# metadata-coding-challenge

The object is:

Coding challenge for book processing related roles. Please complete the challenge and send it back within 24 hours, either a zip of the source code or a link to the git repository. Tech stack

To complete the challenge, installaion of node.js (https://nodejs.org/en/) and npm (https://nodejs.org/en/) will be required. Min version of Node 8 is recommended. Coding challenge

The challenge is to build a metadata extractor for all the project Gutenberg titles which are available here: https://www.gutenberg.org/wiki/Gutenberg:Feeds (https://www.gutenberg.org/cache/epub/feeds/rdf-files.tar.zip).

Each book has an RDF file which will need to be processed to extract the:

id (will be a number with 0-5 digits)

title

publisher (value will always be Gutenberg)

publication date

language

license rights

author/s

subject/s

Note: For some books all of the data won't be available. Tasks

Write a function that reads a single file in and outputs the correct output. The libraries that might be useful: https://www.npmjs.com/package/xml2js, https://www.npmjs.com/package/xmldom

Store the output in a database of your choice locally for later querying. Use ORM of your choice (like Sequlize , Mongoose ), to define database models as well.

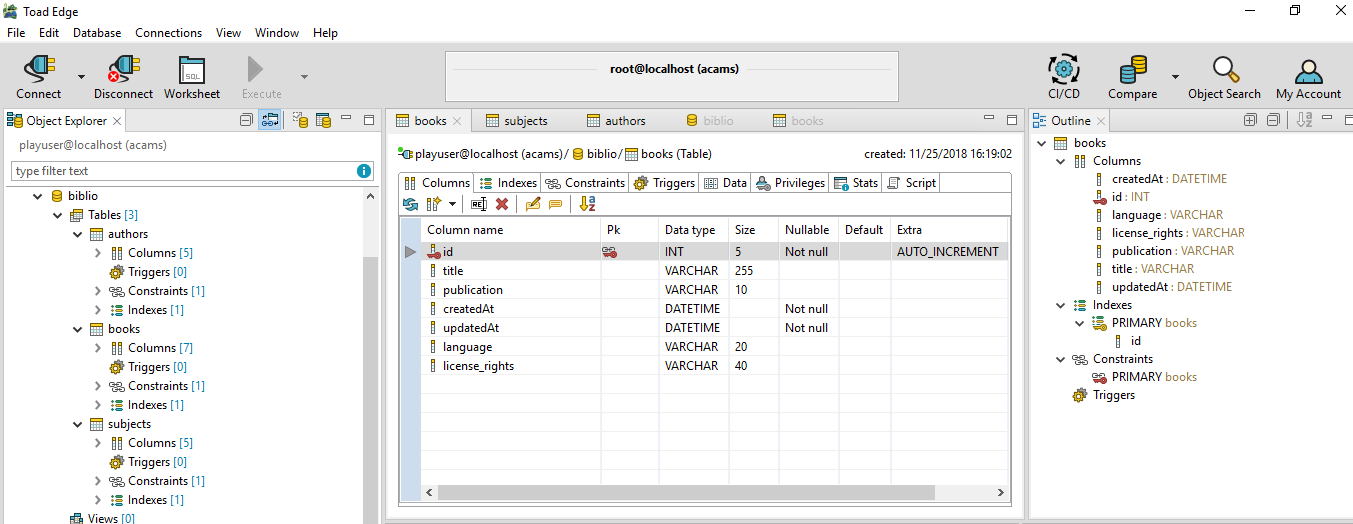
Write unit tests (use test suite libraries like mocha or jest ) for the code and use code coverage analysis tool (built-in jest one, or libraries like Istanbul ).

Process all metadata for the titles for later querying.

-----------------------------------------------------------------------------------------------------------------------------------

First, we create the database in MySQL using Sequalize,

The biblio DB is created with the books, authors and subjects tables in MySQL:



The book contains,

id (will be a number with 0-5 digits)

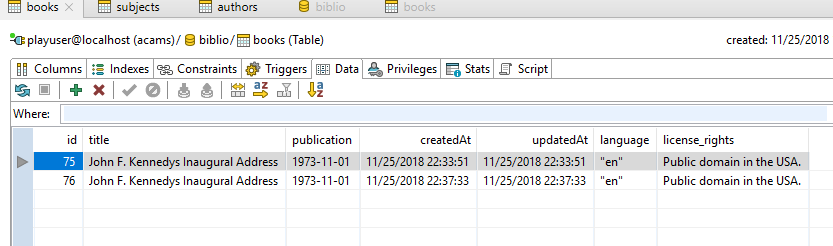
title

publisher (value will always be Gutenberg)

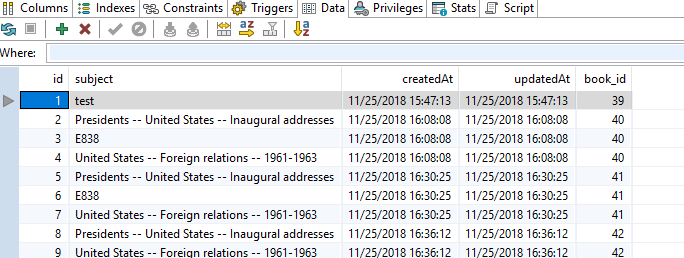
publication date

language

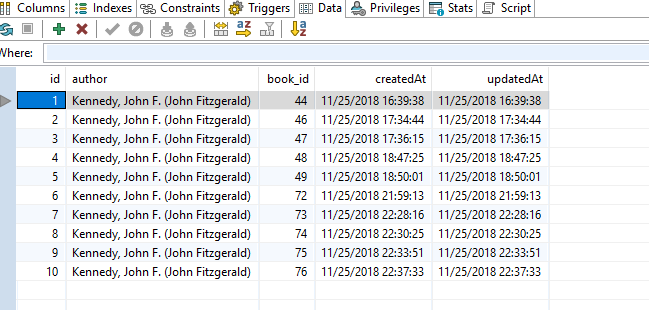
license rights



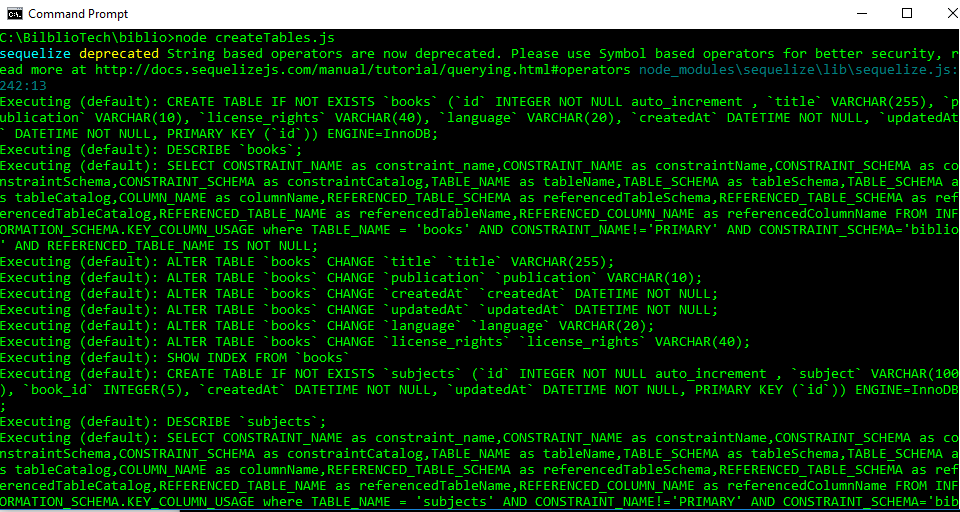
Subjects and Authors are different tables as there is “Many-to-One” relationship to the book. There can be many subject and authors per book. The book\_id maps to the book by Id in the books table.



The Authors follow the same structure:



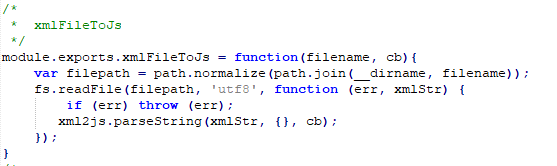
To create these tables and DB in the MySQL, the createTables.js is called from the command line, also there is a Dump20181125.sql with related SQL that can also be used to load the DB.



So, we got a Database.

Now, we have to parse out the RDF, I started with a RDFLib that helped a lot, but the instructions references 'xml2js', so I used that instead.

There is a parsing function in the utils.js called ‘xmlFileToJs’, this takes the RDF and puts it in JSON object form.

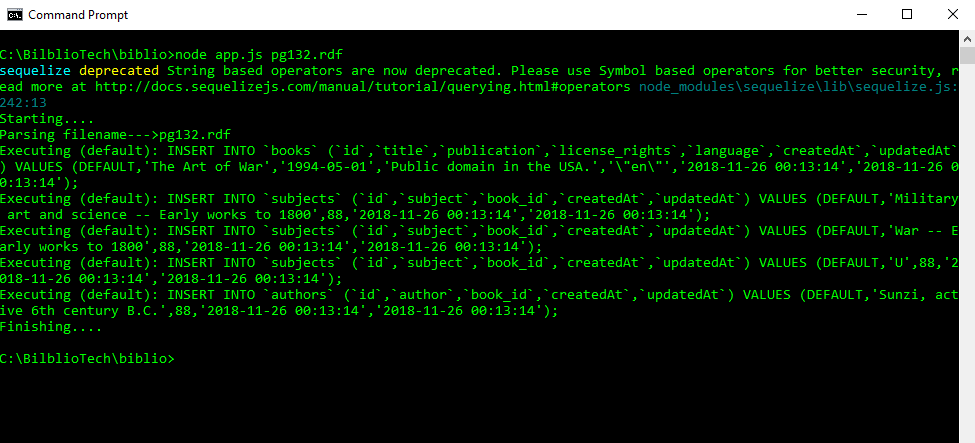


Once in JSON Object form, then the associated JSON objects are populated.

The utils.js has a few functions to retrieve the data, it will populate a string array of both subjects and authors. The subjects are populated through the ‘getSubjects’ function, and the authors are populated through the ‘getAuthors’ function. The rest of the book material, such as title, license, language and publication data (called publication), are returned through the ‘getBook’ function. This function will populate the Book model, found as books.js.

Once the subjects, authors and book models are populated, they are saved to the database using the app.js ‘saveToDB’ function.

At the command line. C:\BilblioTech\biblio>node app.js pg132.rdf where ‘pg132.rdf’ is one of the test files that we pass at the command line.

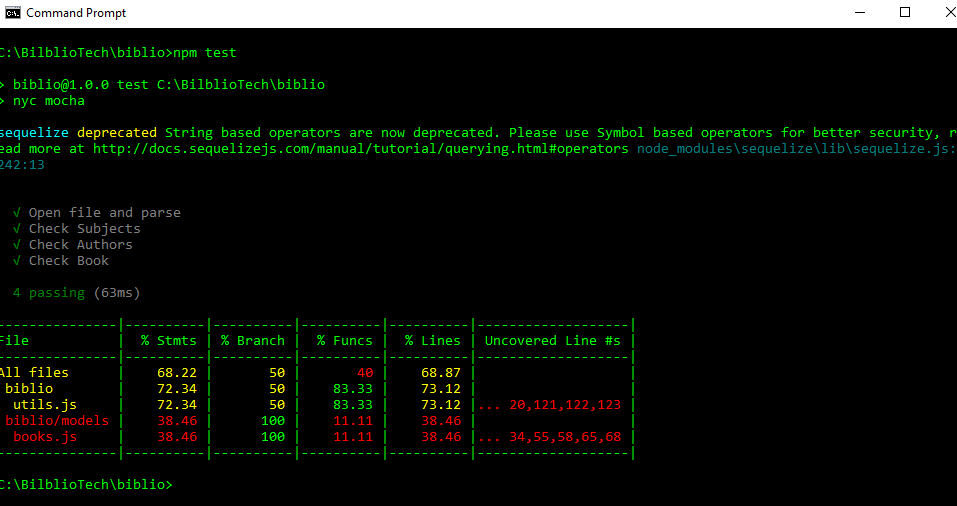


The RDF is parsed and inserted into the biblio DB through Sequaeizer. There is timer for 5 seconds to ensure that the process exists after that time. Most files parsed well within 5 seconds and this ensured that the process completed. Running the app.js will populated the database with data.

So now we have some Data.

To test, Mocha and Istanbul was used. Which are saved in the test.js file.

Running the ‘npm test’, we see that the tests are executed:



There are 4 tests, parsing the file, reading the subjects material, reading the authors material, and reading the book material, from a sample files.

Now we have unit tests and coverage analysis.

Enclosed is a picture of the directory structure:

