MIDTERM REPORT

Done by: Gao WanQian

Link to lab:
https://hub.labs.coursera.org:443/connect/sharedszolnxit?forceRefresh=false&path=%2F%3Ffolder%3D%2Fhome%2Fcoder%2Fproject

Steps to run file:

1) node create_db.js ------ create the database
2) node populate_db.js ------ connect to the database, create tables and populate them with csv data
3) node index.js ------ run the webapp

Stage 1: Find and critique a dataset

Dataset: https://www.kaggle.com/datasets/farehasultan/bookstore?resource=download

For this midterm project, I'm using a single csv file, whose data is open and not yet normalised. This csv file is a book Store dataset is downloaded from Kaggle, whose data is retrieved from BookDespository. It contains 14 columns, and 537 rows of data.

Columns:
Title – Stores book titles
URL – Stores the URL of the book from bookdepository.com
Image – Stores the image URL of the book
Format – States the book format
Year – Stores the year the book is published
Price – Stores the price of the book
Author – Stores the author of the book
Rating – Stores the rating of the book
Publisher – Stores the publisher company of the book
Length – Stores the length of the book (no. of pages)
ISBN – Stores the unique ISBN of the book
City/Country – Stores the author of the book
Categories – Stores the category the book belongs in
Quantity – Stores the amount of stocks of the book

At a glance, there seems to be quite a number of repetitive data in the csv file, as well as multiple data in a simple column. All of these have to be dealt with to ensure that the database meets the normalization requirements.

Assessing the dataset:

The dataset should first be assessed based on its quality, detail, documentation, interrelation, use, discoverability. Firstly, for the quality of the dataset, it is quite a complete dataset with 14 columns and 500 over rows, with minimal blank entries or data in the wrong place. The dataset is quite detailed, with the necessary data (title, isbn, author, publisher, category, quantity, price), as well as some extra details for further clarity (year, length, rating, city/country, url, image, format). These data are detailed enough, yet not overly complicated, for me to create a database for a bookstore.

Even though there is no documentation of what the dataset means, there were links provided for each book, which leads to bookdepository.com, where the data was retrieved from. The site contains an explanation on what the columns of the dataset means under its product details section. Since this dataset is rather sufficient for this midterm project to build a database, I would not be utilizing other datasets. It also not be a good choice to utilize other datasets with this bookstore dataset, as it requires the data to be correlated, whereby both dataset refers to the same books.

It was rather difficult to find a dataset that is suitable for this project. Many of them had either too much irrelevant or confusing data, or having too little data for the normalization process. There were other alternative datasets, but I went ahead with this as I felt that it was the most complete, with all the relevant fields.

Copyright/Licence:

The above image is from BookDepository.com's copyright notice. It states that :

"Permission is granted to electronically copy and print in hard copy portions of this Website for the sole purpose of placing an order with The Book Depository, or using this Website as a shopping resource."

Since I'm using their data electronically as a shopping resource, it doesn't violate their terms and conditions.

The dataset is rather confusing when displayed as a csv file, and there are quite a bit of repetitive data.

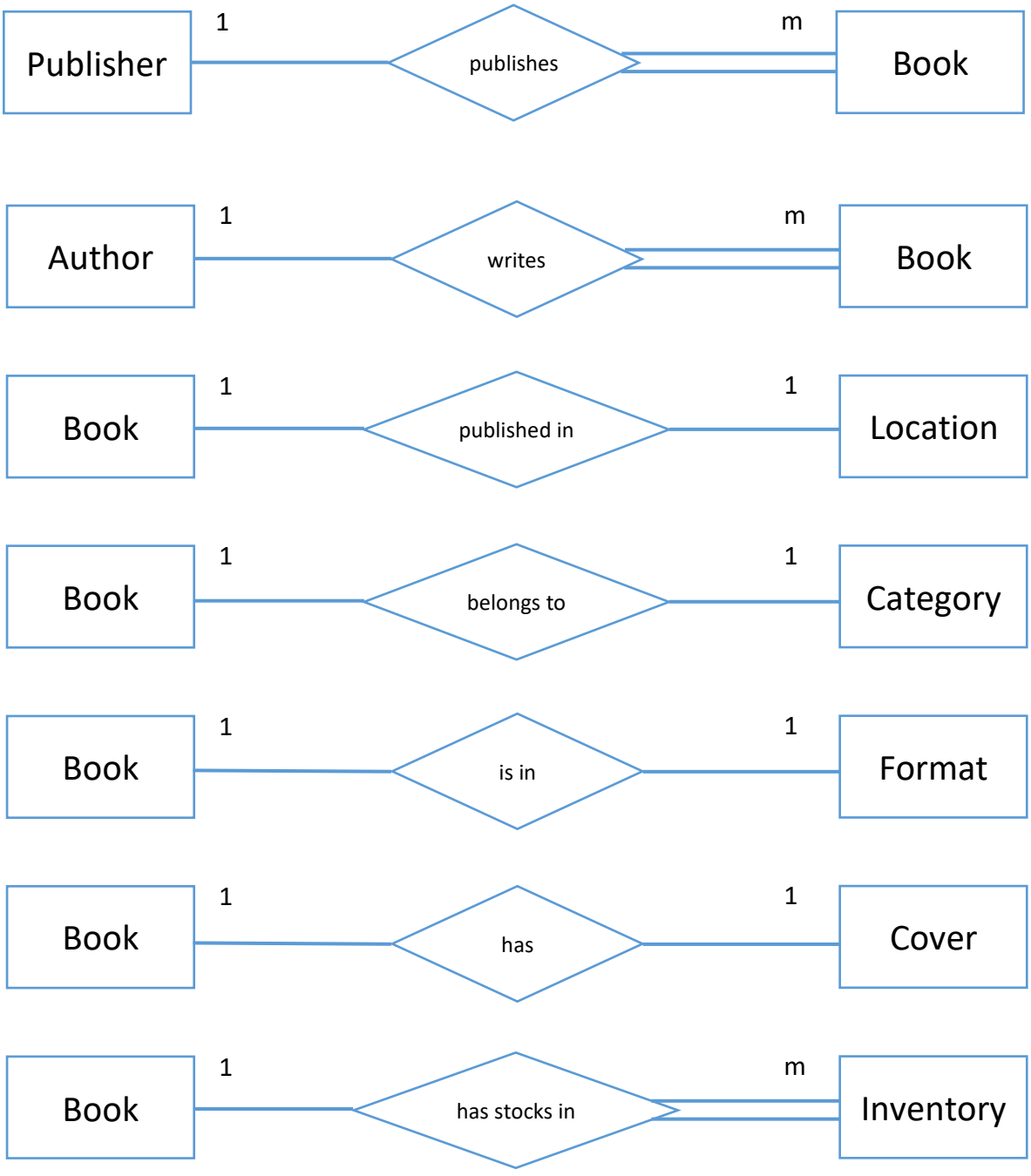
Questions I would like to ask the dataset but can't:

- What if I would want to sort the rows by a particular column? (in either ascending or descending order)
- What if I would want to search for a book title?
- What if I would want to search for books written by a particular author?
- What if I would want to search for books published by a particular publisher?
- What if I would want to search for books by a particular category?
- What if I only want to look through a particular column? (eg. authors, publishers, formats available)

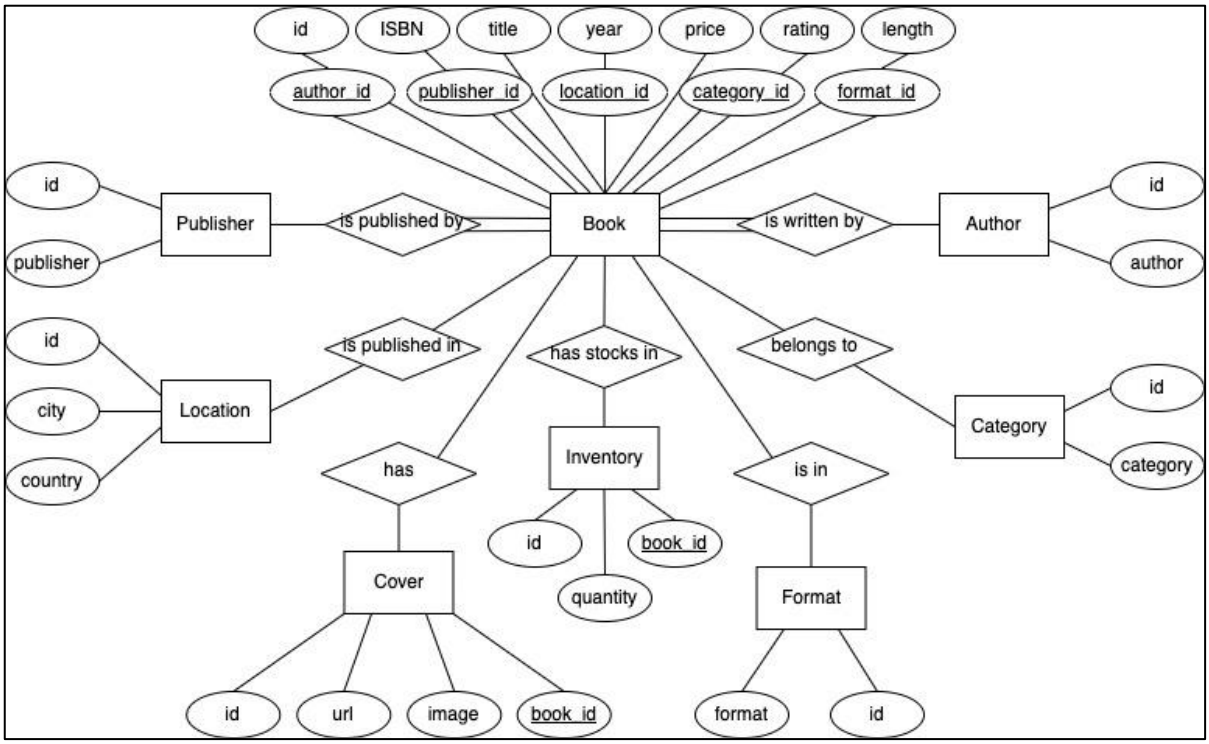
There questions could not be answered with just a plain and unorganized csv data, therefore the need to normalize it and create a simple web application for it.

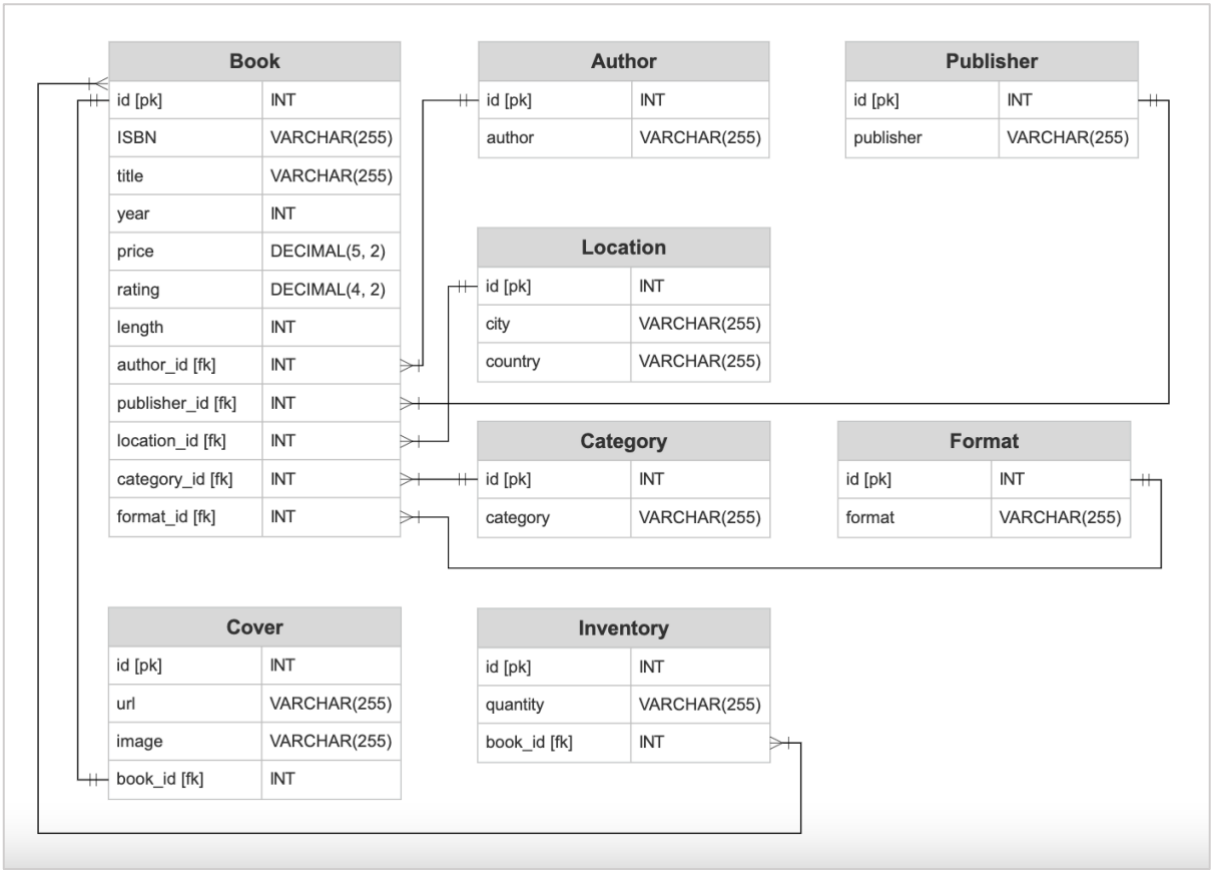
Stage 2: Model your data

E/R model:



Book and Inventory should have an m:n relationship, but for this dataset, a 1:m relationship is sufficient as each book is only stored in 1 inventory.


id
ISBN
title
year
price
rating
length
author_id
publisher_id
location_id
category_id
format_id
Book
Publisher
id
publisher
is published by
is written by
Author
id
author
Location
id
city
country
is published in
has stocks in
belongs to
Category
id
category
has
Inventory
is in
id
book_id
quantity
Cover
Format
id
url
image
book_id
format
id


Book
id [pk] INT
ISBN VARCHAR(255)
title VARCHAR(255)
year INT
price DECIMAL(5, 2)
rating DECIMAL(4, 2)
length INT
author_id [fk] INT
publisher_id [fk] INT
location_id [fk] INT
category_id [fk] INT
format_id [fk] INT
Author
id [pk] INT
author VARCHAR(255)
Publisher
id [pk] INT
publisher VARCHAR(255)
Location
id [pk] INT
city VARCHAR(255)
country VARCHAR(255)
Category
id [pk] INT
category VARCHAR(255)
Format
id [pk] INT
format VARCHAR(255)
Cover
id [pk] INT
url VARCHAR(255)
image VARCHAR(255)
book_id [fk] INT
Inventory
id [pk] INT
quantity VARCHAR(255)
book_id [fk] INT

Each Book is written by mandatory one Author, while each Author can write one to many books.
Each Book is published by mandatory one Publisher, while each Publisher can publish one to many books.
Each Book is published in mandatory one Location, while each Location can have one to many published Books.
Each Book is belongs in mandatory one Category, while each Category can have one to many Books.
Each Book is in mandatory one Format, while each Format can have many Books.
Each Book has mandatory one Cover.
Each Book belongs in mandatory one Inventory, while each Inventory have one to many Books.

city/country
London, United Kingdom
London, United Kingdom
London, United Kingdom
Edinburgh, United Kingdom
London, United Kingdom
New York, United States
London, United Kingdom
Ferntree Gully, VIC, Australia
San Francisco, United States
New York, United States

For this column in the csv file, it contains both city and country data. I split them up into separate fields to store in the location table. This is to ensure that the database meets the First Normal Form requirements (1NF), whereby all of its attributes are scalar values (no column contains an array or an object).

To meet the Second Normal Form(2NF) requirements, every non-key attribute must be irreducibly dependant on the primary key. Since each field in the table should have partial dependency with the primary key, I created 2 new tables, namely Cover and Inventory. Cover table stores all information about the book (url, image), while Inventory table stores the quantity/stocks the book has.

For a database to be in Third Normal Form(3NF), every non-key attribute is non-transitively (directly) dependant on the primary key. Author, publisher, location, category and format in the csv data are rather repetitive. Thus I created new tables for each of these fields, where only unique data is stored in the tables.

Stage 3: Create the database

Creating the database:
I have included a create_db.js file that create a database called Book when the file is ran:

```
db.query("CREATE DATABASE IF NOT EXISTS Books", function (err, result)
  if (err) throw err;
  console.log("Books database created");
});
```

Command to run file: node create_db.js

Creating the tables:
After the database is created, I will first need to connect to the database and create tables. These code are stored in populate_db.js file, which creates and populates the database.

```
const db = mysql.createConnection({
    host: "localhost",
    user: "root",
    password: "password",
    database: "Books"
});
```

Connect to Books database with root user and password.

Next create the author, publisher, location, category and format tables. Each field would store data in the appropriate data structure.

```
var authorTable = "CREATE TABLE IF NOT EXISTS author (" +
                  "id INT UNSIGNED AUTO_INCREMENT NOT NULL," +
                  "author VARCHAR(255) NOT NULL," +
                  "PRIMARY KEY(id))";
```

```
var publisherTable = "CREATE TABLE IF NOT EXISTS publisher (" +
                     "id INT UNSIGNED AUTO_INCREMENT NOT NULL," +
                     "publisher VARCHAR(255) NOT NULL," +
                     "PRIMARY KEY(id))";
```

```
var locationTable = "CREATE TABLE IF NOT EXISTS location (" +
                    "id INT UNSIGNED AUTO_INCREMENT NOT NULL," +
                    "city VARCHAR(255) NOT NULL," +
                    "country VARCHAR(255) NOT NULL," +
                    "PRIMARY KEY(id))";
```

```
var categoryTable = "CREATE TABLE IF NOT EXISTS category (" +
                    "id INT UNSIGNED AUTO_INCREMENT NOT NULL," +
                    "category VARCHAR(255) NOT NULL," +
                    "PRIMARY KEY(id))";
```

```
var formatTable = "CREATE TABLE IF NOT EXISTS format (" +
                  "id INT UNSIGNED AUTO_INCREMENT NOT NULL," +
                  "format VARCHAR(255) NOT NULL," +
                  "PRIMARY KEY(id))";
```

The book table have to be created after the 5 tables above, as it utilizes foreign keys that references to the primary key of the 5 tables.

```
var bookTable = "CREATE TABLE IF NOT EXISTS book (" +
                "id INT UNSIGNED AUTO_INCREMENT NOT NULL," +
                "ISBN VARCHAR(255) NOT NULL," +
                "title VARCHAR(255) NOT NULL," +
                "year INT UNSIGNED NOT NULL," +
                "price DECIMAL(5, 2) UNSIGNED NOT NULL," +
                "rating DECIMAL(4, 2) UNSIGNED NOT NULL," +
                "length INT UNSIGNED," +
                "author_id INT UNSIGNED NOT NULL," +
                "publisher_id INT UNSIGNED NOT NULL," +
                "location_id INT UNSIGNED NOT NULL," +
                "category_id INT UNSIGNED NOT NULL," +
                "format_id INT UNSIGNED NOT NULL," +
                "PRIMARY KEY(id)," +
                "FOREIGN KEY (author_id) REFERENCES author(id)," +
                "FOREIGN KEY (publisher_id) REFERENCES publisher(id)," +
                "FOREIGN KEY (location_id) REFERENCES location(id)," +
                "FOREIGN KEY (category_id) REFERENCES category(id)," +
                "FOREIGN KEY (format_id) REFERENCES format(id))";
```

author_id, publisher_id, location_id, category_id, format_id are foreign keys of the book table.

Next up is the cover table that stores the book's url, cover image, and a foreign key referencing to the book table. This is to link the book data to the cover correctly.

```
var coverTable = "CREATE TABLE IF NOT EXISTS cover (" +
                 "id INT UNSIGNED AUTO_INCREMENT NOT NULL," +
                 "url VARCHAR(255) NOT NULL," +
                 "image VARCHAR(255) NOT NULL," +
                 "PRIMARY KEY(id)," +
                 "book_id INT UNSIGNED NOT NULL," +
                 "FOREIGN KEY (book_id) REFERENCES book(id))";
```

Finally, for the inventory table, it stores the quantity of the book. This table also has a foreign key called book_id that references book table's primary key.

```
var inventoryTable = "CREATE TABLE IF NOT EXISTS inventory (" +
                     "id INT UNSIGNED AUTO_INCREMENT NOT NULL," +
                     "quantity INT UNSIGNED NOT NULL," +
                     "PRIMARY KEY(id)," +
                     "book_id INT UNSIGNED NOT NULL," +
                     "FOREIGN KEY (book_id) REFERENCES book(id))";
```

Populating the database:

```
// CSV file name
const csvFile = "data/book_dataset.csv";

csvtojson().fromFile(csvFile).then(source => {
    // initialise empty list to store unique fields
    var author_list = [];
    var author_full =[];

    var publisher_list = [];
    var publisher_full =[];

    var location_list = [];
    var location_full =[];
```

After setting up the tables and database, I used csvtojson() function to read the csv file and populate the database. Empty lists are initialized to store each field of data.

```
// get field data csv
for (var i = 0; i < source.length; i++) {
    // Fetching author name data from each row
    var author = source[i]["author"];

    // Fetching publisher name data from each row
    var publisher = source[i]["publisher"];

    // Fetching location table data from each row
    var location = source[i]["city/country"];
```

By looping through the data, each field data is fetched and pushed into their respective lists afterwards.

```
if (author_list.includes(author) === false) author_list.push(author);
if (publisher_list.includes(publisher) === false) publisher_list.push(pu
if (category_list.includes(category) === false) category_list.push(categ
if (format_list.includes(format) === false) format_list.push(format);
```

Some of the fields have repetitive data, and therefore only unique data are pushed into the lists.

For these fields, there are also lists initialized to save the full 500+ rows of repetitive data, in order. These are used to get the index of the rows in the later part.

```
// get author field data (stores author's name) from csv
for (var i = 0; i < author_list.length; i++) {
    // insert command
    var sqlinsert = `INSERT INTO author values(?, ?)`;
    // data to be inserted. since the if starts at 1, increment i by 1
    var data = [i+1, author_list[i]];
    //console.log(data);

    // Inserting data of current row into database
    db.query(sqlinsert, data,
        (err, results, fields) => {
        if (err) {
            console.log("Unable to insert author item at row ", i + 1);
            return console.log(err);
        }
    });
}
```

The data are than inserted by looping through the populated lists.

For the location column in the csv file, I have to split the city and country data up, and insert them separately into the location table.

```
var city_country_pair = location_list[i].split(", ");

// empty string in row, push NULL into the lists
if(city_country_pair == "") {
    city.push("NULL");
    country.push("NULL");
}
// only have 1 element, push NULL into city list and c
// for this csv data, if there is only 1 element in th
else if(city_country_pair.length == 1) {
    city.push("NULL");
    country.push(city_country_pair[0]);
}
// 2 elements, push city and country into the lists
else if(city_country_pair.length == 2) {
    city.push(city_country_pair[0]);
    country.push(city_country_pair[1]);
}
```

-- split data with comma as separator

Push null if there is no location data in that row

If there is only 1 element, push it into country, and push null into city.

If there are exactly 2 elements, that means that both city and country data exist, push them both normally into city and country lists.

However, things gets a little complicated if there were more than 2 elements in the split data.

```
else if(city_country_pair.length > 2) {
    for(var x = 0; x < city_country_pair.length - 1; x++) {
        // if there is only 1 last element to concatenate, dont add
        if(x == city_country_pair.length - 2) {
            city_temp = city_temp + city_country_pair[x];
        }
        // add comma behind each city
        else {
            city_temp = city_temp + city_country_pair[x] + ", ";
        }

    }
    // push the concatenated string, city_temp, into the city list
    city.push(city_temp);
    // push last element as country
    country.push(city_country_pair[city_country_pair.length-1]);
}
```

A city_temp variable is create at the beginning. This variable is used to store the concatenated city data. A loop is used to check how many more elements are there in the field. If there is more than 1 left to concatenate to the variable, add a comma behind it, else if there is only 1 element left, simple add it at the back of the string.

In this case, the country data would be the last element in the split string.

Since there are 5 foreign keys in the book table, their data have to be retrieved correctly too. To do so, using the full field data (repetitive), compare it to the unique field data (non-repetitive), and get the correct id.

```
for(var a = 0; a < author_list.length; a++) {
    if(author_full[i] == author_list[a]) {
        author_id = a+1;
    }
}

for(var a = 0; a < publisher_list.length; a++) {
    if(publisher_full[i] == publisher_list[a]) {
        publisher_id = a+1;
    }
}
```

This id is than inserted into the book, along with the other book field data.

```
// insert command
var sqlinsert = `INSERT INTO book values(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
// data to be inserted. since the if starts at 1, increment i by 1
var data = [i+1, ISBN_list[i], title_list[i], year_list[i], price_list[i],
            author_id, publisher_id, location_id, category_id, format_id];
//console.log(data);

// Inserting data of current row into database
db.query(sqlinsert, data,
    (err, results, fields) => {
    if (err) {
        console.log("Unable to insert book item at row ", i + 1);
        return console.log(err);
    }
});
```

The database is now correctly populated.

Command to run file: node populate_db.js

Book table:

(full table of first 5 rows)

```
mysql> SELECT * FROM book LIMIT 5;
+----+---------------+-----------------------------------------------------------------+------+-------+--------+--------+-----------+--------------+-------------+-----------+
| id | ISBN          | title                                                           | year | price | rating | length | author_id | publisher_id | location_id | category_id | format_id |
+----+---------------+-----------------------------------------------------------------+------+-------+--------+--------+-----------+--------------+-------------+-------------+-----------+
|  1 | 9781398515628 | The Spanish Love Deception                                      | 2021 | 15.74 |   5.00 |    480 |         1 |            1 |           1 |           1 |         1 |
|  2 | 9781471156267 | It Ends With Us: The most heartbreaking novel you'll ever read  | 2016 | 16.77 |   4.42 |    384 |         2 |            1 |           1 |           2 |         1 |
|  3 | 9781529360684 | Immune                                                          | 2021 | 50.66 |   4.58 |    368 |         3 |            2 |           1 |           3 |         2 |
|  4 | 9781786892737 | The Midnight Library                                            | 2021 | 15.00 |   4.09 |    304 |         4 |            3 |           2 |           2 |         1 |
|  5 | 9781471136726 | Ugly Love                                                       | 2016 | 19.28 |   4.26 |    352 |         2 |            1 |           1 |           2 |         1 |
+----+---------------+-----------------------------------------------------------------+------+-------+--------+--------+-----------+--------------+-------------+-------------+-----------+
```

(for clearer view)

```
mysql> SELECT * FROM book LIMIT 5;
+----+---------------+-----------------------------------------------------------------+------+-------+
| id | ISBN          | title                                                           | year | price |
+----+---------------+-----------------------------------------------------------------+------+-------+
|  1 | 9781398515628 | The Spanish Love Deception                                      | 2021 | 15.74 |
|  2 | 9781471156267 | It Ends With Us: The most heartbreaking novel you'll ever read  | 2016 | 16.77 |
|  3 | 9781529360684 | Immune                                                          | 2021 | 50.66 |
|  4 | 9781786892737 | The Midnight Library                                            | 2021 | 15.00 |
|  5 | 9781471136726 | Ugly Love                                                       | 2016 | 19.28 |
+----+---------------+-----------------------------------------------------------------+------+-------+

--------+--------+-----------+--------------+-------------+-------------+-----------+
 rating | length | author_id | publisher_id | location_id | category_id | format_id |
--------+--------+-----------+--------------+-------------+-------------+-----------+
   5.00 |    480 |         1 |            1 |           1 |           1 |         1 |
   4.42 |    384 |         2 |            1 |           1 |           2 |         1 |
   4.58 |    368 |         3 |            2 |           1 |           3 |         2 |
   4.09 |    304 |         4 |            3 |           2 |           2 |         1 |
   4.26 |    352 |         2 |            1 |           1 |           2 |         1 |
--------+--------+-----------+--------------+-------------+-------------+-----------+
```

Even though the database had now been populated and normalized, there are some parts that are not as intuitive. For example the book table above, if we want to know the author for a particular book, we have to look for the author data in the author table, which is equal to the author_id provided in the book table.

Therefore the web app is made to be able to look at all the fields at one glance, able to sort them by each field, and search boxes to locate books/authors/publishers/categories. The sql select query returns the foreign key datas correctly.

With the author, publisher, location, category and format tables, it reduces the amount of space needed for the database, as there are now little to no data being pointlessly repeated. Minimizing the amount of repetitive data is good, especially when storage is limited or when the database gets large with more data being inserted in.

<u>These questions are answered with the sql queries stores in routes/main.js file</u>

- What if I would want to sort the rows by a particular column? (in either ascending or descending order)

```
function sortTable(n) {
    var table = document.getElementById("table");
    var count = 0;
    var switching = true;

    // Order is set as ascending
    var direction = "ascending";
```

This is from public/js/scripts.js file. This function sorts the tables in by ascending or descending order. The parameter n is the signify based on which column the table should be sorted on.

```
<button onclick="sortTable(2)">Sort by title</button>
```

A button is created and when clicked, the function runs and sorts the table according to the n value parsed in.

- What if I would want to search for a book title?
- What if I would want to search for books written by a particular author?
- What if I would want to search for books published by a particular publisher?
- What if I would want to search for books by a particular category?

These questions can be answered with similar sql queries. An example below shows the search via book title:

```
<form action="/" method="POST">
    <input type="text" name="keyword" placeholder="Search by title">
    <input type="submit" value="Search">
</form>
```

In the index.ejs file, a form that acts as a search box is created, which POST the keyword entered to the path('/').

```
p.post("/", function (req, res) {
  // search for the keyword in the database
  let keyword = [req.body.keyword]
  //get request that sends the render of the index.ejs page
  let sqlquery = "SELECT id, ISBN, title, year, price, rating, length FROM book WHERE title LIKE '%" + keyword + "%';" +
                 "SELECT author.author FROM author INNER JOIN book ON author.id = author_id WHERE title LIKE '%" + keyword + "%';" +
                 "SELECT publisher.publisher FROM publisher INNER JOIN book ON publisher.id = publisher_id WHERE title LIKE '%" + keyword
                 "SELECT location.city, location.country FROM location INNER JOIN book ON location.id = location_id WHERE title LIKE '%" +
                 "SELECT category.category FROM category INNER JOIN book ON category.id = category_id WHERE title LIKE '%" + keyword + "%'
                 "SELECT format.format FROM format INNER JOIN book ON format.id = format_id WHERE title LIKE '%" + keyword + "%';" +
                 "SELECT cover.url, cover.image FROM cover INNER JOIN book ON book.id = book_id WHERE title LIKE '%" + keyword + "%';" +
                 "SELECT inventory.quantity FROM inventory INNER JOIN book ON book.id = book_id WHERE title LIKE '%" + keyword + "%';"

  db.query(sqlquery, function (err, result, fields) {
      if (err) throw err;
      else {
          //console.log(result[0]);
          //console.log(result[1]);
          res.render("index.ejs", {book: result[0], author: result[1],
                                   publisher: result[2], location: result[3],
                                   category: result[4], format: result[5],
                                   cover: result[6], inventory: result[7], search: keyword});
      }
  });
```

In the main.js file, there get and post requests are managed with sql queries that returns the data back to the ejs file for display.

The foreign key are inner joined with book where the foreign key id matched the referenced table id. In this case, the keyword in title field is than searched through these inner joined tables, and returned. It would result in the relevant rows being returned and displayed.

- What if I only want to look through a particular column? (eg. authors, publishers, formats available)

```
//get request that sends the render of the author.ejs page
app.get("/author", function (req, res) {
    // get all author fields
    let sqlquery = "SELECT * FROM author"
    db.query(sqlquery, function (err, result, fields) {
        if (err) throw err;
        else {
            res.render("author.ejs", {author: result});
        }
    });
});


app.post("/author", function (req, res) {
    // search for the keyword in the database
    let keyword = [req.body.keyword]
    let sqlquery = "SELECT * FROM author WHERE author LIKE '%" + keyword + "%'"
    // execute sql query
    db.query(sqlquery, function (err, result, fields) {
        if (err) throw err;
        else {
            res.render("author.ejs", {author: result, search: keyword});
        }
    });
});
```

Each table in the database has a separate ejs file. For example, all the data in the author table are selected and returned as result to the ejs page, which is than displayed as a table on the webpage. There is also a search box that allows the searching of, in this case, author.

Stage 4: Create a simple web application

User privileges:

create_db.js:

```
CREATE USER 'root'@'127.0.0.1';
ALTER USER 'root'@'127.0.0.1' IDENTIFIED WITH mysql_native_password BY 'password';
GRANT ALL ON *.* TO 'root'@'127.0.0.1' WITH GRANT OPTION;

user: root
password: password
```

Use root user that has access to all privileges to create the database.

populate_db.js:

```
CREATE USER 'user'@'127.0.0.1';
ALTER USER 'user'@'127.0.0.1' IDENTIFIED WITH mysql_native_password BY 'password';
GRANT INSERT, SELECT, DROP, CREATE, REFERENCES ON Books.* TO 'user'@'127.0.0.1';

user: user
password: password
```

Use user that has access to insert, select, drop, create, references, which are necessary to create tables and populate them.

index.js:

```
CREATE USER 'test_user'@'127.0.0.1';
ALTER USER 'test_user'@'127.0.0.1' IDENTIFIED WITH mysql_native_password BY
'password';
GRANT SELECT ON Books.* TO 'test_user'@'127.0.0.1';

user: test_user
password: password
```

Use test_user that only has access to select, as the only priviledge needed to get the data from the database and display It on the webapp is select. Limiting user privilege ensures that data in the database won't be altered or deleted either by accidents or attacks.

Command to run file: node index.js