BDDS Final Project Report

Vasyl Korzavatykh, ID: 158669

Description

For the final project I developed a console application simulating the reservation process in the cinema. I used Apache Cassandra for this project.

Project structure

The project has 5 main files:

1. **docker-compose.yml** – run with *docker compose up -d* to add and start the nodes cas1 and cas2 (from the image cassandra:latest)
2. **utils.py** – file with all the utility functions shared among files described below
3. **main.py** – file that creates initial keyspace creates / re-creates the tables and populates the tables with sample movies / screenings
4. **stress_tests.py** – file that runs 3 stress tests listed in the project requirements
5. **user_console.py** – entrypoint for a sample user that can perform any of the actions described in the section below

Actions

Sample user can:

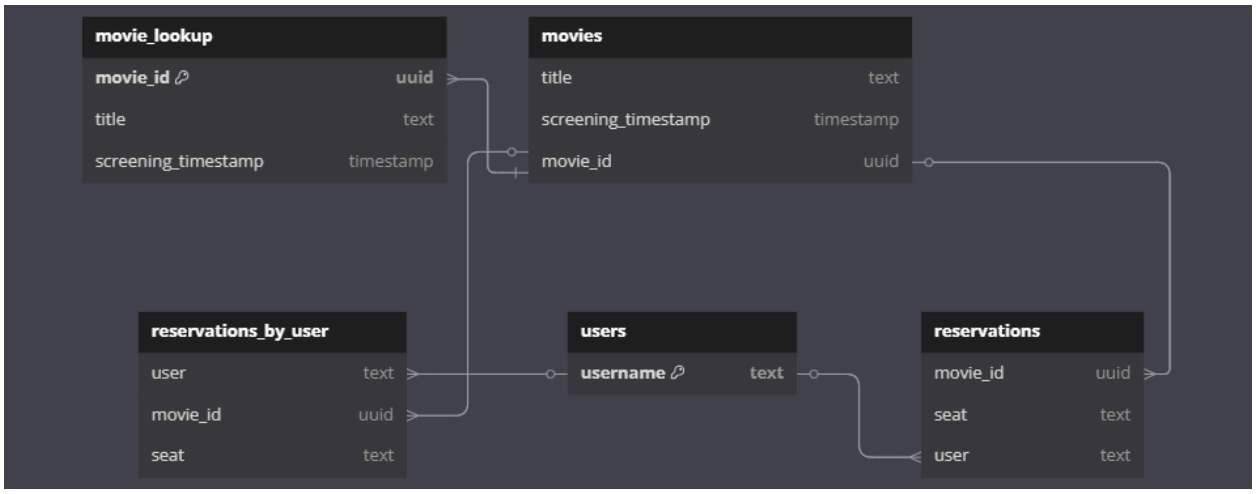
- Make a reservation (choose the movie, screening (time), and a seat)
- View all reservations (made by all the users in the system)
- View their reservations
- Cancel their reservation

Technical setup

- There are 2 nodes working (cas1 and cas2)
- System functions on any node (the setup is made using cassandra_driver)
- The replication factor is set to 2 in the main.py
- If one node is down then the system stays functionable, however, operations that require a quorum can't be executed e.g., one can view reservations, however, they can't create one because operations of writing are partly restricted in that scenario

Database schema

The database schema is (partly) shown below (partly because dbdiagram.io wouldn't understand Cassandra's primary key notation). Interactive diagram with comments is available via the link: https://dbdiagram.io/d/cassandra-cinema-reservation-6842fde9a845fcbe14b011b4



The database has 3 main tables and 2 supplementary ones. The main tables are:

- **users** – table with all the usernames (system is simplistic so there are no passwords)
- **movies** – table where each screening is saved (e.g., movie Inception at 17:00 on June 16, 2025)
- **reservations** – table of reservations where each reservation has corresponding movie_id, seat number, and user name of a person making the reservation

Supplementary tables are added to improve the efficiency of the database and not to use ALLOW FILTERING or indices. There are two of them:

- **reservations_by_user** – the user is a partition key while movie_id and seat are clustering keys. This table helps to find all the movies for a given user
- **movie_lookup** – this table allows to get title and time of a screening knowing the movie_id (primary key)

Problems encountered

I can't pinpoint any of the serious problems encountered during the project completion, however, what raised a question was a situation when I tried to use 3 nodes with replication factor staying at 2. When I stopped one of the nodes I supposed that the system would stay fully operational but it seems that the node I turned down was the one keeping the replicas and those replicas won't move to the remaining nodes even after some waiting time. I discovered that operations requiring SERIAL consistency wouldn't work because quorum for Paxos wasn't achieved. I believe this is in line with Cassandra functionality, however, not fully intuitive.