

Lab 2: SJSU Movie Database using MongoDB

Guided By : Professor Simon Shim

Sadakha Narnur
Masters in Data Analytics
San Jose State University
California
sadakha.narnur@sjsu.edu

Nikitha Bramadi
Masters in Data Analytics
San Jose State University
California
nikitha.bramadi@sjsu.edu

Pallavi Jain
Masters in Data Analytics
San Jose State University
California
pallaviamrutkumar.jain@sjsu.edu

Abstract— SJSU Movie Database is implemented using MongoDB . The system is modularized into Movies, Users and Service Request modules. Each of which are organized into collections and queried. The Users collection follows an embedded document structure with references to the Movies and Service Request collections.

Keywords—Collection, Documents, Schema, ObjectId, Aggregation.

DatasetURL: <https://www.kaggle.com/PromptCloudHQ/imdb>

I. PROBLEM STATEMENT

The IMDb movie database serves as a useful repository of movie information for our application system. In order to create a low latency solution for viewing movies, creating users and profiles and providing support, there is a need for efficient document design.

II. SOLUTION REQUIREMENTS

The goal of our application would be to present users with access to the movies for a monthly subscription of 19.95 and allow 5 members access. Users will be logged in to the application and presented with features like adding or deleting profiles, watching movies, adding to their movie list, access watch history, view monthly bills and support with any account related issues.

III. CONCEPTUAL DATABASE DESIGN

Schema Design

In MongoDB there are two different kinds of schema representations. One being the **Embedding** and the other being **Referencing** between the collections. Modeling the data depends on the application's data access patterns. We should structure the data to match the ways that our application queries and updates it. Every application has unique needs and requirements, so the schema design should reflect the needs of that particular application.

Here we will discuss the needs of our application from 3 different point of views:

1. A registered SJSU movie DB user should be able to perform the following activities:
 - Access movies hosted by the SJSU movie database
 - Create upto 5 profiles under one registered user. Each profile can independently choose to stream their own movies
 - Each profile can create their own list called my_list.
 - A user pays the monthly bills for the access to continue. And choose to discontinue if needed. User can choose their own payment method
 - Create an SR in order for any kind of request for help from the database owners
2. The service agent who is also the owner of the database, takes the ownership of the following dependent requirements:
 - When a user profile raises a service request, make sure that the complaint is assigned to an available service agent. At the same time take necessary actions towards that request
3. The application itself will be taking care of the following:
 - Each user profile history will be recorded
 - Bill will be generated for every month
 - Tracks the user registration date
 - Tracks the number of profiles created

For the easy and efficient outcome of these duties mentioned above we have used a combination of both Embedded and Referencing.

All the required operations in the application are done efficiently by our hierarchical approach to the system. The application is constructed into an embedded collection and in 2 cases references have been added to the rest of the collections.

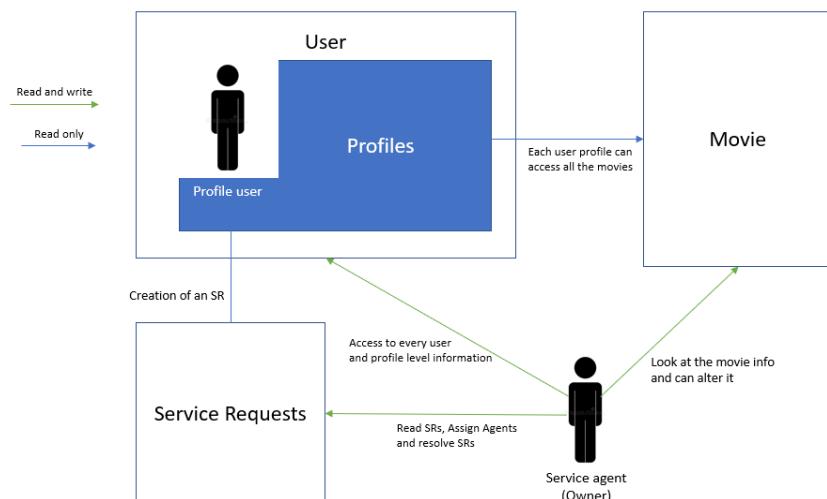


Fig 1. System design in terms of collections

B. Conceptual Design

Keeping the requirements in mind, we have created the following 3 collections.

1. users
2. movie
3. service_requests

Collection	Schema of the data
users	Embedding profiles and their activities, References the movie collection and References the service_requests
movie	Embeds all the information including the movie actors and genres
service_requests	Embeds all the information related to service requests

IV. DOCUMENT STRUCTURE

The core functionalities of the application are divided among the three collections.

```
[Atlas atlas-100xuq-shard-0 [primary] lab2> show collections;
movies
service_requests
users
```

Movies Collection :

- It is the base collection that has references from the User collection.
- This has complete data of the movies which is used by another collection. It has fields of different data types including Arrays.
- New movie records can be added and all the details for movies are available, which can be updated and deleted by the admin.
- Genre and Actors are the fields that are arrays of strings.

Field	Datatype
_id	ObjectId
imdb_rank	int
title	String
genre	Array[String]

actors	Array[String]
year	Int
runtime (minutes)	Int
rating	Float
votes	Int
revenue (millions)	float
metascore	int

The sample Movie document structure is as follows:

```

_id: ObjectId('6275909b42b44ac7f586752f')
Rank: 2
Title: "Prometheus"
Genre: Array
  0: "Adventure"
  1: "Mystery"
  2: "Sci-Fi"
Description: "Following clues to the origin of mankind, a team finds a structure on ..."
Director: "Ridley Scott"
Actors: Array
  0: "Noomi Rapace"
  1: "Logan Marshall-Green"
  2: "Michael Fassbender"
  3: "Charlize Theron"
Year: "2012"
Runtime (Minutes): 124
Rating: 7
Votes: 485820
Revenue (Millions): 126.46
Metascore: 65

```

User Collection:

- This collection begins with the registration of a new user which inserts a new document into the collection.
- This collection includes all the details of a user, registration date, profiles count, status and etc.
- Profiles is an embedded document that has the profile details associated with a user account.
- Profile_history and Profile_my_list are nested documents within the Profiles embedded document. They have references to the Movies collection.
- Complaints is an array of ObjectId where it has references to the Service_requests collection.

Fields	Datatype
--------	----------

_id	ObjectId
emailid	String
phoneno	String
name	String
dob	Date
password	String
account_create_date	date
no_of_profiles	Int
preferred_language	String
profiles	Array[Struct]
profiles.profile_name	String
profiles.service_id	Int
profiles.profile_history	Array[Struct]
profiles.profile_history.view_timestamp	Date
profiles.profile_history.imdb_rank	Int
profiles.profile_history.movie_id	ObjectId
profiles.profile_my_list	Array[Struct]
profiles.profile_my_list.imdb_rank	Int
profiles.complaints	Array[ObjectId]
profiles.profile_my_list.movie_id	ObjectId
billing_date	date
status	String
payment_method	String

Sample User document structure as follows:

```

_id: ObjectId("627058250e2a53c322e3b170")
emailid: "5BzoRFTP6tdHfs@hotmail.com"
phoneno: "512-974-6533"
name: "michael smith"
dob: "1917-11-19"
password: "319fat5"
account_create_date: "2020-10-01"
no_of_profiles: "1"
preferred_language: "Hindi"
profiles: Array
  ▼ 0: Object
    profile_name: "EON"
    service_id: "400005"
    ▼ profile_history: Array
      ▼ 0: Object
        view_timestamp: "2021-07-29 15:16:17"
        imdb_rank: "67"
        movie_id: ObjectId("62701cf0e2a53c322e3a1d4")
    ▼ profile_my_list: Array
      ▼ 0: Object
        imdb_rank: "4"
        movie_id: ObjectId("62701cf0e2a53c322e3a195")
    ▼ complaints: Array
      ▼ 0: Object
        _id: ObjectId("627057a10e2a53c322e3b0e7")
      > 1: Object
    billing_date: "2020-11-01"
status: "Active"
payment_method: "DEBIT/CREDIT"

```

Service_Requests Collection:

- A profile user raises a service request describing the problem.
- This consists of all the details of a service ticket raised like the description, severity, agent assigned, date of complaint and profiles' service id.
- Any new Service request added will be referenced from the User.profiles.complaint.

Field	Datatype
_id	ObjectId
complaint_severity	String
service_agent_name	String
service_id	Int
date_of_complaint	Date
complaint_description	String
emailid	String

Sample service_request document structure:

```

_id: ObjectId("627057a10e2a53c322e3b0e2")
complaint_severity: "m"
service_agent_name: "Sandeshn"
service_id: "400000"
date_of_complaint: "2022-01-02"
complaint_description: "Buffering"
emailid: "OoececBAnkd@yahoo.com"

```

V. DENORMALIZATION

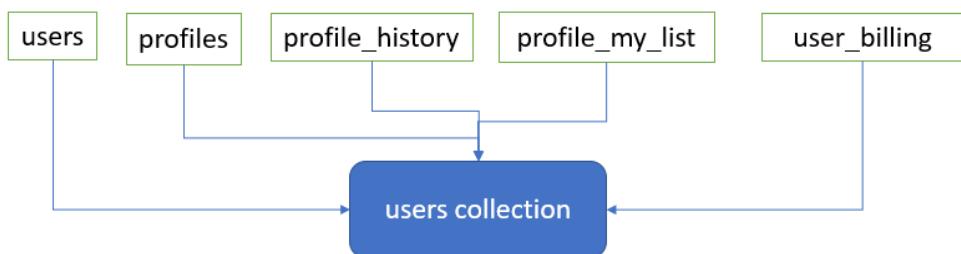
The major consideration that needs to be taken while designing our schema is the weightage of reads to update. It is always a good practice to denormalize data that will be often read and less frequently updated.

We have taken a collection design that optimizes the reads by avoiding joins and performing a \$lookup as much as possible. To avoid join we choose the process of embedding which results in a denormalized structure of the collection.

We can showcase our denormalization process in reference to the database we have created in the MySQL database.

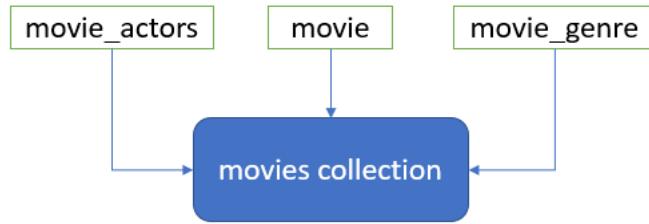
User Collection Denormalization:

All the Profile details are embedded into the User document and the watch history of a profile, the wishlist(my_list) for a profile can be embedded or nested within the Profiles. This led to the denormalizing of fields and merging into a single collection.



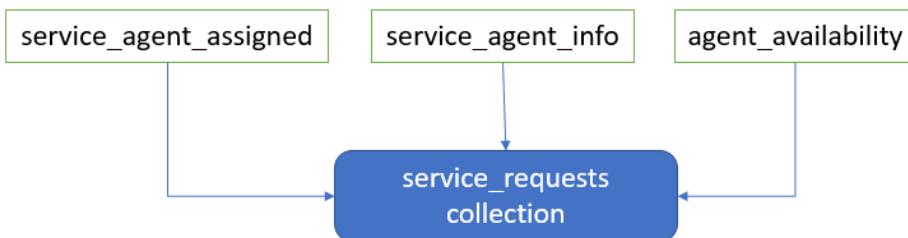
Movies Collection Denormalization:

Actors and Genres are multi value fields that need to be normalized for a relational structure. However since there is a flexibility to add these fields as an array of strings they are denormalized and embedded into the same Movie collection which alleviates the joining requirement.



Service Requests Collection Denormalization:

All the details that are normalized into multiple tables for a relational database can be now merged into a single document that enables Read operations to be done in a very less execution time.



VI. FEATURES

The user of the application will have access to the following features:

- **New user registers into the application :** This is a feature that is implemented through the insertOne query for adding the user document to the User collection. All the required fields will be added to the user document.

```
db.users.insertOne({ "emailid": "new_user@gmail.com", "phoneno": "322-370-1228", "name": "Ratan Harry", "dob": new Date("2001-11-12"), "password": "qwerty12#", "account_create_date": new Date("2010-10-07"), "no_of_profiles": 0, "preferred_language": "English", "profiles": [], "billing_date": new Date("2022-04-03"), "status": "Active", "payment_method": "DEBIT/CREDIT" })
```

```
{
  acknowledged: true,
  insertedId: ObjectId("6273091e95942a06cc811d90")
}
```

```
_id: ObjectId('62755e474fd65fe5ef9f6f88')
emailid: "new_user@gmail.com"
phoneno: "322-370-1228"
name: "Ratan Harry"
dob: 2001-11-12T00:00:00.000+00:00
password: "qwerty12#"
account_create_date: 2010-10-07T00:00:00.000+00:00
no_of_profiles: 0
preferred_language: "English"
> profiles: Array
  billing_date: 2022-04-03T00:00:00.000+00:00
  status: "Active"
  payment_method: "DEBIT/CREDIT"
```

- **User adds a profile into his account:** A user can add up to 5 profiles to his accounts and the application runs an update on no of profiles to increment the count when new profile is added, which will be achieved as follows:

- db.users.updateOne({ "emailid": "new_user@gmail.com" }, { "\$push": { "profiles": { "profile_name": "Profile1", "service_id": 80012, "profile_history": [], "profile_my_list": [], "complaints": [] } } })

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Updating the no_of_profiles:

- db.users.updateOne({ "emailid": "new_user@gmail.com" }, { "\$inc": { "no_of_profiles": 1 } })

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```

_id: ObjectId('62755e474fd65fe5ef9f6f88')
emailid: "new_user@gmail.com"
phoneno: "322-370-1228"
name: "Ratan Harry"
dob: 2001-11-12T00:00:00.000+00:00
password: "qwerty12#"
account_create_date: 2010-10-07T00:00:00.000+00:00
no_of_profiles: 1
preferred_language: "English"
profiles: Array
  0: Object
    profile_name: "Profile1"
    service_id: 80012
    > profile_history: Array
    > profile_my_list: Array
    > complaints: Array
    billing_date: 2022-04-03T00:00:00.000+00:00
    status: "Active"
    payment_method: "DEBIT/CREDIT"

```

- **Adding a movie to Profile watching list:** A movie can be added to profiles' my list that can be later used to list the movie wishlist. The movie name is looked up in the movies collection and the Reference ObjectId is stored in the users document.

- var movie = db.movies.findOne({ "Title": "Guardians of the Galaxy" })
- var movieId = movie._id
- var movie_rank = movie.Rank
- db.users.updateOne({ "profiles.profile_name": "Profile1", "emailid": "new_user@gmail.com" }, { "\$push": { "profiles.\$.profile_my_list": { "imdb_rank": movie_rank, "movie_id": movieId } } })

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```

_id: ObjectId('62755e474fd65fe5ef9f6f88')
emailid: "new_user@gmail.com"
phoneno: "322-370-1228"
name: "Ratan Harry"
dob: 2001-11-12T00:00:00.000+00:00
password: "qwertyl2#"
account_create_date: 2010-10-07T00:00:00.000+00:00
no_of_profiles: 2
preferred_language: "English"
profiles: Array
  0: Object
    profile_name: "Profile1"
    service_id: 80012
    profile_history: Array
    profile_my_list: Array
      0: Object
        imdb_rank: 1
        movie_id: ObjectId('627564ef42b44ac7f586713e')
    complaints: Array
  billing_date: 2022-04-03T00:00:00.000+00:00
  status: "Active"
  payment_method: "DEBIT/CREDIT"

```

- **Adding a movie to Profiles' history:** Everytime a profile user watches a movie it is added into their Profile_history with the watched timestamp.

- var movie = db.movies.findOne({ "Title": "Prometheus" })
- var movieId = movie._id
- var movie_rank = movie.Rank

```

db.users.updateOne({ "profiles.profile_name" : "Profile1",
"emailid": "new_user@gmail.com"}, { "$push": { "profiles.$profile_history": {
"view_timestamp": new Date(Date.now()), "imdb_rank": movie_rank,
"movie_id": movieId} } })

```

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```

_id: ObjectId('62755e474fd65fe5ef9f6f88')
emailid: "new_user@gmail.com"
phoneno: "322-370-1228"
name: "Ratan Harry"
dob: 2001-11-12T00:00:00.000+00:00
password: "qwerty12#"
account_create_date: 2010-10-07T00:00:00.000+00:00
no_of_profiles: 2
preferred_language: "English"
profiles: Array
  0: Object
    profile_name: "Profile1"
    service_id: 80012
    profile_history: Array
      0: Object
        view_timestamp: 2022-05-06T18:41:28.011+00:00
        imdb_rank: 1
        movie_id: ObjectId('627564ef42b44ac7f586713e')
    profile_my_list: Array
      0: Object
        imdb_rank: 1
        movie_id: ObjectId('627564ef42b44ac7f586713e')
    complaints: Array
  billing_date: 2022-04-03T00:00:00.000+00:00

```

- **Deleting a Profile:** A user can delete a profile from their account which is done as follows:

```

● db.users.update({ "emailid": "0omsUvRjUHZlzEPT@msn.com" }, { $pull: { "profiles": { "profile_name": "sush" } } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}

```

Updating the no_of_profiles:

- db.users.updateOne({ "emailid": "0omsUvRjUHZlzEPT@msn.com" }, { "\$inc": { "no_of_profiles": -1 } })

```

_id: ObjectId('6271ee61bb962b8695dd11af')
emailid: "0omsUvRjUHZlzEPT@msn.com"
phoneno: "512-974-2015"
name: "john martin"
dob: 1992-05-03T00:00:00.000+00:00
password: "lunareon"
account_create_date: 2017-12-19T00:00:00.000+00:00
no_of_profiles: 3
preferred_language: "Hindi"
profiles: Array
  0: Object
    profile_name: "niva"
    service_id: 400234
    profile_history: Array
    profile_my_list: Array
    complaints: Array
  1: Object
    profile_name: "sush"
    service_id: 400235
    profile_history: Array
    profile_my_list: Array
    complaints: Array
  2: Object
    profile_name: "shat"
    service_id: 400236
    profile_history: Array
    profile_my_list: Array
    complaints: Array
  billing_date: 2018-01-19T00:00:00.000+00:00
  status: "Active"
  payment_method: "DEBIT/CREDIT"

```

- **Deleting a movie from Profiles' watching list:** A movie can be deleted from profiles' my list as follows:

Before deleting:

```

_id: ObjectId('62755e474fd65fe5ef9f6f88')
emailid: "new_user@gmail.com"
phoneno: "322-370-1228"
name: "Ratan Harry"
dob: 2001-11-12T00:00:00.000+00:00
password: "qwerty12#"
account_create_date: 2010-10-07T00:00:00.000+00:00
no_of_profiles: 1
preferred_language: "English"
profiles: Array
  0: Object
    profile_name: "Profile1"
    service_id: 80012
    profile_history: Array
      0: Object
        view_timestamp: 2022-05-06T20:30:55.858+00:00
        imdb_rank: 2
        movie_id: ObjectId('627564ef42b44ac7f586713f')
    profile_my_list: Array
      0: Object
        imdb_rank: 1
        movie_id: ObjectId('627564ef42b44ac7f586713e')
    complaints: Array
  billing_date: 2022-04-03T00:00:00.000+00:00
  status: "Active"
  payment_method: "DEBIT/CREDIT"

```

Let's say the profile user requested the movie on his list 'Guardians of the Galaxy' to be removed from his list. Hence we will be doing the following steps

```

var movie = db.movies.findOne({ "Title": "Guardians of the Galaxy" })
var movieId = movie._id
var movie_rank = movie.Rank
db.users.update({ "emailid": "new_user@gmail.com", "profiles.profile_name": "Profile1" }, {
$pull: { "profiles.$profile_my_list": {imdb_rank: movie_rank} } }, { multi: true })

```

After executing the query:

```

_id: ObjectId('62755e474fd65fe5ef9f6f88')
emailid: "new_user@gmail.com"
phoneno: "322-370-1228"
name: "Ratan Harry"
dob: 2001-11-12T00:00:00.000+00:00
password: "qwerty12#"
account_create_date: 2010-10-07T00:00:00.000+00:00
no_of_profiles: 1
preferred_language: "English"
profiles: Array
  0: Object
    profile_name: "Profile1"
    service_id: 80012
    profile_history: Array
      0: Object
        view_timestamp: 2022-05-06T20:52:15.948+00:00
        imdb_rank: 2
        movie_id: ObjectId('627564ef42b44ac7f586713f')
    profile_my_list: Array
    complaints: Array
  billing_date: 2022-04-03T00:00:00.000+00:00
  status: "Active"
  payment_method: "DEBIT/CREDIT"

```

- **Deleting a movie from Profiles' history:** A movie can be deleted from Profile history as follows:

The application deletes the history whenever the user requests. It can be done in 2 ways:

1. Delete entire history

```
db.users.updateOne({"emailid":"new_user@gmail.com","profiles.profile_name":"Profile1"},{$unset:{"profiles.$profile_history":""}},{"multi:true})
```

```
_id: ObjectId('62755e474fd65fe5ef9f6f88')
emailid: "new_user@gmail.com"
phoneno: "322-370-1228"
name: "Ratan Harry"
dob: 2001-11-12T00:00:00.000+00:00
password: "qwerty12#"
account_create_date: 2010-10-07T00:00:00.000+00:00
no_of_profiles: 1
preferred_language: "English"
profiles: Array
  0: Object
    profile_name: "Profile1"
    service_id: 80012
    profile_my_list: Array
    complaints: Array
    billing_date: 2022-04-03T00:00:00.000+00:00
    status: "Active"
    payment_method: "DEBIT/CREDIT"
```

2. Delete a specific movie from the history

Before:

```
_id: ObjectId('62755e474fd65fe5ef9f6f88')
emailid: "new_user@gmail.com"
phoneno: "322-370-1228"
name: "Ratan Harry"
dob: 2001-11-12T00:00:00.000+00:00
password: "qwerty12#"
account_create_date: 2010-10-07T00:00:00.000+00:00
no_of_profiles: 1
preferred_language: "English"
profiles: Array
  0: Object
    profile_name: "Profile1"
    service_id: 80012
    profile_history: Array
      0: Object
        view_timestamp: 2022-05-06T20:52:15.948+00:00
        imdb_rank: 2
        movie_id: ObjectId('627564ef42b44ac7f586713f')
      1: Object
        view_timestamp: 2022-05-06T23:47:00.139+00:00
        imdb_rank: 1
        movie_id: ObjectId('6275909b42b44ac7f586752e')
    profile_my_list: Array
    complaints: Array
    billing_date: 2022-04-03T00:00:00.000+00:00
    status: "Active"
    payment_method: "DEBIT/CREDIT"
```

- var movie = db.movies.findOne({"Title": "Guardians of the Galaxy"})
- var movie_rank = movie.Rank
- db.users.update({"emailid": "new_user@gmail.com", "profiles.profile_name": "Profile1"}, { \$pull: {"profiles.\$profile_history": {imdb_rank: movie_rank} } }, { multi: true })

After:

```

_id: ObjectId('62755e474fd65fe5ef9f6f88')
emailid: "new_user@gmail.com"
phoneno: "322-370-1228"
name: "Ratan Harry"
dob: 2001-11-12T00:00:00.000+00:00
password: "qwerty12#"
account_create_date: 2010-10-07T00:00:00.000+00:00
no_of_profiles: 1
preferred_language: "English"
profiles: Array
  0: Object
    profile_name: "Profile1"
    service_id: 80012
    profile_history: Array
      0: Object
        view_timestamp: 2022-05-06T20:52:15.948+00:00
        imdb_rank: 2
        movie_id: ObjectId('627564ef42b44ac7f586713f')
    > profile_my_list: Array
    > complaints: Array
billing_date: 2022-04-03T00:00:00.000+00:00
status: "Active"
payment_method: "DEBIT/CREDIT"

```

- **Adding a new movie:** The admin can add new movies to the movie collection.
 - db.movies.insertOne({ "imdb_rank": "1001", "Title": "Fantastic Beasts - The Secrets of Dyumbledore", "Genre": ["Action", "Adventure", "Sci-Fi"], "Description": "Professor Albus Dumbledore knows the powerful, dark wizard Gellert Grindelwald is moving to seize control of the wizarding world. ", "Director": "David Yates", "Actors": "", "Year": "2022", "Runtime (Minutes)": "121", "Rating": "8.1", "Votes": "757074", "Revenue (Millions)": "333.13", "Metascore": "80" })

```

_id: ObjectId("6276f9069caf477874466b98")
imdb_rank: "1001"
Title: "Fantastic Beasts – The Secrets of Dyumbledore"
> Genre: Array
Description: "Professor Albus Dumbledore knows the powerful, dark wizard Gellert Gri..."
Director: "David Yates"
Actors: ""
Year: "2022"
Runtime (Minutes): "121"
Rating: "8.1"
Votes: "757074"
Revenue (Millions): "333.13"
Metascore: "80"

```

- **Deleting a movie from Movie Collection:** Admin can delete a movie from the movie collection as follows:
 - db.movies.deleteOne({ "Title": "Fantastic Beasts - The Secrets of Dyumbledore" });

{ acknowledged: true, deletedCount: 1 }

- **Creating a Service request:** A user raises a service request with their service id.

When a user raises a service request.

```
db.service_requests.insertOne({service_id:400019, date_of_complaint:new Date(Date.now()), complaint_description:"Problems with the subtitles or captions, including quality or availability", emailid:"uqCdy@hotmail.com"})
```

```
{
  _id: ObjectId("62703429bb962b8695dd0d24"),
  service_id: 400019,
  date_of_complaint: ISODate("2022-01-21T00:00:00.000Z"),
  complaint_description: 'Problems with the subtitles or captions, including quality or availability',
  emailid: "uqCdy@hotmail.com"
},
```

When the service agent acknowledges the request raised.

```
db.service_requests.updateOne({service_id:400019}, {$set:{complaint_severity: "h", service_agent_name:"Preetha"}},{upsert:true})
```

```
[
  {
    _id: ObjectId("62703429bb962b8695dd0d24"),
    complaint_severity: 'm',
    service_agent_name: 'Preetha',
    service_id: 400019,
    date_of_complaint: ISODate("2022-01-21T00:00:00.000Z"),
    complaint_description: 'Problems with the subtitles or captions, including quality or availability',
    emailid: "uqCdy@hotmail.com"
  }
]
```

Adding a reference to this complaint raised in the data of the profile

```
var serviceid = db.service_requests.findOne({"service_id":400019})
var complaintId = serviceid._id
db.users.updateOne({ "profiles.service_id" : 400019, "emailid":"uqCdy@hotmail.com"}, { "$push": { "profiles.$ .complaints": { _id :complaintId}}});
```

```
{
  _id: ObjectId("6271ee61bb962b8695dd12f9"),
  emailid: 'uqCdy@hotmail.com',
  phoneno: '512-974-5022',
  name: 'james williams',
  dob: ISODate("1982-01-01T00:00:00.000Z"),
  password: 'zr1659',
  account_create_date: ISODate("2017-10-01T00:00:00.000Z"),
  no_of_profiles: 3,
  preferred_language: 'English',
  profiles: [
    {
      profile_name: 'Paid for the Account',
      service_id: 400019,
      profile_history: [
        {
          view_timestamp: ISODate("2020-01-21T05:00:00.000Z"),
          imbd_rank: 366,
          movie_id: ObjectId("625ddf2139b486fd700dd6c2")
        }
      ],
      profile_my_list: [],
      complaints: [
        { _id: ObjectId("62703429bb962b8695dd0d24") },
        { _id: ObjectId("62703429bb962b8695dd0d74") },
        { _id: ObjectId("627340c030b048a05181c75b") }
      ]
    },
    {
      profile_name: 'My Profile',
      service_id: 400019,
      profile_history: [
        {
          view_timestamp: ISODate("2020-01-21T05:00:00.000Z"),
          imbd_rank: 366,
          movie_id: ObjectId("625ddf2139b486fd700dd6c2")
        }
      ],
      profile_my_list: [],
      complaints: [
        { _id: ObjectId("62703429bb962b8695dd0d24") },
        { _id: ObjectId("62703429bb962b8695dd0d74") },
        { _id: ObjectId("627340c030b048a05181c75b") }
      ]
    }
  ]
}
```

- **Deleting a user:** Admin may delete any user from the users collection.

```
db.users.deleteOne( { emailid : "2Q@msn.com" } )
```

```
Atlas atlas-j2215d-shard-0 [primary] sjsu_movie_db> db.users.deleteOne( { emailid : "2Q@msn.com" } )
{ acknowledged: true, deletedCount: 1 }
```

- **Displaying a user bill:** A user can view their bill for the month.

```
db.users.find({emailid:'3Gx@aol.com'}, {billing_date:1,status:'active',payment_method:1,
emailid:1,_id:0});
```

```
Atlas atlas-j2215d-shard-0 [primary] sjsu_movie_db> db.users.find({emailid:'3Gx@aol.com'}, {billing_date:1,status:'active',
payment_method:1,emailid:1,_id:0})
[
  {
    emailid: '3Gx@aol.com',
    billing_date: ISODate("2020-01-31T00:00:00.000Z"),
    payment_method: 'DEBIT/CREDIT',
    status: 'active'
  }
]
```

VII. QUERIES

1. To find directors who have directed more than 1 movie

- db.movies.aggregate([{ "\$group": { "_id": "\$Director", "count": { "\$sum": 1 } } }, { "\$match": { "count": { "\$gte": 2 } } }])

```
Atlas atlas-xokgc4-shard-0 [primary] sjsu_movie_db> db.movies.aggregate([ { "$group": { "_id": "$Director",
"count": { "$sum": 1 } } }, { "$match": { "count": { "$gte": 2 } } } ])
[
  { _id: 'Ang Lee', count: 2 },
  { _id: 'Mike Flanagan', count: 4 },
  { _id: 'Paul McGuigan', count: 2 },
  { _id: 'Michael Bay', count: 6 },
  { _id: 'F. Gary Gray', count: 2 },
  { _id: 'George Nolfi', count: 2 },
  { _id: 'Christopher McQuarrie', count: 2 },
  { _id: 'Denis Villeneuve', count: 5 },
  { _id: 'Gavin Hood', count: 3 },
  { _id: 'Will Gluck', count: 3 },
  { _id: 'Wes Anderson', count: 3 },
  { _id: 'Barry Sonnenfeld', count: 2 },
  { _id: 'Kenneth Branagh', count: 3 },
  { _id: 'Glenn Ficarra', count: 3 },
  { _id: 'Frank Coraci', count: 2 },
  { _id: 'David Lynch', count: 2 },
  { _id: 'Steven Spielberg', count: 4 },
  { _id: 'Edgar Wright', count: 3 },
  { _id: 'Michael Patrick King', count: 2 },
  { _id: 'George Miller', count: 2 }
]
```

2. To find which movies that were released between the years 2018 to 2020

- db.movies.find({ "Year": { "\$gte": "2016", "\$lte": "2018" } }, { Title:1, Description:1, Year:1, _id:0 }).pretty()

3. Showing Deactivated Users

- db.users.find({"status":"Deactivated"},{ emailid : 1,name:1, status:1})

```
[Atlas atlas-xokgc4-shard-0 [primary] sjsu_movie_db> db.users.find({ "status": "Deactivated"}, { "emailid": 1, "name": 1, "status": 1})
[{"_id": ObjectId("62711618901ad4b149608132"), "emailid": "7aW0msn.com", "name": "joseph miller", "status": "Deactivated"}, {"_id": ObjectId("62711618901ad4b149608159"), "emailid": "CmSOuF@yahoo.com", "name": "jeniffer thompson", "status": "Deactivated"}, {"_id": ObjectId("62711618901ad4b149608163"), "emailid": "dH1Mt@gmail.com", "name": "elizabeth moore", "status": "Deactivated"}]
```

4. Actors who have performed in more than 1 movie.

- db.movies.aggregate([{\$unwind": "\$Actors"}, {"\$group": {"_id": "\$Actors", "count": {"\$sum": 1}}}, {"\$match": {"count": {"\$gt": 1}}}]

```
[{"_id": "William Hurt", "count": 2}, {"_id": "Joseph Gordon-Levitt", "count": 9}, {"_id": "Marion Cotillard", "count": 5}, {"_id": "James Marsden", "count": 4}, {"_id": "Ben Foster", "count": 6}, {"_id": "Matthew Goode", "count": 2}, {"_id": "Liam Hemsworth", "count": 7}, {"_id": "Benedict Cumberbatch", "count": 4}, {"_id": "Ethan Hawke", "count": 7}, {"_id": "Jai Courtney", "count": 4}, {"_id": "Octavia Spencer", "count": 2}, {"_id": "Theo James", "count": 6}, {"_id": "Kate Hudson", "count": 3}, {"_id": "Jon Favreau", "count": 3}, {"_id": "Clive Owen", "count": 3}, {"_id": "Dave Bautista", "count": 2}, {"_id": "Viggo Mortensen", "count": 3}, {"_id": "Blake Lively", "count": 5}, {"_id": "Channing Tatum", "count": 12}, {"_id": "Tony Shalhoub", "count": 2}]
```

5. Getting the user details who raised the service request

- db.service_requests.aggregate([{"\$lookup": {"from": "users", "localField": "emailid", "foreignField": "emailid", "as": "users"}}, {"\$project": {"users.password": 0, "users.dob": 0, "users.billing_date": 0, "users.status": 0, "users.profiles": 0, "users.payment_method": 0}}])

```
{
  _id: ObjectId("62703429bb962b8695dd0d22"),
  complaint_severity: 'm',
  service_agent_name: 'Sunay',
  service_id: 400017,
  date_of_complaint: ISODate("2022-01-19T00:00:00.000Z"),
  complaint_description: 'Episodes or seasons out of order',
  emailid: 'EGmvTx@yahoo.com',
  users: [
    {
      _id: ObjectId("6271ee61bb962b8695dd1215"),
      emailid: 'EGmvTx@yahoo.com',
      phoneno: '512-974-4119',
      name: 'michael johnson',
      account_create_date: ISODate("2021-01-01T00:00:00.000Z"),
      no_of_profiles: 3,
      preferred_language: 'Japanese'
    }
  ],
  {
    _id: ObjectId("62703429bb962b8695dd0d23"),
    complaint_severity: 'm',
  }
}
```

6. Top 5 rated movies

- db.movie.aggregate([{\$sort: {Rating:-1}}, {\$limit:5}, {\$project: {"imdb_rank":1, "Title":1,"Rating":1,"Votes":1}}])

```
Atlas atlas-j2215d-shard-0 [primary] sjsu_movie_db> db.movies.aggregate([{$sort:{Rating:-1}},{$limit:5},{$project:{"imdb_rank":1, "Title":1,"Rating":1,"Votes":1}}])
[
  {
    _id: ObjectId("6275909b42b44ac7f5867564"),
    Title: 'The Dark Knight',
    Rating: 9,
    Votes: 1791916
  },
  {
    _id: ObjectId("6275909b42b44ac7f58675a3"),
    Title: 'Dangal',
    Rating: 8.8,
    Votes: 48969
  },
  {
    _id: ObjectId("6275909b42b44ac7f586757e"),
    Title: 'Inception',
    Rating: 8.8,
    Votes: 1583625
  },
  {
    _id: ObjectId("6275909b42b44ac7f586758e"),
    Title: 'Kimi no na wa',
    Rating: 8.6,
    Votes: 34110
  },
  {
    _id: ObjectId("6275909b42b44ac7f5867552"),
    Title: 'Interstellar',
    Rating: 8.6,
    Votes: 1047747
  }
]
```

7. Types of service complaints raised by the user with their counts:

- db.service_requests.aggregate([{"\$group":{ "_id" : "\$complaint_description","count" : {"\$sum" : 1}}}))

```
Atlas atlas-xokgc4-shard-0 [primary] sjsu_movie_db> db.service_requests.aggregate([{"$group": {"_id": "$complaint_description", "count": {"$sum": 1}}})
[{"_id": "Not able to access the subtitles", "count": 1}, {"_id": "Volume levels", "count": 21}, {"_id": "Buffering", "count": 3}, {"_id": "Problems with the maturity rating or classification", "count": 21}, {"_id": "Artwork or description errors", "count": 1}, {"_id": "Problems with the subtitles or captions, including quality or availability", "count": 6}, {"_id": "Problems with the video quality or aspect ratio", "count": 7}, {"_id": "Problems with the audio or dubbing, including quality or availability", "count": 7}, {"_id": "Episodes or seasons out of order", "count": 42}]
```

8. Generate the next bill due date:

- db.users.aggregate([{\$project: {"next_bill_due": {\$dateAdd: {startDate:"\$billing_date", unit:"month"}, amount: 1}}}, name:1,billing_date:1,amount:{\$literal:"\$19.95"}}, {\$sort: {billing_date:-1}}, {\$limit:5}])

```
Atlas: atlas-32215d-shard-0 [primary] ssj5_movie_db db.user.aggregate([{$project:{`next_bill_due`:{$dateAdd:{`startDate`:$Billing_date, `unit`:`month`, `amount`:1}},`name`:_id,`Billing_date`:_id,`amount`:$literal:19.96}},{$sort:{`Billing_date`:-1,`$limit`5}},{`_id`:{`$objectID`:"627964f724e2eb13f7192f38a"},`name`:"brando saga",`Billing_date`:{`$ISODate`:"2022-04-28T00:00:00.000Z"},`next_bill_due`:{`$ISODate`("2022-05-28T00:00:00.000Z")},`amount`：“$19.96”},{`_id`:{`$objectID`:"627964f724e2eb13f7192f396"},`name`:"rober N",`Billing_date`:{`$ISODate`("2022-04-28T00:00:00.000Z")},`next_bill_due`:{`$ISODate`("2022-05-28T00:00:00.000Z")},`amount`：“$19.96”},{`_id`:{`$objectID`:"627964f724e2eb13f7192f405"},`name`:"shant video",`Billing_date`:{`$ISODate`("2022-04-28T00:00:00.000Z")},`next_bill_due`:{`$ISODate`("2022-05-28T00:00:00.000Z")},`amount`：“$19.96”},{`_id`:{`$objectID`:"627964f724e2eb13f7192f3c6"},`name`:"ram",`Billing_date`:{`$ISODate`("2022-04-28T00:00:00.000Z")},`next_bill_due`:{`$ISODate`("2022-05-28T00:00:00.000Z")},`amount`：“$19.96”},{`_id`:{`$objectID`:"627964f724e2eb13f7192f37b"},`name`:"christian",`Billing_date`:{`$ISODate`("2021-11-30T00:00:00.000Z")},`next_bill_due`:{`$ISODate`("2021-12-30T00:00:00.000Z")},`amount`：“$19.96”}],{`$sort`:{`Billing_date`:-1}}])
```

9. Daily service request counts :

- db.service_requests.aggregate([{"\$group": {"_id": "\$date_of_complaint", "count": {"\$sum": 1}}}])

```
atlas atlas-j2215d-shard-0 [primary] sjsu_movie_db> db.service_requests.aggregate([{"$group": {"_id": "$date_of_complaint", "count": {"$sum": 1}}})
[{"_id": ISODate("2022-01-19T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-02-03T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-01-25T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-01-18T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-01-17T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-01-05T00:00:00.000Z"), count: 2}, {"_id": ISODate("2022-02-16T00:00:00.000Z"), count: 3}, {"_id": ISODate("2022-02-04T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-03-03T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-03-04T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-01-24T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-01-21T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-02-22T00:00:00.000Z"), count: 5}, {"_id": ISODate("2022-03-01T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-02-07T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-03-02T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-02-02T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-02-19T00:00:00.000Z"), count: 5}, {"_id": ISODate("2022-01-15T00:00:00.000Z"), count: 1}, {"_id": ISODate("2022-01-03T00:00:00.000Z"), count: 2}]
```

10. Movie language preferred by the user with count

- db.users.aggregate([{"\$group": {"_id": "\$preferred_language", "count": {"\$sum": 1}}])

```
[{"_id": "Japanese", "count": 40}, {"_id": "english", "count": 1}, {"_id": "English", "count": 148}, {"_id": "Korean", "count": 74}, {"_id": "Hindi", "count": 100}, {"_id": "Chinese", "count": 40}, {"_id": "Tamil", "count": 1}]
```

11. View profiles my list

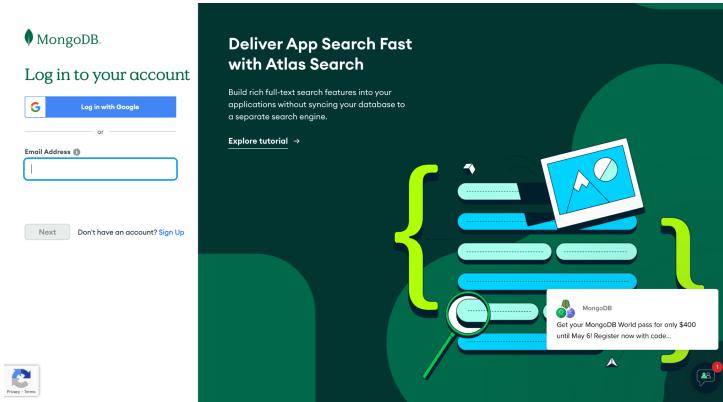
- db.users.aggregate([{\$project: {"profiles": 1, "_id": 0}}, {\$unwind: "\$profiles"}, {\$unwind: "\$profiles.profile_history"}, {\$lookup: {"from": "movies", "localField": "profiles.profile_history.movie_id", "foreignField": "_id", "as": "movie_title"}}, {\$project: {"profiles.profile_name": 1, "movie_title.Title": 1, "_id": 0}}])

```
[{"profiles": [{"profile_name": "doggo", "movie_title": [{"Title": "Rogue One"}]}, {"profiles": [{"profile_name": "cat", "movie_title": [{"Title": "Moana"}]}, {"profiles": [{"profile_name": "niva", "movie_title": [{"Title": "What If"}]}, {"profiles": [{"profile_name": "sush", "movie_title": [{"Title": "Sleeping with Other People"}]}, {"profiles": [{"profile_name": "shat", "movie_title": [{"Title": "Split"}]}, {"profiles": [{"profile_name": "joy", "movie_title": [{"Title": "Hidden Figures"}]}]}]}]}]}]
```

VIII. CONNECTIVITY TO MONGODB CLOUD

MongoDB Atlas allows our data to be deployed in the cloud and connect with our local terminal or Compass to insert, read and update the data using queries.

- Login to <https://cloud.mongodb.com> to deploy the data.



- Create a cluster from the homepage.

This screenshot shows the MongoDB Atlas Database Deployments page for a 'Cluster0' in the 'SADAKHYA'S ORO' project. The page displays various metrics such as connections (25.8), throughput (In 105.1 B/s, Out 1.4 KB/s), and data size (72.9 KB). It also shows deployment details like version (5.0.8), region (AWS / N. Virginia (us-east-1)), and cluster tier (M0 Sandbox (General)). A 'Create Index' button is visible at the bottom right.

- Provide the required information, cloud as aws and create a new cluster.

This screenshot shows the 'Create a Shared Cluster' wizard in MongoDB Atlas. The 'Shared' tab is selected. It includes a note about learning in a sandbox environment and upgrading to a dedicated cluster. The 'Cloud Provider & Region' section shows AWS (selected) and Azure as options. Below, a map highlights the 'N. Virginia (us-east-1)' region as the recommended one. Other regions listed include Oregon, Paris, Sydney, Frankfurt, Stockholm, Ireland, London, Milan, Tokyo, Seoul, Bahrain, and Jakarta.

Cluster Tier

M0 Sandbox (Shared RAM, 512 MB Storage) Encrypted

Base hourly rate is for a MongoDB replica set with 3 data bearing servers.

Shared Clusters for development environments and low-traffic applications

Tier	RAM	Storage	vCPU	Base Price
M0 Sandbox	Shared	512 MB	Shared	Free forever
M0 clusters are best for getting started, and are not suitable for production environments.				
500 max connections Low network performance		100 max databases 500 max collections		
M2	Shared	2 GB	Shared	\$7 / MONTH
M5	Shared	5 GB	Shared	\$25 / MONTH

Additional Settings

MongoDB 5.0, No Backup

Turn on Backup (M2 and up)

See Backup Solutions for Paid Clusters (M2+)

Cluster Name

Lab2

One time only: once your cluster is created, you won't be able to change its name.

Cluster names can only contain ASCII letters, numbers, and hyphens.

- Connect to the cluster by clicking on the connect option.

Connect to Cluster0

Setup connection security Choose a connection method Connect

Choose a connection method View documentation

Get your pre-formatted connection string by selecting your tool below.

- Connect with the MongoDB Shell
- Connect your application
- Connect using MongoDB Compass

Go Back Close

- Select “connect with MongoDB shell” to connect with the command line on the system.
Select the OS and install mongosh on the system. Then Run Connection string in the command line.

Connection String:

```
mongosh "mongodb+srv://cluster0.vyvoo.mongodb.net/myFirstDatabase"
--apiVersion 1 --username root
```

Connect to Cluster0

✓ Setup connection security > ✓ Choose a connection method > Connect

I do not have the MongoDB Shell installed

I have the MongoDB Shell installed

- 1 Select your operating system and download the mongosh

macOS ▾

Install via Homebrew

brew install mongosh



Homebrew is a package manager for macOS. [Install Homebrew](#)

- 2 Run your connection string in your command line

Use this connection string in your application:

```
mongosh "mongodb+srv://cluster0.vyvoo.mongodb.net/myFirstDatabase" --apiVersion 1  
--username root
```



Replace **myFirstDatabase** with the name of the database that connections will use by default. You will be prompted for the password for the Database User, **root**. When entering your password, make sure all special characters are [URL encoded](#).

Having trouble connecting? [View our troubleshooting documentation](#)

Go Back

Close

- Select “Connect using Mongodbc compass” to connect with compass. Copy the connection string and connect from the compass.

Connect to Cluster0

✓ Setup connection security > ✓ Choose a connection method > Connect

I do not have MongoDB Compass

I have MongoDB Compass

- 1 Choose your version of Compass:

1.12 or later ▾

See your Compass version in “About Compass”

- 2 Copy the connection string, then open MongoDB Compass.

mongodb+srv://root:<password>@cluster0.vyvoo.mongodb.net/test



You will be prompted for the password for the **root** user’s (Database User) username. When entering your password, make sure that any special characters are [URL encoded](#).

Having trouble connecting? [View our troubleshooting documentation](#)

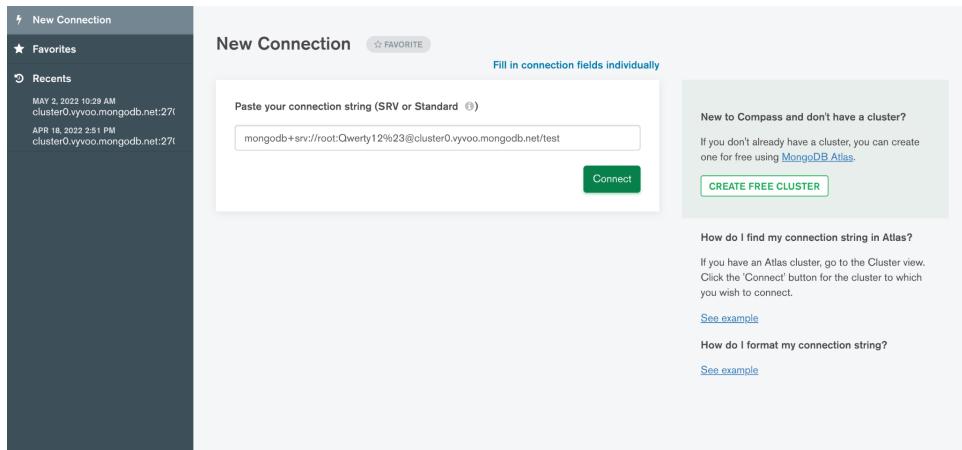
Go Back

Close

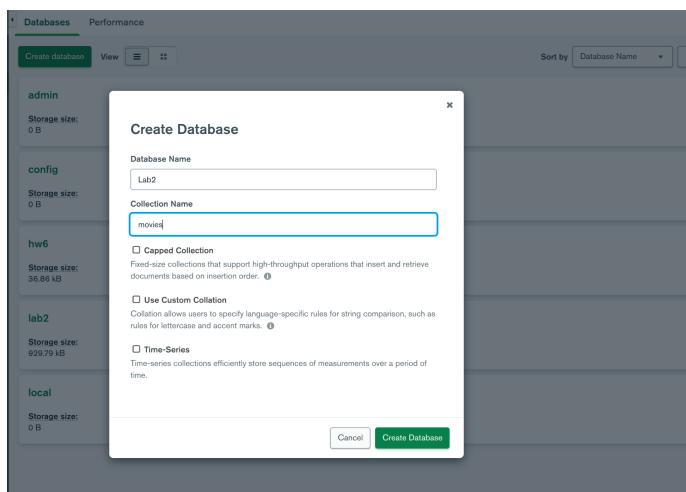
In MongoDB compass click on “New Connection” and paste your connection string with URL encoded password.

Connection String:

`mongodb+srv://root:Qwerty12%23@cluster0.vyvoo.mongodb.net/test`

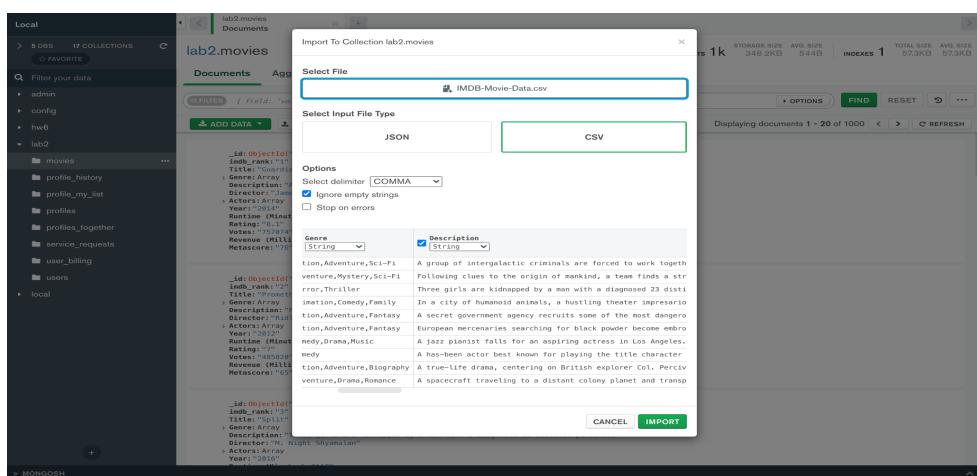


- In compass click on create Database to add collections to the new DB.



- Create 3 collections named Users, movies and service_requests. Once the Collections are created, add the documents.

Import the IMDb movie dataset into the collection.



We are denormalizing the multi value fields of actors and Genre into arrays as follows:

- db.movies.aggregate([{"\$addFields": {"Genre": { "\$split": ["\$Genre", ","]}}}, {"\$out: "movie"}])
- db.movies.aggregate([{"\$addFields": {"Actors": { "\$split": ["\$Actors", ","]}}}, {"\$out: "movie"}])

- Create Users collection and add documents using Insert query.

From shell:

```
db.users.insertOne({
    "emailid": "new_user@gmail.com",
    "phoneno": "322-370-1228",
    "name": "Ratan Harry",
    "dob": "2001-11-12",
    "password": "qwert12#",
    "account_create_date": "2010-10-07",
    "no_of_profiles": "0",
    "preferred_language": "English",
    "profiles": [],
    "billing_date": "2022-04-03",
    "status": "Active",
    "payment_method": "DEBIT/CREDIT"
});
```

From Compass:

Select Add Data-> Insert Document

```
lab2.users
DOCUMENTS 404
STORAGE SIZE 106.5KB AVG. SIZE 758B INDEXES 1 TOTAL SIZE 28.7KB AVG. SIZE 28.7KB

ADD DATA ...
```

Insert to Collection lab2.users

```

1 {
2   "_id": {
3     "$oid": "6233ae50e2053c322e3b313"
4   },
5   "emailid": "new_user@gmail.com",
6   "phoneno": "322-370-1228",
7   "name": "Ratan Harry",
8   "dob": "2001-11-12",
9   "password": "qwert12#",
10  "account_create_date": "2010-10-07",
11  "no_of_profiles": "0",
12  "preferred_language": "English",
13  "profiles": [],
14  "billing_date": "2022-04-03",
15  "status": "Active",
16  "payment_method": "DEBIT/CREDIT"
17 }
18
19 {
20   "_id": {
21     "$oid": "6233ae50e2053c322e3b313"
22   },
23   "emailid": "new_user@gmail.com",
24   "phoneno": "322-370-1228",
25   "name": "Ratan Harry",
26   "dob": "2001-11-12",
27   "password": "qwert12#",
28   "account_create_date": "2010-10-07",
29   "no_of_profiles": "0",
30   "preferred_language": "English",
31   "profiles": [],
32   "billing_date": "2022-04-03",
33   "status": "Active",
34   "payment_method": "DEBIT/CREDIT"
35 }
36
37 {
38   "_id": {
39     "$oid": "6233ae50e2053c322e3b313"
40   },
41   "emailid": "new_user@gmail.com",
42   "phoneno": "322-370-1228",
43   "name": "Ratan Harry",
44   "dob": "2001-11-12",
45   "password": "qwert12#",
46   "account_create_date": "2010-10-07",
47   "no_of_profiles": "0",
48   "preferred_language": "English",
49   "profiles": [],
50   "billing_date": "2022-04-03",
51   "status": "Active",
52   "payment_method": "DEBIT/CREDIT"
53 }
54
55 {
56   "_id": {
57     "$oid": "6233ae50e2053c322e3b313"
58   },
59   "emailid": "new_user@gmail.com",
60   "phoneno": "322-370-1228",
61   "name": "Ratan Harry",
62   "dob": "2001-11-12",
63   "password": "qwert12#",
64   "account_create_date": "2010-10-07",
65   "no_of_profiles": "0",
66   "preferred_language": "English",
67   "profiles": [],
68   "billing_date": "2022-04-03",
69   "status": "Active",
70   "payment_method": "DEBIT/CREDIT"
71 }
72
73 {
74   "_id": {
75     "$oid": "6233ae50e2053c322e3b313"
76   },
77   "emailid": "new_user@gmail.com",
78   "phoneno": "322-370-1228",
79   "name": "Ratan Harry",
80   "dob": "2001-11-12",
81   "password": "qwert12#",
82   "account_create_date": "2010-10-07",
83   "no_of_profiles": "0",
84   "preferred_language": "English",
85   "profiles": [],
86   "billing_date": "2022-04-03",
87   "status": "Active",
88   "payment_method": "DEBIT/CREDIT"
89 }
90
91 {
92   "_id": {
93     "$oid": "6233ae50e2053c322e3b313"
94   },
95   "emailid": "new_user@gmail.com",
96   "phoneno": "322-370-1228",
97   "name": "Ratan Harry",
98   "dob": "2001-11-12",
99   "password": "qwert12#",
100  "account_create_date": "2010-10-07",
101  "no_of_profiles": "0",
102  "preferred_language": "English",
103  "profiles": [],
104  "billing_date": "2022-04-03",
105  "status": "Active",
106  "payment_method": "DEBIT/CREDIT"
107 }
```

OPTIONS FIND RESET ...

Displaying documents 1 - 20 of 405 < > C REFRESH

- Create Service Request collection and add documents using Insert query.

From shell:

```
db.service_requests.insertOne({
    "complaint_severity": "m",
    "service_agent_name": "Sandeshn",
    "service_id": "400000",
    "date_of_complaint": "2022-01-02",
    "complaint_description": "Buffering",
    "emailid": "OoececBAnkd@yahoo.com"
})
```

From Compass:

Select Add Data-> Insert Document

The screenshot shows the MongoDB Compass interface. The top navigation bar has tabs for 'Documents', 'Aggregations', 'Schema', 'Explain Plan', 'Indexes', and 'Validation'. Below the navigation is a search bar with a filter dropdown set to '_id: "v000003"'. The main area displays a table of 109 documents. A modal window titled 'Insert to Collection lab2.service_requests' is open in the center. The modal contains a code editor with the following JSON document:

```

{
    "_id": "62733be00e2a53c322e3b316",
    "complaint_severity": "m",
    "service_agent_name": "Romegha",
    "service_id": "400000",
    "date_of_complaint": "2022-01-02",
    "complaint_description": "Cannot get subtitles",
    "emailid": "OoececBAnkd@yahoo.com"
}

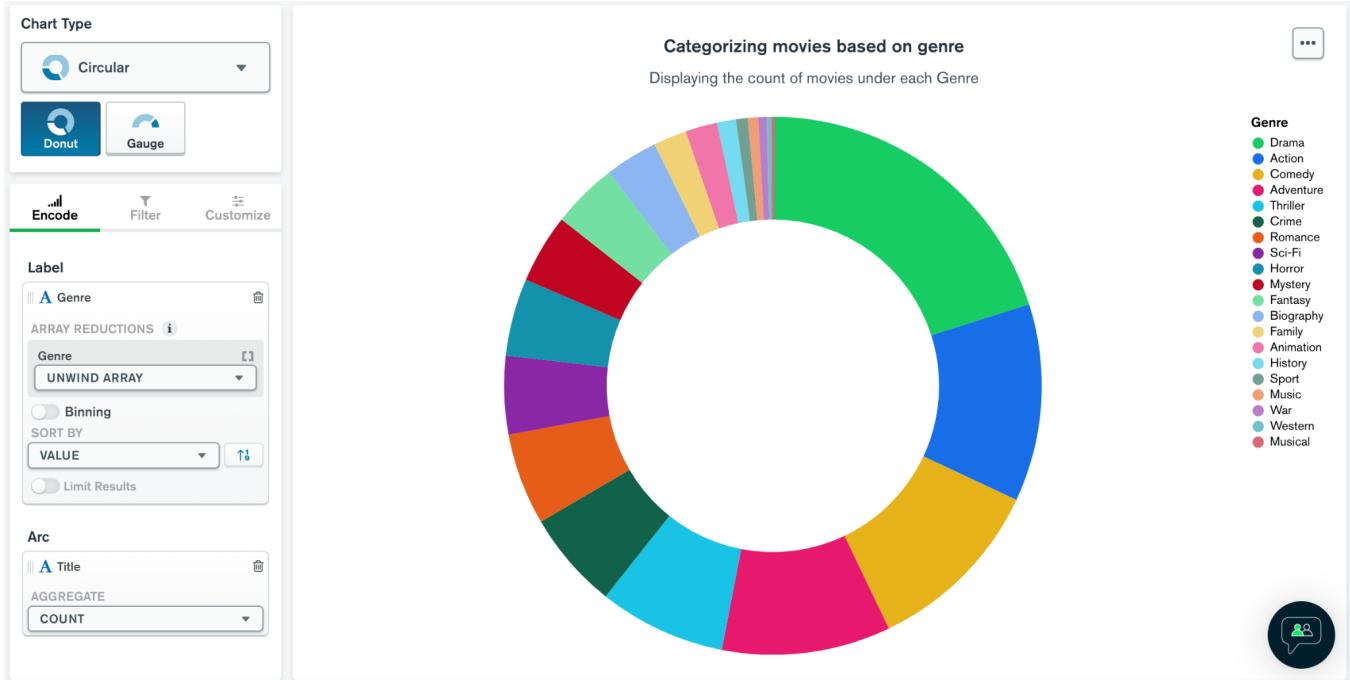
```

At the bottom of the modal are 'Cancel' and 'Insert' buttons.

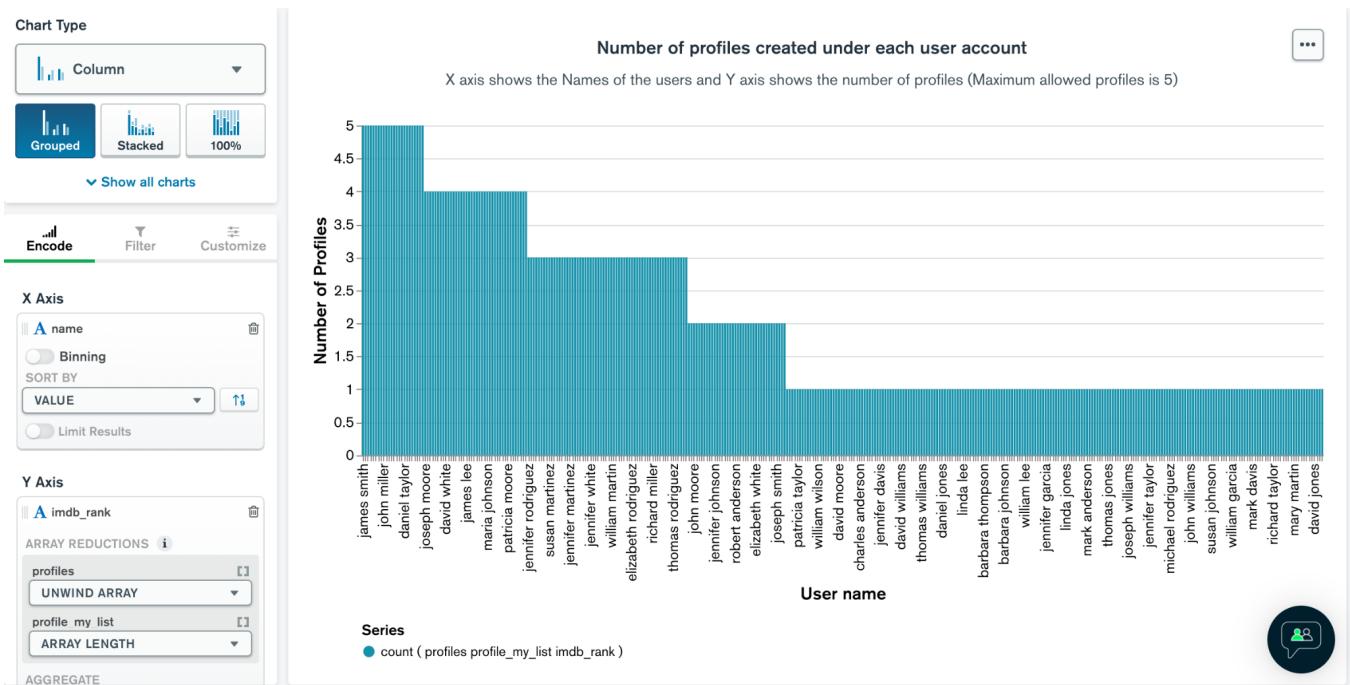
IX. VISUALIZATION

Mongodb Atlas provides built-in support for Charts. Drag and Drop of the fields for aggregations can be done to visualize the data.

1. **Categorizing movies based on genre:** Displaying the count of movies under each Genre.

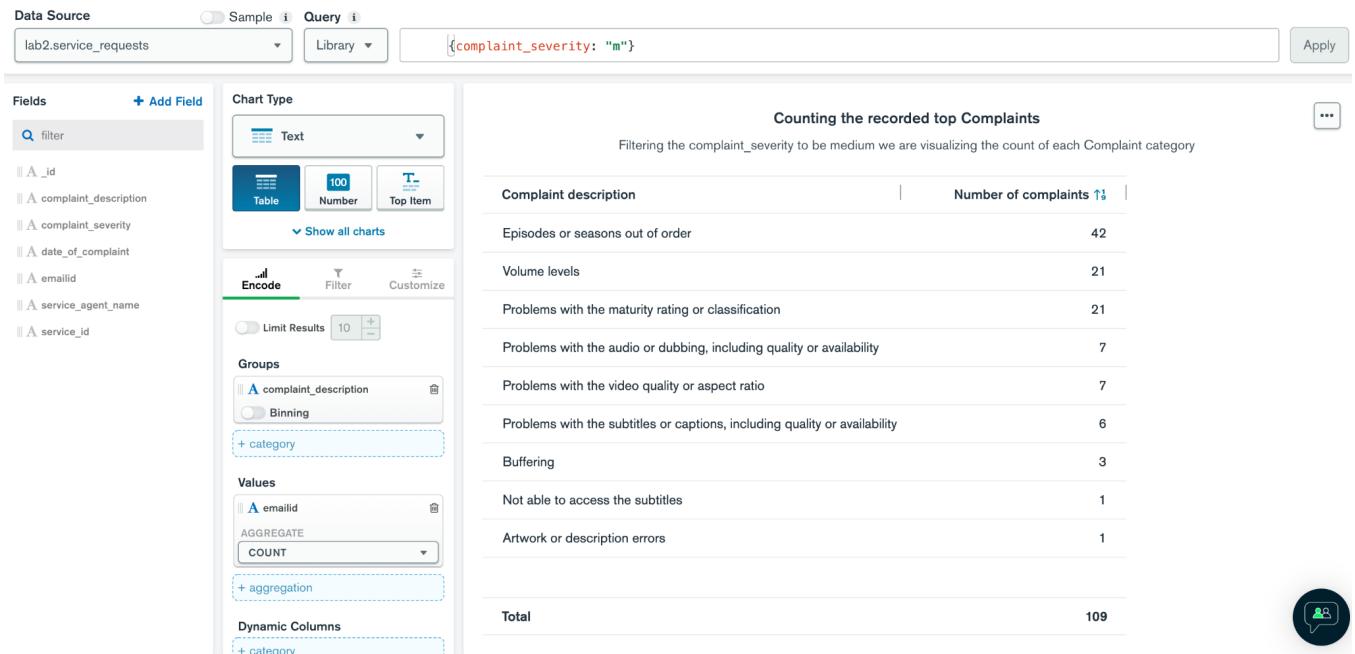


2. Displaying the count of movies under each Genre: X axis shows the Names of the users and Y axis shows the number of profiles (Maximum allowed profiles is 5)



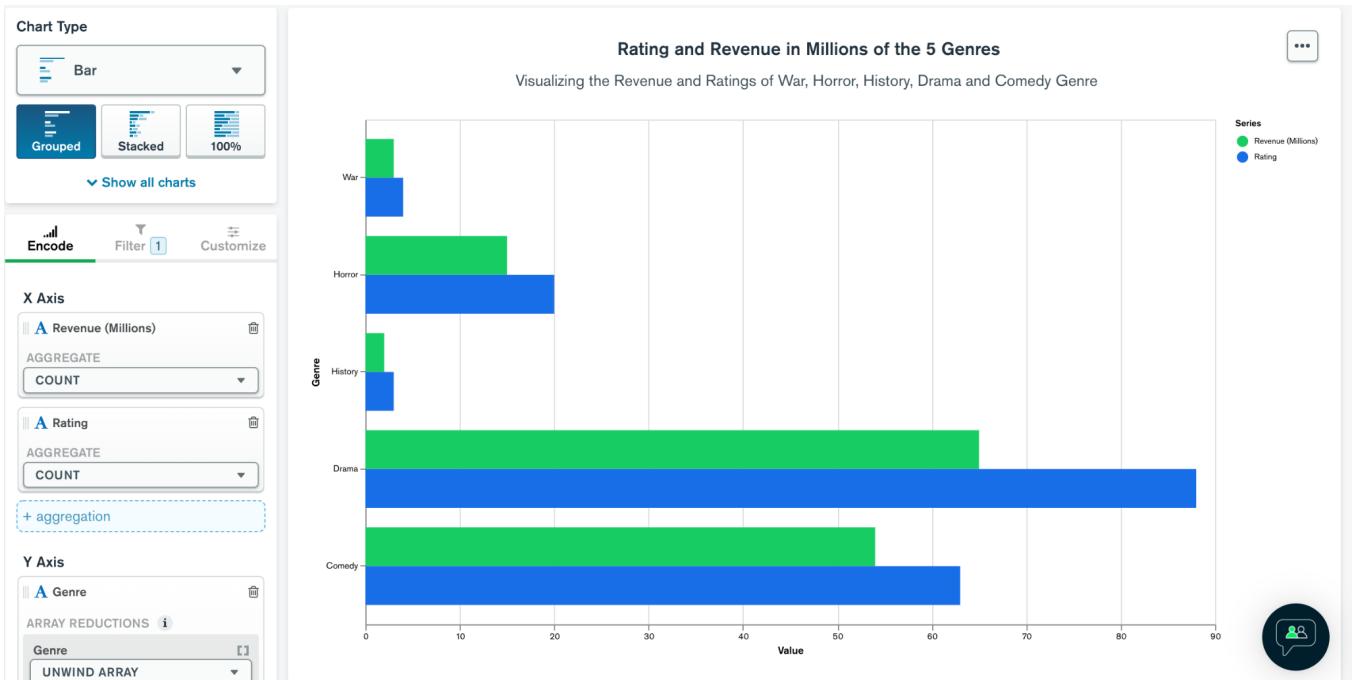
3. Counting the recorded top Complaints: Filtering the complaint_severity to be medium we are visualizing the count of each Complaint category.

Initially filter : {complaint_severity: "m"} is applied.

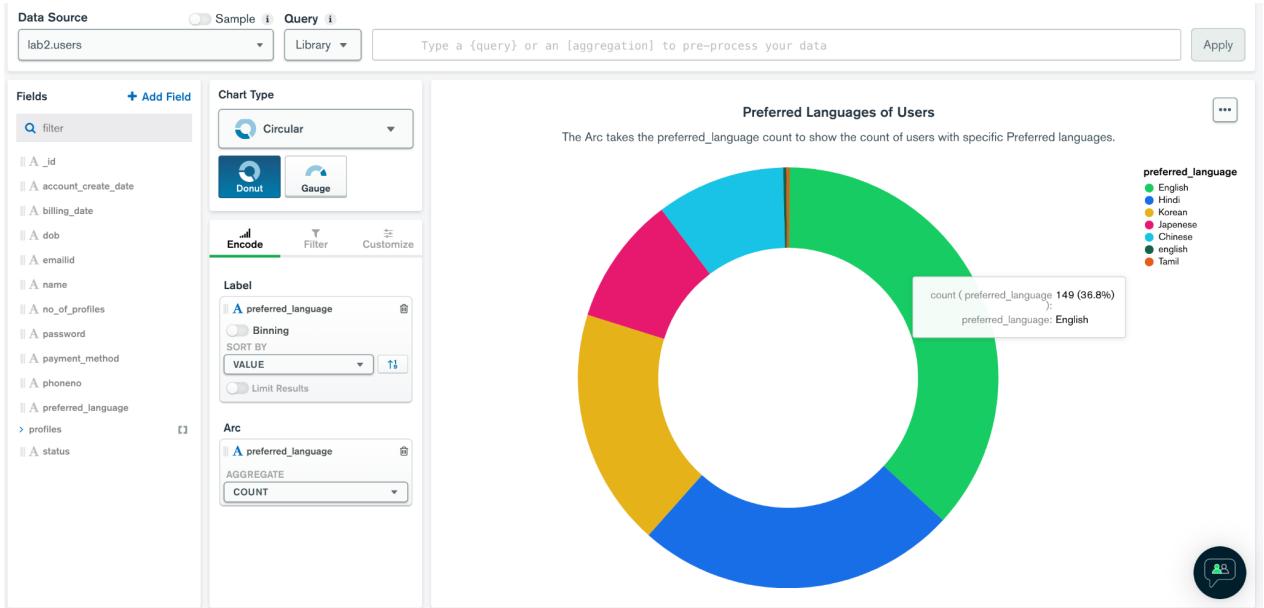


4. Rating and Revenue in Millions of the 5 Genres: Visualizing the Revenue and Ratings of War, Horror, History, Drama and Comedy Genre.

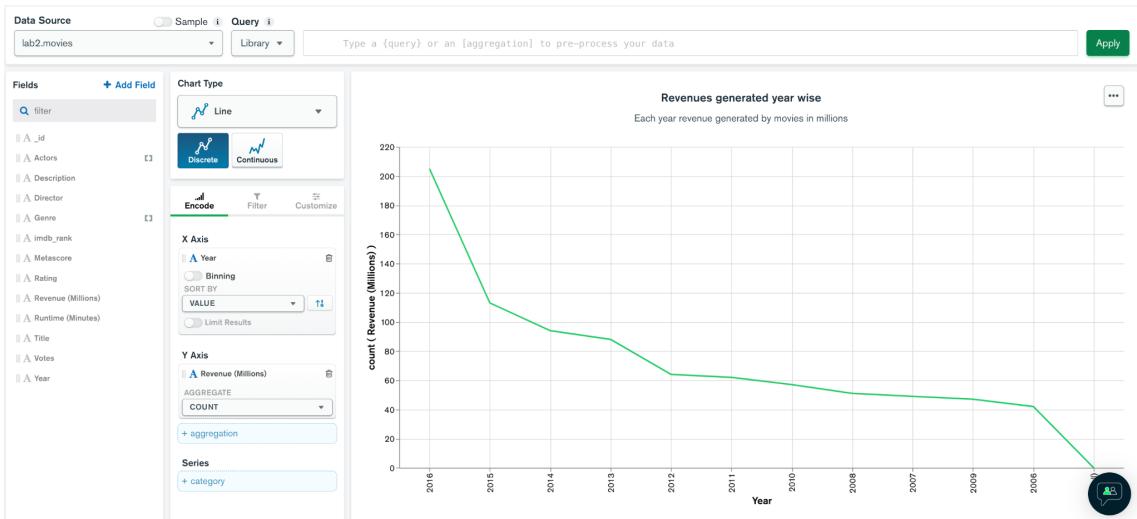
Adding count aggregate on Revenue(Millions), count aggregate on Rating in Xaxis and applying unwind aggregation on Genre array in Y axis and filter with 5 Genres.



5. Language preferred by the user for a movie: The below donut chart shows the language preference of the user for watching a movie and it shows English is the most preferred language which takes around 37% from other languages.



6. Revenue generated by the movies in that year : The line chart shows the revenue as compared to the previous years. The X Axis takes the Years and Y Axis takes Revenue.



X. MONGODB QUERIES PERFORMANCE MEASUREMENT

MongoDB document storage provides efficient querying of data. It provides an O(1) lookup for documents.

Query Performance without indexing:

Executing a find query that filters based on an email and projects a few fields in Mongodbd will usually scan 405 documents to return 1 matching document with 0ms execution time.

- db.users.find({emailid:'3Gx@aol.com'}, {billing_date:1,status:'active',payment_method:1, emailid:1,_id:0});

The screenshot shows the MongoDB Compass interface with the 'Explain Plan' tab selected. The query is:

```

{
  "$filter": {
    "expression": {
      "$eq": [
        "$emailid",
        "3Gx@aol.com"
      ]
    }
  },
  "$project": {
    "_id": 1,
    "status": "active",
    "payment_method": 1,
    "emailid": 1
  },
  "$sort": {
    "emailid": -1
  }
}
  
```

Indexes used:

- COLLATION: { locale: "simple" }

Performance Summary:

- Documents Returned: 1
- Index Keys Examined: 0
- Documents Examined: 405
- Actual Query Execution Time (ms): 0
- Sorted in Memory: no
- No index available for this query.

Projection Details:

- PROJECTION_DEFAULT: nReturned: 1, Execution Time: 0 ms
- COLLSCAN: nReturned: 1, Execution Time: 0 ms, Documents Examined: 405

Query Performance with index:

Mongodb default does indexing by the `_id` which is the primary key for the document. We added an index for `emailid` named `user`.

Create Index Dialog:

Choose an index name: `user`

Configure the index definition: `emailid` (asc)

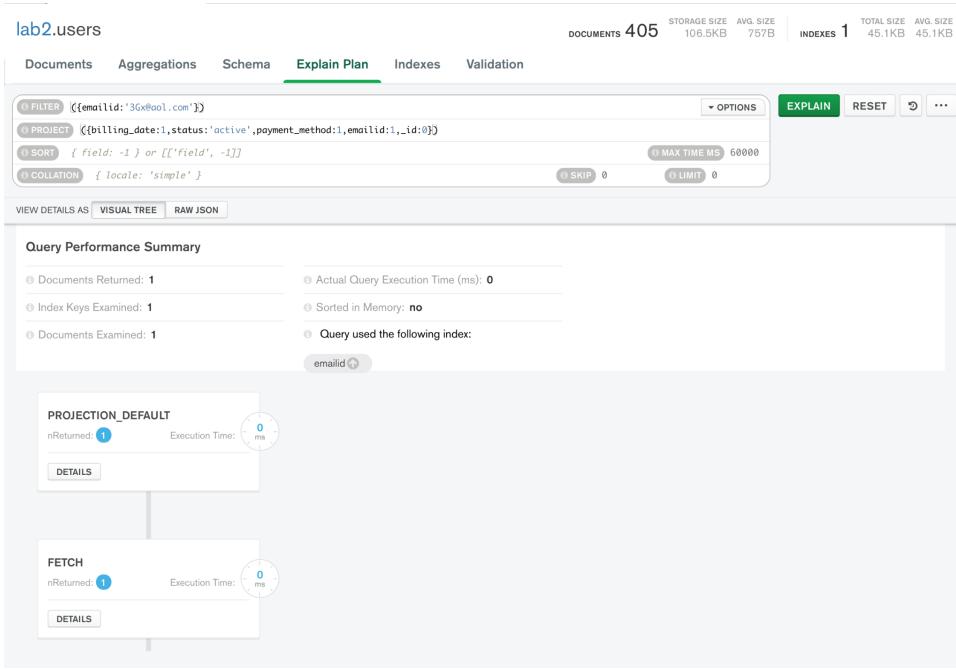
Options:

- Build index in the background
- Create unique index
- Create TTL
- Partial Filter Expression
- Use Custom Collation
- Wildcard Projection

Indexes List:

Name and Definition	Type	Size	Usage	Properties
<code>_id</code>	REGULAR	45.1 KB	43 since Mon May 02 2022	UNIQUE
<code>user</code>	REGULAR	287 KB	2 since Thu May 05 2022	UNIQUE

Now that an index is added the performance is greatly improved. In place of scanning all the documents it scans 1 index entry and 1 document to return 1 matching document which is very efficient querying.



Comparing MongoDB Vs MySQL query Performance:

MongoDB gives efficient performance when most of the queries are read operations. The update query operations will be comparatively slow on embedded documents. However this can be seen as a tradeoff for the complex join operations that are avoided using the embedding of documents.

For performing the same operation of viewing the billing details of a user in mysql the following query needs to be run:

```
> SELECT
    u.emailid, b.billing_date, b.payment_method, b.status
FROM
    users u
JOIN
    user_billing b ON u.emailid = b.emailid
WHERE
    u.emailid = '3Gx@aol.com';
```

This query needs details from 2 tables hence it applies JOIN on both the tables and get the details on first scanning all the rows in Users table for the matching user. This overall takes a lot of execution time.

```
1
2 • SELECT
3     u.emailid, b.billing_date, b.payment_method, b.status
4 FROM
5     users u
6     JOIN
7         user_billing b ON u.emailid = b.emailid
8 WHERE
9     u.emailid = '3Gx@aol.com';

100% 1:1

Result Grid Filter Rows: Search Export ▾ Result Grid Form Editor Field Types

emailid billing_date payment_method status
▶ 3Gx@aol.com 2020-01-31 DEBIT?CREDIT ACTIVE

Result 5 Read Only

Action Output Action Response Duration / Fetch Time
Time Action
1 16:45:06 select u.emailid, b.billing_date, b.payment_method, b.status from users u join user_billing b on u.emailid=b.emailid 1 row(s) returned 0.00045 sec / 0.000...
```

Hence we can see a clear improvement in querying MongoDB documents with embedded structure where the billing table is embedded into the users table.