My Home Library Database, Round 1

Paul R. Potts

October 2023

The Library Database Project

I've long wanted to keep some sort of database of our entire library. Years ago I was using a Mac application called Delicious Library, which had some really nifty features. I could type in a title, and it would look up different editions via Amazon's database. I could then select the edition you had, and it would download the data and put a little image of the book's cover on a virtual shelf. I could rearrange these virtual shelves to match our real shelves. I could even use a camera to scan the bar codes on books and CDs, and it would look them up, getting them right about 80% of the time. Unfortunately, a few years ago, Amazon enshittified their database API, and those features stopped working.

Delicious Library was never perfect; it was often extremely slow, taking several minutes to update a book. I think it used Berkeley DB under the hood, but it had very limited export options. You can export a CVS file containing dozens of columns, but the data is very irregular and hard to work with. It might be possible to get at the database files that the application manages behind the scenes and open them up with Berkeley DB, but Berkeley DB is now some kind of Oracle product and I don't want to get into all that nonsense. I might be able to import the exported CSV file into Excel, or in my case actually LibreOffice Calc, but it is such a mess it may just be easier to start over.

I don't actually want to keep thousands of books in a spreadsheet, and I certainly don't want to use a proprietary application. What I really want is a true relational database, something that I can query using SQL, and create different views of the data. And I want a lot of flexibility to import and export files. The tools all have to be open source and the formats have to be supported by multiple, readily-available tools.

SQLite

What I settled on is SQLite. SQLite is a library and a standard file format that allows applications to work with small databases. (Even our library, with thousands of books, CDs, and videos, is still a small database for this kind of purpose; even my laptop will easily hold the whole thing in memory, and queries should be pretty much instantaneous).

The "front end" I'm currently using is a program for Mac called DB Browswer for SQLite. Although it does crash occasionally after running for a long while and doing lots of editing, for the most part it is very nice and easy to work with. It's a graphical user interface, but when you're making tables or performing queries, it's using SQL under the hood. And not very far under the hood; you can open up a window and see what it's doing. You can also write your own SQL and execute it if you want to.

The database file is on my Synology file server. I save my work frequently and the server backs itself up to a cloud storage service every week, so I'm not terribly worried about losing work due to an occasional crash. The files are so small that it is very easy to periodically make a ZIP file of the whole folder and give it a date. The latest one, which includes all the data, as well as a number of exported CSV and HTML files, takes up only 86KiB. Text files are very small compared to image files and video files.

SQL

SQL stands for "Structured Query Language," and it is a very old, but still widely used, tool for talking to databases and getting them to do your bidding. Unfortunately there are a lot of slightly incompatible dialects of the language.

Years ago, when I was in college, I was an English major, not a Computer Science major. I took all the programming classes I could fit into my schedule, and so got a minor in Computer Science. Some of my classmates who actually completed the Computer Science major took a course on databases, but I did not. So that was a bit of a hole in my education.

A few years later I learned how to write front ends for Oracle using Apple's WebObjects, and then later to debug stored procedures written in PL/SQL. Along the way I did teach myself a bit about databases, and I had to use basic SQL to work with Oracle and PostgreSQL, but I didn't ever consider myself an expert.

I last used SQL in a job over twenty years ago. So it's been an interesting challenge to try to get back into it. I don't really love the syntax of SQL, and I often have trouble figuring out where to put clauses to make them work, but there's no denything that it is an incredibly useful and powerful tool, and one can't really be a well-rounded programmer without knowing at least a little bit about how to use databases and SQL.

A home library might seem like it would require only a very simple database, but our home library is unusually large. Also, as your friendly neighborhood computer geek, I want the database to be structured "just so" — just the way I want it, not the way somebody else wants it. After a little bit of searching, I was unable to find a sample database schema that would do what I wanted, so I have come up with my own. I've refined it over time, as I gradually add data to the database.

One of the nice things about this GUI tool is that if I want to change the table design, which I've done as I gradually worked out what I wanted to do, it's easy to do. If for some reason I can't modify the existing table, which happens sometimes if I want to make major changes, I can dump the data out to CSV files, destroy the table, make a new table, and import the data back into the new table. So I've been able to rip up and rearrange the database structure as I have made incremental design changes, although it has settled down for the time being.

I'm pretty sure that I'm not using keys in an optimally efficient way, but given the size of the database, that doesn't really matter. I've also had to accept the idea that I'm not going to be able to complete this project quickly. Right now I've got just over three hundred paper books in the database. I estimate that there may be five thousand to go. The data entry is quite tedious. I try to chip away at it, doing a few books a day. As I go on, I'm getting more efficient at it. And as I go on, I may be able to find ways to speed it up, such as importing chunks of data from the aforementioned CSV files dumped from Delicious Library.

Let's take a look at my database design.

My Library Database Design

I would like to have some nice entity relationship diagrams to include here, that show how the tables and keys fit together. There are a number of tools that will make such diagrams from SQL, including some open-source programs that run on Macs, and some web-based tools that supposedly come with free trials. I tried an open-source tool, implemented in Java, but had endless problems getting it to work, due to compatibility problems with JDBC libraries on Apple Silicon. I was once pretty good at solving problems with Java, but that was over 20 years ago and I was paid to do it. I also tried one of the online tools that looked promising. I signed up for a free trial, and the tool would import my SQL, but I could not get it to make the diagrams I was hoping for. So, for the time being I have given up on making nice diagrams from my SQL.

Instead, I'll just show you my SQL. If you want to play with this design, it ought to be pretty easy to use just about any relational database, such as PostgreSQL, to create a database with the same schema.

Authors The authors table can be created using the following SQL:

```
CREATE TABLE "Authors" (
    "Name" TEXT NOT NULL,
    "Short Description" TEXT NOT NULL,
    "Bio URL" TEXT,
    "Notes" TEXT,
    PRIMARY KEY("Name")
);
```

The "Authors" table exists to maintain a set of of canonical author names. They are entered as first name, last name. The name is the primary key. Other tables can link to these authors using foreign keys, and these references must match canonical author names. The required short description contains years of birth and death (if the author has died), and nationality. The optional "Bio URL" field contains links to a biography on Wikipedia or other online source describing the author. In the optional "Notes" field, I sometimes include information about pen names and alternate spellings of foreign names. Currently, there are 196 authors in this table. The first entry is for James Agee, and the fields look like this:

- Agee, James
- American novelist and journalist 1909-1955
- https://en.wikipedia.org/wiki/James_Agee
- NULL

(The NULL indicates there is nothing in the "Notes" column.)

Publishers The publisher table is generated by the following SQL:

```
CREATE TABLE "Publishers" (
    "Name" TEXT NOT NULL,
    "URL" TEXT,
    "Notes" TEXT,
    PRIMARY KEY("Name")
);
```

The name in this case is the name of the publisher. A typical entry looks like this:

- Berkley
- https://www.penguin.com/berkley-overview/
- Now an imprint of Penguin

Due to the truly massive amount of consolidation that has occurred in the publishing industry, the majority of the publisher names on the spines of my books now refer to companies that are either imprints of other publishers, subsidiaries of other publishers (sometimes two or three layers deep, after a long history of mergers or acquisitions), or completely defunct. This consolidation has been a disaster for small publishers and diversity in the book market. By recording the publisher name as it is appears on the books, as well as some current information about the publisher, if I can find any, I'm trying to leave bread crumbs, which might help someone in the future to find more information about an edition, even in cases where the publisher displayed on the book has changed its name, often more than once.

Paper Book Types This table contains canonical descriptions of different types of paper books. The SQL is very simple:

```
CREATE TABLE "Paper Book Types" (
    "Book Type" TEXT NOT NULL,
    PRIMARY KEY("Book Type")
);
```

The editions table refers to the book type as a foreign key, so that the book types follow a canonical set of descriptions. Here are a few of them:

- Chapbook
- Hardcover
- Trade Paperback
- Mass Market Paperback
- Jacketed Hardcover with Traycase
- Jacketed Hardcover with Slipcase
- Hardcover with Slipcase, No Jacket as Issued
- Paperback Book with Unusual Features

"Paperback Book with Unusual Features" covers special editions like Haruki Murakami's *The Strange Library*, a slim little art book with covers that fold and wrap around the entire book, so that you open it as if you were unwrapping a present. This list of paper book types will grow as I add books that are unusually sized, boxed sets, books that are collections of pamphlets, issues of magazines, etc. If the book is more-or-less a standard kind of book, like a slipcased hardcover edition, but has interesting features like foil printing, a psychedelic cover, a fabric-wrapped cover, die cutting on the cover, dust jacket, or slipcase, silk-screening on the page edges, tip-ins, included floppy discs or CD-ROMs, etc., I describe those features in the "Notes" field.

Paper Book Shelving Areas This is another simple table:

```
CREATE TABLE "Paper Book Shelving Areas" (
   "Description" TEXT NOT NULL,
    PRIMARY KEY("Description")
);
```

I use this table to hold a canonical list of shelving *areas*, not shelving units. For example, I currently have two bookshelves devoted to Library of America books. They get a single name in this table. Sometimes the names just refer to a distinct section of books within a shelving unit, such as our set of books from New York Review Books, which are shelved together because they form a rainbow of matching volumes. Technically I could interleave them with other books in the other shelving areas by author, but they look cool shelved together:

Paper Book Editions This table is where things start to get interesting, since three of the columns contain foreign keys that refer to columns in other tables; this is where the "relational" in "relational databases" really comes into play.

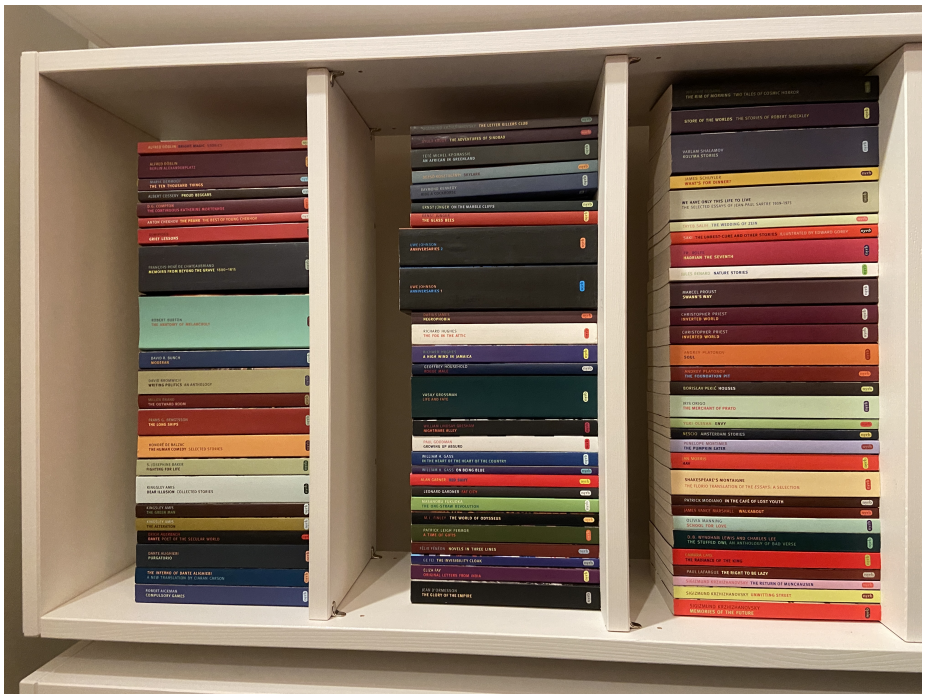

Figure 1: “New York Review Books, the First Three Shelves”

```
CREATE TABLE "Paper Book Editions" (
    "Author" TEXT,
    "Other Contributors" TEXT,
    "Title" TEXT,
    "Publisher" TEXT,
    "Book Type" TEXT,
    "ISBN-13" TEXT UNIQUE,
    "Edition Notes" TEXT,
    PRIMARY KEY("ISBN-13"),
    FOREIGN KEY("Publisher") REFERENCES "Publishers"("Name"),
    FOREIGN KEY("Book Type") REFERENCES "Paper Book Types"("Book Type"),
    FOREIGN KEY("Author") REFERENCES "Authors"("Name")
);
```

The primary key for this table is the ISBN-13, "canonicalized" into the form ###-##########-#. Different sources show ISBN-13 numbers with the dashes in different places. Why did I choose this form? It's complicated, but it has to do with the way ISBNs are broken into a heirarchy of variable-length fields, and I wanted a generic format that would be consistent for all of them.

ISBN doesn't work for everything. Editions can have the same ISBN with different cover art, or different physical formats. Some books have no ISBN number. Folio Society editions don't have them. Some Subterranean Press editions don't have them. And many older books don't have them. In this case I make up a number that includes the publisher name followed by a number, for example, Folio-001. If there is anything else that seems important to distinguish this edition from other editions of the same book, I put it in the notes field.

The "Other Contributors" gives me a field I can use for translators, editors, illustrators, and people who contribute introductions, prefaces, forewords, afterwords, or whatever, as specified on the title page or cover.

This scheme allows me to have multiple editions of books with the same title and author. For example, I might have editions of the same original book translated by two different translators. It also allows me to have multiple copies of the same book, since this table does not actually hold the books themselves. They're in the next table.

Paper Books At last, the books. This table mostly references other tables:

```
CREATE TABLE "Paper Books" (
    "ID" INTEGER UNIQUE,
    "Brief Author & Title" TEXT,
    "ISBN-13" TEXT,
    "Shelving Area" TEXT,
    "Shelf Number" INTEGER,
    "Shelf Order" INTEGER,
    "Copy Notes" TEXT,
```

```
    PRIMARY KEY("ID" AUTOINCREMENT),
    FOREIGN KEY("Shelving Area") REFERENCES "Paper Book Shelving Areas"("Description"),
    FOREIGN KEY("ISBN-13") REFERENCES "Paper Book Editions"("ISBN-13")
);
```

The "Brief Author & Title" field is not the full title or the canonical author or publisher. The full title is in the "Paper Book Editions" table. Indirectly, the editions table references the authors and publishers, so this field is technically redundant and not strictly necessary, but because I can't read ISBNs while I'm entering data into the "Paper Books" table, it is easy to get confused and and lose my place. To avoid that, I wanted a readable way to easily identify the books in this table. So, the "Brief Author & Title" contains the author's last name and the book's title, as in "Proust: Swann's Way."

I'm not entirely happy with this design, due to the redundancy, but in practice it has proven helpful. If I could easily edit the data in the "Paper Books" table while I could *see* the indirectly linked author and title on the screen, populated after I enter the ISBN-13 which is used as a foreign key, I wouldn't need this field. I'm sure there are ways to do this, but they probably involve tools that build forms. The lightweight DB Browser for SQLite tool doesn't seem to do that, and I'm not sure what tools I might try next.

The "Shelf Order" field needs a bit of explanation. Sometimes, author and title aren't enough to specify the order that I use for books on the shelf. This is true of anthologies and omnibus editions containing work by a single author. For example, I have Library of America editions of the novels of Philip K. Dick called *Four Novels of the 1960s* and *Five Novels of the 1960s and 70s*. On the shelf, I put the books in chronological order, rather than alphabetical order by title. But the database normally sorts books in alphabetical order by title, after shelf number and author. Thus, they appear out of order in the database, since alphabetically, "Five" comes before "Four."

"Shelf Order" can be NULL, and is NULL in most cases, so that I don't have to specify a value for every book; alphabetical order is the default. But this allows me to specify an ordering for a sequence of books by an author, when alphabetical order isn't what I want.

Sample Views These are the first six shelves of our Library of America collection:

Here's a SQL statement to create a view to show the Library of America books that have authors (that is, they are single-author volumes, not anthologies or collections of some sort):

```
CREATE VIEW "loa-shelves-single-author-volumes-view" AS select
    'Paper Books'.'Shelving Area',
    'Paper Books'.'Shelf Number',
    'Paper Book Editions'.Author,
```

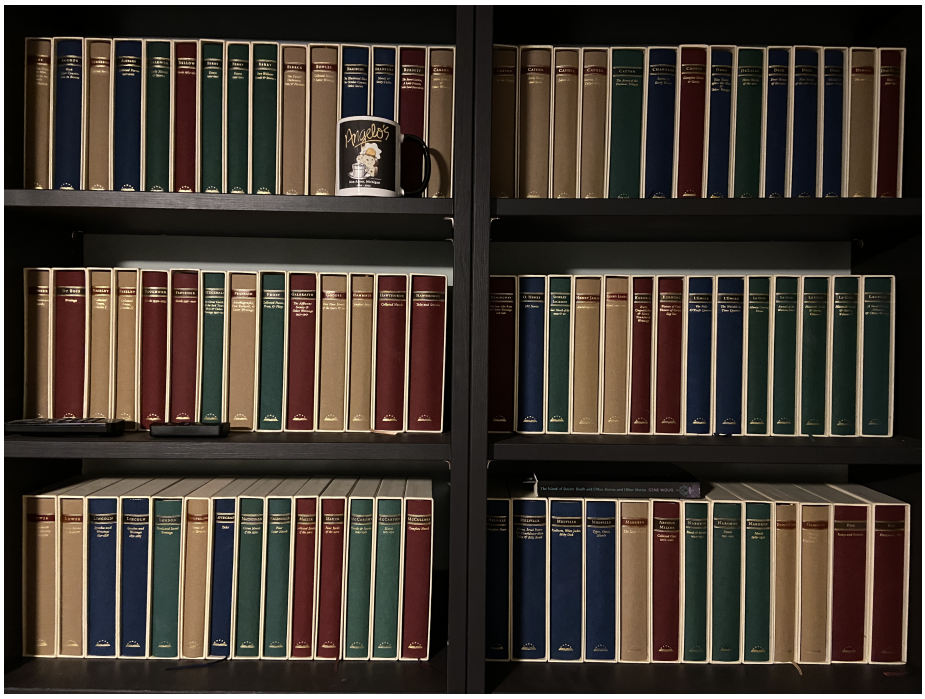

Figure 2: “Library of America First Six Shelves”

```
    'Paper Book Editions'.Title,
    'Paper Book Editions'.Publisher,
    'Paper Book Editions'.'ISBN-13',
    'Paper Book Editions'.'Book Type',
    'Paper Book Editions'.'Edition Notes',
    'Paper Books'.'Copy Notes'
from
    'Paper Books'
    inner join 'Paper Book Editions'
        on 'Paper Book Editions'.'ISBN-13'='Paper Books'.'ISBN-13'
     where 'Paper Books'.'Shelving Area'="Library of America" and 'Paper Book Editions'.Auth
    order

        by 'Paper Books'.'Shelving Area', 'Paper Books'.'Shelf Number', 'Paper Book Editions
            case 'Paper Books'.'Shelf Order' when null then 'Paper Book Editions'.Title else
```

That did not come easily; I had to read some SQL reference materials and scrutinize a number of different examples to fit those pieces together, but it works. The view makes it look like I have a nice simple table of just the Library of America single-author volumes, when really it is pulling data from the database and organizing columns from multiple tables.

Here are the first few rows produced by the above view, in CSV form. CSV is a bit awkward to read because some fields are quoted, and some aren't. Quoting is normally used when the field contains the separator character, which in this case is a comma. For readability I have exported these records using a space as the field separator, which makes the data a bit more legibleWhen it's done this way all strings that contain spaces are quoted. I've also inserted blank lines to make it clearer where the rows begin and end.

```
"Library of America" 1 "Agee, James" "Let Us Now Praise Famous Men, A Death in the Family, a

"Library of America" 1 "Alcott, Louisa May" "Work, Eight Cousins, Rose in Bloom, Stories & C

"Library of America" 1 Anderson,Sherwood "Collected Stories" "Library of America" 978-159853
```

I can use **pandoc** to convert the CSV to Markdown, and put the Markdown in this document for processing by **pandoc**. That will produce a nice HTML table, although it is very wide, and with the style sheet I'm using for this newsletter, the HTML likely won't be very readable in your browser, and in the PDF, the table will march right off the edge of the page. But here it is anyway:

Shelving Area	Shelf Number	Author	Title
Library of America	1	Agee, James	Let Us Now Praise Famous Men, A Death in the
Library of America	1	Alcott, Louisa May	Work, Eight Cousins, Rose in Bloom, Stories & C
Library of America	1	Anderson,Sherwood	Collected Stories

The point is not so much not that I can read the data in an ideal way, but that using a database like this gives me a lot of options for different kinds of reports.

One thing I haven't figured out is how to ignore leading articles like "the," "a," and "an" while sorting. There are ways to do this, but I haven't tried them yet, and I'm not sure which ones are best practices (or even which ones work) in SQLite. For now, if I have a problem with sorting, I fix it using the "Shelf Order" field, but I don't want to have to rely on this solution as the database gets larger; in some cases I have a lot of books by specific authors, and I don't want to have to maintain shelf order numbers for all of them.

For the Library of America anthologies, the view of those books is created a little bit differently; I don't want to show the author, which is NULL; in fact, I select only the Library of America books with empty author fields:

```
CREATE VIEW "loa-shelves-anthologies-view" AS select
    'Paper Books'.'Shelving Area',
    'Paper Books'.'Shelf Number',
    'Paper Book Editions'.'Other Contributors',
    'Paper Book Editions'.Title,
    'Paper Book Editions'.Publisher,
    'Paper Book Editions'.'ISBN-13',
    'Paper Book Editions'.'Book Type',
    'Paper Book Editions'.'Edition Notes',
    'Paper Books'.'Copy Notes'
 from
    'Paper Books'
    inner join 'Paper Book Editions'
        on 'Paper Book Editions'.'ISBN-13'='Paper Books'.'ISBN-13'
     where 'Paper Books'.'Shelving Area'="Library of America" and 'Paper Book Editions'.Auth
    order

        by 'Paper Books'.'Shelving Area', 'Paper Books'.'Shelf Number',
            case 'Paper Books'.'Shelf Order' when null then 'Paper Book Editions'.Title else
```

How Long Will This Take?

I've completed about 22 individual shelves. There are 130 more shelves of books to catalog, not counting the children's books in the family room, the DVDs, the Blu-rays, and the CDs. So, this is obviously going to take a while. I'll be looking for shortcuts, but there's going to be a lot of data entry and editing. I am hoping that the catalog might be fully complete by this time next year. As I go, I'm taking photographs of each shelf. These are going into folders on the server as a reference. These can be updated periodically. I don't expect them to stay perfectly synchronized with the database, since the collection slowly changes. As a shortcut, if I add books, I can just add photos of the new books to the folders, rather than taking a whole new set of pictures of the shelves.

It would be nice if I could incorporate the photos right into the view, but so far my experiments putting JPEG file into "blobs" in the database made the DB Browser application run incredibly slowly, even though the JPEG files really aren't that large. So I'll be avoiding that for the time being.

The fact that this is all built on SQL and text suggests that I might be able to come up with future uses for it in the future, such as a web-based front end for checking out books. But I'm not going to dive into that anytime soon, and I'm not going to "overdesign" for future front-ends that may never happen.
