Big Data - Page Python Project

Mateusz Tabaszewski 151945 Bartłomiej Pukacki 151942

This is the report for the final Big Data project - Page Python. Page Python is an application utilizing a distributed database system with the help of Cassandra to allow for fast and convenient booking of library books. The system utilizes both the positives of Cassandra and our own Python implementations to ensure fast and reliable booking, update, and cancellation of book reservations.

The system utilizes three nodes, connected with one another. The exact method of creating those nodes is shown in the docker-compose.yaml file and it includes a command to expose appropriate ports to speed up the described operations. The system can, if necessary, be extended to more nodes. The system utilizes a RetryPolicy and an ExponentialReconnectionPolicy. Furthermore, when testing the system the consistency level of “TWO” is utilized along with a replication factor of two, however, this could be changed depending on the size of the system. The system allows the users to add a new reservation, cancel an existing reservation, and update the already existing reservation which includes updating either the user who made the reservation (in case of an error on the part of the user logging the reservation) or updating the reserved book.

To allow for such manipulations to be done in an efficient way we have defined a schema as shown below in Figure 1. We have defined three tables: reservations, users, and books. Reservations hold the information about the currently existing reservations. The reservations table includes the information about the reservation_id (primary key), user_id, user_name, book_name, and book_id. Furthermore, the users table includes the same information as the reservations table, however, its primary key is the user_id and the clustering column is reservation_id. This setup should allow for quick recovery of the information about the concrete user’s reservations. The last table named “books” holds the information about book_id, book_name, and is_reserved. The is_reserved field holds the information on whether or not the book is currently reserved as a boolean value.

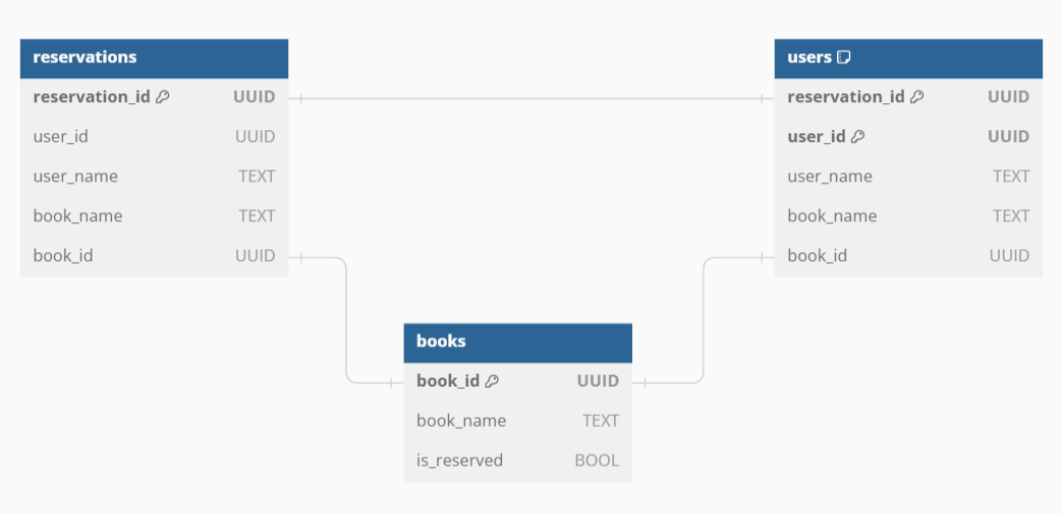

Figure 1. Database schema for the Page Python distributed library system. It is made up of three tables: reservations, users, and books with reservations and users holding the same information but under different keys with the primary key for reservations being the reservation_id the primary key for users being the user_id, and reservation_id being the clustering column.

It is important to note that stress tests have been conducted on the database to check the viability of our solution. The tests included:

- The client makes the same request very quickly (update the reservation book 10000 times).
- Two clients make the possible requests randomly (10000 times).
- Immediate occupancy of all seats/reservations by two clients.
- Constant cancellations and reservation of a book (for the same reservation 10000 times).
- Update of 1000 reservations.

All tests have been passed successfully and swiftly with the single longest test taking at most around 90 seconds and the shortest test taking around 17 seconds. However, please take into account that running the tests as a whole can take much longer due to creating and setting up the database for the appropriate conditions for testing.

In the process of creating the application stress tests revealed an issue with the initial database schema we implemented mostly due to working with a user table containing lists of user's reservation IDs which took a prohibitively long time to update. Due to this concern, we had to propose another way of handling the user table which is presented in Figure 1. This new approach proved to be significantly faster and helped us finish stress tests in a reasonable amount of time. Other issues that appeared during testing were solved by adjusting the policy of the system and introducing a thread lock in immediate reservation occupancy by two clients. Another problem was caused by issues with the customtkiner and tkinter python libraries which happened to have broken functionality concerning copying values to clipboard. This obstacle was circumvented by allowing for printing IDs to the console from which the values can be copied.

In conclusion, we believe that we have managed to accomplish the task of creating a distributed system for a library using Cassandra. Furthermore, we hope that our efforts may be useful for anyone willing to check our solution for guidance on building a simple distributed system.