PART 4

Objective:

Implement distributed transaction management in MongoDB to ensure data consistency and concurrency control for a specific application.

Tools Used:

- MongoDB (with Replica Set)
- PyMongo (Python MongoDB Driver)
- Python (with threading for concurrency simulation)

Implementation Details:

MongoDB Replica Set Setup:

- Configured a MongoDB replica set with three nodes to support distributed transactions.
- Used port numbers 27017, 27018, and 27019 on localhost.

Database and Collections:

Database: streaming_dds
Collections: playlists and users

Concurrency Control:

- Utilized MongoDB's default concurrency control mechanisms (document-level locking).
- Simulated concurrent transactions using Python's threading module.

Transaction Handling:

- Implemented ACID-compliant transactions using PyMongo's session and transaction controls.
- Transactions involve updating playlists_collection and user_collection.

Concurrency Testing:

- Simulated five concurrent transactions, randomly selecting playlist_id and user_id within a given range.
- Each thread starts a MongoDB session and executes a transaction that updates the status in playlists_collection and increments playlist_count in user_collection.

Code Structure:

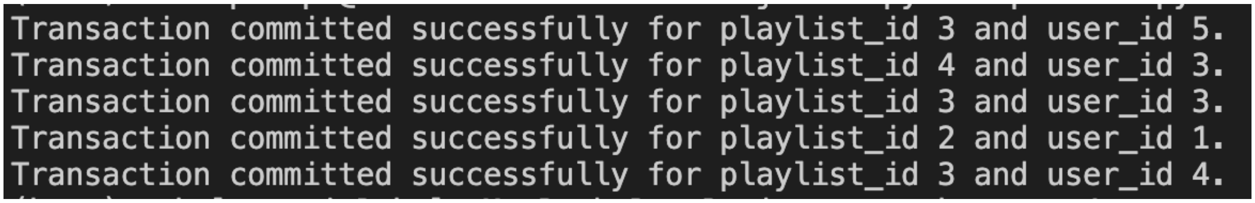
- Connection to MongoDB replica set.
- Definition of transaction function execute_transaction that encapsulates the transaction logic.
- Function simulate_concurrent_transactions to execute transactions in separate threads.
- Main script body to initiate and manage concurrent threads.

Observations:

- The script successfully demonstrates the ability of MongoDB to handle multiple concurrent transactions while maintaining data integrity.
- The script logs the outcome of each transaction, providing visibility into the transaction process.

Result and Insights:

The implementation successfully demonstrates MongoDB's robust handling of distributed transactions within a replica set environment. The script's execution resulted in all five simulated transactions committing successfully, each involving different combinations of playlist_id and user_id. This outcome highlights MongoDB's effectiveness in managing concurrent transactions, ensuring data consistency and integrity across the database.



```
Transaction committed successfully for playlist_id 3 and user_id 5.
Transaction committed successfully for playlist_id 4 and user_id 3.
Transaction committed successfully for playlist_id 3 and user_id 3.
Transaction committed successfully for playlist_id 2 and user_id 1.
Transaction committed successfully for playlist_id 3 and user_id 4.
```

These results provide valuable insights into MongoDB's capability to handle simultaneous transactional operations in a high-concurrency environment. The ACID compliance of these transactions within a distributed setup illustrates the database's reliability for complex, real-world applications requiring robust data management.