ENPM818T - 0101 - Variable Topics in Engineering; Data Storage and Databases

FINAL PROJECT - SUBJECT 2

Arden Matikyan - 116250909
Bavan Mooganahally Yadunath - 121301802
Shankar Narayan Saiprasad - 121128041
Sri Harsha Chayanulu Varahabhatla - 120478345

This kv2_design document describes the strategy for creating keys and the structure of the keys' values that are used to achieve the functionalities below.

Part 2 3
- 3.4 Player-Centric Functionality 3
  - 1. View another player's match history 3
  - 2. List all future games a player is scheduled to play 3
  - 3. Check if a friend is a member by email (Bloom Filter) 3
  - 4. View match history between two players 3
  - 5. View most used opening by a player 4
- 3.5 Leaderboard Functionality 4
  - 1. Top 10 most successful players (most by victories) 4
  - 2. Bottom 10 least successful players (by most losses) 4
- 3.6 Game Functionality 5
  - 1. Search for a three-move sequence by player 5
  - 2. Search for a three-move sequence globally 5
- 3.7 Analytics 5
  - 1. Most common three-move sequence 5
  - 2. Least common three-move sequence (non-zero) 6
  - 3. Shortest game by number of moves 6
  - 4. Number of checks in a game 6
  - 5. Most frequently used opening overall 6

Setup & Execution Guide 8
- 1. Create a Virtual Environment 8
- 2. Install Required Dependencies 8
- 3. Ensure Redis is Running 8
- 4. Load Data into Redis 8
- 5. Test Each Functional Module 9

Part 2

3.4 Player-Centric Functionality

1. View another player's match history
    - Key: player:{user_id}:games
    - Value: [{game_id_1}, {game_id_2}, ...]
    - Write/Update: When a new game is created, extract white_player_id and black_player_id, and set game_id as the value of player:{user_id}: games for both players
    - Read: Read player:{target_player_id}:games and return the list of game IDs
    - Interactions: 2 writes on game creation, 1 read on view
2. List all future games a player is scheduled to play
    - Key: player:{player_id}:scheduled_games
    - Value: [{game_id_1}, {game_id_2}, ...]
    - Write/Update: When a future game is scheduled, set game_id as the value of player:{user_id}:scheduled_games for both players. A helper function is used to automatically remove a game_id after the set time expires.
    - Read: Read player:{target_user_id}:scheduled_games and return the list of scheduled game IDs
    - Interactions: 2 writes on schedule, 1 read on view
3. Check if a friend is a member by email (Bloom Filter)
    - Key: email_user:{email_id}
    - Value: {user_id}
    - Write/Update: When a user registers, set email_user:{email} = {user_id}, also update the bloom filter for quick access
    - Read: Read from the bloom filter first, if it returns true, check the key email_user:{email} and return user_id if it exists
    - Interactions:
        - Registration: 2 writes (SET + BF.ADD)
        - Lookup: 1-2 reads (BF.EXISTS + possibly GET)
4. View match history between two players
    - Key: player_versus:{playerA_id}:{playerB_id}
    - Value: [{game_id_1}, {game_id_2}, ...]
    - Write/Update: On new game, determine playerA and playerB as white_player_id and black_player_id, and set game_id as the value of player_versus:{playerA}:{playerB}
    - Read: Read and return union of player_versus:{playerA}:{playerB} and player_versus:{playerB}:{playerA}
    - Interactions: 1 write on game, 2 reads on view

  - Note: "player A" and "player B" could have been written lexicographically in the writes, but this implementation allows us to easily add the feature of viewing versus games while specifying which player is white and which is black.

5. View most used opening by a player
   - Keys: player:{user_id}:opening:{opening_eco}:count, player:{user_id}:openings
   - Value: {user_id_opening_eco_count}, [{game_id_1}, {game_id_2}, ...]
   - Write/Update: For each new game, increment eco_count for both players and update a set to include a list of all the openings played by a player and also increment the global counter for that opening
   - Read: Get all the openings the player has ever used, For each opening, retrieve the count of how many times they've used it, Compare all counts to find the opening with the highest usage
   - Interactions: N (based on the number of different openings the player has used) reads on view and 5 writes during game insert

---

3.5 Leaderboard Functionality

1. Top 10 most successful players (most by victories)
   - Keys: leaderboard:wins:{player_id}, leaderboard:top_players
   - Value: player victory count as an integer, leaderboard: sorted list of user_ids based on the number of wins
   - Write/Update: When a new game record with a decisive outcome (not a draw) is added, determine the winner and increment the win counter. Update the top players in the leaderboard as necessary.
   - Read: Return the first 10 from the leaderboard:top_players, also with their win count
   - Interactions: During writes, 1 INCR operation for updating the win counter, 1 LRANGE operation to get current leaderboard players, up to 10 GET operations to retrieve win counts, 1 DELETE and up to 10 RPUSH operations to update leaderboard. During reads, 1 LRANGE operation to get the ordered player list and up to 10 GET operations to retrieve individual win counts
2. Bottom 10 least successful players (by most losses)
   - Keys: leaderboard:losses:{player_id}, leaderboard:bottom_players
   - Value: player loss count as integer, sorted list of user_ids
   - Write/Update: When a new game record with a decisive outcome (not a draw) is added, determine the loser and increment the loss counter. Update the bottom players in the leaderboard as necessary.
   - Read: Return the first 10 from the leaderboard:bottom_players, also with their loss count
   - Interactions: During writes, 1 INCR operation for updating the loss counter, 1 LRANGE operation to get current leaderboard players, up to 10 GET operations

to retrieve loss counts, 1 DELETE and up to 10 RPUSH operations to update leaderboard. During reads, 1 LRANGE operation to get the ordered player list and up to 10 GET operations to retrieve individual loss counts

---

3.6 Game Functionality

1. Search for a three-move sequence by player
    - Key: sequence:{move1}>{move2}>{move3}, sequence:{move1}>{move2}>{move3}:games, player:{user_id}:sequences
    - Value: count of the sequence, set of game Ids, Set of three move sequences used by a player
    - Write/Update: For each 3-move subsequence in game, determine the global count and increment it, Add the game ID to the set of games containing this sequence and add the sequence to each player's set of used sequences
    - Read: Get all games containing the sequence, get all games the player participated in, find the intersection of these two setsInteractions: For write operations, for each new game with N moves:Up to (N-2) SET operations for sequence counts, up to (N-2) SADD operations for sequence:games sets, up to 2(N-2) SADD operations for player sequence sets, up to 2 SET operations for most/least common sequence updates. For reads, 1 SMEMBERS operation
2. Search for a three-move sequence globally
    - Key: sequence:{move1}>{move2}>{move3}, sequence:{move1}>{move2}>{move3}:games
    - Value: count of the sequence, set of game Ids
    - Write/Update: For each 3-move subsequence in game, determine the global count and increment it, Add the game ID to the set of games containing this sequence
    - Read: Retrieve all games containing the sequence
    - Interactions: For write operations, for each new game with N moves:Up to (N-2) SET operations for sequence counts, up to (N-2) SADD operations for sequence:games sets, up to 2 SET operations for most/least common sequence updates. For reads, 1 SMEMBERS operation

---

3.7 Analytics

1. Most common three-move sequence
    - Keys: analytics:most_common_sequence, sequence:{move1}>{move2}>{move3}
    - Value: String of the sequence in format "move1>move2>move3", integer count for each sequence

- Write/Update: On a new game, identify all three-move sequences, SET/INCR their counts, compare with current most common sequence and update analytics:most_common_sequence if needed
- Read: GET analytics:most_common_sequence and sequence:{result_of_first_get}
- Interactions: For each game with N moves, up to (N-2) SET/INCR operations for sequence counts, up to 2 GET operations, and potentially 1 SET operation. For reads, 2 GET operations

2. Least common three-move sequence (non-zero)
    - Keys: analytics:least_common_sequence, sequence:{move1}>{move2}>{move3}
    - Value: String of the sequence in format "move1>move2>move3", integer count for each sequence
    - Write/Update: On a new game, identify all three-move sequences, SET/INCR their counts, compare with current least common sequence and update analytics:least_common_sequence if needed
    - Read: GET analytics:least_common_sequence and sequence:{result_of_first_get}
    - Interactions: For each game with N moves, the sequence count operations are shared with the most common sequence process. Additional comparison logic requires up to 2 GET operations, and potentially 1 SET operation. For reads, 2 GET operations
3. Shortest game by number of moves
    - Keys: analytics:shortest_game, analytics:shortest_game_turns
    - Value: game_id {string} of the current shortest game, number of turns {integer} of the current shortest game
    - Write/Update: On a new game, if the move count (determined from list of moves) is smaller than the current shortest, update the value of both analytics:shortest_game and analytics:shortest_game_turns
    - Read: Read analytics:shortest_game and analytics:shortest_game_turns
    - Interactions: For each game, there can be at most 2 SET operations. For reads, 2 GET operations
4. Number of checks in a game
    - Keys: game:{game_id}:analytics:check_count
    - Value: Integer representing the number of check moves in the game
    - Write/Update: On a new game, count occurrences of '+' in the moveset and SET game:{game_id}:analytics:check_count
    - Read: GET game:{game_id}:analytics:check_count for the specific game
    - Interactions: For writes, 1 SET operation for each record. For reads, 1 GET operation
5. Most frequently used opening overall
    - Keys: analytics:most_frequent_opening, opening:{eco_code}
    - Value: ECO code string of the most frequent opening, integer count for each opening

- Write/Update: On a new game, INCR opening:{eco_code}, then compare with current most frequent opening and update analytics:most_frequent_opening if needed
- Read: GET analytics:most_frequent_opening and opening:{result_of_first_get}
- Interactions: For each game, 1 INCR operation, up to 2 GET operations and potentially 1 SET operation. For reads, 2 GET operations

Setup & Execution Guide

1. Create a Virtual Environment

macOS / Linux

```
python3 -m venv venv
source venv/bin/activate
```

Windows

```
python -m venv venv
venv\Scripts\activate
```

2. Install Required Dependencies

`pip install -r requirements.txt` (It will automatically install all necessary dependencies)

3. Ensure Redis is Running

Make sure a Redis server is running on localhost:6379.

On macOS (Homebrew):

```
brew install redis
brew services start redis
```

On Linux:

```
sudo apt install redis
sudo service redis start
```

With Docker: (use this to avoid problems with bloom filter)

```
docker pull redis:latest
docker run -d --name <some-name> -p 6379:6379 redis
```

4. Load Data into Redis

```
python3 load_transform.py
```

You should see a prompt

```
This will delete all Redis data. Continue? (y/n):
```

Type y and press Enter.

This will:

- Flush existing Redis data
- Load players, schedules, and game records from the data/ folder
- Initialize analytics, leaderboard, and expiration timers

The script will stay active to manage 72-hour scheduled game expirations.
You should get

```
INFO - Data loading complete!

INFO - The loader will remain alive to manage scheduled games.
Use ^C to exit (WARNING: Scheduled games will not be managed upon
exit).
```

***Note**: Keep this terminal open while testing.*

5. Test Each Functional Module

Open a new terminal tab while keeping the loader alive and since you're using a virtual environment , first make sure it's activated:

macOS/Linux:

```
source venv/bin/activate
```

Windows:

```
venv\Scripts\activate
```

Each script will print results from sample data stored in Redis.

```
python3 graph_functions.py

(These are interactive scripts and will require user input)
python3 player_functions.py
python3 leaderboard_functions.py
python3 game_functions.py
python3 analytics_functions.py
```