MongoDB Performance Optimization

Techniques Used:

1. Indexing
2. Query Refinement
3. Sharding (Data Partitioning)

🔹 Indexing in MongoDB

MongoDB uses indexes to enhance query efficiency by reducing the number of documents it needs to scan.

Common Scenarios for Indexing:

- Filtering queries (find())
- Sorting (sort())
- Aggregations
- Join operations ($lookup)

🛠 Example:

Suppose we have a `products` collection with a frequent query on the `category` field

Before Optimization:

```
db.products.find({ category: "electronics" });
```

This scans every document in the collection — slow with millions of entries.

After Optimization:

```
db.products.createIndex({ category: 1 });
```

Now, MongoDB uses the index to rapidly access documents matching "electronics".

🔹 Query Refinement

Efficient querying isn't just about indexes—query structure plays a big role. Reducing payload and matching fields precisely speeds up performance.

🛠 Example:

Let's say we have a customers collection.

Before:

```
db.customers.find({ active: true });
```

Returns all fields — even unnecessary ones, increasing response time and memory usage.

After:

```
db.customers.find(
  { active: true },
  { _id: 0, name: 1, email: 1 }
);
```

Returns only essential fields — lighter and faster.

🔹 Sharding: Data Partitioning in MongoDB

MongoDB handles large-scale data through **sharding**—distributing data across multiple servers. This improves horizontal scalability and performance for high-throughput workloads.

🛠 Example:

We have an events collection logging user activity across years.

Before:

```
db.events.find({ timestamp: { $gte: ISODate("2023-01-01") } });
```

With no partitioning, all documents are scanned — slow**.**

After:

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

db.events.createIndex({ timestamp: 1 });

sh.shardCollection("analytics.events", { timestamp: 1 });
```

Now, queries targeting timestamp will only hit relevant shards.