**Experiment 7: Implement Functions: Count – Sort – Limit – Skip – Aggregate using MongoDB**

| Function | Method | Purpose |
|---|---|---|
| Count | countDocuments() | Counts matching documents. |
| Sort | sort() | Sorts documents by fields. |
| Limit | limit() | Limits the number of documents in results. |
| Skip | skip() | Skips a specified number of documents. |
| Aggregate | aggregate() | Performs complex queries using pipelines. |

These methods provide powerful tools for querying and manipulating data in MongoDB. Depending on your requirements, you can use them individually or in combination for more complex operations.

**Insert the Collection into MongoDB: users**

```
db.users.insertMany([
  { "user_id": 1, "name": "John", "age": 30, "interests": ["coding", "sports"] },
  { "user_id": 2, "name": "Alice", "age": 25, "interests": ["reading", "music"] },
  { "user_id": 3, "name": "Bob", "age": 35, "interests": ["traveling", "sports"] },
  { "user_id": 4, "name": "Eve", "age": 28, "interests": ["cooking", "reading"] },
  { "user_id": 5, "name": "Charlie", "age": 40, "interests": ["music", "gardening"] }
]
```

**1. Count**

**Query:**

Count the number of users older than 30:

```
db.users.countDocuments({ age: { $gt: 30 } })
```

**2. Sort**

**Query:**

Sort users by age in descending order:

```
db.users.find().sort({ age: -1 })
```

**3. Limit**

**Query:**

Fetch the first 2 users:

db.users.find().limit(2)

**4. Skip**

**Query:**

Skip the first 2 users and fetch the next 2:

db.users.find().skip(2).limit(2)

**5. Aggregate**

**Query 1: Group by Age (Count Users per Age Group)**

```
db.users.aggregate([
  { $group: { _id: "$age", count: { $sum: 1 } } }
])
```

Query 2: Find the Oldest User, the document with the highest age is returned.

```
db.users.aggregate([
  { $sort: { age: -1 } },
  { $limit: 1 }
])
```

Query 3: Average Age of Users

```
db.users.aggregate([
  { $group: { _id: null, averageAge: { $avg: "$age" } } }
])
```

**Combining Functions**

**Query: Fetch the second page of 2 users sorted by age in descending order**

```
db.users.aggregate([
  { $sort: { age: -1 } },
  { $skip: 2 },
  { $limit: 2 } ])
```

**Questions**

**1. Count**

Count the number of users who live in Chicago and are interested in "sports."

**2. Sort**

Sort users by `salary` in descending order, and then by `age` in ascending order (in case of ties).

**3. Limit and Skip**

Fetch the third and fourth highest-paid users:

**4. Aggregation**

**Query 1: Average Salary by City**
Find the average salary of users grouped by city:

**Query 2: Users Interested in "Sports" with Total Salary**
Find users who are interested in "sports" and calculate their total salary:

**Query 3: List Users with Selected Fields**
Return only the name, age, and salary fields of users, sorted by age in ascending order:

**5. Complex Filter: Multiple Conditions**

Find users who are:

- Above 30 years old
- Living in "Chicago" or "New York"
- Interested in "sports"

**6. Add a New Field to Users**

- Add a new field called `status` that labels users as either "High Earner" (salary >= 75000) or "Low Earner":