ETL Project

Kenny Dao
Sean Pei

Our task was to source, process and load the data to a production database. We used Jupyter notebook to pull API from selected data source, Pandas library for our data manipulation and creating DataFrames. Then, we utilized PgAdmin to load the data into Postgresql database and make it accessible online using Google Cloud Platform.

Data source:

1. Kaggle: https://www.kaggle.com/bahramjannesarr/goodreads-book-datasets-10m?select=book900k-1000k.csv (dataset is being updated every 2 days)
2. API: https://www.goodreads.com/api
3. https://www.kaggle.com/pelinsoylu/amazon-the-most-read-books-of-the-2019-dataset (bestsellers with categories.csv)

EXTRACT

We used the books data (book900k-1000k.csv) from Kaggle. It's a csv file with over 40k rows and 20 columns that content lot of information and a good starting point for us.

We also used the bestsellers with categories data to create the API request (from Goodreads API) about information about authors.

TRANSFORM

Having found the books dataset on Kaggle, we have chosen to make more information relating to books available for inquiries, such as finding the most popular book titles (bestsellers title by year), the highest-rated books, recently released titles, etc. For books table, 11 columns were removed and cleaned up/convert to suitable forms. The focused columns are BookId, Title, Rating, ISBN, PublishY, PublishM, PublishD and Language.

We have pulled API from Goodreads to get more information about authors. The data is in XML file so we converted into dataframe using Jupyter notebook and created an authors table, includes AuthorId, AuthorName, Gender, and Hometown columns. At present, the authors table has 244 records.

In [8]:
```
authors_name_and_ids = []
for name in author_list:

    query=name
    url = f"https://www.goodreads.com/api/author_url/{query}?key={api_key}"

    r = requests.get(url)
    root = ET.fromstring(r.text)
    tree = ET.ElementTree(root)
    tree.write("list.xml")

    with open("list.xml") as xml_file:
        data_dict = xmltodict.parse(xml_file.read())
        xml_file.close()

    json_data = json.dumps(data_dict)
    with open("data.json","w") as json_file:
        json_file.write(json_data)


    with open("data.json") as f:
        data = json.load(f)
        authors_name_and_ids.append(data)
```

In [15]:
```
author_id_list=[]
author_name_list=[]
author_link_list=[]

for info in authors_name_and_ids:
    if "author" in info['GoodreadsResponse']:
        author_id = info['GoodreadsResponse']['author']['@id']
        author_name = info['GoodreadsResponse']['author']['name']
        author_link = info['GoodreadsResponse']['author']['link']

        author_id_list.append(info['GoodreadsResponse']['author']['@id'])
        author_name_list.append(author_name)
        author_link_list.append(author_link)
    else:
        pass
```

In [10]:
```
df_author = pd.DataFrame({"author_id":author_id_list,
                          "author_name":author_name_list
                         })
```

In [24]:
```
# merge authors_df with author_id_df

authors_df = authors_info.merge(authors_id, on = 'AuthorId')
```

In [25]:
```
authors_df = authors_df[['AuthorId','AuthorName', 'Gender', 'Hometown']]
authors_df.head()
```

Out[25]:

	AuthorId	AuthorName	Gender	Hometown
0	14980615	Chip Gaines	female	Albuquerque, New Mexico
1	33280	Dav Pilkey	NaN	Cleveland, OH
2	61105	Dr. Seuss	male	Springfield, MA
3	3362	Eric Carle	male	Syracuse, New York
4	4880403	Emily Winfield Martin	female	NaN

We then processed and merged the data from books data, bestseller data, authors (created above) to form an author_book dataframe that make a relation between author and book, adding BestSeller feature (to determine whether a book title is one of the bestseller books).

```
bestsellers_df = bestsellers_df.merge(books_df, on='AuthorName')
#bestsellers_df.shape
```

```
#bestsellers_df.columns
```

```
bestsellers_df.drop(columns=['Unnamed: 0','Title_x', 'Rating_x', 'PublishY_x',
                             'PublishD', 'PublishM', 'Publisher', 'Language'], inplace=True)
```

```
#bestsellers_df = bestsellers_df.merge(books_df, on='AuthorName')
# bestsellers_df.columns
```

```
bestsellers_df = bestsellers_df.rename(columns = {'Title_y':'Title', 'Rating_y':'Rating', 'PublishY_y':'PublishY'})
#bestsellers_df.head()
```

```
bestsellers_df.drop_duplicates(subset='BookId', inplace=True)
bestsellers_df.shape
```

```
# save to file
#bestsellers_df.to_csv('Resources/bestsellers_book.csv', index=False)
```

```
# verify if BookId in author_book is bestseller and store value to BestSeller column in author_book (True/False)

# assign BookId from author_book to book_list
book_list = author_book_df['BookId']

# assign BookId from bestseller_df to bestSellers_list
bestSellers_list = bestsellers_df['BookId']
```

```
# checking if the book_list is in bestseller_list
bestseller = book_list.isin(bestSellers_list)
bestseller
```

```
# storing result back into author_book_df
author_book_df['BestSeller'] = bestseller
author_book_df.head()
```

In [67]:
```
# storing result back into author_book_df
author_book_df['BestSeller'] = bestseller
author_book_df.head()
```

Out[67]:

	AuthorId	BookId	AuthorName	BestSeller
0	61105	900821	Dr. Seuss	True
1	3362	920783	Eric Carle	True
2	5086768	958347	Bill Martin Jr.	True
3	311467	994192	Mark R. Levin	True
4	6785	993176	Sebastien Braun	False

LOAD

We loaded our CSV into Postgresql as our database and made it available on-line using google cloud platform. After that, we have tested out with some queries as below.

In summary, based on our observation and finding, there are multiple books dataset available. However, the data format is vastly different and still much of missing data (null value), which we can further update a long the way. Applying Postgresql to store database about books would be an alternative to present data in a structured data for inquiries.

postgres/postgres@goodreads_db

Query Editor | Query History

```
-- Join three tables to list the detail info about bestseller books from 1990 - 2019 in the dataset
SELECT b.Title, au.AuthorName, b.Rating, b.PublishY
FROM books b
JOIN author_book ab
ON b.BookId = ab.BookId
JOIN authors au
ON ab.AuthorId = au.AuthorId
WHERE   ab.BestSeller = true
        AND (b.PublishY >= 1990 OR b.PublishY <= 2019);
```

Data Output | Explain | Messages | Notifications

	title character varying (500)	authorname character varying (50)	rating double precision	publishy integer
1	The Lorax	Dr. Seuss	4.35	1971
2	The Very Busy Spider	Eric Carle	4.21	1985
3	Here Are My Hands	Bill Martin Jr.	3.88	1998
4	Harry Potter and the Chamber...	J.K. Rowling	4.42	1999
5	Rescuing Sprite: A Dog Lover'...	Mark R. Levin	3.93	2007
6	The Five People You Meet in ...	Mitch Albom	3.94	2006
7	For One More Day	Mitch Albom	4.1	2006
8	To Kill a Mockingbird	Harper Lee	4.28	1961
9	Seabiscuit: An American Lege...	Laura Hillenbrand	4.22	2002

postgres/postgres@goodreads_db

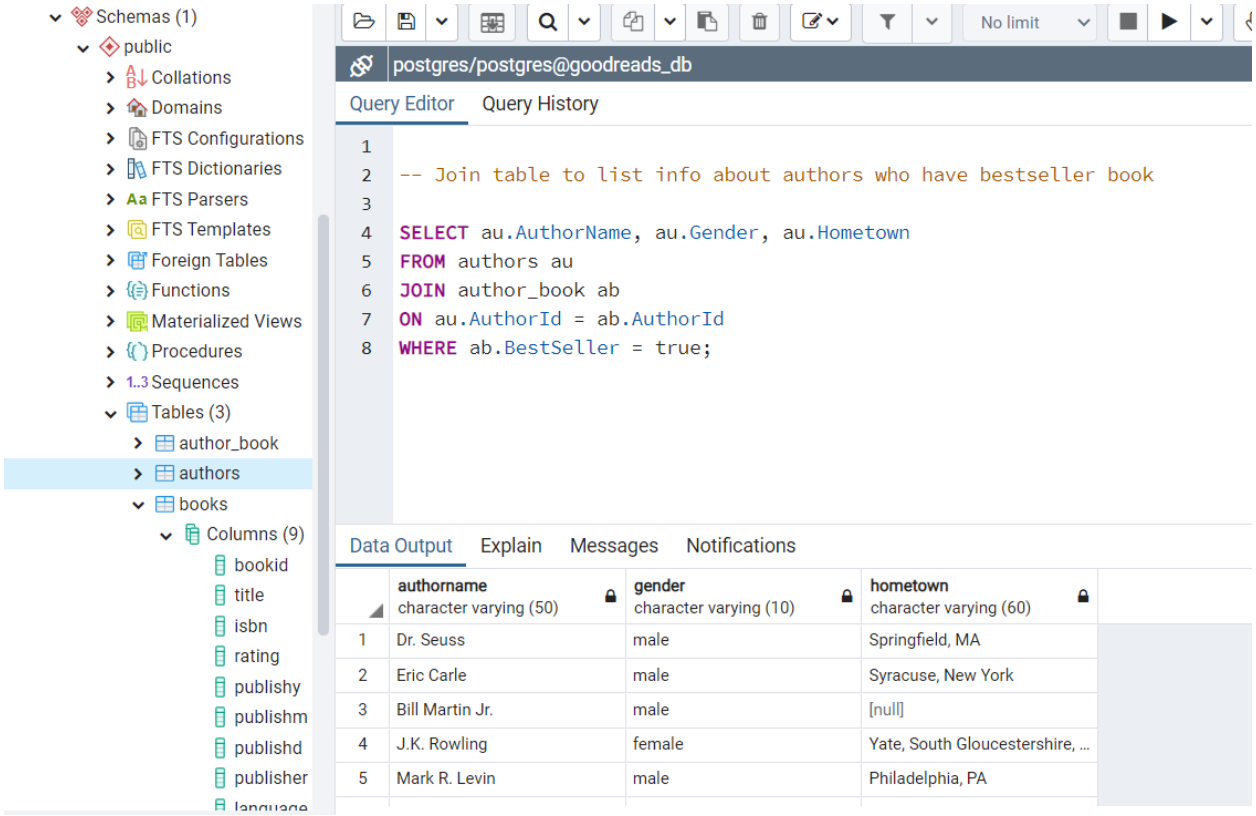
Query Editor | Query History

```

-- Join table to list the detail info about bestseller books from 1990 - 2019 in the dataset
SELECT b.Title, b.Rating, b.PublishY
FROM books b
JOIN author_book ab
ON b.BookId = ab.BookId
WHERE   ab.BestSeller = true
        AND (b.PublishY >= 1990 OR b.PublishY <= 2019);
```

Data Output | Explain | Messages | Notifications

	title character varying (500)	rating double precision	publishy integer
1	The Lorax	4.35	1971
2	The Very Busy Spider	4.21	1985
3	Here Are My Hands	3.88	1998
4	Harry Potter and the Chamber...	4.42	1999
5	Rescuing Sprite: A Dog Lover'...	3.93	2007
6	The Five People You Meet in ...	3.94	2006
7	For One More Day	4.1	2006
8	To Kill a Mockingbird	4.28	1961
9	Seabiscuit: An American Lege...	4.22	2002


Schemas (1)
public
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Procedures
Sequences
Tables (3)
author_book
authors
books
Columns (9)
bookid
title
isbn
rating
publishy
publishm
publishd
publisher
No limit
postgres/postgres@goodreads_db
Query Editor
Query History
-- Join table to list info about authors who have bestseller book
SELECT au.AuthorName, au.Gender, au.Hometown
FROM authors au
JOIN author_book ab
ON au.AuthorId = ab.AuthorId
WHERE ab.BestSeller = true;
Data Output
Explain
Messages
Notifications
authorname character varying (50)
gender character varying (10)
hometown character varying (60)
1 Dr. Seuss male Springfield, MA
2 Eric Carle male Syracuse, New York
3 Bill Martin Jr. male [null]
4 J.K. Rowling female Yate, South Gloucestershire, ...
5 Mark R. Levin male Philadelphia, PA