E-T-L Project - Book Information  
Building A database To Find Trends Among Various Sources Of Book Data

Karen McGee | Andrea Morgan | Stacey Smith

horizontal line

**Finding Data Sources to Use**

Our topic focuses on finding sources of book data with the goal of being able to find information, events, and reviews of various books based on best-seller lists.

We found four sources of data to be able to start building our database:

* A CSV file exported from GoodReads, found at <https://www.kaggle.com/jealousleopard/goodreadsbooks>
* The ISBN Database API, at <https://api2.isbndb.com>
* The New York Times Books API, at <https://developer.nytimes.com/docs/books-product/1/overview>
* The Penguin/Random House Publishing Books API, at <http://www.penguinrandomhouse.biz/webservices/rest/>

After looking at the data that we could extract from each source, we worked on what our table structure would be for the final database. Each data source includes a unique identifier for books, the 13-digit ISBN number. We figured the ISBN number would be a good way to link tables when running joins for queries, since it is the only data we could pull that would be completely unique. Based on our initial exploration of the data, our table structure is as follows:

* BOOKS table - contains book\_id (primary key, isbn number), title, and authors
* EVENTS table - contains event\_id (primary key, unique number for each event), event source, event isbn (foreign key to books table), event participants, event date, event location, event city, event state, event description, and event market
* REVIEWS table - contains reviews id (primary key, unique for each review), rank, weeks on list, book id (foreign key to books table), publisher, book review link, category, current as of date

**Extracting and Transforming the Data**

**GoodReads**

Kaggle provides a CSV of book data that a user had pulled from GoodReads. It includes over 47,000 different books with the title, author(s), average GoodReads rating, the 10-digit ISBN number, the 13-digit ISBN number, the book’s language code, the number of pages, and both the ratings and text reviews counts. We imported the CSV into Jupyter Notebook and dropped many of the columns, leaving only the ISBN13, title, and authors columns. This provided the foundation for joining the New York Times Books API and Penguin/Random House API. By using the ISBN as the Primary Key, we would be able to have a uniform version of the title and author(s) as we joined these two rather than having slightly different versions of title and authors from each of the other two tables.

Unfortunately, when we went to join the NYT and PRH data, we were surprised to find that very few records from those two tables matched the data on the books table. After some research, we learned that each different edition of a book has a different ISBN. For example, Little Women does not just have one ISBN but separate ISBNs for the hardcover version, paperback version, each edition, etc. While we could have settled on still using this source and adding the caveat about editions and ISBN numbers, we decided to see if there was another source for book data that we could use instead.

**ISBN Database API**

This API required that header information be supplied with each request. Rather than use Postman, I was able to use the sample code supplied by the ISBN API website to directly pass the header information in the API request.

I pulled data by publisher name with the idea that focusing on books from specific publishers might give us more relevant information to join with our other sources of data.

To retain the correct ISBN number, it was necessary to delete the null values from the dataframe and then convert that column to float before exporting to csv.

**New York Times (NYT) Books API**

Pulling from the NYT API was fairly straightforward. I began with a list of the current Best Sellers lists, showing how frequently each was updated.

From the lists, I was able to determine the book categories to use when pulling review information. To keep the scope within reasonable limits, I ultimately decided to pull review information for just five of the most current Best Seller list categories.

Transforming the data to conform with the booksDB dataframe (and ultimately the books database) took a bit more work. After reducing the amount of columns and data in the dataframe, I had to rename the isbn column to match the booksDB dataframe and also change the data type from ‘object’ to ‘int64.’

Additionally, I created two new columns to house the category name and the datetime the information was pulled so we could later reference it if needed.

**Penguin/Random House (PRH) API**

Even though the documentation provided from PRH states that the data can be extracted in JSON format through the API, it was actually only available in XML format. Using the python module XML Element Tree, I was able to find the tree, root, and child elements of the data tree provided through the PRH API.

I had to load each attribute separately because there were not the same number of attributes available for all records. For example, one event may have had a description while another did not, in which case the attribute would not be listed. To prevent the lists I was creating from being different lengths, I ran a “try/except” loop for each attribute that would append the attribute if it was present, and would append ‘none’ if it was not.

The data was pretty clean from the API, so the only transforming I had to do was making the column names consistent with our table schema. I also cleaned some minor characters (‘\r’, ‘\n’) from the location column. The biggest transforming that I had to do was on the event source column, which was a column of dictionaries. The data would not load into the database as a list of dictionaries, so I had to extract only the values from each dictionary and put those in the dataframe.

**Loading the Data into PostgreSQL**

We decided the best option for this type of data is the relational database PostgreSQL, since our information can be linked by ISBN number across tables. Once the data was extracted and transformed, we created the table schema within our “booksETL” database and loaded the data in.

One issue we ran into right away is that since both the reviews and events table refer to the books table for a foreign key, if there were ISBN numbers in those tables that were not in the books table, our data would not load into Postgres because of a “Data Integrity” error. We decided to merge the books table with each the events and reviews tables in Jupyter Notebook so that the only information remaining would be the ISBNs that are shared between the tables, which allowed our data to load correctly.

We discussed the implications of leaving the other data out of the database, and came to the conclusion that for the goal of our database, the only data that is relevant is data that can be shared between each table.

This is why there is no data in the reviews table; due to the foreign key constraint, there were no matches between the ISBN in the NYT Reviews table and the main books table.

The final step was to merge the event dataframe with the books dataframe from GoodReads. Unfortunately, there were not many ISBNs that the data shared with the books table, so the database does not have a lot of information on book events.