GROUP 3

NETFLICKS

START

TABLE OF CONTENTS

OVERVIEW

- **Top 10 weekly report:**
    - **TV shows & Movies**
- **Ranked by viewership**
- **Rankings and filters are based on:**
    - Genre
    - Release year
    - Duration
    - Rating
    - Country
    - Subtitles availability
    - Awards
    - IMDb rating

INDEXES

- Indexes in MongoDB are data structures that **improve the speed of data retrieval operations** by allowing the database to **quickly locate documents** based on indexed fields.
- For example, creating an index on the "type" field enables faster retrieval of TV shows or movies.
- Visualizing indexes as a **library catalog** can help understand how they organize data for efficient retrieval.

TOP 10 TV SHOWS OF THE WEEK

Ranked by viewership

Query #1:
Top 10 TV shows of the week, ranked by viewership.

Chosen Query:

```
db.movies.createIndex({ "type": 1 })
db.movies.createIndex({ "type": 1, "weekly_viewership.week_start_date": 1 })
```

```
db.movies.find(
  { "type": "TV Show", "weekly_viewership.week_start_date": "2024-04-01" }
  { "title": 1, "weekly_viewership.views": 1, "_id": 0 }
   ).sort({ "weekly_viewership.views": -1 }).limit(10)
```

Why We Chose This Query

Readability
more readable, straightforward and intuitive

Performance
for simple retrieval and sorting tasks,
this query with find() function can be faster since it's a basic query operation.

Considering the dataset
since weekly_viewership is an embedded document, and we are performing a straightforward query with sorting and limiting, this query with find() is sufficient and likely to perform better

TOP 10
MOVIES & TV SHOWS

In a specific country,
ranked by viewership

Query #2:
Top 10 TV Shows & Movies in a Country, Ranked by Viewership

Chosen Query:

```
db.movies.aggregate
([{$match:{country:"US"}}, {$sort:{weekly_viewership:-1}}, {$limit:10}])
```

- `{$match:{country:"US"}}`
  Filters the documents to include only the particular country

- `{$sort:{weekly_viewership:-1}}`
  Sorts filtered documents by weekly views in descending order

- `{$limit:10}`
  Limits the results to the top 10 movies or shows

Why We Chose This Query

Efficienty
Uses a compound index, which is one of the fastest ways to run the query

```
db.movies.createIndex({country:1, weekly_viewership:-1})
```

High read-to-write ratio
Provides insight into how well certain titles are performing in specific regions,
so we can understand audience preferences

Improves customer experience
Allows us to make popular titles more readily available,
reducing latency and leading to higher customer loyalty and retention rates

Future Index Scenarios Based on More Information

Diverse Patterns Across Fields

We would consider creating **multiple single-field indexes or additional compound indexes** that include other frequently queried fields like genre or release_year.

This would cater to a broader range of query patterns.

Change in Data

If the system evolves to have a higher frequency of data updates or additions, we might opt for **fewer indexes or more selective indexing strategies** to balance the write and read performance

TOP 10 MOVIES
BY IMDB RANKING

Query #3:
Top 10 Movies by IMDb rank

We needed to create a compound index that includes both the “Type” to filter for movies, and “imdb_rating” to return the titles ranked in descending order.

Index creation command

```
db.movies.createIndex({ "imdb_rating": -1, "type": 1 })
```

In order to test our new index, we ran the following query:

```
db.movies.find({ "type": "Movie" }).sort({ "imdb_rating": -1 }).limit(10)
```

To observe the performance of our index and monitor both the execution time and resource usage

```
db.movies.aggregate([{$indexStats:{}}])
```

Why We Chose This Query

Selectivity
We created this index such that it results in high selectivity.
To ensure selectivity, our query limits the number of possible documents with the indexed field, thus reducing the size and computing power needed.

Recognization
We used this command to find movies, and this efficient method quickly gets you the most recommended films without sifting through everything.

Maintain Efficiency
Accessing files quickly and having accurate counts to show how often and when each index is used helps to maintain an efficient filing system.

Future Index Scenarios Based on More Information

Diversified Query Patterns

As the platform grows, we may need to query movies by additional attributes such as genre, director, or release year. Introducing compound indexes that include these fields will enhance filtering and sorting efficiency, reducing the need for extensive document scans.

Increased Data Volume

As the movie dataset expands, query performance may slow, and index storage needs could increase. To manage this, we would use partial indexes that only index movies released in the last 10 years, optimizing performance and storage.

THANK YOU