with the published interface spec, SQL syntax, and database file format. The source code is absolutely free to anybody who wants it, but professional support is also available.

We the developers hope that you find SQLite useful and we charge you to use it well: to make good and beautiful products that are fast, reliable, and simple to use. Seek forgiveness for yourself as you forgive others. And just as you have received SQLite for free, so also freely give, paying the debt forward.

When SQLite is for you ..

Appropriate Uses For SQLite

SQLite is not directly comparable to client/server SQL database engines such as MySQL, Oracle, PostgreSQL, or SQL Server since SQLite is trying to solve a different problem.

Client/server SQL database engines strive to implement a shared repository of enterprise data. They emphasis scalability, concurrency, centralization, and control. SQLite strives to provide local data storage for individual applications and devices. SQLite emphasizes economy, efficiency, reliability, independence, and simplicity.

SQLite does not compete with client/server databases. SQLite competes with fopen().

Situations Where SQLite Works Well

… and Software Carpentry has nice lessons for it :-)

- **Embedded devices and the internet of things**

  Because an SQLite database requires no administration, it works well in devices that must operate without expert human support. SQLite is a good fit for use in cellphones, set-top boxes, televisions, game consoles, cameras, watches, kitchen appliances, thermostats, automobiles, machine tools, airplanes, remote sensors, drones, medical devices, and robots: the "internet of things".

  Client/server database engines are designed to live inside a lovingly-attended datacenter at the core of the network. SQLite works there too, but SQLite also thrives at the edge of the network, fending for itself while providing fast and reliable data services to applications that would otherwise have dodgy connectivity.

- **Application file format**

  SQLite is often used as the on-disk file format for desktop applications such as version control systems, financial analysis tools, media cataloging and editing suites, CAD packages, record keeping programs, and so forth. The traditional File/Open operation calls sqlite3_open() to attach to the database file. Updates happen automatically as application content is revised so the File/Save menu option becomes superfluous. The File/Save_As menu option can be implemented using the backup API.

  There are many benefits to this approach, including improved application performance, reduced cost and complexity, and improved reliability. See technical notes here and here for details.

- **Websites**

  SQLite works great as the database engine for most low to medium traffic websites (which is to say, most websites). The amount of web traffic that SQLite can handle depends on how heavily the website uses its database. Generally speaking, any site that gets fewer than 100K hits/day should work fine with SQLite. The 100K hits/day figure is a conservative estimate, not a hard upper bound. SQLite has been demonstrated to work with 10 times that amount of traffic.

  The SQLite website (https://www.sqlite.org/) uses SQLite itself, of course, and as of this writing (2015) it handles about 400K to 500K HTTP requests per day, about 15-20% of which are dynamic pages touching the database. Each dynamic page does roughly 200 SQL statements. This setup runs on a single VM that shares a physical server with 23 others and yet still keeps the load average below 0.1 most of the time.

- **Data analysis**

  People who understand SQL can employ the sqlite3 command-line shell (or various third-party SQLite access programs) to analyze large datasets. Raw data can be imported from CSV files, then that data can be sliced and diced to generate a myriad of summary reports. More complex analysis can be done using simple scripts written in Tcl or Python (both of which come with SQLite built-in) or in R or other languages using readily available adaptors. Possible uses include website log analysis, sports statistics analysis, compilation of programming metrics, and analysis of experimental results. Many bioinformatics researchers use SQLite in this way.

  The same thing can be done with an enterprise client/server database, of course. The advantage of SQLite is that it is easier to install and use and the resulting database is a single file that can be written to a USB memory stick or emailed to a colleague.

- **Cache for enterprise data**

  Many applications use SQLite as a cache of relevant content from an enterprise RDBMS. This reduces latency, since most queries now occur against the local cache and avoid a network round-trip. It also reduces the load on the network and on the central database server. And in many cases, it means that the client-side application can continue operating during network outages.

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

SQLite

Small. Fast. Reliable.
Choose any three.

About Sitemap Documentation
Download License News Support

Search SQLite Docs... Go

SQLite As An Application File Format

Executive Summary

An SQLite database file with a defined schema often makes an excellent application file format. Here are a dozen reasons why this is so:

1. Simplified Application Development
2. Single-File Documents
3. High-Level Query Language
4. Accessible Content
5. Cross-Platform
6. Atomic Transactions
7. Incremental And Continuous Updates
8. Easily Extensible
9. Performance
10. Concurrent Use By Multiple Processes
11. Multiple Programming Languages
12. Better Applications

https://www.sqlite.org/appfileformat.html

SQLite as an Application Format

- Fred Brooks, in his all-time best-selling computer science text, *The **Mythical Man-Month*** says:
  - *Representation is the essence of computer programming.*

    ...

    *Show me your flowcharts and conceal your tables, and I shall continue to be mystified. **Show me your tables**, and I won't usually need your flowcharts; they'll be obvious.*
- Rob Pike, in his ***Rules of Programming*** expresses the same idea this way:
  - *Data dominates. If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident. Data structures, not algorithms, are central to programming.*
- Linus Torvalds used different words to say much the same thing on the Git mailing list on 2006-06-27:
  - *Bad programmers worry about the code. Good programmers worry about data structures and their relationships.*

https://www.sqlite.org/appfileformat.html

Found In...

- Every Android phone and device
- Every iPhone and iOS device
- Every Mac
- Every Firefox, Chrome, and Safari browser
- Every installation of iTunes, Skype, Dropbox, QuickBooks, TurboTax
- Python and PHP
- Most TV sets and set-top cable boxes
- Automotive multimedia systems
- Countless millions of other applications....

Source: Dr. Richard Hipp, creator of SQLite

SQLite By The Numbers

- 1 file of C89 code: “sqlite3.c”
- 3 developers
- 15 years in development
- 100% MC/DC testing
- 22,000 branch opcodes
- 94,000 SLOC
- 500,000 bytes compiled
- 1,000,000 applications use it
- 177,729,556 tests
- 2,000,000,000 new installs during 2014

Source: Dr. Richard Hipp, creator of SQLite

SQLite Features

- Well-defined Cross-platform File Format
- Power-safe Transactions
- Triggers
- Foreign Keys
- Full-Text Indexes
- R-Tree Indexes
- Recursive Common Table Expressions
- Partial Indexes

Source: Dr. Richard Hipp, creator of SQLite

SQLite Limits

- 1 writer and N readers per database
- 64-way joins
- 1 gigabyte strings and/or BLOBs
- 140 terabytes per database
- 125 databases per connection

Source: Dr. Richard Hipp, creator of SQLite

Flexible Datatypes

- SMALL INT will actually hold any size integer.
- VARCHAR(5) will hold a 1-billion byte string
- No native DATETIME format:
  - ISO8601 text: '2015-02-12 14:23:51.142'
  - Julian day number: 2457068.09989748
  - Unix time: 1423923831.142
- Integers work as BOOLEAN, as in C/C++

Source: Dr. Richard Hipp, creator of SQLite

SQLite Date and Time formats

SQLite supports five date and time functions as follows:

1. **date(***timestring, modifier, modifier, ...***)**
2. **time(***timestring, modifier, modifier, ...***)**
3. **datetime(***timestring, modifier, modifier, ...***)**
4. **julianday(***timestring, modifier, modifier, ...***)**
5. **strftime(***format, timestring, modifier, modifier, ...***)**

All five date and time functions take a time string as an argument. The time string is followed by zero or more modifiers. The strftime() function also takes a format string as its first argument.

The date and time functions use a subset of IS0-8601 date and time formats. The date() function returns the date in this format: YYYY-MM-DD. The time() function returns the time as HH:MM:SS. The datetime() function returns "YYYY-MM-DD HH:MM:SS". The julianday() function returns the Julian day - the number of days since noon in Greenwich on November 24, 4714 B.C. (Proleptic Gregorian calendar). The strftime() routine returns the date formatted according to the format string specified as the first argument. The format string supports the most common substitutions found in the strftime() function from the standard C library plus two new substitutions, %f and %J. The following is a complete list of valid strftime() substitutions:

%d	day of month: 00
%f	fractional seconds: SS.SSS
%H	hour: 00-24
%j	day of year: 001-366
%J	Julian day number
%m	month: 01-12
%M	minute: 00-59
%s	seconds since 1970-01-01
%S	seconds: 00-59
%w	day of week 0-6 with Sunday==0
%W	week of year: 00-53
%Y	year: 0000-9999
%%	%

Notice that all other date and time functions can be expressed in terms of strftime():

Function	**Equivalent strftime()**
date(...)	strftime('%Y-%m-%d', ...)
time(...)	strftime('%H:%M:%S', ...)
datetime(...)	strftime('%Y-%m-%d %H:%M:%S', ...)
julianday(...)	strftime('%J', ...)

Source: Dr. Richard Hipp, creator of SQLite

software carpentry

Teaching basic lab skills for research computing

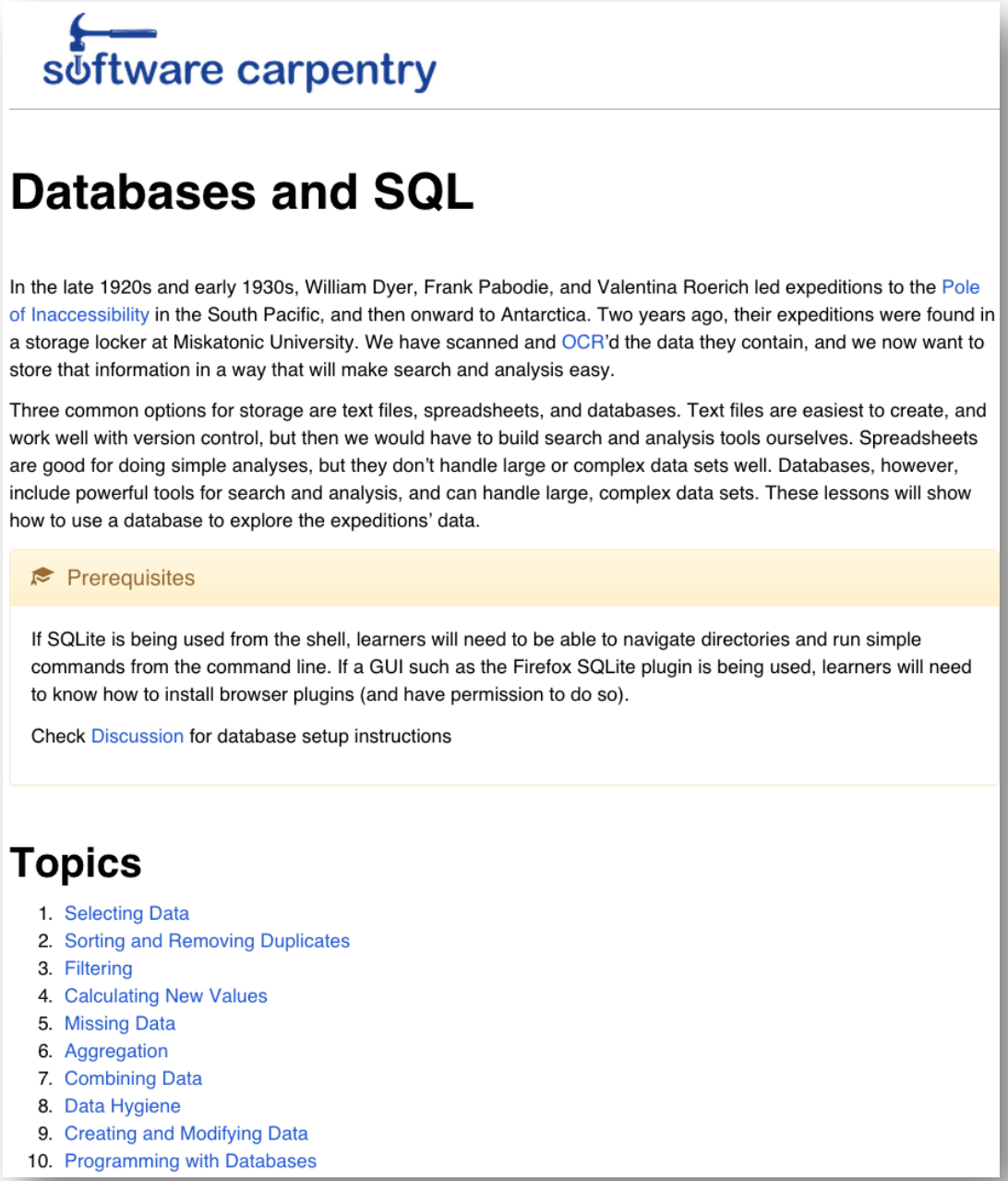
Our Lessons

Our lessons are developed collaboratively on GitHub. You can check the status of each lesson on our dashboard, view the nightly build, or look at older releases. You may also enjoy Data Carpentry's lessons, which focus on data organization, cleanup, analysis, and visualization.

Lesson	Site	Repository	Reference	Maintainer(s)
The Unix Shell				Gabriel Devenyi, Ashwin Srinath
Version Control with Git				Ivan Gonzalez, Daisie Huang
Version Control with Mercurial				Doug Latornell
Using Databases and SQL				Abigail Cabunoc Mayes, Sheldon McKay
Programming with Python				Trevor Bekolay, Valentina Staneva
Programming with R				Daniel Chen, Harriet Dashnow
R for Reproducible Scientific Analysis				Thomas Wright, Naupaka Zimmerman
Programming with MATLAB				Isabell Kiral-Kornek, Ashwin Srinath
Automation and Make				Mike Jackson

SQL(ite) at Software Carpentry

- … teaser now!
- Self-paced!



software carpentry

Databases and SQL

In the late 1920s and early 1930s, William Dyer, Frank Pabodie, and Valentina Roerich led expeditions to the Pole of Inaccessibility in the South Pacific, and then onward to Antarctica. Two years ago, their expeditions were found in a storage locker at Miskatonic University. We have scanned and OCR'd the data they contain, and we now want to store that information in a way that will make search and analysis easy.

Three common options for storage are text files, spreadsheets, and databases. Text files are easiest to create, and work well with version control, but then we would have to build search and analysis tools ourselves. Spreadsheets are good for doing simple analyses, but they don't handle large or complex data sets well. Databases, however, include powerful tools for search and analysis, and can handle large, complex data sets. These lessons will show how to use a database to explore the expeditions' data.

Prerequisites

If SQLite is being used from the shell, learners will need to be able to navigate directories and run simple commands from the command line. If a GUI such as the Firefox SQLite plugin is being used, learners will need to know how to install browser plugins (and have permission to do so).

Check Discussion for database setup instructions

Topics

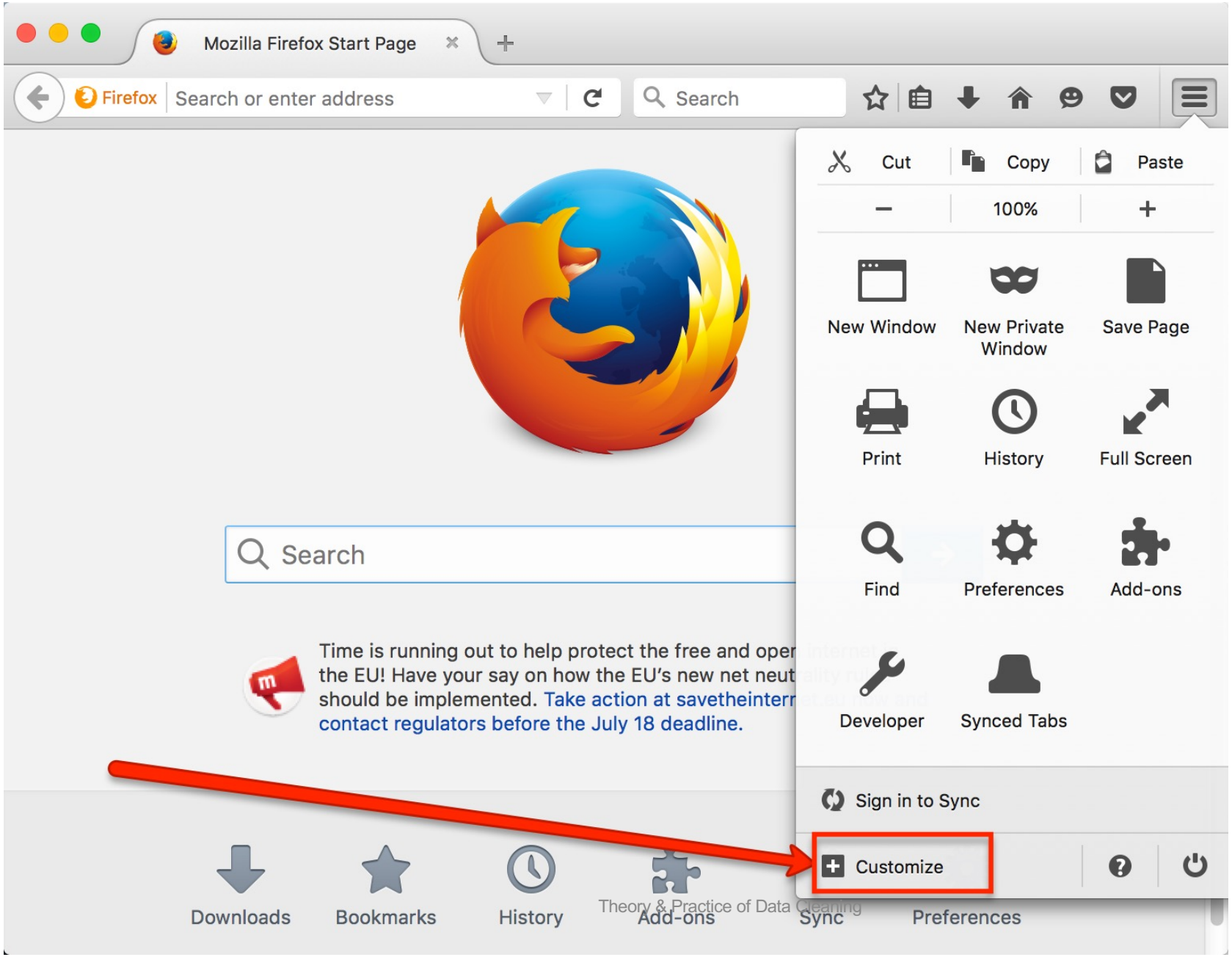
1. Selecting Data
2. Sorting and Removing Duplicates
3. Filtering
4. Calculating New Values
5. Missing Data
6. Aggregation
7. Combining Data
8. Data Hygiene
9. Creating and Modifying Data
10. Programming with Databases

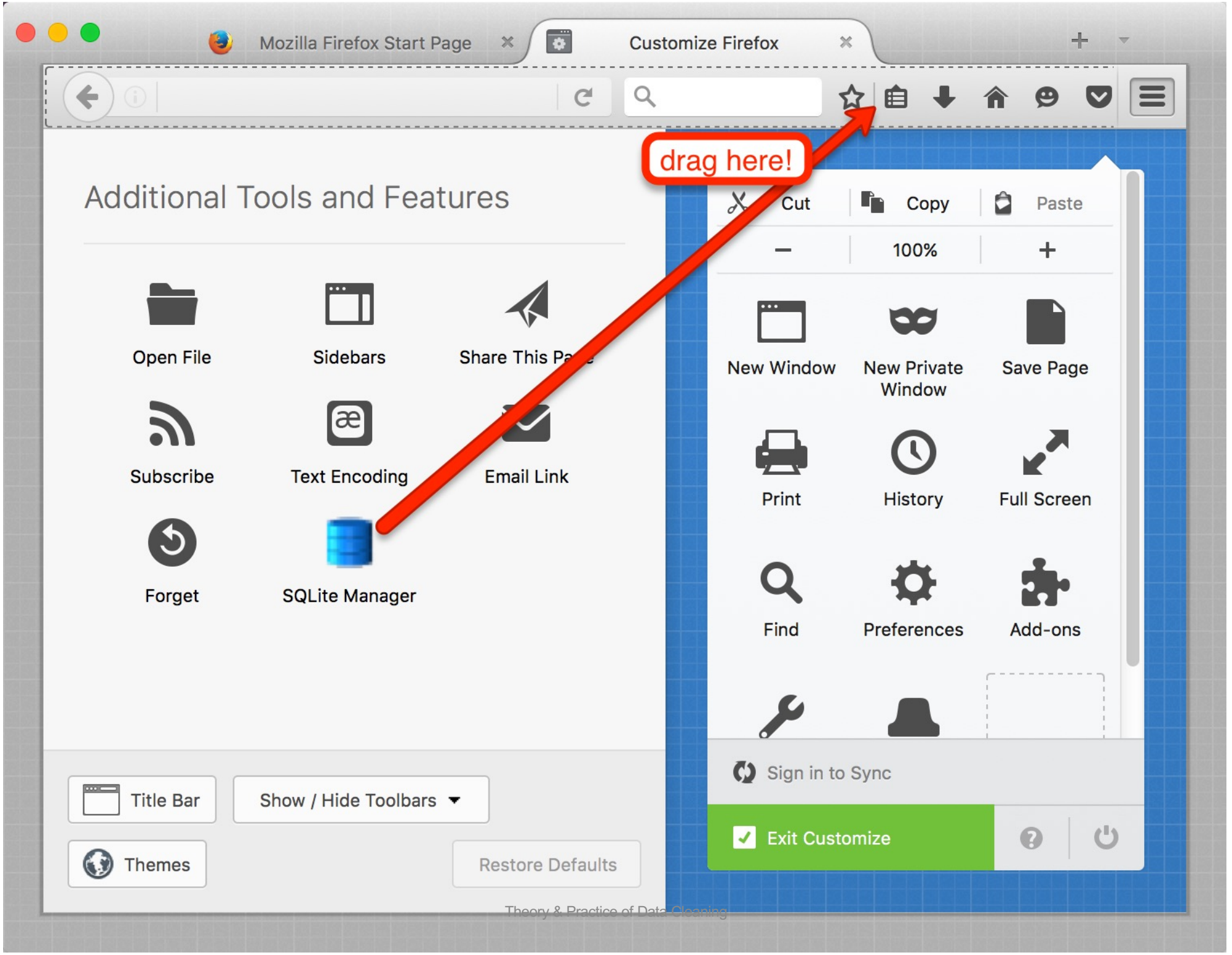
http://swcarpentry.github.io/sql-novice-survey

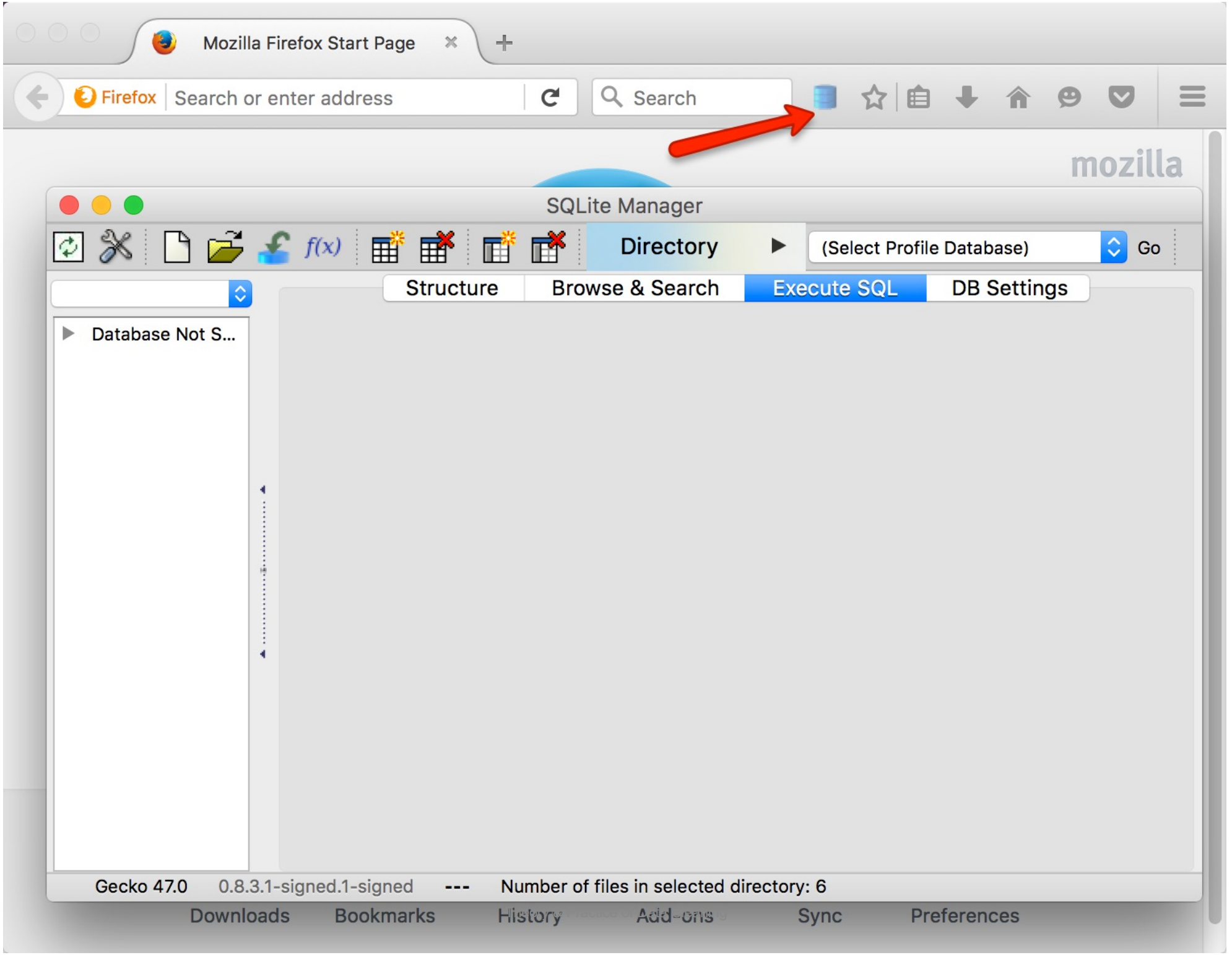
Software Carpentry: Databases & SQL

1. Selecting Data
2. Sorting and Removing Duplicates
3. Filtering
4. Calculating New Values
5. Missing Data
6. Aggregation
7. Combining Data
8. Data Hygiene
9. Creating and Modifying Data
10. Programming with Databases

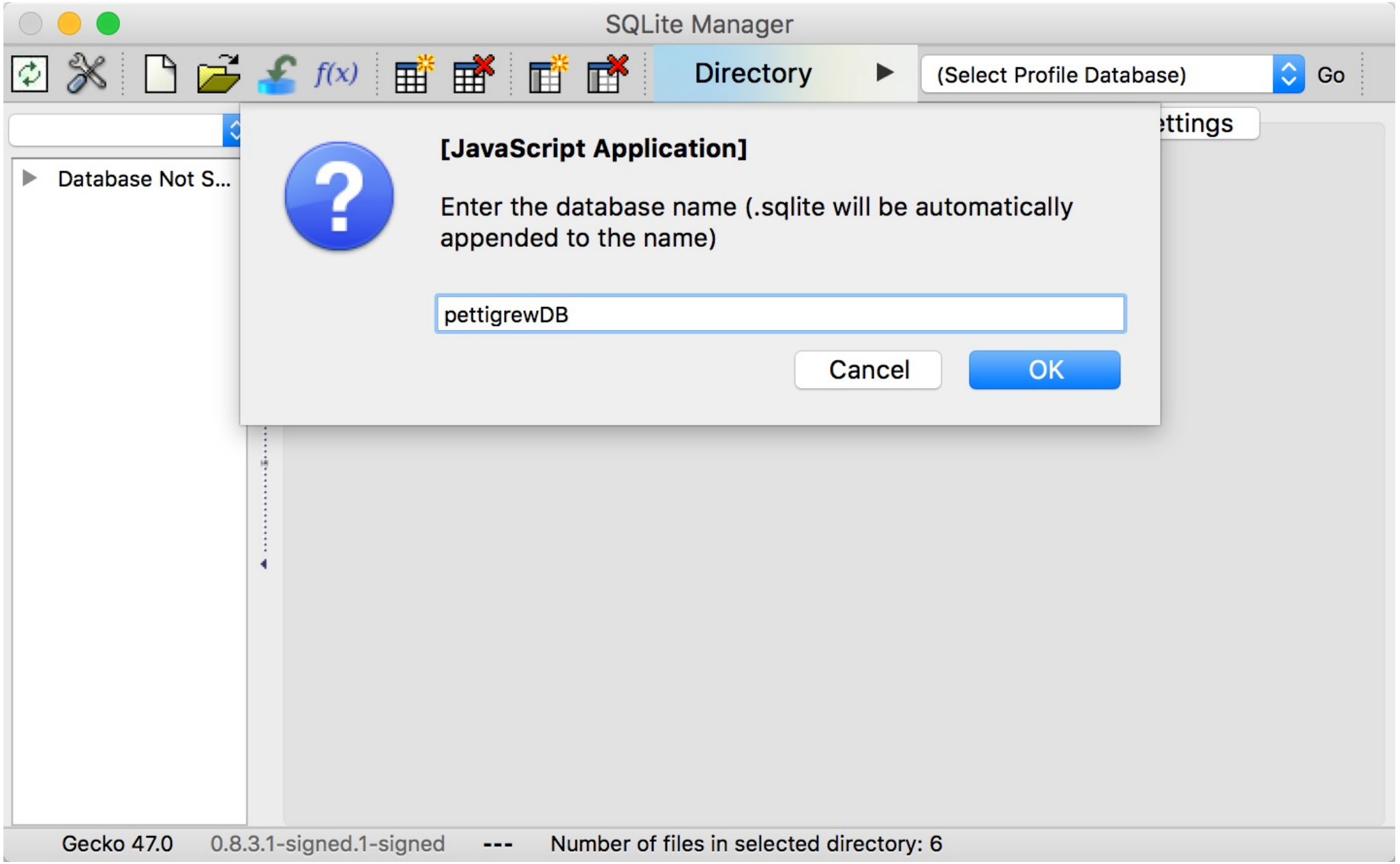
SQLite client as a Firefox extension



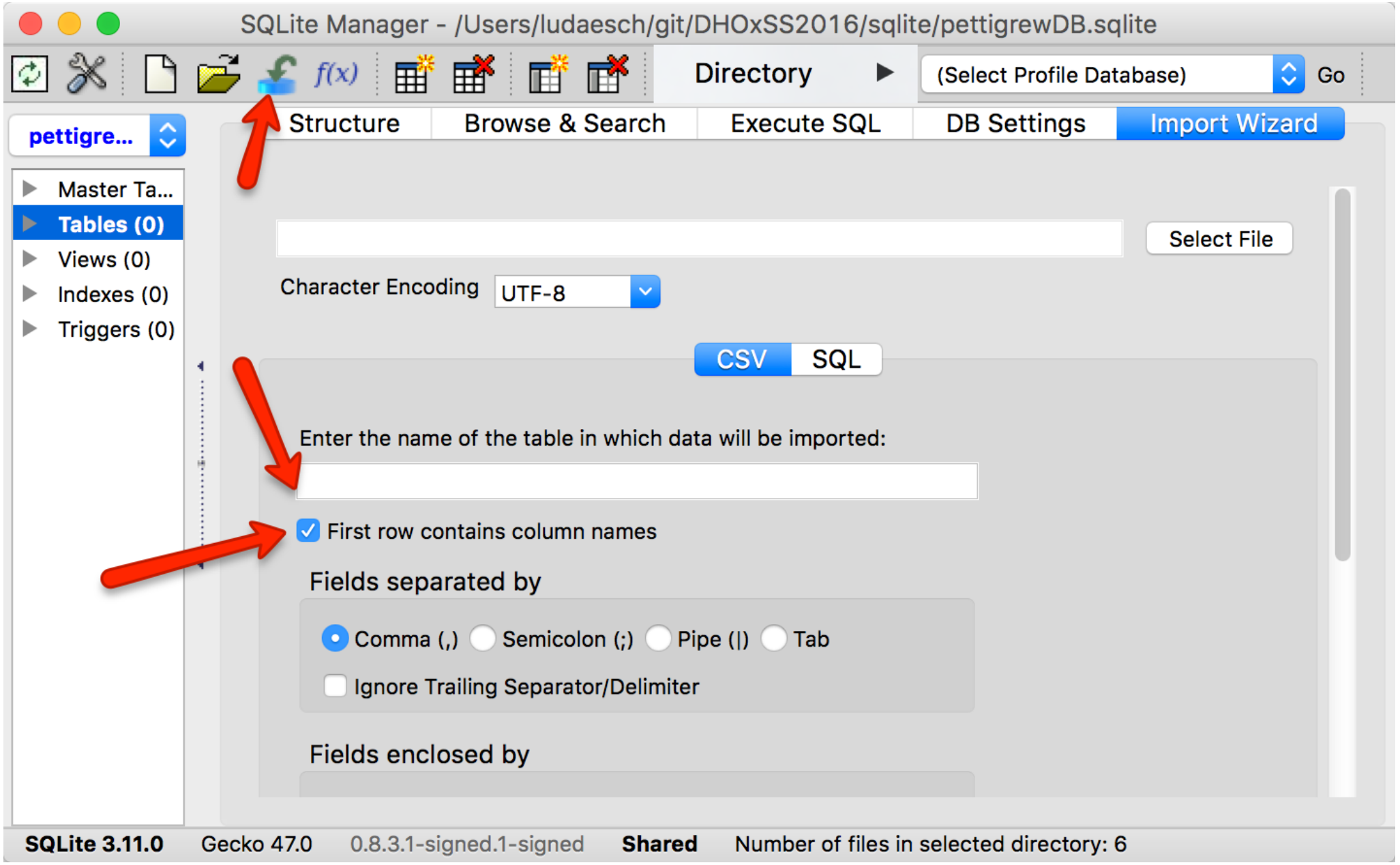

Mozilla Firefox Start Page
Customize Firefox
drag here!
Additional Tools and Features
Open File
Sidebars
Share This Page
Subscribe
Text Encoding
Email Link
Forget
SQLite Manager
Cut
Copy
Paste
100%
New Window
New Private Window
Save Page
Print
History
Full Screen
Find
Preferences
Add-ons
Sign in to Sync
Exit Customize
Title Bar
Show / Hide Toolbars
Themes
Restore Defaults
Theory & Practice of Data Cleaning

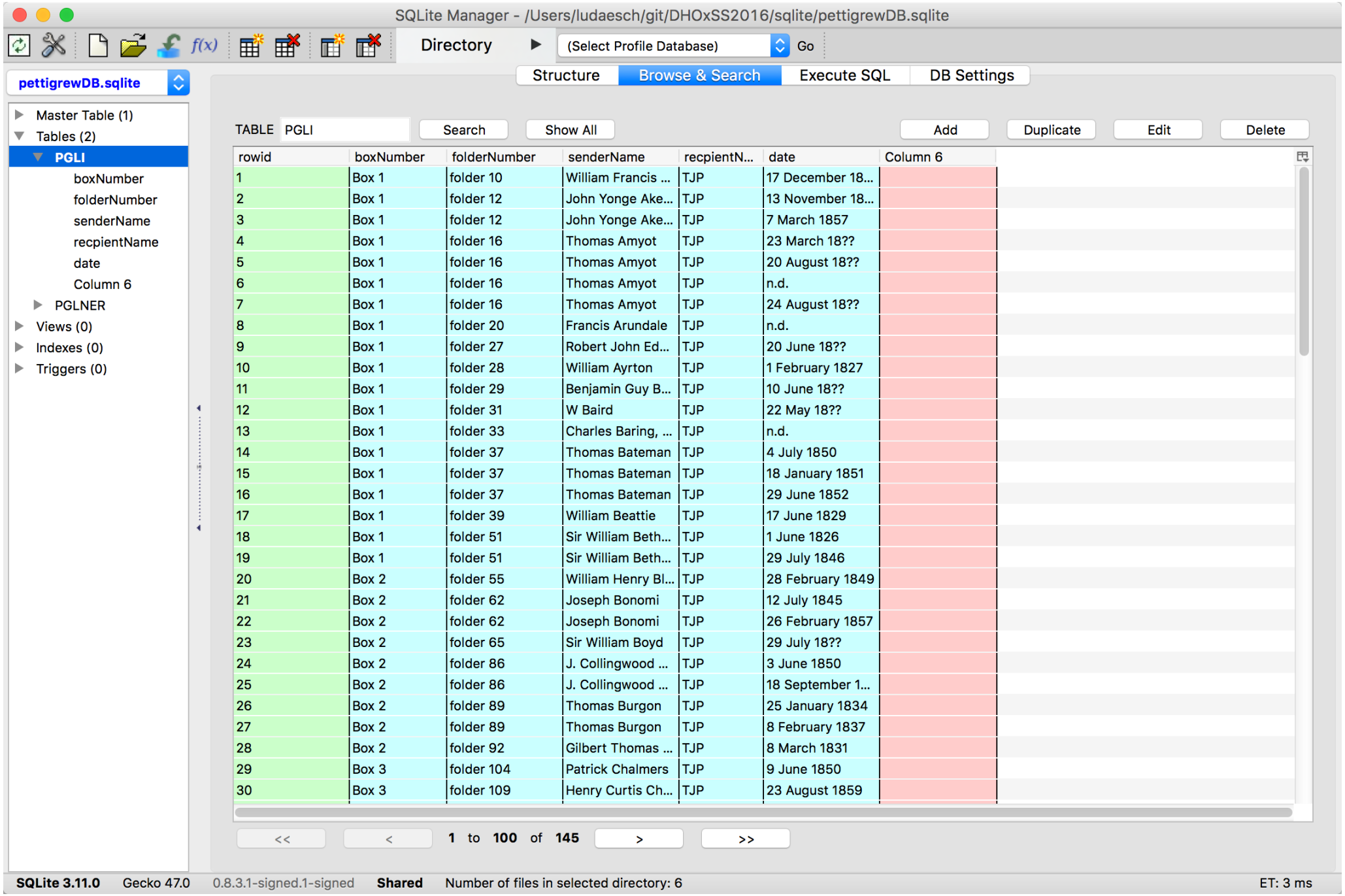

Mozilla Firefox Start Page
Firefox
Search or enter address
Search
mozilla
SQLite Manager
Directory
(Select Profile Database)
Go
Structure
Browse & Search
Execute SQL
DB Settings
Database Not S...
Gecko 47.0
0.8.3.1-signed.1-signed

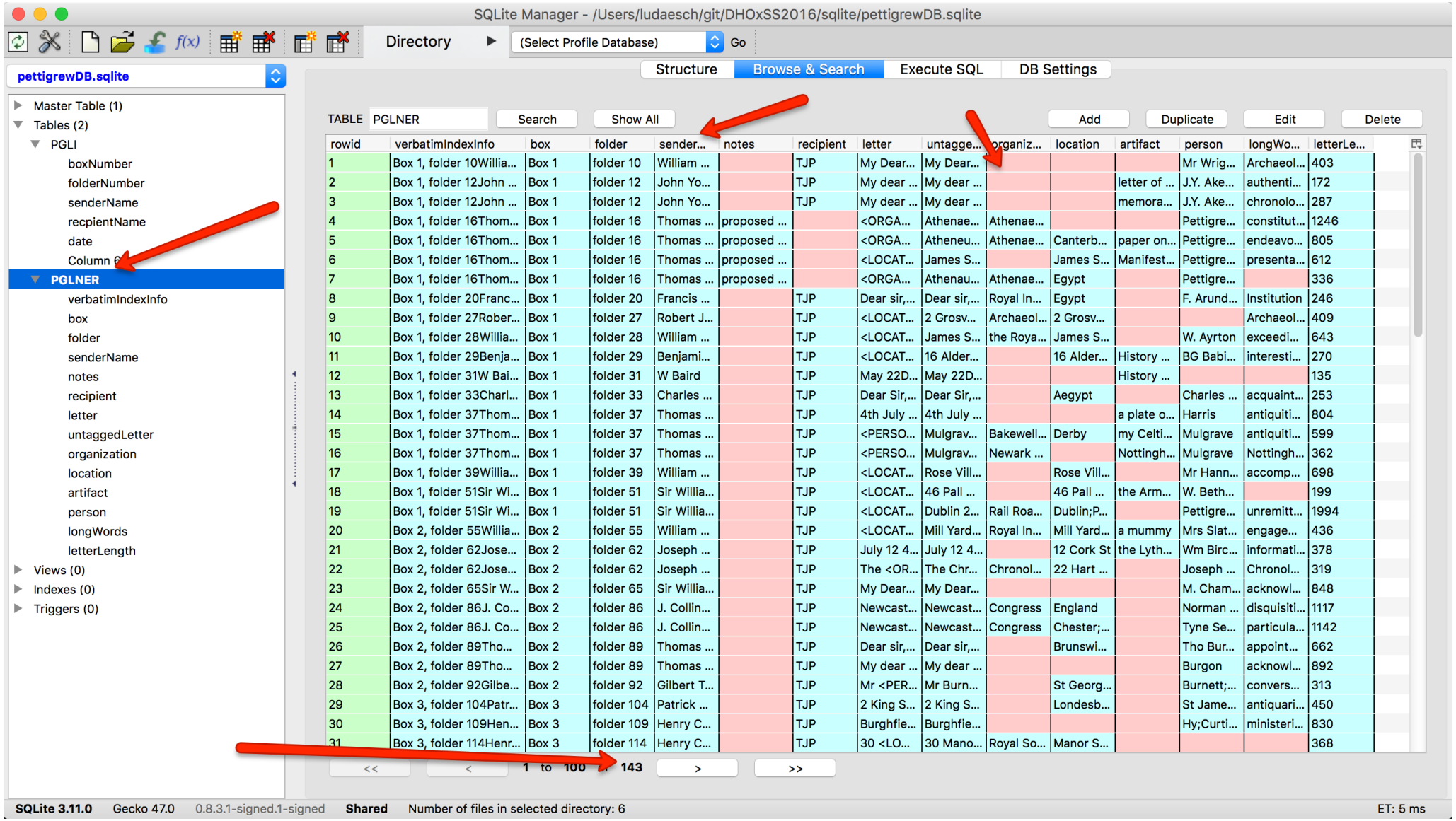
Number of files in selected directory: 6
Downloads
Bookmarks
History
Add-ons
Sync
Preferences

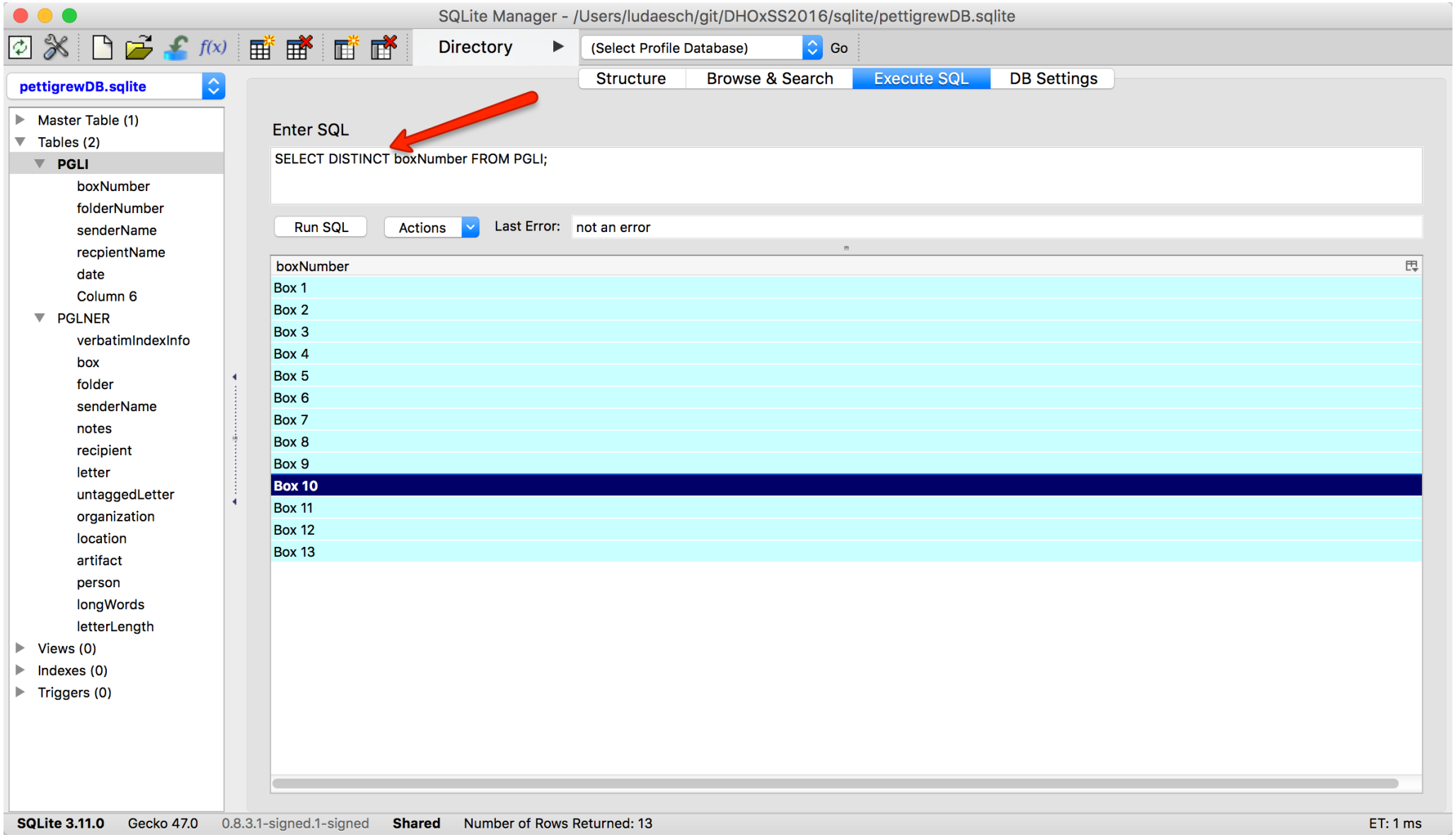

SQLite Manager
Directory
(Select Profile Database)
Go
Database Not S...
[JavaScript Application]
Enter the database name (.sqlite will be automatically appended to the name)
pettigrewDB
Cancel
OK
Gecko 47.0
0.8.3.1-signed.1-signed

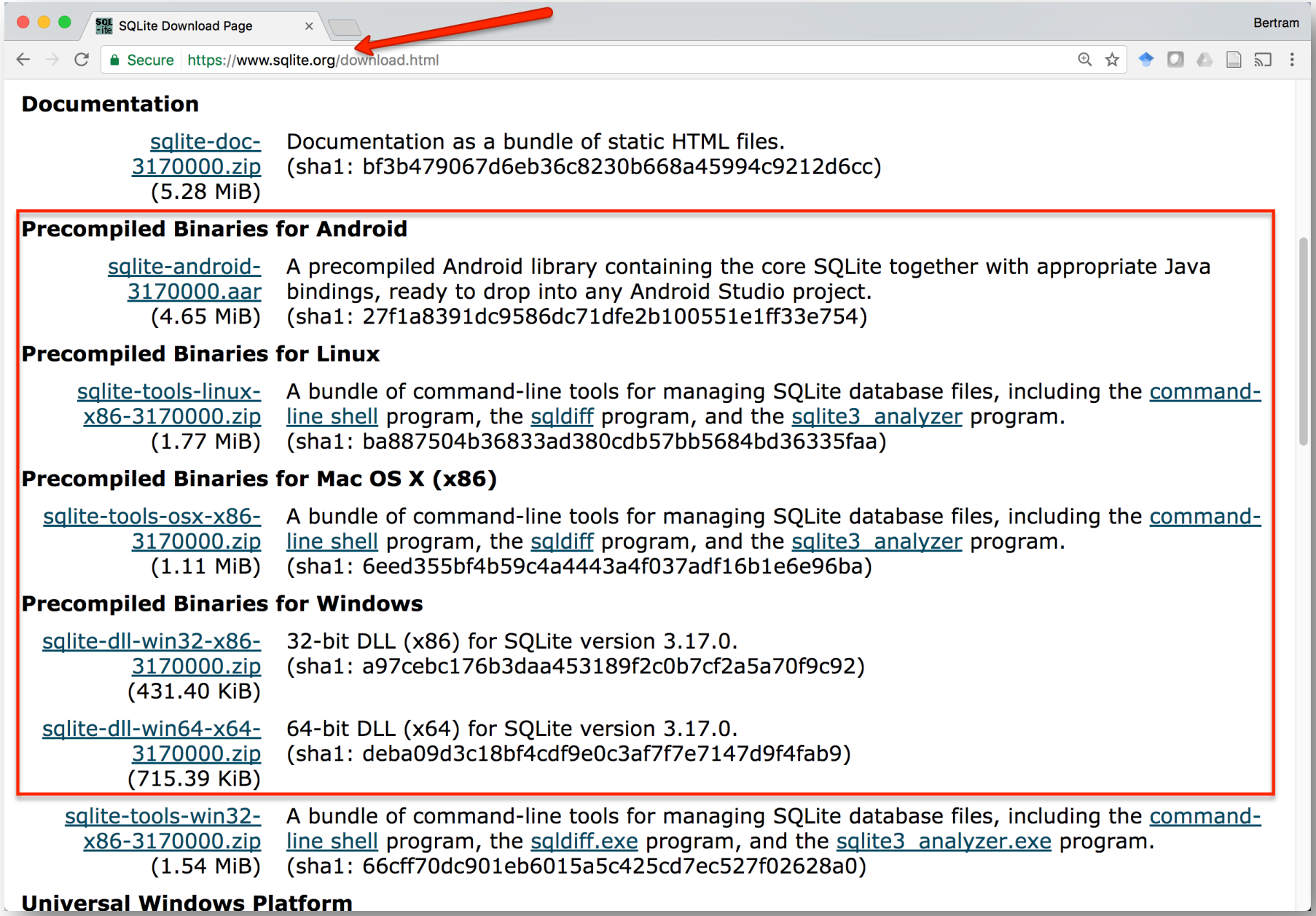
Number of files in selected directory: 6


SQLite Manager - /Users/ludaesch/git/DHOxSS2016/sqlite/pettigrewDB.sqlite
Directory
(Select Profile Database)
Go
pettigre...
Structure
Browse & Search
Execute SQL
DB Settings
Import Wizard
Master Ta...
Tables (0)
Views (0)
Indexes (0)
Triggers (0)
Select File
Character Encoding
UTF-8
CSV
SQL
Enter the name of the table in which data will be imported:
First row contains column names
Fields separated by
Comma (,)
Semicolon (;)
Pipe (|)
Tab
Ignore Trailing Separator/Delimiter
Fields enclosed by
SQLite 3.11.0
Gecko 47.0
0.8.3.1-signed.1-signed
Shared
Number of files in selected directory: 6

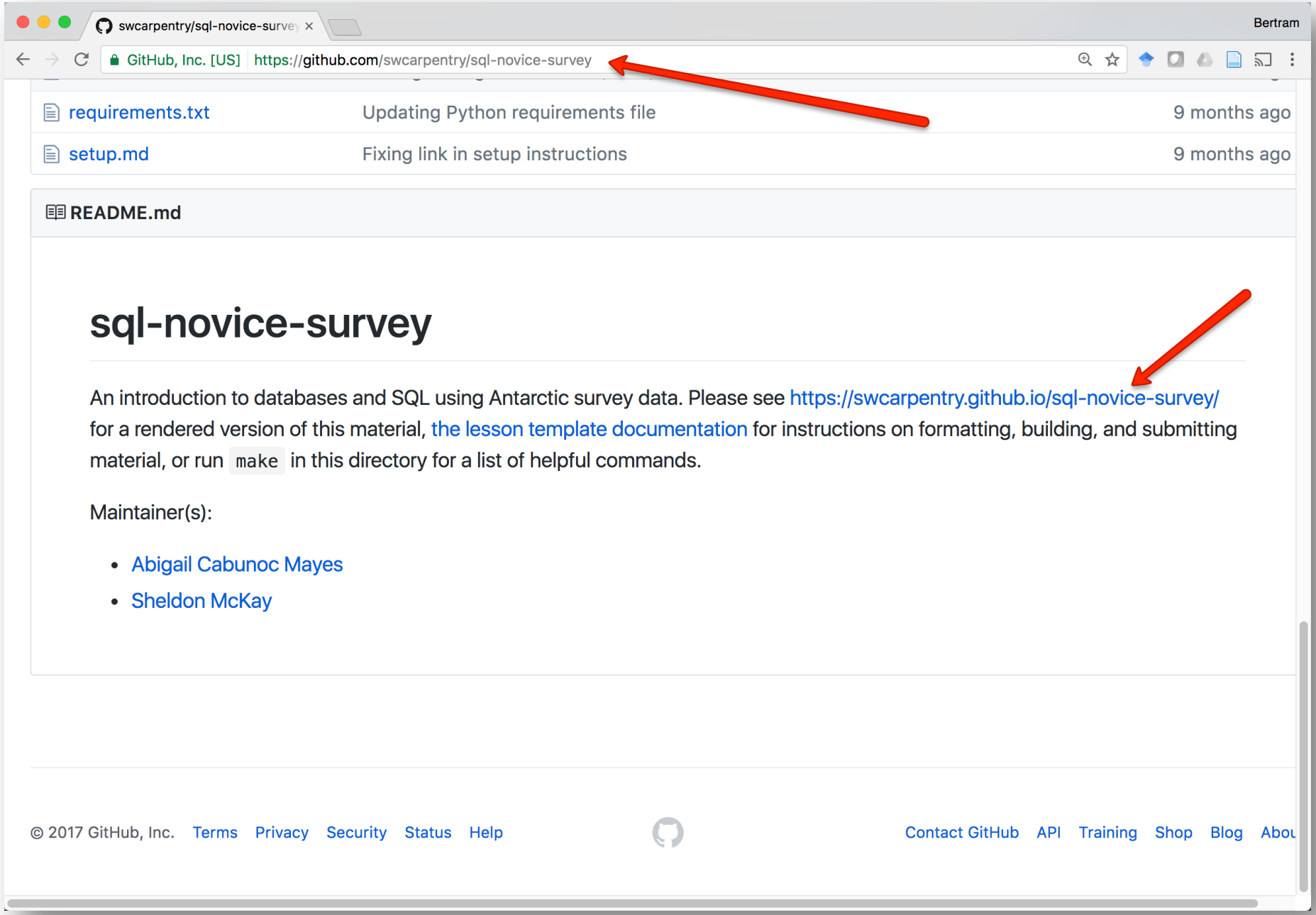

SQLite Manager - /Users/ludaesch/git/DHOxSS2016/sqlite/pettigrewDB.sqlite
Directory
(Select Profile Database)
Go
pettigrewDB.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (2)
PGLI
boxNumber
folderNumber
senderName
recpientName
date
Column 6
PGLNER
Views (0)
Indexes (0)
Triggers (0)
TABLE PGLI
Search
Show All
Add
Duplicate
Edit
Delete
rowid boxNumber folderNumber senderName recpientN... date Column 6
1 Box 1 folder 10 William Francis ... TJP 17 December 18...
2 Box 1 folder 12 John Yonge Ake... TJP 13 November 18...
3 Box 1 folder 12 John Yonge Ake... TJP 7 March 1857
4 Box 1 folder 16 Thomas Amyot TJP 23 March 18??
5 Box 1 folder 16 Thomas Amyot TJP 20 August 18??
6 Box 1 folder 16 Thomas Amyot TJP n.d.
7 Box 1 folder 16 Thomas Amyot TJP 24 August 18??
8 Box 1 folder 20 Francis Arundale TJP n.d.
9 Box 1 folder 27 Robert John Ed... TJP 20 June 18??
10 Box 1 folder 28 William Ayrton TJP 1 February 1827
11 Box 1 folder 29 Benjamin Guy B... TJP 10 June 18??
12 Box 1 folder 31 W Baird TJP 22 May 18??
13 Box 1 folder 33 Charles Baring, ... TJP n.d.
14 Box 1 folder 37 Thomas Bateman TJP 4 July 1850
15 Box 1 folder 37 Thomas Bateman TJP 18 January 1851
16 Box 1 folder 37 Thomas Bateman TJP 29 June 1852
17 Box 1 folder 39 William Beattie TJP 17 June 1829
18 Box 1 folder 51 Sir William Beth... TJP 1 June 1826
19 Box 1 folder 51 Sir William Beth... TJP 29 July 1846
20 Box 2 folder 55 William Henry Bl... TJP 28 February 1849
21 Box 2 folder 62 Joseph Bonomi TJP 12 July 1845
22 Box 2 folder 62 Joseph Bonomi TJP 26 February 1857
23 Box 2 folder 65 Sir William Boyd TJP 29 July 18??
24 Box 2 folder 86 J. Collingwood ... TJP 3 June 1850
25 Box 2 folder 86 J. Collingwood ... TJP 18 September 1...
26 Box 2 folder 89 Thomas Burgon TJP 25 January 1834
27 Box 2 folder 89 Thomas Burgon TJP 8 February 1837
28 Box 2 folder 92 Gilbert Thomas ... TJP 8 March 1831
29 Box 3 folder 104 Patrick Chalmers TJP 9 June 1850
30 Box 3 folder 109 Henry Curtis Ch... TJP 23 August 1859
<< < 1 to 100 of 145 > >>
SQLite 3.11.0 Gecko 47.0 0.8.3.1-signed.1-signed Shared Number of files in selected directory: 6 ET: 3 ms

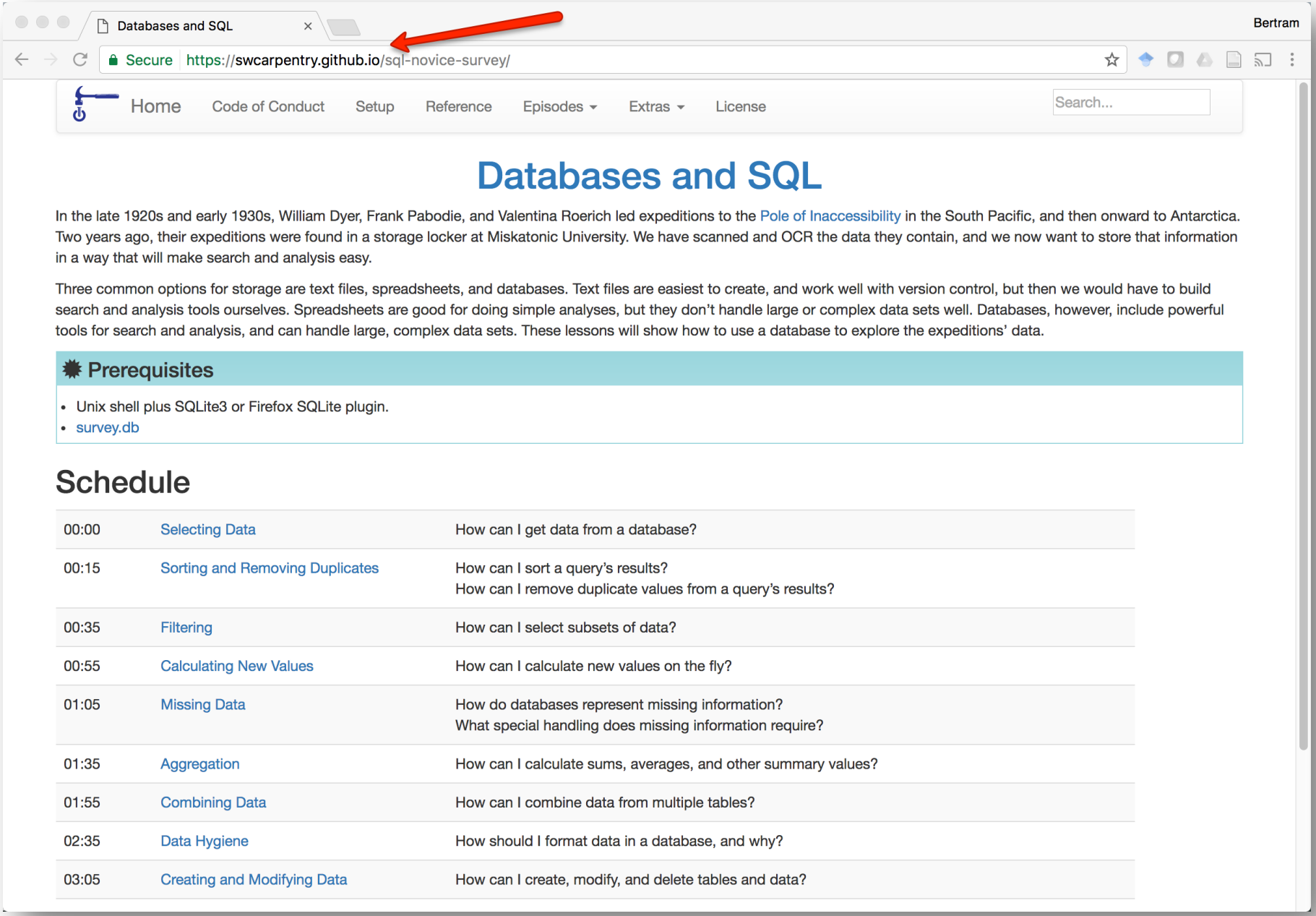

SQLite Manager - /Users/ludaesch/git/DHOxSS2016/sqlite/pettigrewDB.sqlite
Directory
(Select Profile Database)
Go
pettigrewDB.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (2)
PGLI
boxNumber
folderNumber
senderName
recpientName
date
Column 6
PGLNER
verbatimIndexInfo
box
folder
senderName
notes
recipient
letter
untaggedLetter
organization
location
artifact
person
longWords
letterLength
Views (0)
Indexes (0)
Triggers (0)
TABLE PGLNER
Search
Show All
Add
Duplicate
Edit
Delete
rowid | verbatimIndexInfo | box | folder | sender... | notes | recipient | letter | untagge... | organiz... | location | artifact | person | longWo... | letterLe...
1 | Box 1, folder 10Willia... | Box 1 | folder 10 | William ... | | TJP | My Dear... | My Dear... | | | | Mr Wrig... | Archaeol... | 403
2 | Box 1, folder 12John ... | Box 1 | folder 12 | John Yo... | | TJP | My dear ... | My dear ... | | | letter of ... | J.Y. Ake... | authenti... | 172
3 | Box 1, folder 12John ... | Box 1 | folder 12 | John Yo... | | TJP | My dear ... | My dear ... | | | memora... | J.Y. Ake... | chronolo... | 287
4 | Box 1, folder 16Thom... | Box 1 | folder 16 | Thomas ... | proposed ... | | <ORGA... | Athenae... | Athenae... | | | Pettigre... | constitut... | 1246
5 | Box 1, folder 16Thom... | Box 1 | folder 16 | Thomas ... | proposed ... | | <ORGA... | Atheneu... | Athenae... | Canterb... | paper on... | Pettigre... | endeavo... | 805
6 | Box 1, folder 16Thom... | Box 1 | folder 16 | Thomas ... | proposed ... | | <LOCAT... | James S... | | James S... | Manifest... | Pettigre... | presenta... | 612
7 | Box 1, folder 16Thom... | Box 1 | folder 16 | Thomas ... | proposed ... | | <ORGA... | Athenau... | Athenae... | Egypt | | Pettigre... | | 336
8 | Box 1, folder 20Franc... | Box 1 | folder 20 | Francis ... | | TJP | Dear sir,... | Dear sir,... | Royal In... | Egypt | | F. Arund... | Institution | 246
9 | Box 1, folder 27Rober... | Box 1 | folder 27 | Robert J... | | TJP | <LOCAT... | 2 Grosv... | Archaeol... | 2 Grosv... | | | Archaeol... | 409
10 | Box 1, folder 28Willia... | Box 1 | folder 28 | William ... | | TJP | <LOCAT... | James S... | the Roya... | James S... | | W. Ayrton | exceedi... | 643
11 | Box 1, folder 29Benja... | Box 1 | folder 29 | Benjami... | | TJP | <LOCAT... | 16 Alder... | | 16 Alder... | History ... | BG Babi... | interesti... | 270
12 | Box 1, folder 31W Bai... | Box 1 | folder 31 | W Baird | | TJP | May 22D... | May 22D... | | | History ... | | | 135
13 | Box 1, folder 33Charl... | Box 1 | folder 33 | Charles ... | | TJP | Dear Sir,... | Dear Sir,... | | Aegypt | | Charles ... | acquaint... | 253
14 | Box 1, folder 37Thom... | Box 1 | folder 37 | Thomas ... | | TJP | 4th July ... | 4th July ... | | | a plate o... | Harris | antiquiti... | 804
15 | Box 1, folder 37Thom... | Box 1 | folder 37 | Thomas ... | | TJP | <PERSO... | Mulgrav... | Bakewell... | Derby | my Celti... | Mulgrave | antiquiti... | 599
16 | Box 1, folder 37Thom... | Box 1 | folder 37 | Thomas ... | | TJP | <PERSO... | Mulgrav... | Newark ... | | Nottingh... | Mulgrave | Nottingh... | 362
17 | Box 1, folder 39Willia... | Box 1 | folder 39 | William ... | | TJP | <LOCAT... | Rose Vill... | | Rose Vill... | | Mr Hann... | accomp... | 698
18 | Box 1, folder 51Sir Wi... | Box 1 | folder 51 | Sir Willia... | | TJP | <LOCAT... | 46 Pall ... | | 46 Pall ... | the Arm... | W. Beth... | | 199
19 | Box 1, folder 51Sir Wi... | Box 1 | folder 51 | Sir Willia... | | TJP | <LOCAT... | Dublin 2... | Rail Roa... | Dublin;P... | | Pettigre... | unremitt... | 1994
20 | Box 2, folder 55Willia... | Box 2 | folder 55 | William ... | | TJP | <LOCAT... | Mill Yard... | Royal In... | Mill Yard... | a mummy | Mrs Slat... | engage... | 436
21 | Box 2, folder 62Jose... | Box 2 | folder 62 | Joseph ... | | TJP | July 12 4... | July 12 4... | | 12 Cork St | the Lyth... | Wm Birc... | informati... | 378
22 | Box 2, folder 62Jose... | Box 2 | folder 62 | Joseph ... | | TJP | The <OR... | The Chr... | Chronol... | 22 Hart ... | | Joseph ... | Chronol... | 319
23 | Box 2, folder 65Sir W... | Box 2 | folder 65 | Sir Willia... | | TJP | My Dear... | My Dear... | | | | M. Cham... | acknowl... | 848
24 | Box 2, folder 86J. Co... | Box 2 | folder 86 | J. Collin... | | TJP | Newcast... | Newcast... | Congress | England | | Norman ... | disquisiti... | 1117
25 | Box 2, folder 86J. Co... | Box 2 | folder 86 | J. Collin... | | TJP | Newcast... | Newcast... | Congress | Chester;... | | Tyne Se... | particula... | 1142
26 | Box 2, folder 89Tho... | Box 2 | folder 89 | Thomas ... | | TJP | Dear sir,... | Dear sir,... | | Brunswi... | | Tho Bur... | appoint... | 662
27 | Box 2, folder 89Tho... | Box 2 | folder 89 | Thomas ... | | TJP | My dear ... | My dear ... | | | | Burgon | acknowl... | 892
28 | Box 2, folder 92Gilbe... | Box 2 | folder 92 | Gilbert T... | | TJP | Mr <PER... | Mr Burn... | | St Georg... | | Burnett;... | convers... | 313
29 | Box 3, folder 104Patr... | Box 3 | folder 104 | Patrick ... | | TJP | 2 King S... | 2 King S... | | Londesb... | | St Jame... | antiquari... | 450
30 | Box 3, folder 109Hen... | Box 3 | folder 109 | Henry C... | | TJP | Burghfie... | Burghfie... | | | | Hy;Curti... | ministeri... | 830
31 | Box 3, folder 114Henr... | Box 3 | folder 114 | Henry C... | | TJP | 30 <LO... | 30 Mano... | Royal So... | Manor S... | | | | 368
<< < 1 to 100 > 143 > >>
SQLite 3.11.0 Gecko 47.0 0.8.3.1-signed.1-signed Shared Number of files in selected directory: 6
ET: 5 ms


SQLite Manager - /Users/ludaesch/git/DHOxSS2016/sqlite/pettigrewDB.sqlite
Directory
(Select Profile Database)
Go
pettigrewDB.sqlite
Structure
Browse & Search
Execute SQL
DB Settings
Master Table (1)
Tables (2)
PGLI
boxNumber
folderNumber
senderName
recpientName
date
Column 6
PGLNER
verbatimIndexInfo
box
folder
senderName
notes
recipient
letter
untaggedLetter
organization
location
artifact
person
longWords
letterLength
Views (0)
Indexes (0)
Triggers (0)
Enter SQL
SELECT DISTINCT boxNumber FROM PGLI;
Run SQL
Actions
Last Error:
not an error
boxNumber
Box 1
Box 2
Box 3
Box 4
Box 5
Box 6
Box 7
Box 8
Box 9
Box 10
Box 11
Box 12
Box 13
SQLite 3.11.0
Gecko 47.0
0.8.3.1-signed.1-signed
Shared
Number of Rows Returned: 13
ET: 1 ms



Installation -- Usually: Best to use the latest official release ..


swcarpentry/sql-novice-survey
Bertram
GitHub, Inc. [US] https://github.com/swcarpentry/sql-novice-survey
requirements.txt Updating Python requirements file 9 months ago
setup.md Fixing link in setup instructions 9 months ago
README.md
sql-novice-survey
An introduction to databases and SQL using Antarctic survey data. Please see https://swcarpentry.github.io/sql-novice-survey/ for a rendered version of this material, the lesson template documentation for instructions on formatting, building, and submitting material, or run make in this directory for a list of helpful commands.
Maintainer(s):
Abigail Cabunoc Mayes
Sheldon McKay
© 2017 GitHub, Inc. Terms Privacy Security Status Help
Contact GitHub API Training Shop Blog Abou


Databases and SQL
Bertram
Secure
https://swcarpentry.github.io/sql-novice-survey/
Home
Code of Conduct
Setup
Reference
Episodes
Extras
License
Search...
Databases and SQL
In the late 1920s and early 1930s, William Dyer, Frank Pabodie, and Valentina Roerich led expeditions to the Pole of Inaccessibility in the South Pacific, and then onward to Antarctica. Two years ago, their expeditions were found in a storage locker at Miskatonic University. We have scanned and OCR the data they contain, and we now want to store that information in a way that will make search and analysis easy.
Three common options for storage are text files, spreadsheets, and databases. Text files are easiest to create, and work well with version control, but then we would have to build search and analysis tools ourselves. Spreadsheets are good for doing simple analyses, but they don't handle large or complex data sets well. Databases, however, include powerful tools for search and analysis, and can handle large, complex data sets. These lessons will show how to use a database to explore the expeditions' data.
Prerequisites
Unix shell plus SQLite3 or Firefox SQLite plugin.
survey.db
Schedule
00:00 Selecting Data How can I get data from a database?
00:15 Sorting and Removing Duplicates How can I sort a query's results? How can I remove duplicate values from a query's results?
00:35 Filtering How can I select subsets of data?
00:55 Calculating New Values How can I calculate new values on the fly?
01:05 Missing Data How do databases represent missing information? What special handling does missing information require?
01:35 Aggregation How can I calculate sums, averages, and other summary values?
01:55 Combining Data How can I combine data from multiple tables?
02:35 Data Hygiene How should I format data in a database, and why?
03:05 Creating and Modifying Data How can I create, modify, and delete tables and data?

SQL: **S**tructured **Q**uery **L**anguage

- Based on **relational algebra** and tuple **relational calculus**.
- Consists of a data definition language, data manipulation language, and data control language.
- Scope of SQL includes data **insert**, **query**, **update** and **delete**, schema creation and modification, and data access control.
- SQL is a **declarative** language (4GL), but also includes procedural elements.

So many query languages, so little time: 4-in-1

- SQL — SELECT ... FROM ... WHERE ...
- Relational Algebra (RA) — $\sigma, \pi, \bowtie, \delta, \cup, \setminus$
- Relational Calculus (RC) — $\forall x F,\ \exists x F,\ F \wedge G,\ F \vee G, \neg F$
- **Datalog** — $\approx$ RC + Recursion

EXAMPLE: Given relations `employee(Emp, Salary, DeptNo)` and `dept(DeptNo, Mgr)`, find all (employee, manager) pairs:

- SQL:
```
SELECT Emp, Mgr
FROM employee, dept
WHERE employee.DeptNo = dept.DeptNo
```
- RA: $\pi_{\texttt{Emp,Mgr}}(\texttt{employee} \bowtie \texttt{dept})$
- RC: $F(\texttt{Emp,Mgr}) =$ $\exists \texttt{Salary, DeptNo} : (\texttt{employee(Emp, Salary, DeptNo)} \wedge \texttt{dept(DeptNo,Mgr)})$
- **Datalog**: `boss(Emp,Mgr)` $\leftarrow$ `employee(Emp, Salary, DeptNo), dept(DeptNo,Mgr)`