NoSQL

Problem Statement 1:

You are tasked with designing a NoSQL data model for a social media application. The data should include user profiles, posts, comments, and likes. Create a schema that optimizes query performance for retrieving a user's posts and comments.

NoSQL Data Model for a Social Media Application (Using MongoDB)

A **document-oriented model (MongoDB)** is ideal for this use case as it optimizes read performance by storing related data together.

---

1. Users Collection

Stores user profile details.

```
{
  "_id": "user123",
  "username": "john_doe",
  "email": "john@example.com",
  "profile_pic": "profile123.jpg",
  "bio": "Tech enthusiast and blogger",
  "followers": ["user456", "user789"],
  "following": ["user456", "user999"]
}
```

- **Optimized for quick lookups** (e.g., fetching user details, followers, following list).
- Uses `followers` and `following` as **arrays** for fast retrieval.

---

2. Posts Collection

Stores posts made by users.

```
{
  "_id": "post789",
  "user_id": "user123",
  "content": "Exploring NoSQL databases!",
  "media": "nosql.jpg",
  "timestamp": "2024-02-03T10:00:00Z",
  "likes_count": 20,
  "comments_count": 5,
  "likes": ["user456", "user789"],
  "comments": [
    {
      "comment_id": "cmt001",
      "user_id": "user456",
      "text": "Interesting post!",
```

```
      "timestamp": "2024-02-03T10:05:00Z"
    },
    {
      "comment_id": "cmt002",
      "user_id": "user999",
      "text": "I love NoSQL!",
      "timestamp": "2024-02-03T10:10:00Z"
    }
  ]
}
```

- **Embedded comments** for **fast retrieval** when fetching posts.
- **Likes stored as an array** for quick counting.
- **`likes_count` and `comments_count` maintained** to avoid performance issues with aggregation.

---

3. Query Optimization for Retrieving User Posts & Comments

Fetching a User's Posts Efficiently:

```
db.posts.find({ user_id: "user123" }).sort({ timestamp: -1 });
```

- Uses an **index on `user_id`** for fast lookup.
- **Sorting by `timestamp`** ensures the latest posts appear first.

Fetching a User's Comments Across Posts:

```
db.posts.find({ "comments.user_id": "user123" }, { "comments.$": 1 });
```

- Uses **dot notation** to find comments where `user_id` matches.

---

4. Indexing Strategy

To improve performance:

```
db.posts.createIndex({ user_id: 1 });
db.posts.createIndex({ "comments.user_id": 1 });
```

- Index on `user_id` optimizes queries retrieving posts by a specific user.
- Index on `comments.user_id` speeds up queries for fetching a user’s comments.

Why This Model?

Optimized Reads – Posts include comments & likes, reducing joins.
Efficient Querying – Indexes on `user_id` & `comments.user_id`.
Fast Writes – Append operations for likes/comments are efficient.

This model ensures a balance between **fast retrieval** and **manageable document sizes**.