**Among Us a NoSQL Query Project for Advanced SQL**

**Introduction**

"Among Us" is a multiplayer game that gained massive popularity in 2020, having been initially released in 2018 with little acclaim. In the game, 4 to 10 players are dropped onto an alien spaceship, each assigned the role of either a 'crewmate' or an 'impostor'. The game can have 1 to 3 impostors depending on the total number of players. Crewmates are tasked with completing various tasks on the ship and trying to identify the impostors without being killed. Impostors, conversely, aim to covertly kill crewmates and foster suspicion among them to avoid ejection. Victory for crewmates comes through completing all tasks or successfully ejecting all impostors, while impostors win by reducing the number of crewmates to equal their own number. The game is praised for the thrilling experience it provides, especially when playing as an impostor.

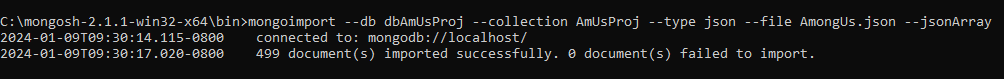
**Dataset**

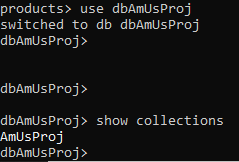
The data set has entries 499 Among Us games.

There are four high-level fields:  
1.Game  
2.Game feed  
3.Player data  
4.Voting data

The game is a unique identifier for each game. The other three fields are divided into nested documents with more granular-level data. A game feed is an array of objects where each object is an event in the game. Each game in Among Us has multiple events.

**Task1. Read the data**

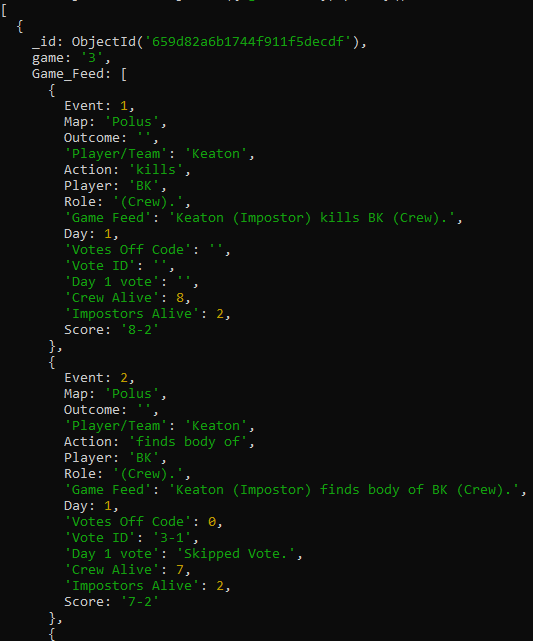
* 1. **Read the data and examine the collections**



* 1. **Display data for matches where the "game" field equals "3".**

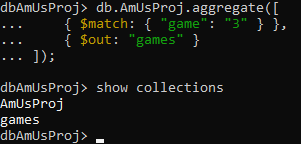


This command is used to retrieve and display information about game 3 from the **AmUsProj** collection within the database in a readable way.



**Task2. Explore the data for game 3**

1. **In this subtask, you are expected to create a new collection with only the document relating to game 3.**

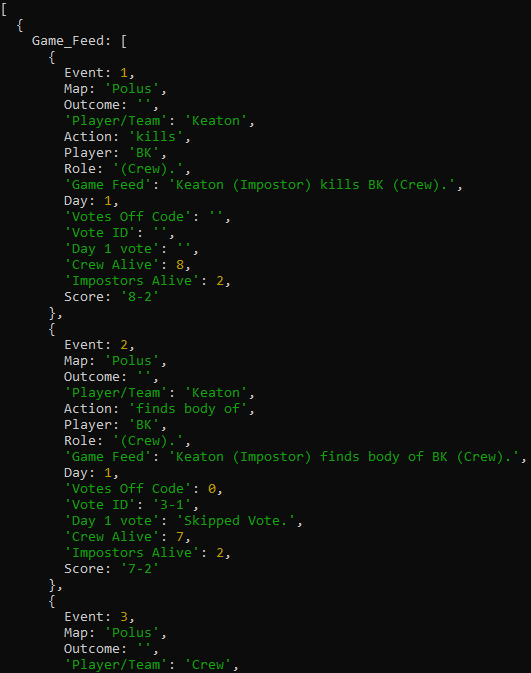


Query where the **AmUsProj** collection is aggregated to filter documents matching 'game' '3' into a new 'games' collection,

* 1. **Show the Game Feed data specifically for game 3 in the newly created collection.**



query that retrieves and formats documents from the 'games' collection where the 'game' field equals '3', specifically including the 'Game\_Feed' field and excluding the '\_id' field.

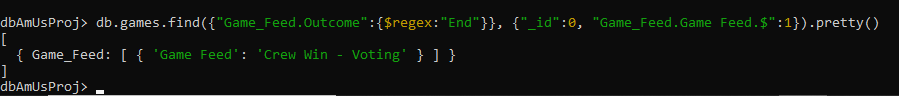


* 1. **Show the most recent event that occurred in game 3.**



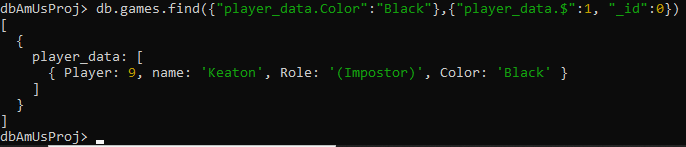
querying the 'games' collection for documents where the 'Game\_Feed.Outcome' field contains the substring "End," selecting only the 'Game\_Feed' field and formatting the output for readability.

* 1. **Determine the winner of game 3 - imposters or crew.**



query that retrieves documents from the 'games' collection where 'Game\_Feed.Outcome' ends with "End," displaying only the first matched 'Game\_Feed' subdocument and excluding the document ID.

* 1. **Identify the player who chose the black color in game 3 and whether they were a crew member or imposter.**



query result indicating that player number 9, named 'Keaton', who chose the color 'Black', is an 'Impostor' in the game

* 1. **Count the number of voting events that took place in game 3.**



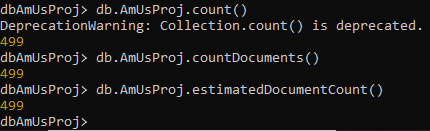
command where the length of the array returned by the **distinct** query on 'voting\_data.Vote\_Event’ field in the 'games' collection is 3, implying there are 3 unique voting events.

* 1. **If you were redesigning this database for easier querying, describe the changes you would make and explain your reasoning.**

Generally, we might normalize the data more, create indexes on commonly queried fields, or even redesign the schema to reduce the need for **$unwind** operations, which can be expensive. We can remove few redundant fields in the nested fields. Create a separate collection for player data and voting data and adding references in the game feed data to lessen the complexity of the database

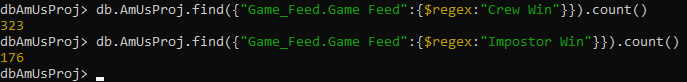
**Task3. Overall aggregation**

**3.1. Calculate the total number of events recorded in this collection across all games.**



Since **.count()** is deprecated, I used 2 different query to check the number of events recorded

**3.2. Compare the crew's wins to the impostors' wins and provide the counts.**



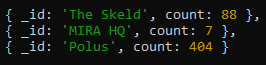
One query counts the number of documents where 'Game\_Feed.Game Feed' matches "Crew Win" (323 occurrences), and the other counts those matching

"Impostor Win" (176 occurrences).

**3.3. List the maps played and the total number of games on each map.**



aggregation pipeline that unwinds the 'Game\_Feed' array, matches documents with a non-empty 'Game\_Feed.Event', and groups the results by 'Game\_Feed.Map' while counting the occurrences.





**3.4. Determine the total instances of crew members skipping a vote across all games.**





aggregation pipeline that unwinds the 'Game\_Feed' array and groups the documents to count the total number of occurrences where 'Game\_Feed.Votes Off Code' equals 0.

**3.5. Calculate the total occurrences of crew members voting against imposters across all matches.**





aggregation pipeline that unwinds the 'Game\_Feed' array and groups the documents to count the total number of occurrences where 'Game\_Feed.Votes Off Code' equals 2.

**3.6. Share your opinion on whether the game is more or less challenging for impostors, supported by insights from the data.**

This would require analyzing the win-loss ratios, kill counts, and voting patterns to determine if impostors are at a disadvantage or not.

In total crew won 323 times and Imposters won 176 times.This statistics alone leads us to believe that the game is easier for crew.

Across 499 matches there were, 3992 crew members and 997 imposters.For every 10 players only 2 are imposters and 8 are crew members which means the imposters have to convince more people to vote against others.

**Task4. Player-level aggregation**

**4.1. Find the count of unique players in the dataset.**



command that counts the number of unique player names in the 'AmUsProj' collection, resulting in a total of 108 unique names.

**4.2. Identify the player considered the best crew member.**



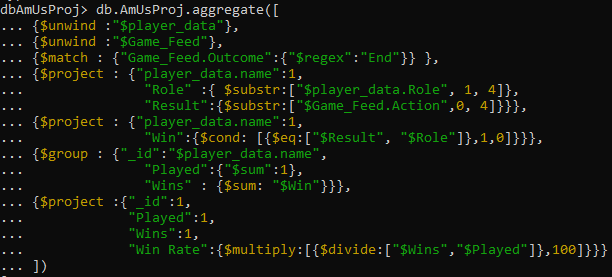
aggregation pipeline that unwinds the 'voting\_data' array, matches documents where 'voting\_data.Vote' contains the phrase "Impostor voted off", groups by 'voting\_data.name’ to count occurrences, sorts by count in descending order, and then limits the output to the single most frequent result.

**4.3. Identify the player regarded as the least effective crew member.**

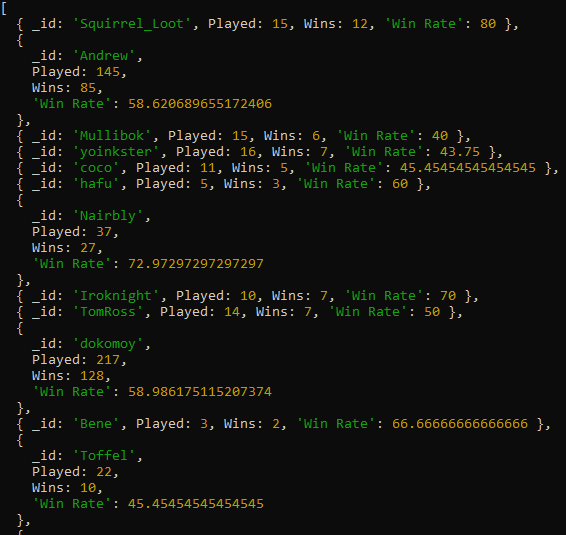


aggregation pipeline that unwinds the 'voting\_data' array, matches documents where 'voting\_data.Vote' contains the phrase "Crew voted off", groups by 'voting\_data.name' to count occurrences, sorts by count in descending order, and then limits the output to the single most frequent result.

**4.4. Compute the win percentage for each player. (Optional)**



Aggregation pipeline that aims to calculate the win rate of players in a game. It starts by separating the nested player data and game feed data into individual documents. The pipeline filters out game results that conclude with 'End', ensuring only finished games are considered. It then extracts the role and the game result for each player, determining if a win condition is met when the role matches the result. Afterward, it aggregates the data by player name, tallying the total games played and wins. Finally, it computes the win rate for each player by dividing the number of wins by the total games played and converts this ratio into a percentage. This pipeline is a powerful tool for analyzing player performance within the game dataset.

 Submitted by: John Erick D. Cadiz