CEP Weighted Assessment 3: NoSQL Database

Loh Yu Chen 2020

Files

- `cli.py`: Main interface for the app
- `library_api.py`: Interface with the MongoDB database
- `config.py`: Credentials for connection to the database
    - `config_template.py`
- `templates.py`: Templates for book and borrower documents
- `insert_books.py`: Script to insert books based on Google Books ID
- `insert_borrowers.py`: Script to insert borrowers from an online API

Data Model

Books

Fields

- `_id`: `ObjectId`
- `title`: `String`
- `authors`: `Array (String)`
- `isbn`: `String`
- `page_count`: `Int32`

Indexes

- Text index of `title`, `authors`, `isbn` to allow & speed up search

Borrowers

Fields

- `_id`: `ObjectId`
- `name`: `String`
- `username`: `String`
- `phone`: `String`

Indexes

- Text index of `name`, `username` to allow & speed up search

Loans

Fields

- `_id`: `ObjectId`
- `book_id`: `ObjectId`
- `borrower_id`: `ObjectId`
- `returned`: `Boolean`

Indexes

- Compound index of `book_id`, `returned` as they are used together to check whether a book is currently on loan
- Compound index of `borrower_id`, `returned` as they are used together to check for books that a borrower still has on loan

Assumptions & Justifications

Usage pattern

- People will likely use the “checkout” and “return” functions the most as they are the main purpose of a library
    - Hence I allowed multiple books to be loaned/returned at the same time: this allows for more convenience while also reducing the number of operations
- When deleting books/borrowers, some loans may still be active, so all of them are returned automatically
- To show books that exist but are on loan, instead of completely hiding them, I disabled the option to select the book, and displayed the current borrower’s username
    - This makes users aware of the existence of this book, so they can potentially borrow it in the future

How you are accessing your data

- Most operations are through searches, such as borrowing/deleting/editing books, and creating/editing/deleting borrowers
    - Hence text indexes are very important as they support text search queries on string content
    - Text indexes:
        - Books: `title`, `authors`, `isbn`
        - Borrowers: `name`, `username`
- When tracking number of books checked out by a given user, the username is used to identify
    - Hence a `username` index is used as well

Which queries are critical in your application

```
db.loans.find_one({"book_id": book_id, "returned": False})
```

- **`is_checked_out`**: This checks the loans collection for active loans on a given `book_id`, deeming whether it is currently available to check out, otherwise returning the current borrower who has loaned it
    - This makes use of the compound index, making the query efficient

```
db.loans.find({"borrower_id": borrower_id, "returned": False})
```

- `get_borrowed_books`: This gets all active loans of a borrower from the loans collection, for use when creating the “return books” menu, and tracking number of books checked out by a borrower
    - This makes use of the compound index, making the query efficient

```
db[collection].find({"$text": {"$search": query}})
```

- `search_function`: This makes use of the text indexes created in the book and borrower collections, to find documents efficiently and accurately

Ratios between reads and writes

- Reads should far exceed writes
    - Other than the create operations, all other operations need to perform at least one query
    - This includes the commonly-used search function, which reads many documents at one go
- Writes are only used upon creating, editing, loaning and returning

Amount of information stored and how it grows with time

- Each loan is stored in a separate document
    - Hence, documents don’t grow
        - This ensures there are no problems in older versions of MongoDB
        - Better practice to keep documents a relatively consistent size
    - Loans can be easily extended in the future with more information when needed
        - e.g. Date of borrowing/returning
    - However, this will require an additional query and the server has to search around for documents
        - Hence, indexes are used as much as possible (mentioned earlier) to speed up queries and try to mitigate this impact (though, indexes also come with a cost, especially if the database were to grow larger)

Reflection

What are the challenges you faced while doing this project?

- Coming up with a database model: I was conflicted in choosing an appropriate method but I settled for pure linking as I was familiar with it, and it is suitable for a small-scale project like this: Though an extra index has to be created and 2 queries have to be executed, the performance impact is negligible for a small library collection like this project, while making it a lot easier to implement. Hence I thought it was a worthy tradeoff and chose pure linking for this project. Also, because documents don't grow, this model is compatible even with older versions of MongoDB.
- Deciding what to do with missing values and null fields: I was not sure how to handle missing fields after being removed, or fields left empty. In the end, I decided to display both in the app as `<null>`.
- Command line interface: It was difficult to create an easy-to-use interface through just the command line. I ended up using `PyInquirer` to display input questions, allowing arrow navigation and checkbox selections.

Does knowledge of relational design methods help you in any way, or otherwise?

Yes, especially since I used pure linking - a relational approach which I was already familiar with, thanks to earlier relational database experience and lessons. However, I also had to get used to the document and collection model of NoSQL, compared to the tables and rows in relational databases.

Do you think this type of application is more suitable to be implemented as a relational DB or non-relational document-based DB like MongoDB? Why?

I think this type of application is more suitable to be implemented as a non-relational database. Firstly, the document model allows new fields and information to be added to books/borrowers at ease, while using a relational table would require schema changes and such. Sometimes, a document may not need the same columns as another too. Also, if this application were to scale, for example to accommodate a larger public library, it would be a lot easier in NoSQL, besides providing superior performance[1].

Any other comments?

- Missing fields and null values are both displayed as `<null>`

[1] https://www.mongodb.com/nosql-explained/advantages

- When creating/editing a new book/borrower, fields left empty will not be set (creating a missing field)
- When deleting a book/borrower, all loans related to it are automatically returned
- The feature “track which user has checked out a book” is embedded in the “checkout” function: when a book that comes up in search is on loan, it will show the username of the borrower
- A limitation of my data model is that a query is required for each book to determine whether it is on loan, which can be very expensive