Advanced_SQL_Project_Report

AmongUS

Project by: Archana Tirumala Narasimhan

Instructor: Maryam Kazemi

Introduction

Among Us is a multiplayer PC and mobile game that has suddenly exploded in popularity, becoming one of the hit video games of 2020. Created by an indie game company in 2018, Among Us remained under the radar until the summer of the pandemic. The game involves 4-10 players dropped onto an alien spaceship, each assigned as a 'crewmate' or 'impostor'. Crewmates must complete tasks and root out impostors to win, while impostors must discreetly eliminate crewmates. Players can be voted off the ship, adding a layer of strategy and survival.

Data Understanding

In this project, we analyze a dataset from 499 Among Us games, seeking detailed insights into gameplay using MongoDB. The dataset contains four main fields:

Game, Game Feed, Player Data, and Voting Data, each offering unique perspectives on the gameplay.

Data Import

The dataset was imported into MongoDB using the following command:

mongoimport -d dbAmongUs -c Among_Us_data --type json --file AmongUs.json –jsonArray

- 1.1. Read the data and examine the collections.

Procedure:

- Database Selection: Initiated by selecting the dbAmongUs database to direct all operations to the relevant data.
- Data Retrieval: Utilized the find() method with a limit(5) to efficiently extract a manageable subset of records for examination.

```
// 1.1. Read the data and examine the collections.

//Answer:
use('dbAmongUs')
db.Among_Us_data.find().limit(5)
```

Result Sample:

- Document Examination: Observed a hierarchical document structure, showcasing MongoDB's ability to store complex, nested data.
- Gameplay Insight: Extracted specific gameplay events, revealing critical moments like eliminations and discoveries within the game.

```
    "_id": {
      "$oid": "658332470347e340b75759e3"
    },
    "game": "1",
    "Game_Feed": [
      {
        "Event": 1,
        "Map": "Polus",
        "Outcome": "",
        "Player/Team": "wrapter",
        "Action": "kills",
        "Player": "LSV",
        "Role": "(Crew).",
        "Game Feed": "wrapter (Impostor) kills LSV (Crew).",
        "Day": 1,
        "Votes Off Code": "",
        "Vote ID": "",
        "Day 1 vote": "",
        "Crew Alive": 8,
        "Impostors Alive": 2,
        "Score": "8-2"
      },
      {
        "Event": 2,
        "Map": "Polus",
        "Outcome": "",
        "Player/Team": "dokomoy",
        "Action": "finds body of",
        "Player": "LSV",
        "Role": "(Crew).",
        "Game Feed": "dokomoy (Crew) finds body of LSV (Crew)
        "Day": 1,
        "Votes Off Code": 0,
        "Vote ID": "1-1",
        "Day 1 vote": "Skipped Vote.",
        "Crew Alive": 7,
        "Impostors Alive": 2,
        "Score": "7-2"
```


➢ 1.2. Display data for matches where the "game" field equals "3".

Procedure:

- Specific Data Filtering: Applied the query db.Among_Us_data.find({"game": "3"}) to filter the dataset and retrieve records where the "game" field equals "3".

- Database Interaction: Engaged with the dbAmongUs database, utilizing MongoDB commands to interact with and manipulate the data.

```
//1.2. Display data for matches where the "game" field equals "3".

//Answer:
use('dbAmongUs')
db.Among_Us_data.find({"game": "3"})
```

Result Sample:

Targeted Game Data: Successfully retrieved documents specific to game "3", focusing the analysis on events from this particular game.

Event Details Uncovered: The sample data reveals individual game events, such as a player named "Keaton" (Impostor) killing another player "BK" (Crew), providing granular details of game "3".


```
"_id": {
  "$oid": "658332470347e340b75759e4"
},
"game": "3",
"Game_Feed": [
  {
    "Event": 1,
    "Map": "Polus",
    "Outcome": "",
    "Player/Team": "Keaton",
    "Action": "kills",
    "Player": "BK",
    "Role": "(Crew).",
    "Game Feed": "Keaton (Impostor) kills BK (
    "Day": 1,
    "Votes Off Code": "",
    "Vote ID": "",
    "Day 1 vote": "",
    "Crew Alive": 8,
    "Impostors Alive": 2,
    "Score": "8-2"
  },
  {
    "Event": 2,
    "Map": "Polus",
    "Outcome": "",
    "Player/Team": "Keaton",
    "Action": "finds body of",
    "Player": "BK",
    "Role": "(Crew).",
    "Game Feed": "Keaton (Impostor) finds body
    "Day": 1,
    "Votes Off Code": 0,
    "Vote ID": "3-1",
    "Day 1 vote": "Skipped Vote.",
    "Crew Alive": 7,
    "Impostors Alive": 2,
    "Score": "7-2"
```

Exploratory Data Analysis

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

- 2.1. Show the Game Feed data specifically for game 3 in the newly created collection.

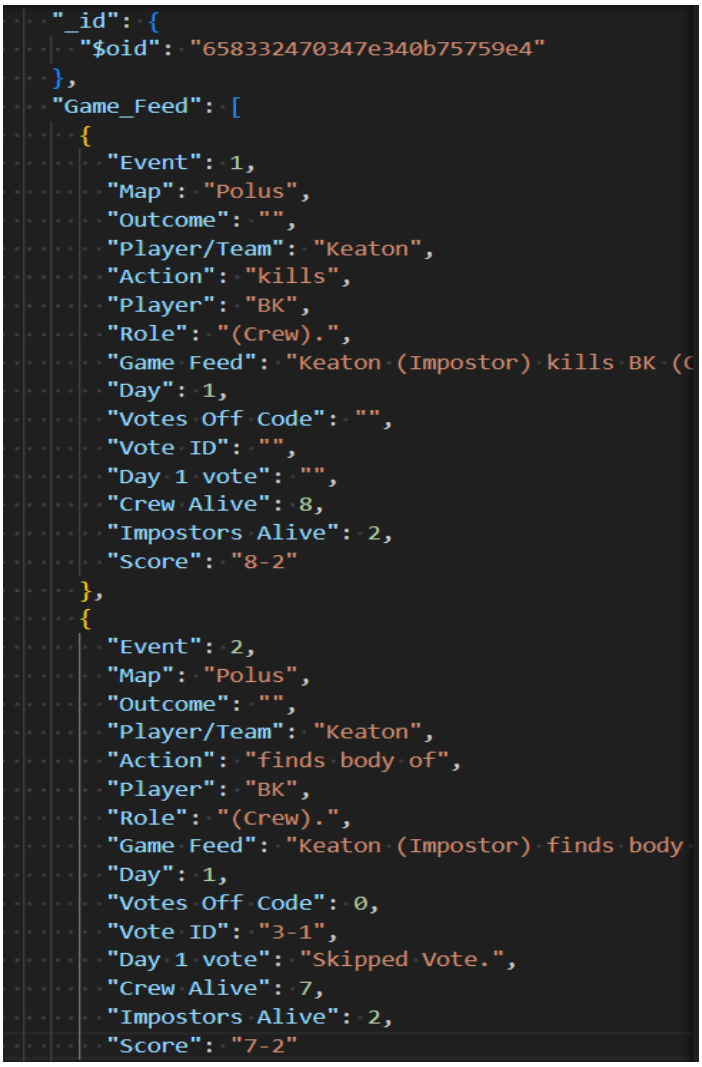
Procedure:

- Data Aggregation: Employed MongoDB's aggregation framework with $match to filter documents where "game" equals "3", and $out to output the results into a new collection named game3Collection.
- Collection Switching: After creating the new collection, used the use command again to switch context and then executed a find operation on game3Collection to display the Game_Feed data.

```
//2. In this subtask, you are expected to create a new
//collection with only the document relating to game 3.

//2.1. Show the Game Feed data specifically for game 3 in the newly creat

  //Answer:
use('dbAmongUs')
db.Among_Us_data.aggregate([
    { $match: {"game": "3"} },
    { $out: "game3Collection" }
  ])
use('dbAmongUs')
db.game3Collection.find({}, { "Game_Feed": 1 })
```

Result Sample:

- Collection Creation: A new collection was successfully created to isolate the documents related to game "3", streamlining further data analysis specifically for that game.
- Game Feed Access: The game feed for game "3" was accessed, providing insights into the sequence of events, such as player actions and the number of crew and impostors remaining at various stages of the game.



➢ 2.2. Show the most recent event that occurred in game 3.

Procedure:

- Aggregation Pipeline Utilization: Implemented the aggregation pipeline in MongoDB to project the most recent event in game3Collection using the $project stage combined with $arrayElemAt to access the last element of the Game_Feed array.
- Database Operation: Reiterated the use of the dbAmongUs database and executed the aggregation command on game3Collection, which is focused on extracting the latest game event.

```
//2.2. Show the most recent event that occurred in game 3.

//Answer:
use('dbAmongUs')
db.game3Collection.aggregate([
    { $project: {
        "Most_Recent_Event": { $arrayElemAt: ["$Game_Feed", -1] }
    }}
])
```

Result :

- Recent Event Extraction: Successfully isolated the most recent event from game3Collection, which provides the final outcome of the game.
- Outcome Determination: The result indicates that the crew won the game by voting, which is a significant piece of information for understanding the game's conclusion and the overall success of the crew team.


```
[
  {
    "_id": {
      "$oid": "658332470347e340b75759e4"
    },
    "Most_Recent_Event": {
      "Event": 10,
      "Map": "Polus",
      "Outcome": "3-End",
      "Player/Team": "",
      "Action": "Crew Win - Voting",
      "Player": "",
      "Role": "",
      "Game Feed": "Crew Win - Voting",
      "Day": 4,
      "Votes Off Code": "",
      "Vote ID": "",
      "Day 1 vote": "",
      "Crew Alive": 5,
      "Impostors Alive": 0,
      "Score": "5-0"
    }
  }
]
```

- 2.3. Determine the winner of game 3 - imposters or crew.

Procedure:

- Aggregation for Outcome: Leveraged MongoDB's aggregation pipeline on game3Collection to unwind the Game_Feed array, sorted the events in descending order, limit the output to the most recent event, and project the Outcome.
- Analytical Approach: Structured the database query to pinpoint the final game event, which directly indicates the winner between imposters and crew.

```
//2.3. Determine the winner of game 3 - imposters or crew.

//Answer:
use('dbAmongUs')
db.game3Collection.aggregate([
    { $unwind: "$Game_Feed" },
    { $sort: { "Game_Feed.Event": -1 } },
    { $limit: 1 },
    { $project: { "Outcome": "$Game_Feed.Action" } }
  ])
```

Result :

- Determination of Winner: The outcome of game 3 is clearly identified as a "Crew Win - Voting," indicating that the crew members emerged victorious in this particular game.
- Aggregation Result: The aggregation pipeline effectively distilled the final result from the game data, demonstrating a focused method for determining the winner.


```
[
  {
    "_id": {
      "$oid": "658332470347e340b75759e4"
    },
    "Outcome": "Crew Win - Voting"
  }
]
```

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

Procedure:

- MongoDB Query for Specific Data: Executed a query on game3Collection to identify the player who chose the black color in game 3, utilizing MongoDB's find method with a condition on the player_data.Color field.
- Projection of Relevant Data: Applied projection to the query to return only the relevant player_data for the player who chose the black color.

```
//2.4. Identify the player who chose the black color in game 3 and whether they we

//Answer:
use('dbAmongUs')
db.game3Collection.find({"player_data.Color": "Black"}, {"player_data.$": 1})
```

Result :

- Player Identification: The query results specify that the player named "Keaton" chose the black color for game 3.
- Role Determination: It is also confirmed that "Keaton" played the role of an Impostor in the game.


```
[
  {
    "_id": {
      "$oid": "658332470347e340b75759e4"
    },
    "player_data": [
      {
        "Player": 9,
        "name": "Keaton",
        "Role": "(Impostor)",
        "Color": "Black"
      }
    ]
  }
]
```

➢ 2.5. Count the number of voting events that took place in game 3.

Procedure:

- Use of Aggregation Pipeline: Utilized MongoDB's aggregation framework with the $project stage to calculate the size of the voting_data.Vote_Event array, which contains the voting events in game 3.
- Counting Array Elements: Employed the $size operator to determine the number of elements within the Vote_Event array, effectively counting the voting events.

```
//2.5. Count the number of voting events that took place in game 3.

//Answer:
use('dbAmongUs')
db.game3Collection.aggregate({$project:{cnt:{$size:'$voting_data.Vote_Event'}}})
```

Result :

- Voting Events Count: The aggregation query successfully counted the number of voting events in game 3, with the result being 24.
- Data Retrieval Outcome: The query provided a clear numeric outcome, signifying that there were 24 voting events that took place during game 3, which can be an indicator of the level of engagement or decision-making activity in the game.


```
[
  {
    "_id": {
      "$oid": "658332470347e340b75759e4"
    },
    "cnt": 24
  }
]
```

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

Answer:

- **Normalized Data Structure:** I will split complex documents into separate collections for games, players, and events, linking them with IDs. This will make my queries simpler and improve overall performance.
- **Indexing:** I'll create indexes on frequently queried fields such as game IDs and player IDs to speed up search operations significantly.
- **Consistent Field Types:** I would ensure that all fields have consistent data types across documents to prevent confusion and errors during querying.
- **Denormalization for Accessibility:** For data that is often accessed together, like some player information in game documents, I'll use a degree of denormalization to facilitate faster access.
- **Timestamps for Events:** will add timestamps to each event in the game feed, enabling easier analysis of the sequence and timing of events.
- **Balanced Use of Subdocuments and Arrays:** will sensibly use subdocuments and arrays for hierarchical or grouped data, and ensuring the efficiency of queries.
- **Schema Design Based on Query Patterns:** would design the schema to align with common query patterns, enhancing the user experience when querying data related to individual games or players.
- **Data Validation Rules:** I'll implement rules for data validation to maintain data integrity, like enforcing the presence of specific fields and setting acceptable ranges for values.
- **Aggregation Framework Optimization:** I will structure the schema to optimize the use of MongoDB's aggregation framework, which is crucial for complex queries and data analysis.

- **IN COMMON**

- **Focused Query:** The query directly searches for a player in a specific game with the specified color. It's more straightforward and focused on the player data.

- **Performance:** By having a separate collection for players, indexed on gameId and color, the database can quickly retrieve relevant player data without searching through nested arrays or subdocuments.
- **Scalability and Flexibility**: This structure allows the database to efficiently manage a large number of players and games. It also provides flexibility for other types of player-related queries.
- **Data Integrity and Clarity:** Separating player data into its own collection can help maintain data integrity and makes understanding the database schema easier.
- **Adaptability for Other Queries:** With a dedicated player collection, queries about player behavior, statistics, and patterns across multiple games become more straightforward.
- By restructuring the data in this way,
- we can make the database schema more efficient for common queries and analyses, enhancing both performance and usability.

Aggregation

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

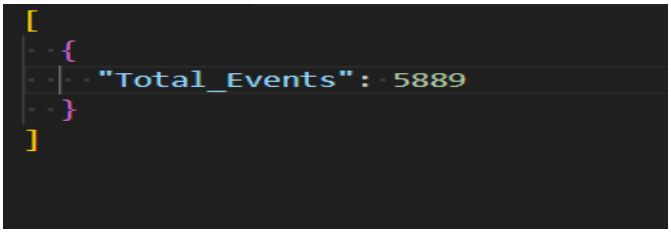
Procedure:

- Data Aggregation for Event Count: Implemented MongoDB's aggregation pipeline with $unwind to deconstruct the Game_Feed array, followed by $count to tally the number of game events across the entire collection.
- Database Context Specification: Made sure to use the correct database context dbAmongUs to apply the aggregation pipeline.

```
//3.1. Calculate the total number of events recorded in this collection across all

//Answer:
use('dbAmongUs')
db.Among_Us_data.aggregate([
    { $unwind: "$Game_Feed" },
    { $count: "Total_Events" }
  ])
```

Result :

- Total Event Calculation: The aggregation query successfully calculated the total number of events, yielding 5889 as the result.
- Aggregate Query Efficiency: Demonstrated the effective use of the aggregation framework to provide a summative insight into the volume of game events recorded in the collection.



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

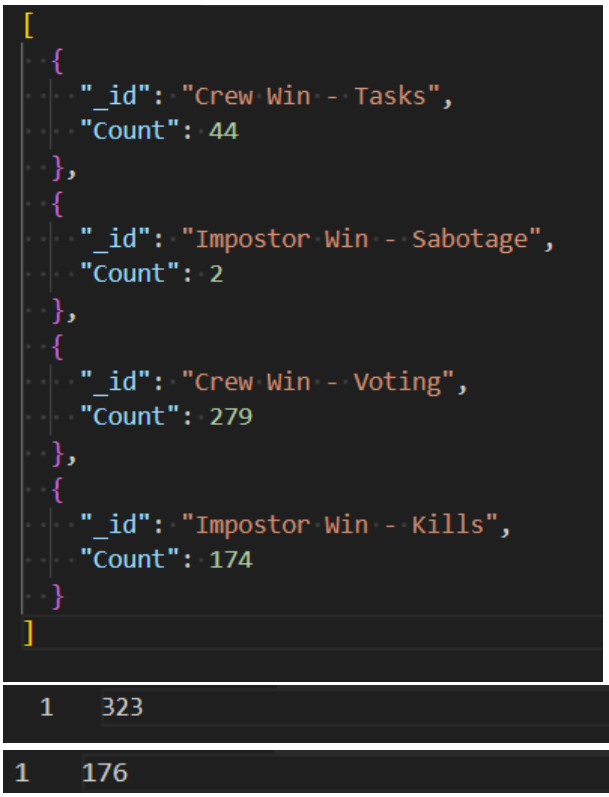
Procedure:

- Aggregation Framework Usage: Deployed MongoDB's aggregation framework for a complex operation that involves projecting the last action from the Game_Feed array and then grouping by the outcome to count the number of wins for both crew and impostors.
- Query Execution for Counts: Used the find method with a regular expression in the Game_Feed.Game Feed field to count occurrences of 'Crew Win' and 'Impostor Win' separately.

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

//Answer:
use('dbAmongUs')
db.Among_Us_data.aggregate([
    { $project: { "Outcome": { $arrayElemAt: ["$Game_Feed.Action", -1] } } },
    { $group: {
        _id: "$Outcome",
        Count: { $sum: 1 }
    }}
])


use('dbAmongUs')
db.Among_Us_data.find({'Game_Feed.Game Feed':{$regex:'Crew Win'}}).count()


use('dbAmongUs')
db.Among_Us_data.find({'Game_Feed.Game Feed':{$regex:'Impostor Win'}}).count()
```

Result:

- Win Counts: The aggregation results show the counts of different win types, with Crew Win - Tasks being 44, Crew Win - Voting being 279, Impostor Win - Sabotage being 2, and Impostor Win - Kills being 174.
- Win Comparison: The separate find counts show there were 323 instances of 'Crew Win' and 176 of 'Impostor Win', highlighting that crew members won more often than impostors in the dataset.



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

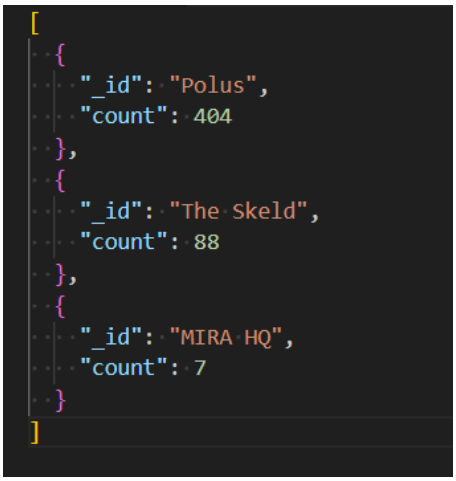
Procedure:

- Aggregation for Map Analysis: Implemented MongoDB's aggregation pipeline to deconstruct the Game_Feed array and used $match to filter for documents with the Event field set to 1, which likely signifies the start of a game. Then, grouped the results by Map using $group to count the total number of games on each map.
- Database Query Execution: Ensured that the dbAmongUs database was in use before executing the aggregation command, which is critical for applying the query to the correct dataset.

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

//Answer:
use('dbAmongUs')
db.Among_Us_data.aggregate([
    { $unwind: "$Game_Feed" },
    { "$match" : {"Game_Feed.Event":1}},
{ "$group" : {"_id": "$Game_Feed.Map","count": { "$sum":1}}}
])
```

Result :

- Map Distribution: The query results reveal the distribution of games across different maps, with Polus having 404 games, The Skeld 88 games, and MIRA HQ 7 games.
- Gameplay Frequency by Map: The results provide a clear indication of the popularity or frequency of each map in the dataset, showing that Polus is the most played map among the three listed.



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

Procedure:

- Utilization of Aggregation Pipeline: Engaged the aggregation pipeline in MongoDB with $unwind to flatten the Game_Feed array, and $match to filter events where crew members skipped voting.

- Counting Specific Events: Used $count to tally the occurrences of skipped votes, providing a total count of such events across all games.

```
//3.4. Determine the total instances of crew members skipping a vote across all g

//Answer:
use('dbAmongUs')
db.Among_Us_data.aggregate([
  { $unwind: "$Game_Feed" },
  { $match: { "Game_Feed.Game Feed": /skips voting/i } },
  { $count: "SkippedVotes" }
])
```

Result :

- Vote Skipping Instances: The result of the aggregation showed that there were 693 instances where crew members chose to skip voting.
- Insight into Player Behavior: This count indicates a significant number of instances where players did not cast a vote during meetings, suggesting potential indecisiveness or strategic choice within the game dynamics.


```
[
  {
    "SkippedVotes": 693
  }
]
```


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

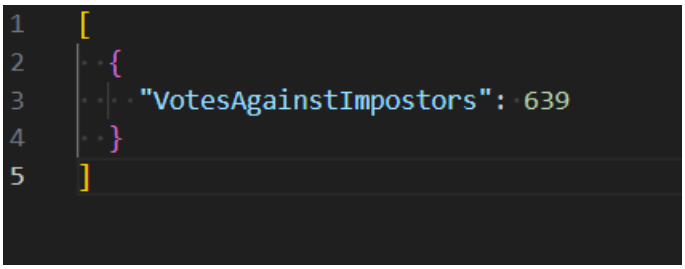
Procedure:

- Event Filtering with Aggregation: Used MongoDB's aggregation pipeline to filter the Game_Feed array for events related to voting against impostors, utilizing $unwind to expand the array and $match with a regular expression to find specific vote-related events.
- Event Counting: Counted the occurrences of votes against impostors by applying the $count stage, which aggregates the number of matching documents.

```
//3.5. Calculate the total occurrences of crew members voting against imposte

//Answer:
use('dbAmongUs')
db.Among_Us_data.aggregate([
  { $unwind: "$Game_Feed" },
  { $match: { "Game_Feed.Day 1 vote": /impostor voted off/i } },
  { $count: "VotesAgainstImpostors" }
])
```

Result :

- Vote Occurrences Calculated: The aggregation resulted in a total of 639 occurrences where crew members voted against impostors throughout the games.
- Gameplay Analysis Insight: This quantitative measure provides insight into how often crew members successfully vote off impostors.



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

Answer:

- Based on the data
- **High Impostor Ejection Rate:** With 893 instances of impostors being voted off, the data suggests that impostors face a considerable challenge in evading detection and maintaining their cover.

- **Crew Diligence:** The frequency of impostor ejections points to crew members being proactive and effective in discussions and voting, thereby increasing the difficulty for impostors to survive votes.
- **Lack of Complete Win/Loss Data:** Without specific win/loss ratios for impostors versus crewmates, the data is incomplete in knowing a full picture of the game's difficulty balance.
- **Implications of Skip Votes:** There were 693 instances of skipped votes, which may indicate a level of uncertainty during voting discussions, potentially providing a slight advantage to impostors.
- **Data-Driven Inference:** Although direct win/loss metrics are missing, the high number of impostor ejections allows us to infer that playing as an impostor could be challenging due to the effectiveness of crew strategy during voting sessions.

Player-Level Aggregation

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

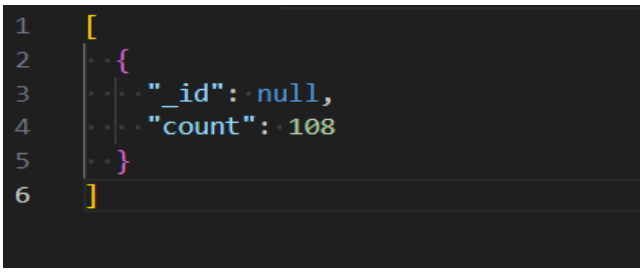
Procedure:

- Unique Player Counting: To find the unique players, the player_data array is unwound using $unwind and then grouped by player name using $group. A second $group is used to count the total number of unique player names.
- Aggregation Pipeline Application: The commands were executed within the dbAmongUs context to apply the aggregation pipeline on the appropriate dataset.

```
//4.1. Find the count of unique players in the dataset.

//Answer:
use('dbAmongUs')
db.Among_Us_data.aggregate([
    { $unwind: "$player_data" },
    { $group: { _id: "$player_data.name" } },
    { $group: { _id: null, count: { $sum: 1 } } }
  ])
```

Result :

- Unique Players Identified: The aggregation query resulted in identifying 108 unique players in the dataset.
- Dataset Player Diversity: This count provides an insight into the diversity of player participation within the dataset.



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

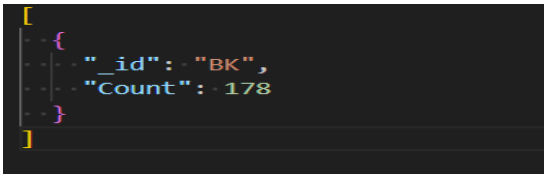
Procedure:

- Identification of Best Crew Member: Executed an aggregation pipeline to first unwind the voting_data array and then used $match to filter for instances where an "Impostor was voted off". This approach identifies the crew members who contributed to voting off impostors.
- Aggregation and Sorting: Grouped the results by player name to count the occurrences of successful votes against impostors and sorted the results in descending order to identify the top player, finally limiting the result to the single best crew member.

```
//4.2. Identify the player considered the best crew member.


//Answer:
use('dbAmongUs')
db.Among_Us_data.aggregate([
    {"$unwind": "$voting_data"},
    {"$match": {"voting_data.Vote": {"$regex": "Impostor voted off"}} },
    {"$group": {"_id": "$voting_data.name", "Count": {"$sum": 1}}},
    {"$sort": {"Count": -1}},
    {"$limit": 1} // Add the $limit stage here
])
```

Result :

- Top Crew Member Recognition: The aggregation determined that the player "BK" had the highest count of voting off impostors, with a total count of 178, suggesting that "BK" could be considered the best crew member based on this metric.
- Strategic Gameplay Insight: This result indicates "BK's" significant contribution to identifying and eliminating impostors, which is a key objective for crew members in the game.



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

Procedure:

- Least Effective Crew Member Analysis: The aggregation pipeline was used to unwind the voting_data array, filter for votes where "Crew was voted off" (indicating a potential mistake by the crew in voting off one of their own), group by player name, and count these occurrences.
- Sorting and Limiting Results: After grouping, the results were sorted in descending order based on the count and then limited to the top result to identify the player with the highest number of such votes, which could suggest being the least effective crew member.

```
//4.3. Identify the player regarded as the least effective crew member.

//Answer:
use('dbAmongUs')
    db.Among_Us_data.aggregate([
        {"$unwind": "$voting_data"},
        {"$match": {"voting_data.Vote": {"$regex": "Crew voted off"}} },
        {"$group": {"_id": "$voting_data.name", "Count": {"$sum": 1}}},
        {"$sort": {"Count": -1}},
        {"$limit": 1} // Add the $limit stage here
    ])
```

Result :

- Identification of Least Effective Member: The player "Sam" was identified as having the highest count of 60 votes for "Crew voted off", suggesting that "Sam" might be regarded as the least effective crew member based on this metric.
- Insight into Voting Patterns: This result provides insight into the game dynamics, where "Sam" either made frequent incorrect accusations or was often misjudged by other crew members.

➢ 4.4. Compute the win percentage for each player.

Procedure:

Complex Aggregation for Win Percentage:

- Unwound both player_data and Game_Feed arrays to work with individual documents.
- Filtered documents to include only those with an outcome indicating the end of a game.
- Projected necessary fields to prepare for calculating wins, including truncating player roles and results to align with win conditions.
- Implemented conditional logic to determine if a player won a particular game.
- Grouped the results by player name to calculate the total number of games and wins for each player.

Win Percentage Calculation:

- Projected the final outcome including player name and win percentage, which was calculated by dividing total wins by total games and multiplying by 100, rounded to two decimal places for readability.

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

//Answer:
use('dbAmongUs');
db.Among_Us_data.aggregate([
    // Unwind the nested arrays to work with individual elements
    { $unwind: "$player_data" },
    { $unwind: "$Game_Feed" },

    // Filter only the games that have an outcome (End of the game)
    { $match: { "Game_Feed.Outcome": { $regex: "End" } } },

    // Project the necessary fields
    { $project: {
        "player_name": "$player_data.name",
        "role": { $substr: ["$player_data.Role", 1, 4] },
        "result": { $substr: ["$Game_Feed.Action", 0, 4] }
    }},

    // Determine if the player won the game
    { $project: {
        "player_name": 1,
        "win": { $cond: [{ $eq: ["$result", "$role"] }, 1, 0] }
    }},

    // Group by player and calculate total games and wins
    { $group: {
        _id: "$player_name",
        total_games: { $sum: 1 },
        total_wins: { $sum: "$win" }
    }},
```

```
    // Calculate win percentage
    { $project: {
        "_id": 0,
        "player_name": "$_id",
        "win_percentage": {
            $round: [{
                $multiply: [{
                    $divide: ["$total_wins", "$total_games"]
                }, 100]
            }, 2]
        }
    }}
]);
```

Result :

Win Percentage Outcomes:

- The aggregation resulted in a list of players with their respective win percentages, providing a clear performance metric for each player.
- For instance, "Isaac" has a win percentage of 100, "DBatterskull" has 50.45, and "ScaldingHotSoup" has 86.67, indicating varying levels of success across different players in the dataset.

Insight into Player Performance:

- These results offer insights into which players are most successful within the game, potentially informing strategies or highlighting skilled players.

- 4.5. Determine the color preferences chosen by all players.

Procedure:

Aggregation for Color Preferences:

- Executed MongoDB's aggregation pipeline with $unwind to deconstruct the player_data array.
- Applied $group with $addToSet to collect unique colors chosen by each player without duplication, indicating their color preferences throughout the games.

Database Context and Query Execution:

- The operation was carried out within the dbAmongUs database to ensure the query was applied to the correct collection.

```
//4.5. Determine the color preferences chosen by all players. (Optional)

//Answer:
use('dbAmongUs')
db.Among_Us_data.aggregate([
    { $unwind: "$player_data" },
    { $group: { _id: "$player_data.name", colors: { $addToSet: "$player_data.Color" } } }
  ])
```

Result Sample:

Player Color Preferences:

- The result of the aggregation reveals each player's unique set of color preferences. For example, "Dreamy" has chosen colors like black, orange, lime, and several others, while "Voxy" has preferred cyan and lime.

Insight into Player Choices:

- These results provide insights into individual players' color choices, which could be used for further analysis on player behavior or preferences within the game.

```
[
  {
    "_id": "Dreamy",
    "colors": [
      "Black",
      "Orange",
      "Lime",
      "Yellow",
      "Pink",
      "Purple",
      "White",
      "Red"
    ]
  },
  {
    "_id": "Voxy",
    "colors": [
      "Cyan",
      "Lime"
    ]
  },
  {
    "_id": "DBatterskull",
    "colors": [
      "Cyan",
      "Blue",
      "",
      "Orange",
      "Lime",
      "Red",
      "Purple",
      "Pink",
      "Green"
    ]
  },
```


Export from MongoDB

5. Create an export from MongoDB in the form given below.

Player name	Games won as imposter	Games won as crew	Win percentage (overall)	Voted Against Imposter	Voted against Crew members	Color preference	Voting rate

Procedure:

Unwinding Nested Arrays:

- The player_data and voting_data arrays were unwound to normalize the data for individual analysis, allowing operations on each element.

Filtering and Aggregation:

- Applied filters to identify specific events, such as impostor ejections and crew votes, followed by aggregation to group results by player name, which is crucial for calculating individual statistics.

Calculating Preferences and Rates:

- Utilized MongoDB's $addToSet and $sum operators within the $group stage to determine color preferences and voting rates, ensuring unique entries and accurate summations.

Merging Collections:

- Performed multiple $lookup operations to join the results of independent aggregation pipelines, merging data on color preferences, win/loss records, and voting statistics into a single collection for each player.

Final Projection and Export:

- Executed a final projection to align the dataset with the desired output format, renaming fields for readability, and then exported the collection to a CSV file, making the data portable and ready for sharing or analysis outside MongoDB.

```
//5 . Create an export from MongoDB in the form given below as player name ,
// games won as imposter, games won as crew, win percentage(overall) voted against imposter,
//voted against crew members , color preference, voting rate

// creating table for color
use('dbAmongUs')
db.Among_Us_data.aggregate([
    { $unwind: "$player_data"},
    { $group: {
        _id: { Player: "$player_data.name", Color: "$player_data.Color"},
        Total: { $sum: 1 }}},
    { $group: {
        _id : "$_id.Player",
        Colors: { $push: { Color: "$_id.Color", Count:"$Total"}} }},
    { $unwind: "$Colors" },
    { $sort: { "Colors.Count": -1 } },
    { $group: { _id: "$_id", details: { $push : "$Colors" }}},
    { $project: { _id:1 , colors: { $first: "$details"}}},
    { $project: { _id:1 , Preference: "$colors.Color" }},
    { $out: "Color"}])


// creating table for win
    use('dbAmongUs')
db.Among_Us_data.aggregate([
    { $unwind: "$player_data"},
    { $unwind: "$Game_Feed"},
    { $match: { "Game_Feed.Outcome": { $regex:"End"}} },
    { $project: {
        "player_data.name": 1,
        Role: { $substr: [ "$player_data.Role", 1, 4 ]} ,
        Result: { $substr: [ "$Game_Feed.Action", 0, 4 ]}}},
    { $project: {
        "player_data.name": 1,
        Win: { $cond: [{ $eq:["$Result","$Role"]}, 1, 0]},
        Crew_Win: { $cond: [{ $and:[{ $eq:["$Result","Crew"]},{ $eq:["$Role","Crew"]}]},1,0]}
```

```
        Imposter_Win: { $cond: [{ $and:[{ $eq:["$Result","Impo"]},{ $eq:["$Role","Impo"]}]},1,0]}},
    { $group: {
        _id: "$player_data.name",
        Played: { $sum:1 },
        Wins: { $sum: "$Win" },
        Win_As_Crew: { $sum: "$Crew_Win" },
        Win_As_Imposter: { $sum: "$Imposter_Win" }}},
    { $project: {
        _id: 1,
        Win_As_Crew: 1,
        Win_As_Imposter: 1,
        Win_Rate:{$multiply:[{ $divide:["$Wins", "$Played"]}, 100]}}},
    { $out: "Win"}])
```

```
    // creating table for 'Voting'
use('dbAmongUs')
db.Among_Us_data.aggregate([
    { $unwind: "$voting_data"},
    { $match: { "voting_data.Is_alive": { $regex:"Yes"}} },
    { $project: {
        "voting_data.name": 1,
        "AgainstCrew":{ $cond:[{ $regexMatch:{ "input":"$voting_data.Vote", "regex": "Crew"}},1,0]},
        "AgainstImpo":{ $cond:[{ $regexMatch:{ "input":"$voting_data.Vote", "regex": "Impostor"}},1,0]} }},
    { $group: {
        _id: "$voting_data.name",
        Voted_Against_Crew: { $sum : "$AgainstCrew" },
        Voted_Against_Impo: { $sum : "$AgainstImpo" },
        Voting_Opportunities: { $sum: 1} }},
    { $project: {
        _id: 1,
        Voted_Against_Crew: 1,
        Voted_Against_Impo: 1,
        Voting_rate:
        { $multiply:[{$divide:[{$add:["$Voted_Against_Crew","$Voted_Against_Impo"]},"$Voting_Opportunities"
    { $out: "Voting"}])
```

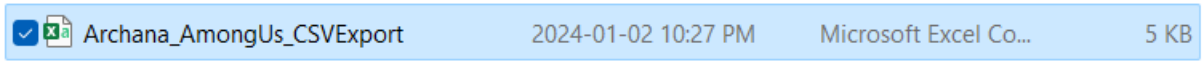
```
// merging all the tables together
use('dbAmongUs')
db.Win.aggregate([
    { $lookup: {
        from: "Voting",
        localField: "_id",
        foreignField: "_id",
        as: "fromVoting" }},
    { $unwind: { path: "$fromVoting", preserveNullAndEmptyArrays: true }},
    { $merge: {
        into: "temp",
        whenMatched: "merge",
        whenNotMatched: "insert" }}])
use('dbAmongUs')
db.Color.aggregate([
    { $lookup: {
        from: "temp",
        localField: "_id",
        foreignField: "_id",
        as: "fromWin_Voting" }},
    { $unwind: { path: "$fromWin_Voting", preserveNullAndEmptyArrays: true }},
    { $merge: {
        into: "Final_data",
        whenMatched: "merge",
        whenNotMatched: "insert" }}])
```

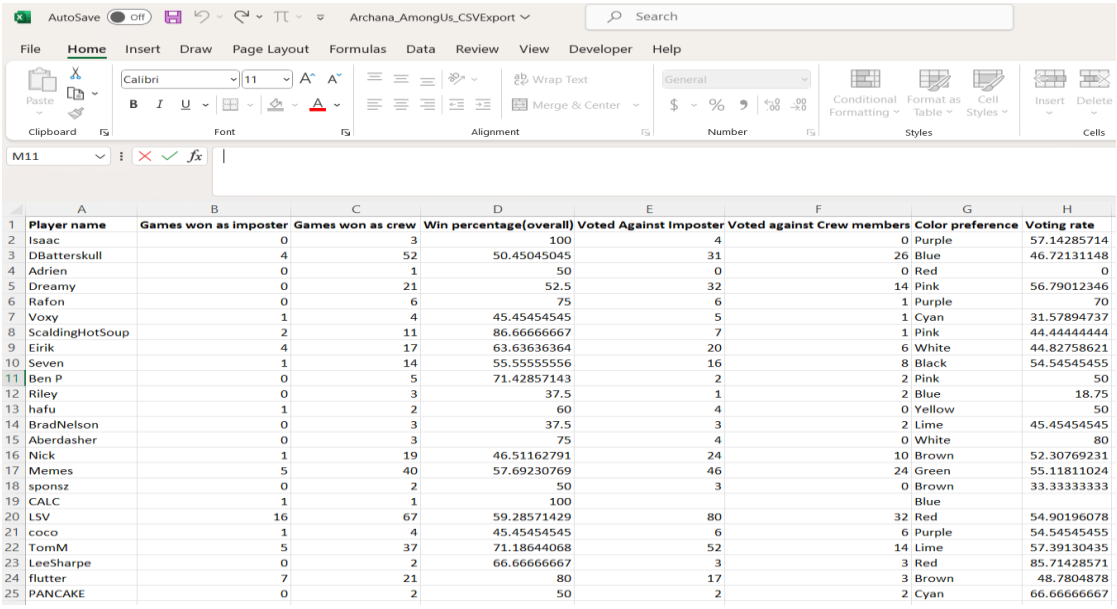
```
///and then rename the fields
use('dbAmongUs')
db.Final_data.aggregate([
    { $project: {
        _id:0,
        "Player name": "$_id",
        "Games won as imposter": "$fromWin_Voting.Win_As_Imposter",
        "Games won as crew": "$fromWin_Voting.Win_As_Crew",
        "Win percentage(overall)": "$fromWin_Voting.Win_Rate",
        "Voted Against Imposter": "$fromWin_Voting.fromVoting.Voted_Against_Impo",
        "Voted against Crew members": "$fromWin_Voting.fromVoting.Voted_Against_Crew",
        "Color preference": "$Preference",
        "Voting rate": "$fromWin_Voting.fromVoting.Voting_rate" }},
    { $out: "AmongUs_CSVExport"}])
```

```
// importing code //
mongoexport --db=dbAmongUs --collection=AmongUs_CSVExport --type=csv --fields="Player name",
"Games won as imposter","Games won as crew","Win percentage(overall)","Voted Against Imposter",
"Voted against Crew members","Color preference","Voting rate"
 --out="C:\Users\ramph\Downloads\Advanced SQL Project\Archana_AmongUs_CSVExport.csv"
```

Result:

- Generated a structured dataset with player names, game wins as impostor and crew, overall win percentages, voting behavior, color preferences, and voting rates, enabling detailed player performance analysis.





	A	B	C	D	E	F	G	H
1	Player name	Games won as imposter	Games won as crew	Win percentage(overall)	Voted Against Imposter	Voted against Crew members	Color preference	Voting rate
2	Isaac	0	3	100	4	0	Purple	57.14285714
3	DBatterskull	4	52	50.45045045	31	26	Blue	46.72131148
4	Adrien	0	1	50	0	0	Red	0
5	Dreamy	0	21	52.5	32	14	Pink	56.79012346
6	Rafon	0	6	75	6	1	Purple	70
7	Voxy	1	4	45.45454545	5	1	Cyan	31.57894737
8	ScaldingHotSoup	2	11	86.66666667	7	1	Pink	44.44444444
9	Eirik	4	17	63.63636364	20	6	White	44.82758621
10	Seven	1	14	55.55555556	16	8	Black	54.54545455
11	Ben P	0	5	71.42857143	2	2	Pink	50
12	Riley	0	3	37.5	1	2	Blue	18.75
13	hafu	1	2	60	4	0	Yellow	50
14	BradNelson	0	3	37.5	3	2	Lime	45.45454545
15	Aberdasher	0	3	75	4	0	White	80
16	Nick	1	19	46.51162791	24	10	Brown	52.30769231
17	Memes	5	40	57.69230769	46	24	Green	55.11811024
18	sponsz	0	2	50	3	0	Brown	33.33333333
19	CALC	1	1	100			Blue	
20	LSV	16	67	59.28571429	80	32	Red	54.90196078
21	coco	1	4	45.45454545	6	6	Purple	54.54545455
22	TomM	5	37	71.18644068	52	14	Lime	57.39130435
23	LeeSharpe	0	2	66.66666667	3	3	Red	85.71428571
24	flutter	7	21	80	17	3	Brown	48.7804878
25	PANCAKE	0	2	50	2	2	Cyan	66.66666667

Additional Statistics

- 6.In tasks 4 and 5, you create individual player statistics.

Discuss additional statistics that might be worth exploring (aside from those already calculated) while selecting players. (There's no requirement to write code for them.)

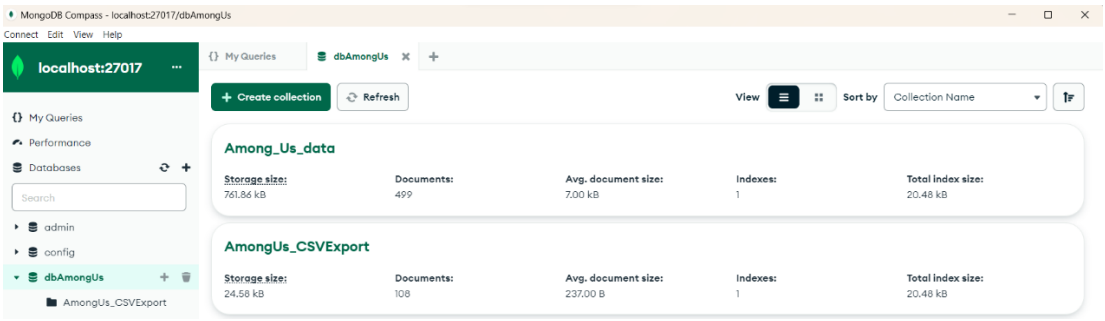
Answers:

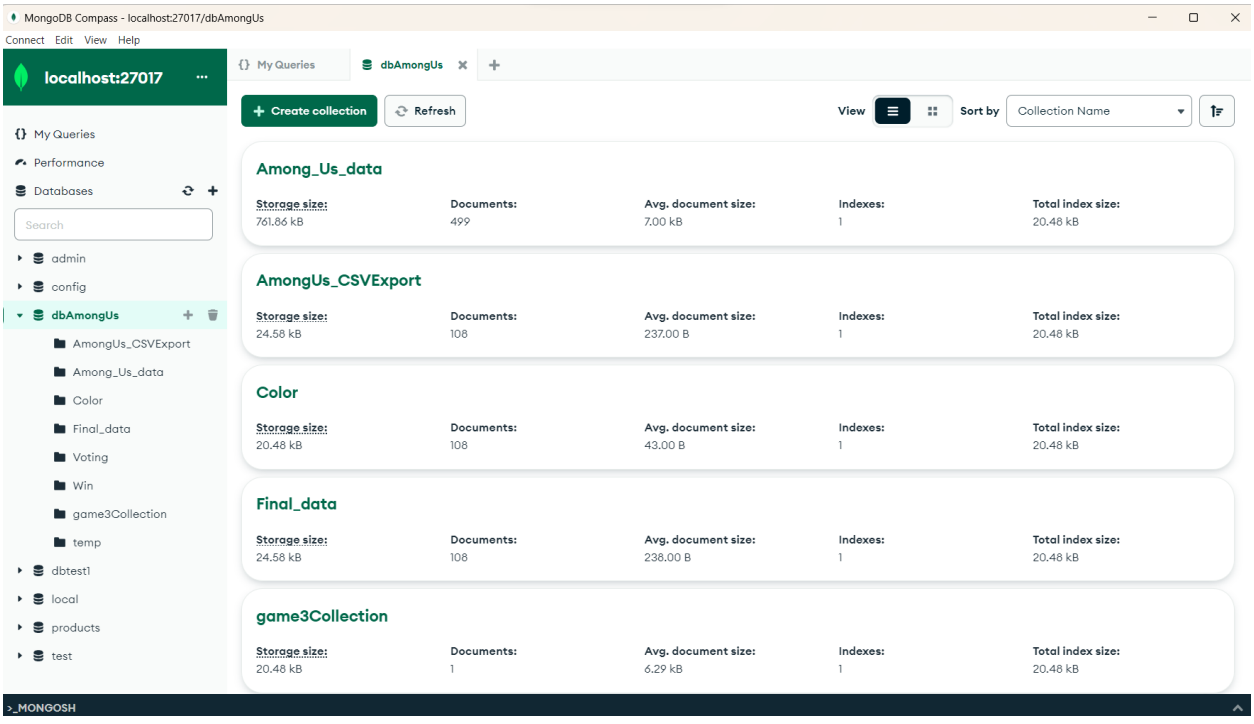
- **Task Efficiency:** We can Calculate how efficiently a player completes tasks as a crew member, indicating their diligence in fulfilling crew duties.
- **Survival Skill:** By Determining the percentage of games a player survives until the end, showcasing their ability to stay alive.
- **Impostor win Rate:** Calculating the percentage of impostor games won by the player, reflecting their effectiveness in deceiving the crew.
- **Crew Victory Rate:** Calculating the percentage of crew member games won by the player, demonstrating their skill in identifying impostors and completing tasks.
- **Meeting Engagement:** We can Count how often a player calls emergency meetings, revealing their involvement in game discussions.
- **Teamwork:** Analyzing if a player prefers working alone or collaborates with others to achieve game goals.
- **Vote Patterns:** Studying a player's voting behavior, such as who they vote for and their preferences for impostors or crew members.
- **Time Played:** Calculating the total time a player spends in the game, indicating their experience level.
- **Communication:** Can analyze in-game chat patterns and communication frequency among players.
- **Consistency:** We can Assess a player's performance consistency across multiple games.
- **Preferred Roles:** To Identify a player's preferred role (crew or impostor) and calculate their performance and win rates based on their preference.

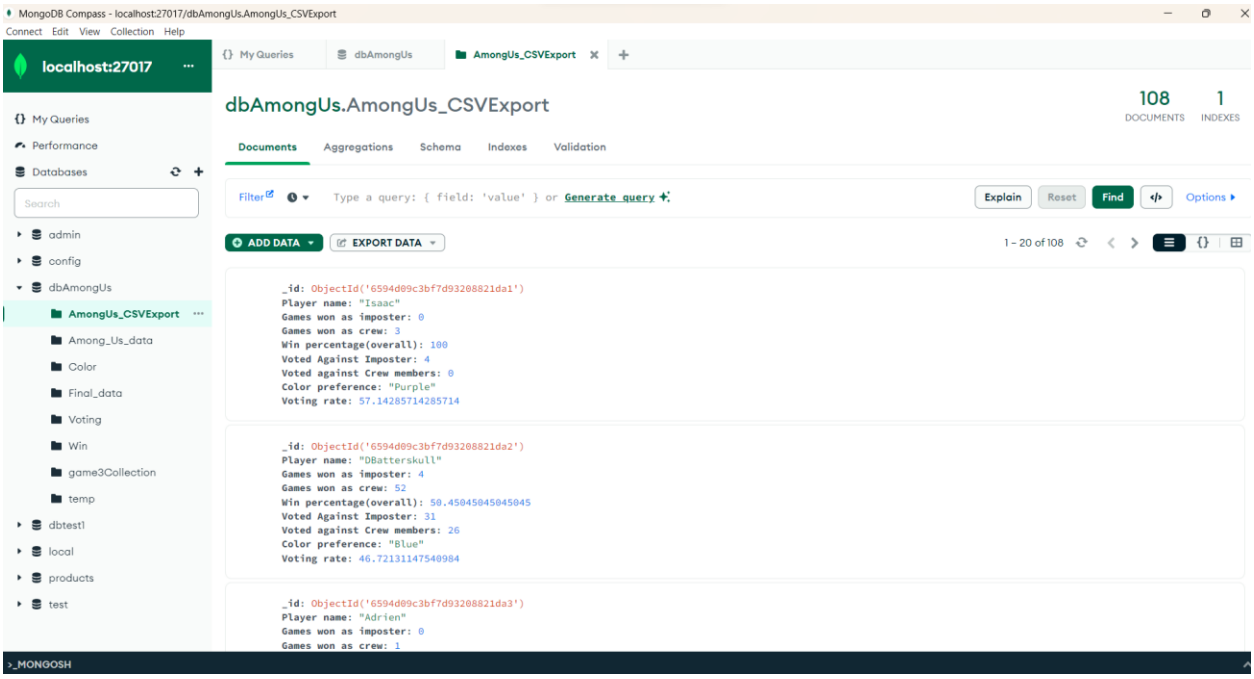
- **Tasks Completed as Impostor:** Calculating the number of tasks completed by the player while playing as an impostor.
- **Task Interruption as Impostor:** We can Count how many times a player disrupts crew members while they are completing tasks as an impostor.
- **Map Proficiency:** Analyzing a player's performance on different maps to identify strengths and weaknesses.
- **Social Interaction:** Can perform a study about player's interactions with others, including alliances, conflicts, and cooperation.

MongoDB Project Report

➢ MongoDB Compass :







Conclusion:

Data Consolidation: The MongoDB Compass screenshots here shows I aggregated data into a single collection named AmongUs_CSVExport, which now holds 108 documents for export.

Export Functionality: The Compass interface indicates the EXPORT DATA option is active, suggesting that the collection is ready for data export, typically to a CSV or JSON file.

Structured Output: The AmongUs_CSVExport collection contains structured data including player stats such as win rates and voting behavior, confirming the successful execution of the data pipeline.

Collection Metrics: The AmongUs_CSVExport collection has 108 documents, each likely representing a unique player, signifying a comprehensive export of player data.

Data Accessibility: The data within AmongUs_CSVExport is presented in a format that is accessible and interpretable, suitable for analysis or reporting.