DATABASE FOR SOCIAL NET.

ADVANCED DATABASE MANAGEMENT SYSTEM
COURSE WORK

FERNANDO L H M
KAHDSE222F-020

CONTENT

design decisions

implementation details

query examples

Code snippets or commands demonstrating database schema creation and basic CRUD operations

example queries to demonstrate the retrieval of data from the database

An explanation of how indexing and scaling techniques can be applied to the chosen NoSQL database

design decisions

Choosing a Suitable NoSQL Database Model

For the requirements of SocialNet, a document store NoSQL database model would be the most suitable choice. Document stores are well-suited for handling semi-structured or unstructured data, which is common in social media applications. They allow for flexible schema design, making it easier to accommodate changes in data structure without impacting existing records. Additionally, document stores provide powerful querying capabilities, which will be essential for efficient retrieval of posts, media, and friendship data.

implementation details

Database schema

Collection: Users

- *Fields:*
  - `_id`: Unique identifier for the user (e.g., MongoDB ObjectID)
  - `username`: User's username (string)
  - `email`: User's email address (string)
  - `date_of_birth`: User's date of birth (date)
  - `profile_picture_url`: URL to the user's profile picture (string)

Collection: Posts

- *Fields:*
  - `_id`: Unique identifier for the post (e.g., MongoDB ObjectID)
  - `content`: Content of the post (string)
  - `timestamp`: Timestamp when the post was created (date/time)
  - `user`: Embedded user information:

- `user_id`: User's unique identifier (corresponding `_id` from the Users collection)
- `username`: User's username (string)
- `profile_picture_url`: URL to the user's profile picture (string)
- `tags`: List of tags associated with the post (array of strings)

Collection: Media

- *Fields:*
  - `_id`: Unique identifier for the media file (e.g., MongoDB ObjectID)
  - `url_or_path`: URL or path to the media file (string)
  - `type`: Type of media (e.g., "photo" or "video") (string)
  - `user`: Embedded user information:
    - `user_id`: User's unique identifier (corresponding `_id` from the Users collection)
    - `username`: User's username (string)
    - `profile_picture_url`: URL to the user's profile picture (string)

Collection: Friendships

- *Fields:*
  - `_id`: Unique identifier for the friendship connection (e.g., MongoDB ObjectID)
  - `user_name_1`: User name of one friend (corresponding `username` from the Users collection)
  - `user_name_2`: User name of the other friend (corresponding `username` from the Users collection)
  - `timestamp`: Timestamp when the friendship was established (date/time)

Denormalization and Embedding:

To optimize performance and avoid additional queries, we have embedded certain related data within documents:

- In the Posts collection, we have embedded user information (username, profile_picture_url) within the `user` field. This way, when displaying a post, we can directly access the user details without making an additional query to the Users collection.

- In the Media collection, we have also embedded user information (username, profile_picture_url) within the `user` field. This allows us to retrieve the user's details along with the media file without the need for extra queries.

- Additionally, we have embedded tags directly within the Posts collection. This embedding enables faster tag-based queries as the tags are readily available within each post document.

Advantages of Denormalization and Embedding:

By denormalizing and embedding data, we achieve better performance for read operations, as we can retrieve all the necessary information with a single query. This approach reduces the need for complex joins or multiple queries when fetching related data. For a social media platform like SocialNet, where retrieving user details, posts, and media is common, denormalization and embedding help in minimizing response times and enhancing the overall user experience.

query examples

Create the Collections

```
db.createCollection("users");
```

```
db.createCollection("posts");
```

```
db.createCollection("media");
```

```
db.createCollection("friendships");
```

Insert Documents

```
db.users.insertOne({
  username: "example_user1",
  email: "user1@example.com",
  date_of_birth: ISODate("1990-01-01"),
  profile_picture_url: "https://example.com/profile1.jpg"
});

db.users.insertOne({
  username: "example_user2",
  email: "user2@example.com",
  date_of_birth: ISODate("1995-03-15"),
  profile_picture_url: "https://example.com/profile2.jpg"
});

db.posts.insertOne({
  content: "Exciting trip to the beach!",
  timestamp: ISODate(),
  user: {
    user_id: ObjectId("64c77fbb46942922ee557996"),
    username: "example_user1",
    profile_picture_url: "https://example.com/profile1.jpg"
  },
  tags: ["#travel", "#beach"]
```

```
});

db.media.insertOne({
  url_or_path: "https://example.com/photo1.jpg",
  type: "photo",
  user: {
    user_id: ObjectId("64c77fbb46942922ee557996"),
    username: "example_user1",
    profile_picture_url: "https://example.com/profile1.jpg"
  }
});

db.friendships.insertOne({
  user_1: "example_user1",
  user_2: "example_user2",
  timestamp: new Date()
});
```

Update and Delete Operations

Updating User Information:

Update the email address of a user

```
db.users.updateOne(
  { username: "example_user1" },
  { $set: { email: "new_email@example.com" } }
```

```
);
```

Update the profile picture URL of a user

```
db.users.updateOne(
  { username: "example_user1" },
  { $set: { profile_picture_url: "https://example.com/new_profile.jpg" } }
);
```

Deleting a User:

```
db.users.deleteOne({ username: "example_user1" });
```

Updating Post Content:

```
db.posts.updateOne(
  { _id: ObjectId("64c7805846942922ee557998") },
  { $set: { content: "Updated post content!" } }
);
```

Deleting a Post:

```
db.posts.deleteOne({ _id: ObjectId("64c7805846942922ee557998") });
```

Updating Media Information:

```
db.media.updateOne(
  { _id: ObjectId("64ca749a46942922ee557999") },
  { $set: { type: "video" } }
```

```
)
```

Deleting Media File:

```
db.media.deleteOne({ _id: ObjectId("64ca749a46942922ee557999") })
```

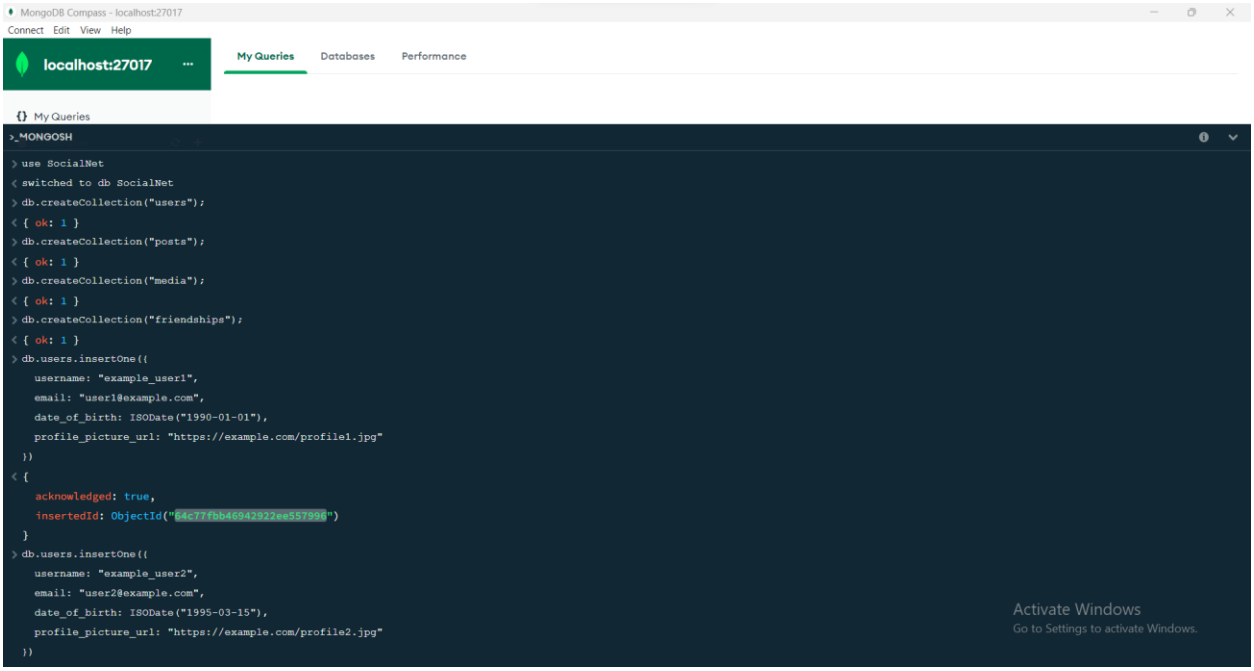
Updating Friendship Timestamp:

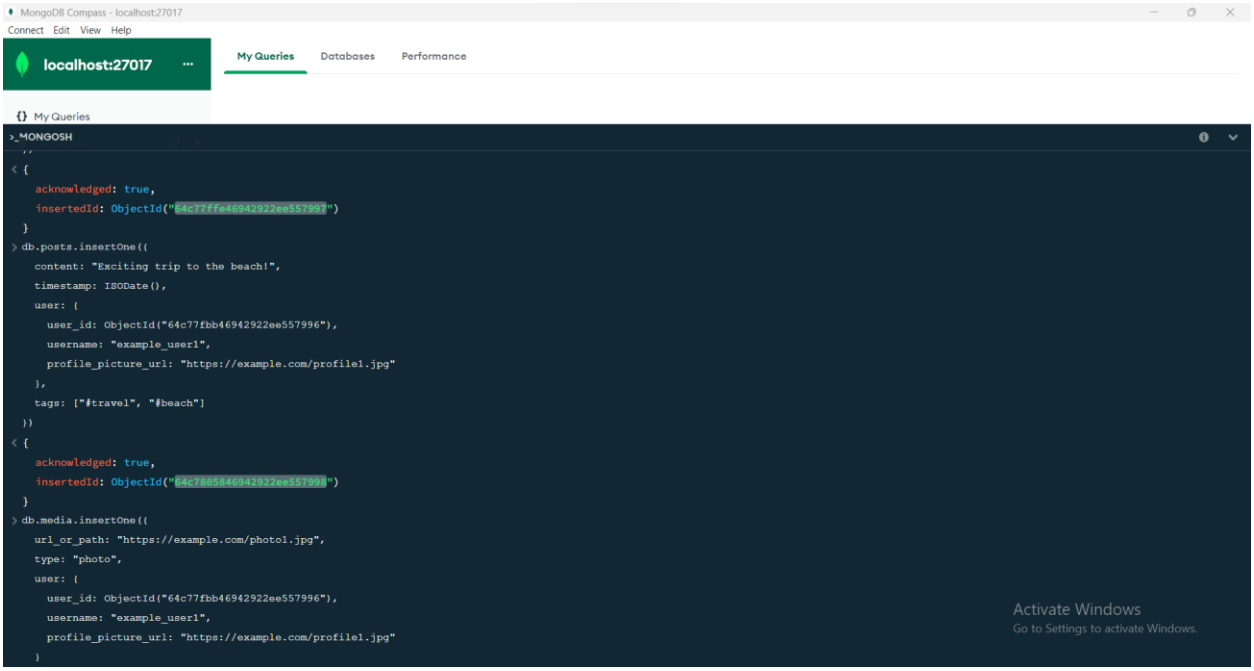
```
db.friendships.updateOne(
  { _id: ObjectId("64caa04f46942922ee55799a") },
  { $set: { timestamp: ISODate("2023-07-30T12:00:00Z") } }
);
```

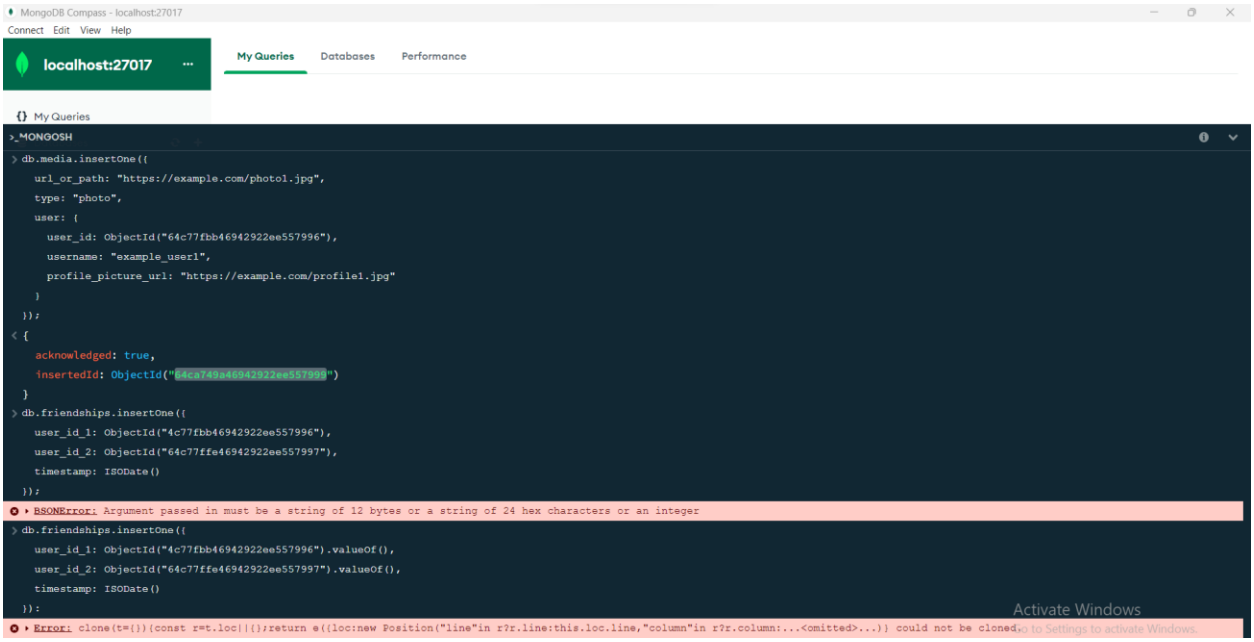
Deleting a Friendship Connection:

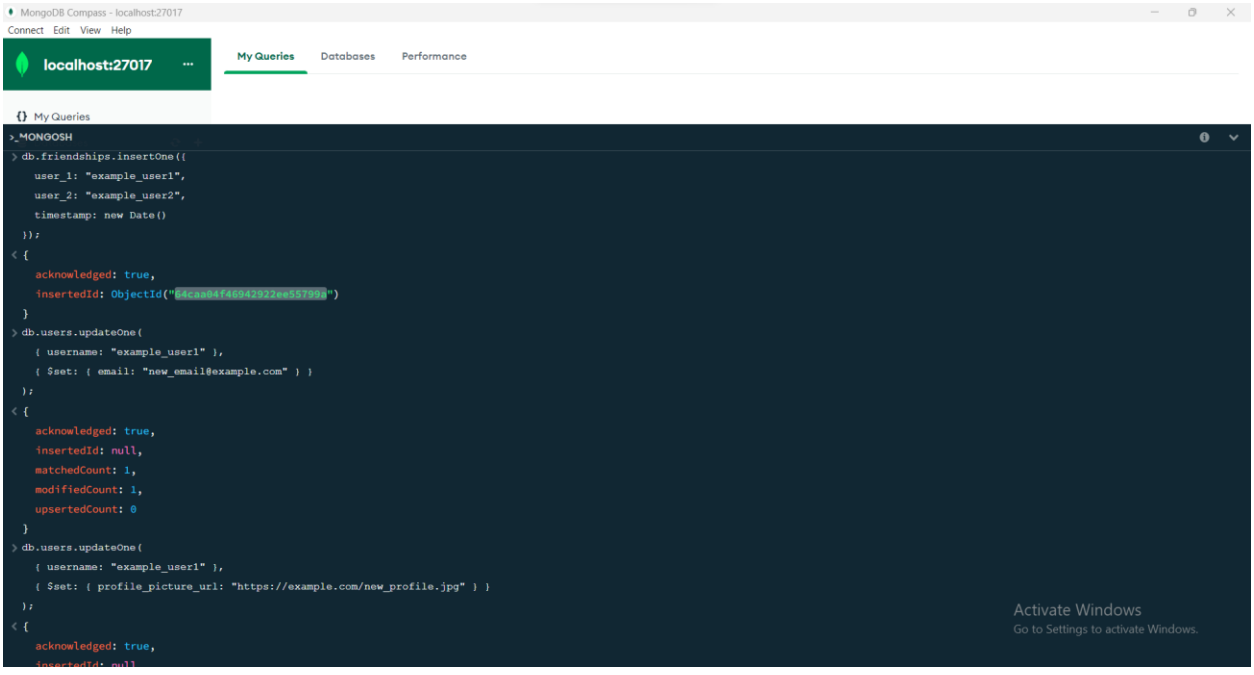
```
db.friendships.deleteOne({ _id: ObjectId("64caac4046942922ee55799f") });
```

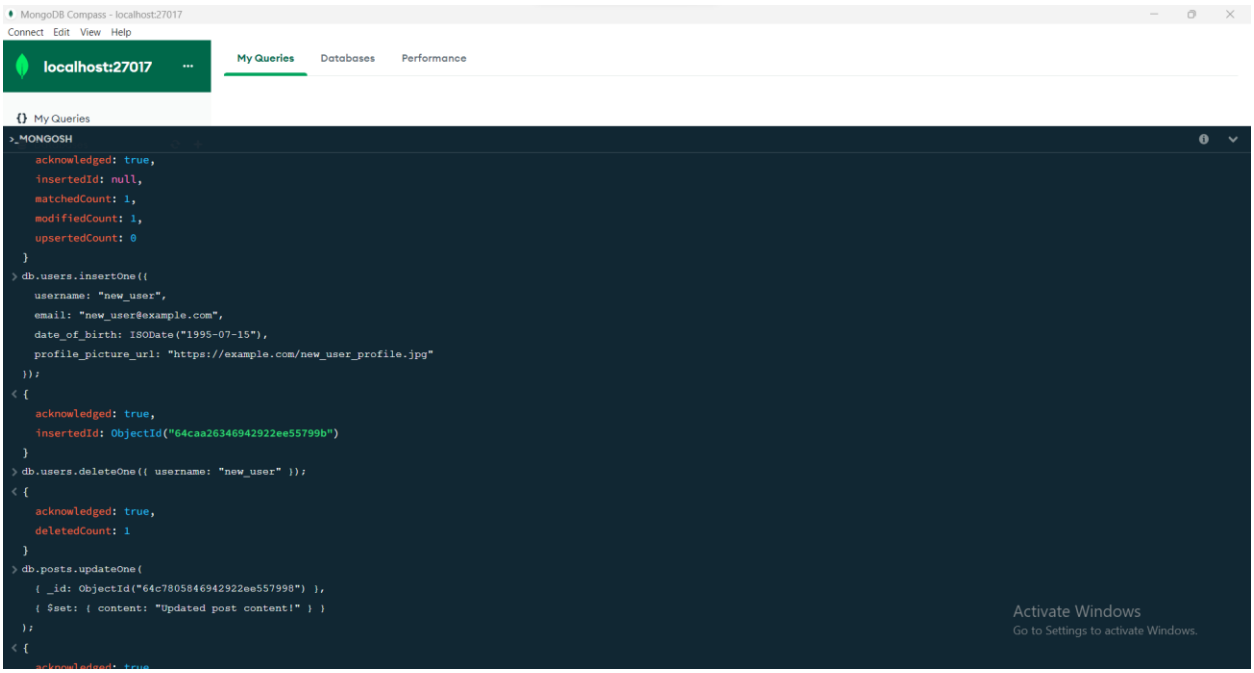
SCREENSHOTS

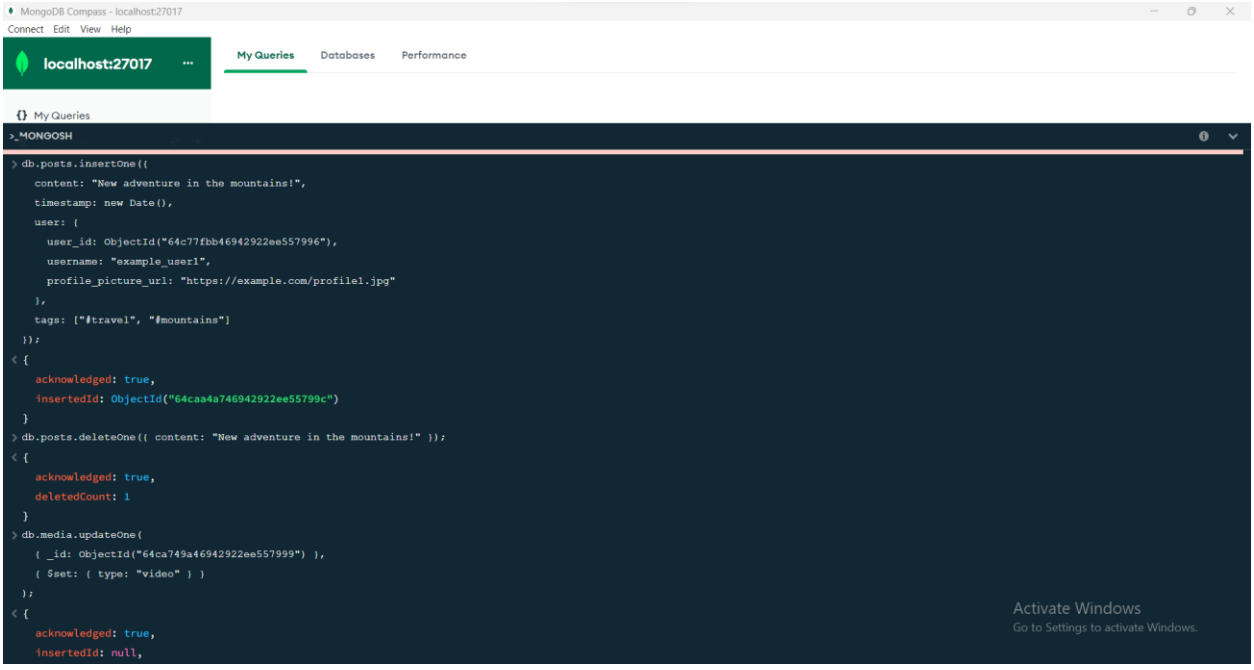


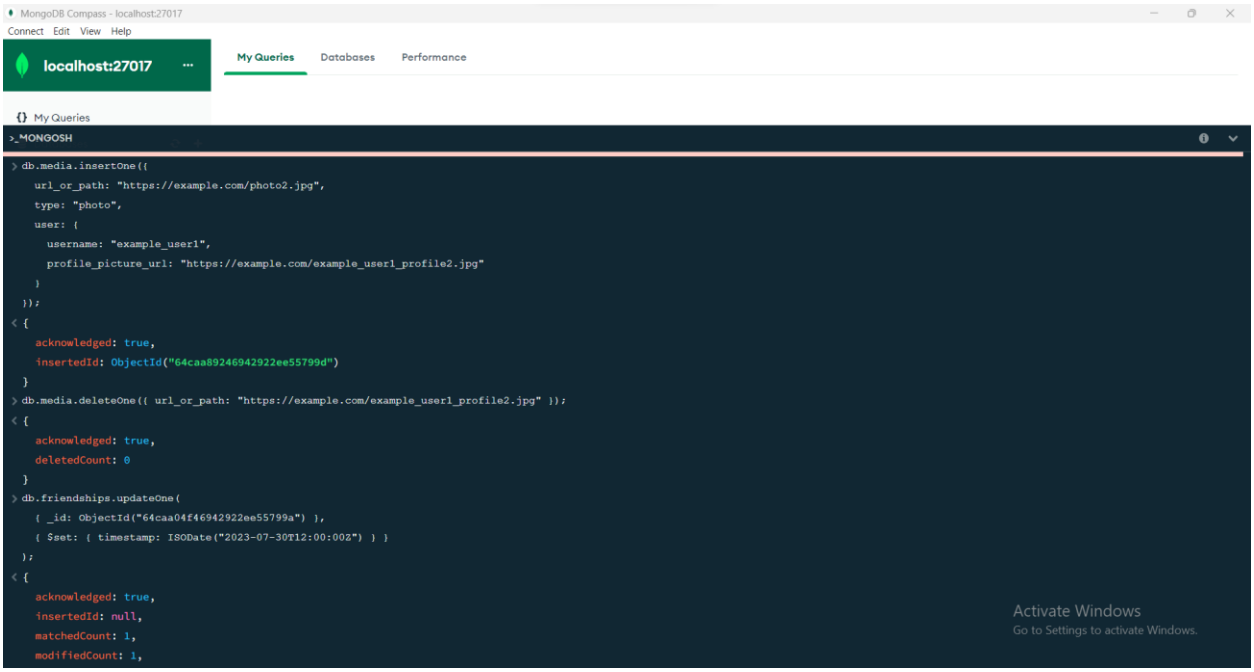


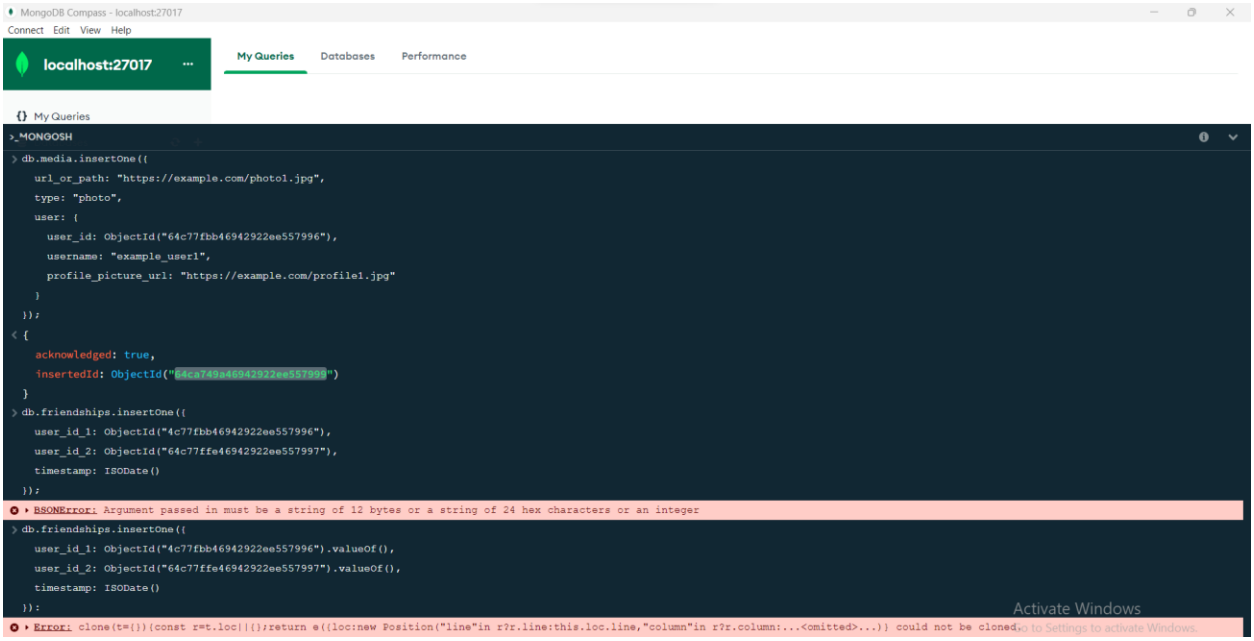

MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
db.media.insertOne({
url_or_path: "https://example.com/photo1.jpg",
type: "photo",
user: {
user_id: ObjectId("64c77fbb46942922ee557996"),
username: "example_user1",
profile_picture_url: "https://example.com/profile1.jpg"
}
});
{
acknowledged: true,
insertedId: ObjectId("64ca749a46942922ee557995")
}
db.friendships.insertOne({
user_id_1: ObjectId("4c77fbb46942922ee557996"),
user_id_2: ObjectId("64c77ffe46942922ee557997"),
timestamp: ISODate()
});
BSONError: Argument passed in must be a string of 12 bytes or a string of 24 hex characters or an integer
db.friendships.insertOne({
user_id_1: ObjectId("4c77fbb46942922ee557996").valueOf(),
user_id_2: ObjectId("64c77ffe46942922ee557997").valueOf(),
timestamp: ISODate()
});
Error: clone(t=[]){const r=t.loc||{};return e({loc:new Position("line"in r?r.line:this$.loc.line,"column"in r?r.column:...<omitted>...)} could not be cloned.
Activate Windows
Go to Settings to activate Windows.


MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
db.friendships.insertOne({
user_1: "example_user1",
user_2: "example_user2",
timestamp: new Date()
});
{
acknowledged: true,
insertedId: ObjectId("64caa04f46942922ee55799a")
}
db.users.updateOne(
{ username: "example_user1" },
{ $set: { email: "new_email@example.com" } }
);
{
acknowledged: true,
insertedId: null,
matchedCount: 1,
modifiedCount: 1,
upsertedCount: 0
}
db.users.updateOne(
{ username: "example_user1" },
{ $set: { profile_picture_url: "https://example.com/new_profile.jpg" } }
);
{
acknowledged: true,
Activate Windows
Go to Settings to activate Windows.


MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
acknowledged: true,
insertedId: null,
matchedCount: 1,
modifiedCount: 1,
upsertedCount: 0
}
db.users.insertOne({
username: "new_user",
email: "new_user@example.com",
date_of_birth: ISODate("1995-07-15"),
profile_picture_url: "https://example.com/new_user_profile.jpg"
});
{
acknowledged: true,
insertedId: ObjectId("64caa26346942922ee55799b")
}
db.users.deleteOne({ username: "new_user" });
{
acknowledged: true,
deletedCount: 1
}
db.posts.updateOne(
{ _id: ObjectId("64c7805846942922ee557998") },
{ $set: { content: "Updated post content!" } }
);
{
Activate Windows
Go to Settings to activate Windows.


MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
db.posts.insertOne({
content: "New adventure in the mountains!",
timestamp: new Date(),
user: {
user_id: ObjectId("64c77fbb46942922ee557996"),
username: "example_user1",
profile_picture_url: "https://example.com/profile1.jpg"
},
tags: ["#travel", "#mountains"]
});
{
acknowledged: true,
insertedId: ObjectId("64caa4a746942922ee55799c")
}
db.posts.deleteOne({ content: "New adventure in the mountains!" });
{
acknowledged: true,
deletedCount: 1
}
db.media.updateOne(
{ _id: ObjectId("64ca749a46942922ee557999") },
{ $set: { type: "video" } }
);
{
acknowledged: true,
insertedId: null,
Activate Windows
Go to Settings to activate Windows.


MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
> db.media.insertOne({
url_or_path: "https://example.com/photo2.jpg",
type: "photo",
user: {
username: "example_user1",
profile_picture_url: "https://example.com/example_user1_profile2.jpg"
}
});
< {
acknowledged: true,
insertedId: ObjectId("64caa89246942922ee55799d")
}
> db.media.deleteOne({ url_or_path: "https://example.com/example_user1_profile2.jpg" });
< {
acknowledged: true,
deletedCount: 0
}
> db.friendships.updateOne(
{ _id: ObjectId("64caa04f46942922ee55799a") },
{ $set: { timestamp: ISODate("2023-07-30T12:00:00Z") } }
);
< {
acknowledged: true,
insertedId: null,
matchedCount: 1,
modifiedCount: 1,
Activate Windows
Go to Settings to activate Windows.


MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
> db.media.insertOne({
url_or_path: "https://example.com/photo1.jpg",
type: "photo",
user: {
user_id: ObjectId("64c77fbb46942922ee557996"),
username: "example_user1",
profile_picture_url: "https://example.com/profile1.jpg"
}
});
< {
acknowledged: true,
insertedId: ObjectId("64ca749a46942922ee557998")
}
> db.friendships.insertOne({
user_id_1: ObjectId("4c77fbb46942922ee557996"),
user_id_2: ObjectId("64c77ffe46942922ee557997"),
timestamp: ISODate()
});
BSONError: Argument passed in must be a string of 12 bytes or a string of 24 hex characters or an integer
> db.friendships.insertOne({
user_id_1: ObjectId("4c77fbb46942922ee557996").valueOf(),
user_id_2: ObjectId("64c77ffe46942922ee557997").valueOf(),
timestamp: ISODate()
});
Error: clone(t={}){const r=t.loc||{};return e({loc:new Position("line"in r?r.line:this.loc.line,"column"in r?r.column:...<omitted>...)} could not be cloned.
Activate Windows
Go to Settings to activate Windows.

example queries to demonstrate the retrieval of data from the database

Example query to retrieve all posts by a specific user:

```
const userPosts = db.posts.find({
  "user.user_id": ObjectId("64c77fbb46942922ee557996")
}).toArray();
printjson(userPosts);
```

Example query to retrieve all media files uploaded by a specific user:

```
const userMedia = db.media.find({
  "user.user_id": ObjectId("64c77fbb46942922ee557996")
}).toArray();
printjson(userMedia);
```

Example query to retrieve all friends of a specific user:

```
const userFriends = db.friendships.find({
  $or: [
    { user_id_1: ObjectId("64c77fbb46942922ee557996") },
    { user_id_2: ObjectId("64c77ffe46942922ee557997") }
  ]
}).toArray();
printjson(userFriends);
```

List of Users with the Count of Their Posts

```
db.users.aggregate([
 {
  $lookup: {
   from: "posts",
   localField: "_id",
   foreignField: "user.user_id",
   as: "user_posts"
  }
 },
 {
  $project: {
   _id: 1,
   username: 1,
   email: 1,
   date_of_birth: 1,
   profile_picture_url: 1,
   post_count: { $size: "$user_posts" }
  }
 }
]);
```

This query performs a lookup between the Users and Posts collections and returns a list of users along with the count of their posts.

List of Tags and the Number of Posts for Each Tag

```
db.posts.aggregate([
 { $unwind: "$tags" },
 {
  $group: {
   _id: "$tags",
   count: { $sum: 1 }
  }
 },
 {
  $sort: { count: -1 }
 }
])
```

This query uses the aggregation pipeline to unwind the `tags` array in the Posts collection, groups the documents by tags, and calculates the number of posts for each tag. The results are sorted in descending order based on the post count.

Users with Most Posts

```
db.users.aggregate([
 {
  $lookup: {
   from: "posts",
   localField: "_id",
   foreignField: "user.user_id",
   as: "user_posts"
  }
```

```
  },
  {
    $project: {
      _id: 1,
      username: 1,
      email: 1,
      date_of_birth: 1,
      profile_picture_url: 1,
      post_count: { $size: "$user_posts" }
    }
  },
  {
    $sort: { post_count: -1 }
  },
  {
    $limit: 1
  }
]);
```

Posts with Most Tags

```
db.posts.aggregate([
  {
    $project: {
      _id: 1,
      content: 1,
      timestamp: 1,
      user: 1,
```

```
    tags: 1,
    tag_count: { $size: "$tags" }
   }
  },
  {
   $sort: { tag_count: -1 }
  },
  {
   $limit: 1
  }
]);
```

Users and Their Friends

```
db.users.aggregate([
 {
  $lookup: {
   from: "friendships",
   let: { user_id: "$_id" },
   pipeline: [
    {
     $match: {
      $expr: {
       $or: [
        { $eq: ["$user_id_1", "$$user_id"] },
        { $eq: ["$user_id_2", "$$user_id"] }
       ]
      }
```

```
        }
       }
      ],
      as: "friends"
    }
  },
  {
    $project: {
      _id: 1,
      username: 1,
      email: 1,
      date_of_birth: 1,
      profile_picture_url: 1,
      friends: {
        $map: {
          input: "$friends",
          as: "friendship",
          in: {
            $cond: {
              if: { $eq: ["$$friendship.user_id_1", "$_id"] },
              then: "$$friendship.user_id_2",
              else: "$$friendship.user_id_1"
            }
          }
        }
      }
    }
  }
]);
```

retrieve all posts by a specific user

```
const userPosts = db.posts.find({
  "user.user_id": ObjectId("64c77fbb46942922ee557996")
}).toArray();
printjson(userPosts);
```

retrieve all media files uploaded by a specific user

```
const userMedia = db.media.find({
  "user.user_id": ObjectId("64c77fbb46942922ee557996")
}).toArray();
printjson(userMedia);
```

Retrieving a post with user details embedded

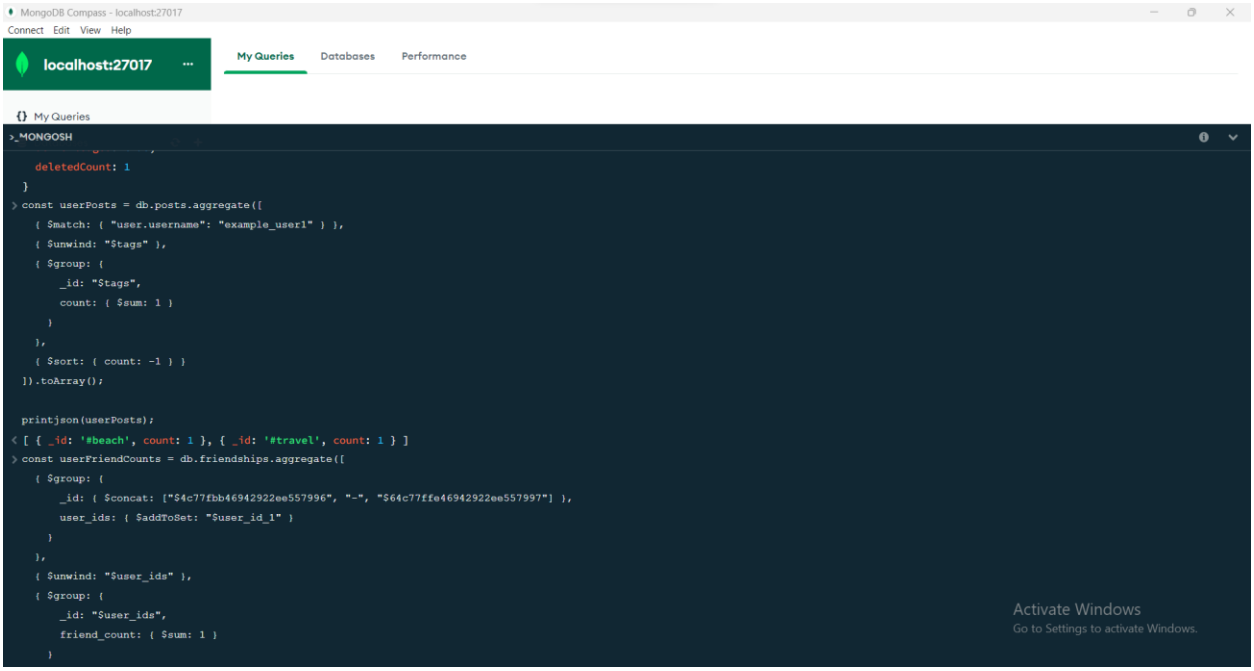
```
db.posts.aggregate([
 {
   $lookup: {
     from: "users",
     localField: "user_id",
     foreignField: "_id",
     as: "user"
   }
 },
 {
   $unwind: "$user"
```

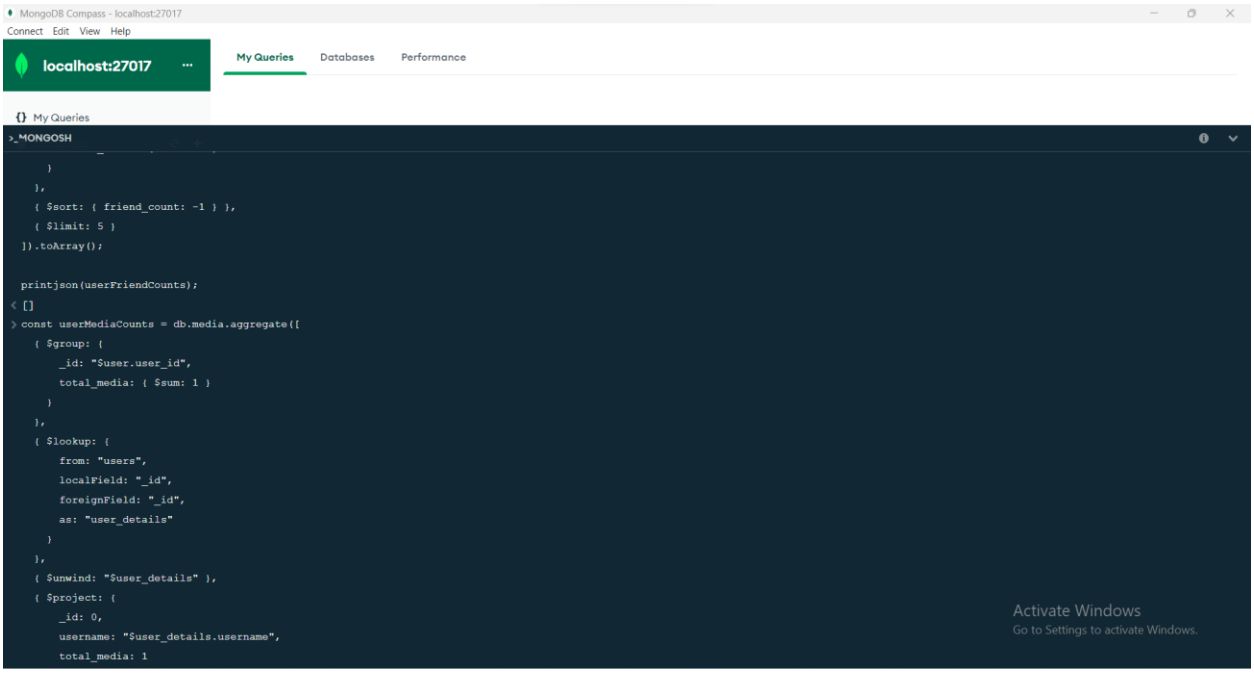
```
  },
  {
    $match: {
      _id: ObjectId("64c7805846942922ee557998")
    }
  }
]);
```

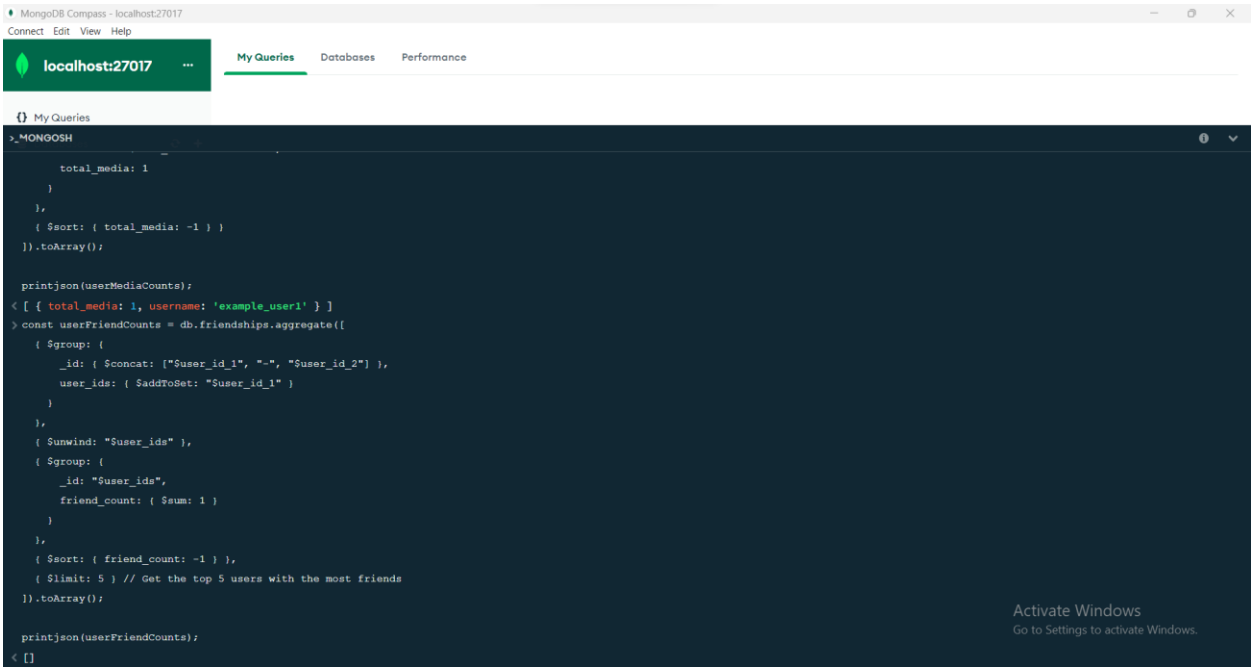
Retrieving posts with a specific tag

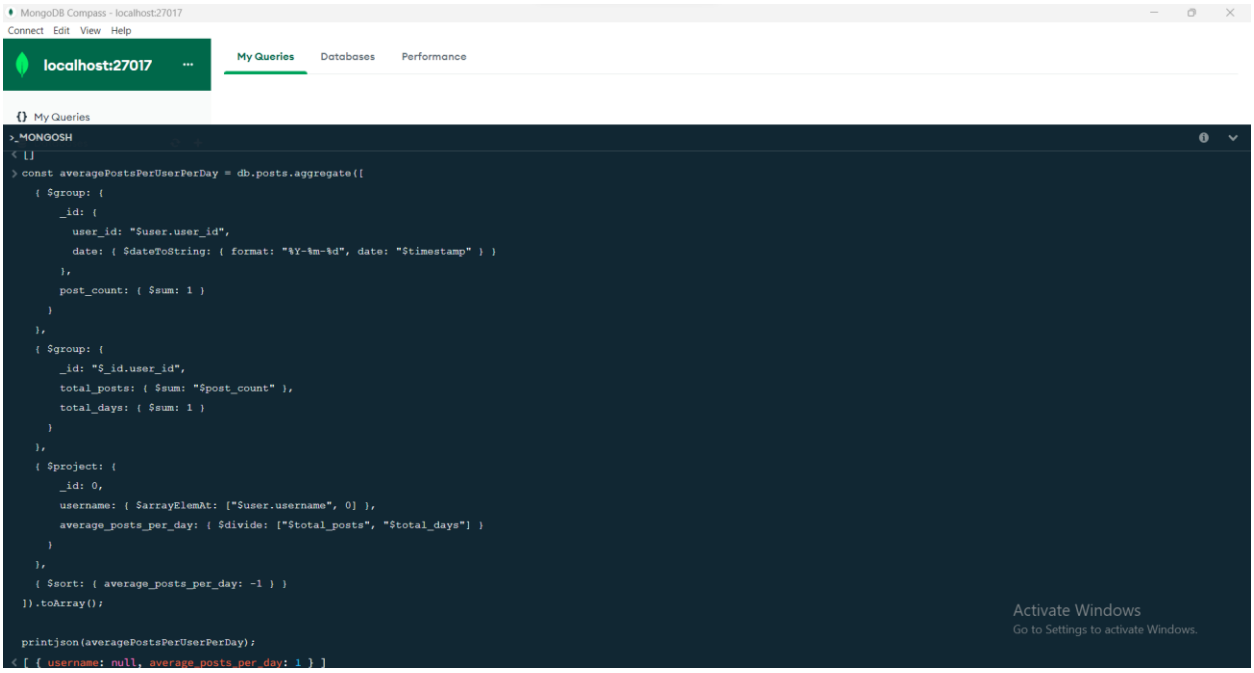
```
db.posts.find({
  tags: "#travel"
});
```

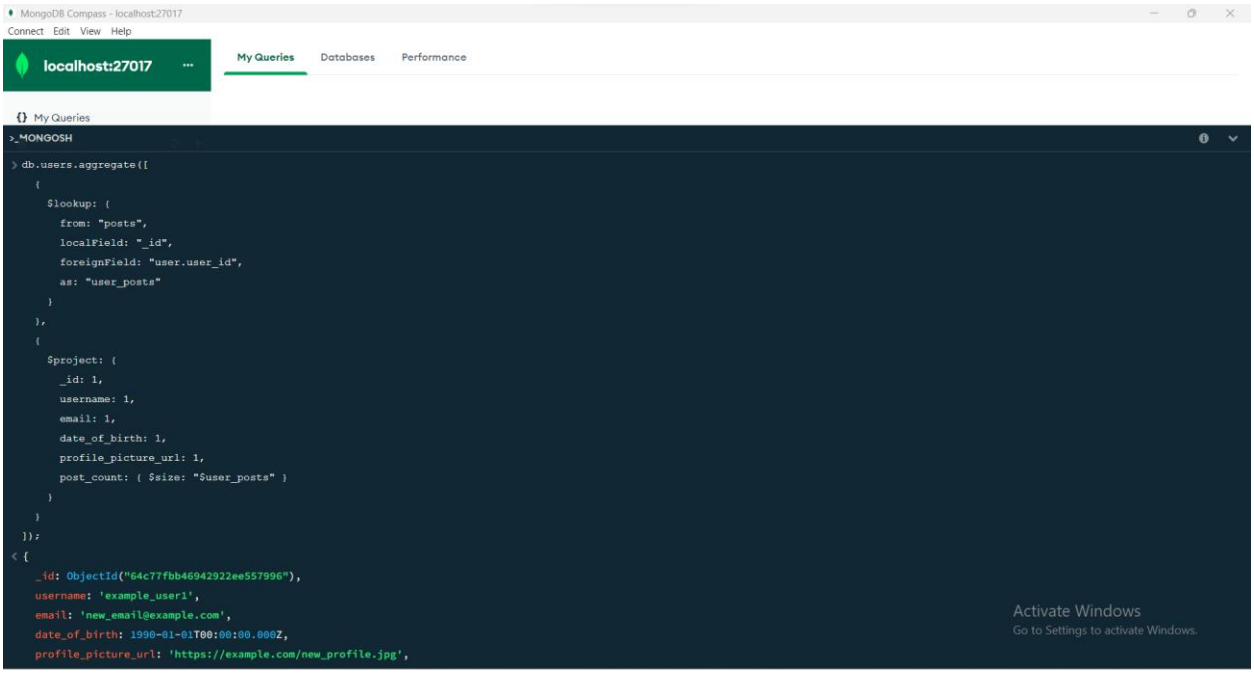
SCREENSHOTS

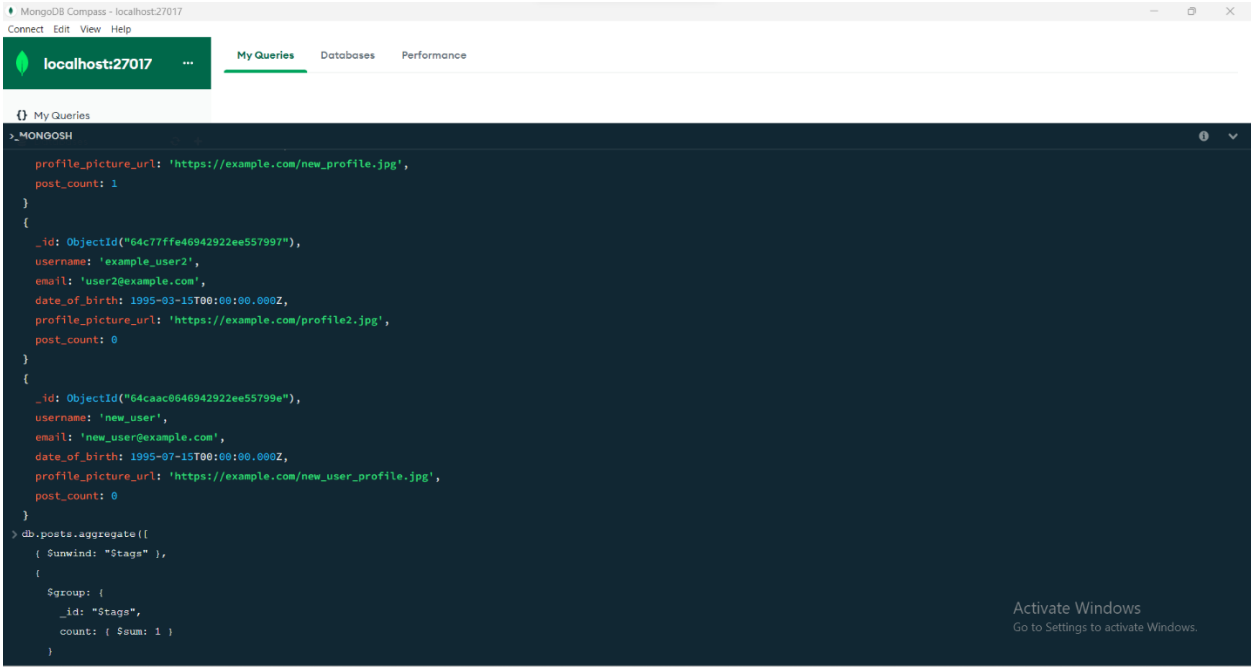


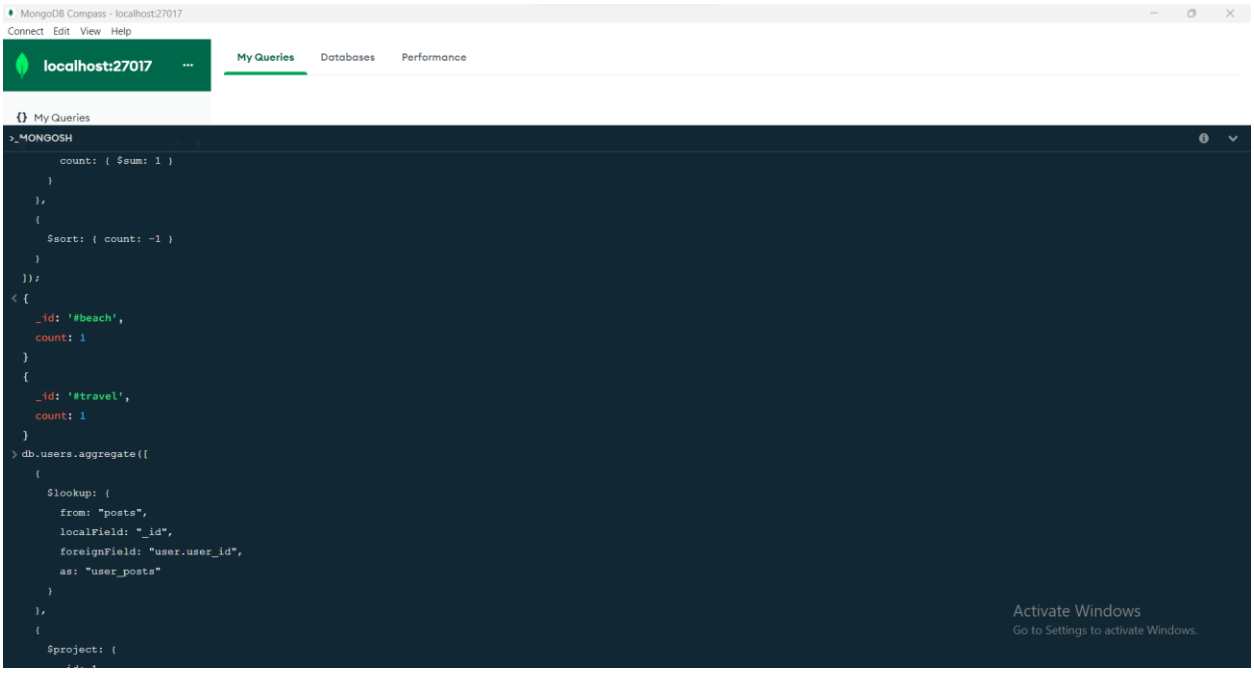


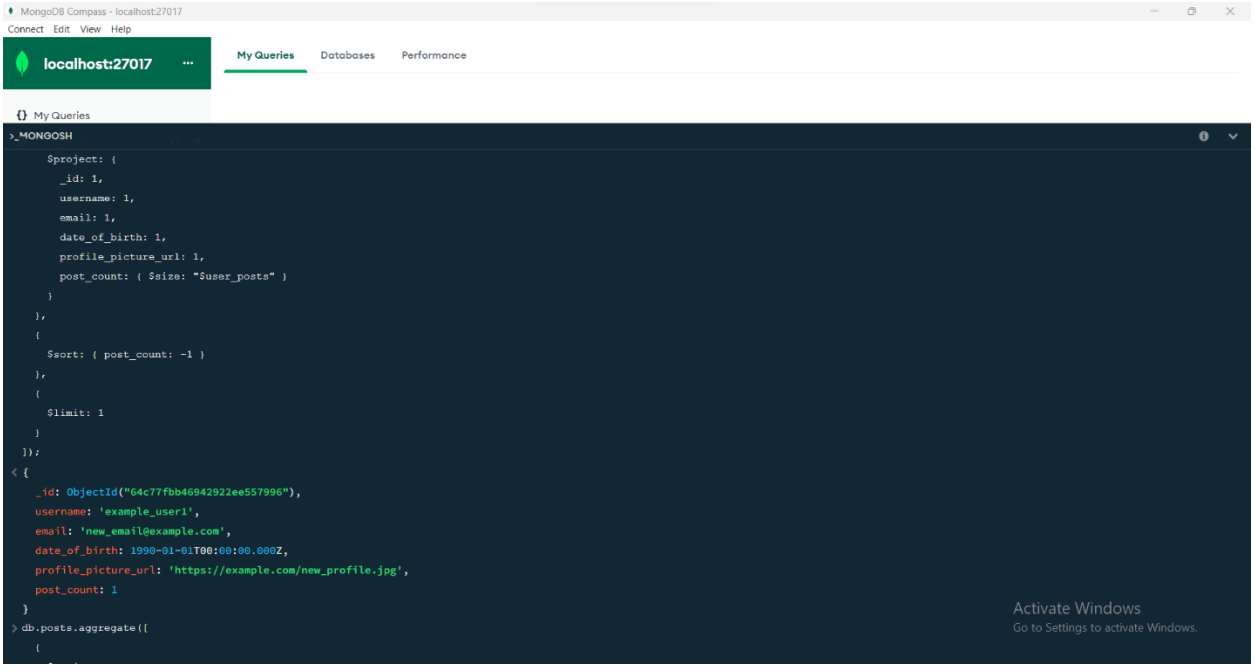

MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
total_media: 1
}
},
{ $sort: { total_media: -1 } }
]).toArray();
printjson(userMediaCounts);
[{ total_media: 1, username: 'example_user1' }]
const userFriendCounts = db.friendships.aggregate([
{ $group: {
_id: { $concat: ["$user_id_1", "-", "$user_id_2"] },
user_ids: { $addToSet: "$user_id_1" }
}
},
{ $unwind: "$user_ids" },
{ $group: {
_id: "$user_ids",
friend_count: { $sum: 1 }
}
},
{ $sort: { friend_count: -1 } },
{ $limit: 5 } // Get the top 5 users with the most friends
]).toArray();
printjson(userFriendCounts);
[]
Activate Windows
Go to Settings to activate Windows.

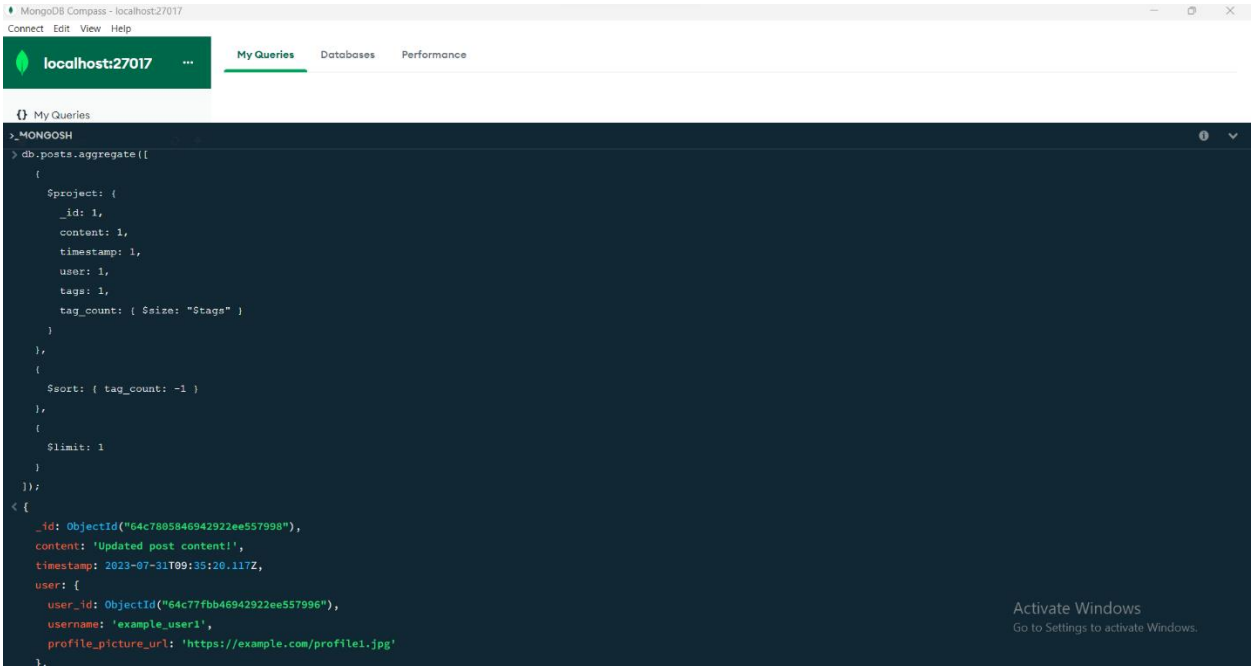

MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
[]
const averagePostsPerUserPerDay = db.posts.aggregate([
{ $group: {
_id: {
user_id: "$user.user_id",
date: { $dateToString: { format: "%Y-%m-%d", date: "$timestamp" } }
},
post_count: { $sum: 1 }
}
},
{ $group: {
_id: "$_id.user_id",
total_posts: { $sum: "$post_count" },
total_days: { $sum: 1 }
}
},
{ $project: {
_id: 0,
username: { $arrayElemAt: ["$user.username", 0] },
average_posts_per_day: { $divide: ["$total_posts", "$total_days"] }
}
},
{ $sort: { average_posts_per_day: -1 } }
]).toArray();
printjson(averagePostsPerUserPerDay);
[{ username: null, average_posts_per_day: 1 }]
Activate Windows
Go to Settings to activate Windows.

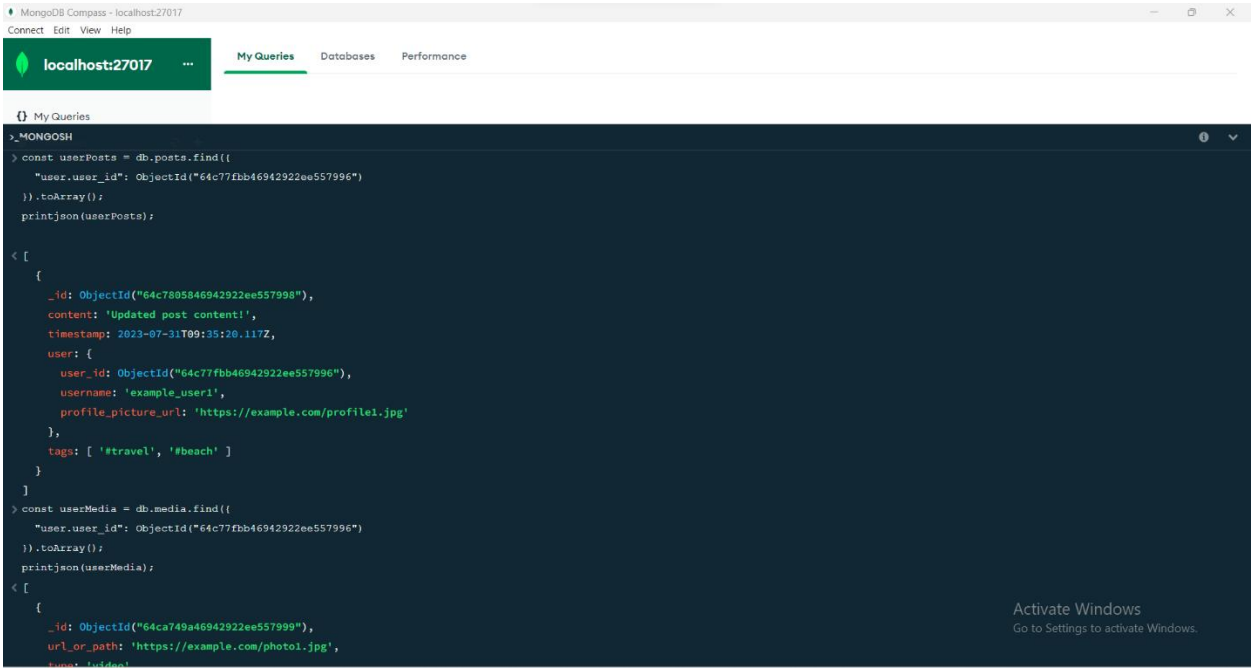

MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
> db.users.aggregate([
{
$lookup: {
from: "posts",
localField: "_id",
foreignField: "user.user_id",
as: "user_posts"
}
},
{
$project: {
_id: 1,
username: 1,
email: 1,
date_of_birth: 1,
profile_picture_url: 1,
post_count: { $size: "$user_posts" }
}
}
]);
< {
_id: ObjectId("64c77fbb46942922ee557996"),
username: 'example_user1',
email: 'new_email@example.com',
date_of_birth: 1990-01-01T00:00:00.000Z,
profile_picture_url: 'https://example.com/new_profile.jpg',
Activate Windows
Go to Settings to activate Windows.

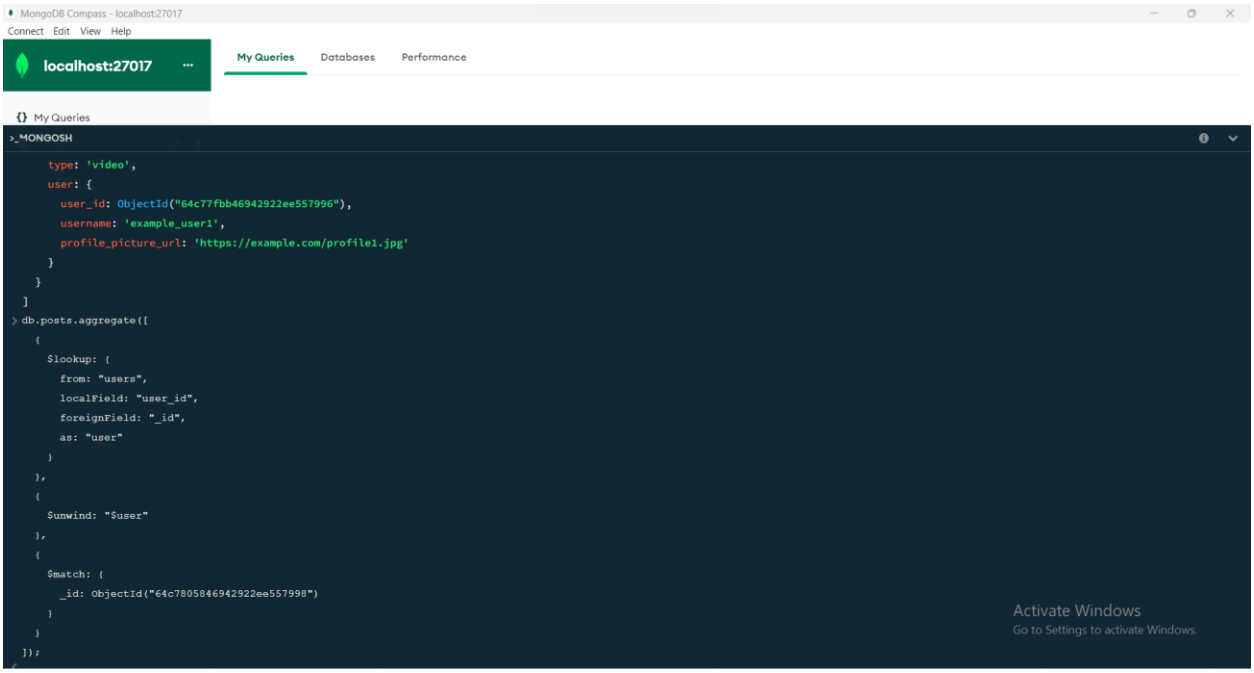

MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
profile_picture_url: 'https://example.com/new_profile.jpg',
post_count: 1
}
{
_id: ObjectId("64c77ffe46942922ee557997"),
username: 'example_user2',
email: 'user2@example.com',
date_of_birth: 1995-03-15T00:00:00.000Z,
profile_picture_url: 'https://example.com/profile2.jpg',
post_count: 0
}
{
_id: ObjectId("64caac0646942922ee55799e"),
username: 'new_user',
email: 'new_user@example.com',
date_of_birth: 1995-07-15T00:00:00.000Z,
profile_picture_url: 'https://example.com/new_user_profile.jpg',
post_count: 0
}
> db.posts.aggregate([
{ $unwind: "$tags" },
{
$group: {
_id: "$tags",
count: { $sum: 1 }
}
Activate Windows
Go to Settings to activate Windows.

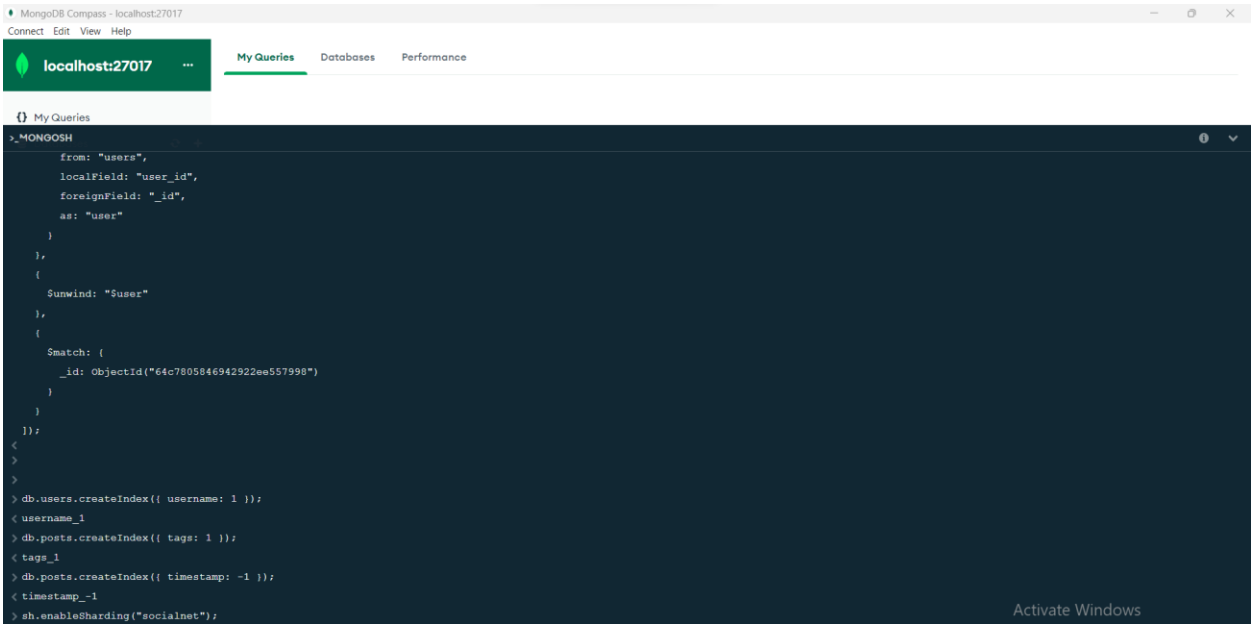

MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
count: { $sum: 1 }
$sort: { count: -1 }
{ _id: '#beach', count: 1 }
{ _id: '#travel', count: 1 }
db.users.aggregate([
$lookup: { from: "posts", localField: "_id", foreignField: "user.user_id", as: "user_posts" }
$project: {
Activate Windows
Go to Settings to activate Windows.


MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
$project: { _id: 1, username: 1, email: 1, date_of_birth: 1, profile_picture_url: 1, post_count: { $size: "$user_posts" } }
$sort: { post_count: -1 }
$limit: 1
{ _id: ObjectId("64c77fbb46942922ee557996"), username: 'example_user1', email: 'new_email@example.com', date_of_birth: 1990-01-01T00:00:00.000Z, profile_picture_url: 'https://example.com/new_profile.jpg', post_count: 1 }
db.posts.aggregate([
Activate Windows
Go to Settings to activate Windows.


MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
> db.posts.aggregate([
{
$project: {
_id: 1,
content: 1,
timestamp: 1,
user: 1,
tags: 1,
tag_count: { $size: "$tags" }
}
},
{
$sort: { tag_count: -1 }
},
{
$limit: 1
}
]);
< {
_id: ObjectId("64c7805846942922ee557998"),
content: 'Updated post content!',
timestamp: 2023-07-31T09:35:20.117Z,
user: {
user_id: ObjectId("64c77fbb46942922ee557996"),
username: 'example_user1',
profile_picture_url: 'https://example.com/profile1.jpg'
},
Activate Windows
Go to Settings to activate Windows.


MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
> const userPosts = db.posts.find({
"user.user_id": ObjectId("64c77fbb46942922ee557996")
}).toArray();
printjson(userPosts);
< [
{
_id: ObjectId("64c7805846942922ee557998"),
content: 'Updated post content!',
timestamp: 2023-07-31T09:35:20.117Z,
user: {
user_id: ObjectId("64c77fbb46942922ee557996"),
username: 'example_user1',
profile_picture_url: 'https://example.com/profile1.jpg'
},
tags: ['#travel', '#beach']
}
]
> const userMedia = db.media.find({
"user.user_id": ObjectId("64c77fbb46942922ee557996")
}).toArray();
printjson(userMedia);
< [
{
_id: ObjectId("64ca749a46942922ee557999"),
url_or_path: 'https://example.com/photo1.jpg',
Activate Windows
Go to Settings to activate Windows.


MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
type: 'video',
user: {
user_id: ObjectId("64c77fbb46942922ee557996"),
username: 'example_user1',
profile_picture_url: 'https://example.com/profile1.jpg'
}
}
]
db.posts.aggregate([
{
$lookup: {
from: "users",
localField: "user_id",
foreignField: "_id",
as: "user"
}
},
{
$unwind: "$user"
},
{
$match: {
_id: ObjectId("64c7805846942922ee557998")
}
}
]);
Activate Windows
Go to Settings to activate Windows.


MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
My Queries
>_MONGOSH
from: "users",
localField: "user_id",
foreignField: "_id",
as: "user"
}
},
{
$unwind: "$user"
},
{
$match: {
_id: ObjectId("64c7805846942922ee557998")
}
}
]);
db.users.createIndex({ username: 1 });
username_1
db.posts.createIndex({ tags: 1 });
tags_1
db.posts.createIndex({ timestamp: -1 });
timestamp_-1
sh.enableSharding("socialnet");
Activate Windows

```
> db.posts.find({
    tags: "#travel"
  });
< {
    _id: ObjectId("64c7805846942922ee557998"),
    content: 'Updated post content!',
    timestamp: 2023-07-31T09:35:20.117Z,
    user: {
      user_id: ObjectId("64c77fbb46942922ee557996"),
      username: 'example_user1',
      profile_picture_url: 'https://example.com/profile1.jpg'
    },
    tags: [
      '#travel',
      '#beach'
    ]
  }
```

Indexing:

Index on username for fast user lookups

```
db.users.createIndex({ username: 1 });
```

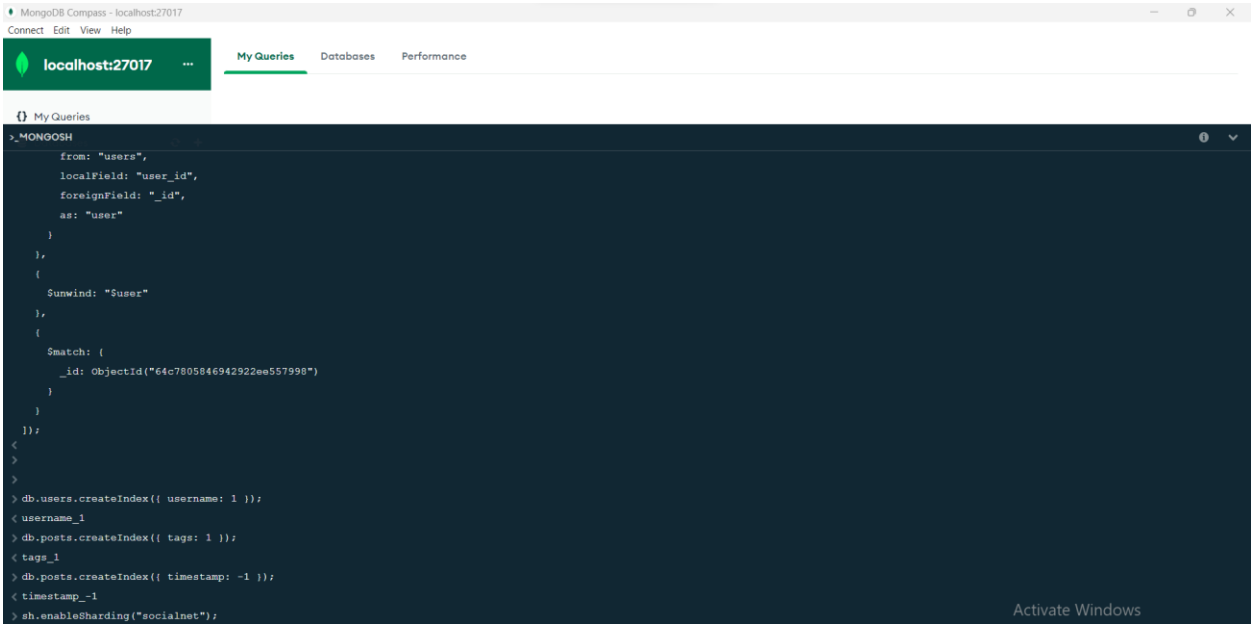
Index on tags for fast tag-based queries

```
db.posts.createIndex({ tags: 1 });
```

Index on timestamp for sorting and filtering posts by time

```
db.posts.createIndex({ timestamp: -1 });
```

SCREENSHOTS



Scaling Techniques:

To handle scalability requirements, we can implement sharding in MongoDB. Sharding allows us to distribute data across multiple servers to handle increased data volume and traffic.

Sharding the Posts Collection:

Assuming we have selected the `user_id` field as the sharding key for the Posts collection:

Enable sharding on the database (replace 'socialnet' with your actual database name)

```
sh.enableSharding("socialnet");
```

Shard the Posts collection based on the 'user_id' field

```
sh.shardCollection("socialnet.posts", { "user.user_id": 1 });
```

Advantages of Sharding:

Horizontal Scaling: Sharding allows the database to scale horizontally, distributing data and read/write operations across multiple servers. This ensures that the system can handle increased traffic and data volume.

Improved Performance: With data distributed across shards, the database can parallelize read and write operations, resulting in improved query performance and reduced response times.

Fault Tolerance: Sharding enhances fault tolerance as each shard is an independent replica set. If one shard goes down, the system can continue to function using the other available shards

Indexing and sharding techniques have been applied to improve query performance and handle scalability requirements as the platform grows.