CSE4701

Project 2, Part 2

Mike Medved

November 24th, 2023

Table of Contents

Introduction

This assignment reuses some of the logic for generating data from the previous assignment. As you can see below, the code is the same from last assignment.

```
// Enumerate all JSON files in the data directory and get their contents.
let files = readdirSync('./data')
    .filter(file => file.endsWith('.json'))
    .map(file => `./data/${file}`)
    .map(path => readFileSync(path, 'utf-8'))
    .map(json => JSON.parse(json))
    .map(payload => payload as BookJson);

// Enumerate all JPG files in the data directory and get their binary contents.
let images = readdirSync('./data')
    .filter(file => file.endsWith('.jpg'))
    .map(file => `./data/${file}`)
    .map(path => ({
        bookId: path.split(/(B d+)/g)[1],
        blob: readFileSync(path)
    }));

// Coalesce the books and images into a map of book ID to book and image.
let bookMap: BookMap = files.reduce((acc, book) => {
    acc[book.id] = { book };

    let image = images.find(image => image.bookId === book.id);
    if (image) acc[book.id].image = image.blob;

    return acc;
}, {} as BookMap);

// Create a database record for each book in the above map.
let records = await Promise.all(
    Object.keys(bookMap).map(async bookId => {
        let { book, imageBlob } = bookMap[bookId];
        let { author,..rest } = book;

        return await client.book.create({
            data: {
                name: book.bookname,
                book_id: bookId,
                image: imageBlob,
                description: rest,
            }
        });
    })
);
```

In order to initialize the SQL database, we will use the *mysqldump* command to dump the state of the database as of Project 1 Part 1, and then reimport it into this database instance. Note that this is required because each assignment's databases are containerized via Docker and thus do not otherwise overlap. The commands used are shown below.

```
# dump from project 1 part 1 database
mysqldump -h 127.0.0.1 -u root -p Book_Loan_DB > proj1p2.sql

# reimport into project 2 part 2 database
mysql -h 127.0.0.1 -u root -p Book_Loan_DB < ./proj1p2.sql
```

Question 1, Cross-Fetching Book Data

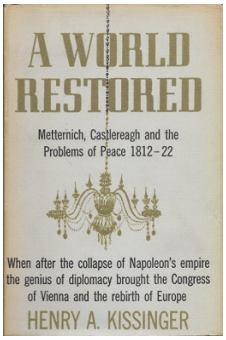
We will be using the *mariadb* npm package to interface with the MySQL database. Omitted from the below code snippet is setting up the connection pool upon which we can get a connection to perform queries. The code below highlights the query we perform as well the MongoDB *findUnique* call to retrieve the image data for the associated book ID.

Once the data from each database is retrieved and paired together, we iterate over all the books that have images and write to an image file by creating a buffer with the base64 encoded image data, then writing it to the filesystem.

```
let connection = await pool.getConnection();
let rows = await connection.query(`
    SELECT
        l.Book_id, b.Title
    FROM Book_Loans l
    INNER JOIN Borrower u
        ON l.Card_no = u.Card_no
    INNER JOIN Book b
        ON l.Book_id = b.Book_id
    WHERE u.Name="Ramesh Narayan";
`);

let matched = await Promise.all(rows.map(async (row: any) => {
    let book = await client.book.findUnique({ where: { book_id: row.Book_id } });
    if (!book) {
        console.warn(`No match for Book ${row.Book_id} in Mongo!`);
        return null;
    }

    return {
       ..row,
        image: book.image
    };
}));

for (let ent of matched) {
    let { Book_id: id, image } = ent;
    if (!image) continue;

    let buf = Buffer.from(image, 'base64');
    let path = `./covers/${id}.jpg`;
    console.log(`Writing ${path}`);
    writeFileSync(path, buf);
}

console.log('Done.');
```

After executing the above code, a *covers* directory is created with two image files, *B3.jpg* and *B5.jpg*. The images are shown below.

B3.jpg



B5.jpg

Question 2, Book Investment by Branch

In order to compute the total investment for each branch's paper books, we will need to query the Book_Copies table, which will give us the Book IDs, corresponding branches, and number of copies per branch.

Next, we will query all of the books from the MongoDB database and get the paperback price for each book that has that data. Lastly, we will get the investment per-book per-branch and sum it up to get the final branch totals.

```
01 | let connection = await pool.getConnection();
02 | let rows = await connection.query(`SELECT * FROM Book_Copies;`);

04 | const books = await client.book.findMany();
05 | const prices = books
06 |     .map(book => book.description as any)
07 |     .map(json => ({
08 |         id: json.id,
09 |         amount: parseFloat(json.format?.paperback?.substring(1)) ?? 0
10 |     }));

12 | const branchBooks = rows.map((row: any) => {
13 |     const price = prices.find(price => price.id === row.Book_id);
14 |     if (!price) return NaN;
15 |     return {
16 |         branchId: row.Branch_id,
17 |         bookId: row.Book_id,
18 |         investment: price.amount * row.No_of_copies
19 |     }
20 | });

22 | let branches = branchBooks.reduce((acc: any, curr: any) => {
23 |     if (!acc[curr.branchId]) acc[curr.branchId] = 0;
24 |     acc[curr.branchId] += Math.round(curr.investment * 100) / 100;
25 |     return acc;
26 | }, {});

28 | console.table(branches);
```

Below, you can see the output of running the above script - note that the column names in the below table were generated by the built-in JavaScript *console.table* function and are not indicative of how the data is stored or anything of the sort.

(index)	Values
BR1	1124.7
BR2	1919.4
BR3	106.33
BR5	151.9
BR4	198.8

Branch Investment