upGrad

MongoDB Practice Solutions

Section 1: CRUD Operations & Filters

1. Insert a new book into the collection with the following details:

- `title: "MongoDB Basics"`
- `isbn: "1234567890"`
- `pageCount: 250`
- `publishedDate: current date`
- `status: "DRAFT"`
- `authors: ["John Doe"]`
- `categories: ["Databases"]`

```JSON
db.books.insertOne({
  title: "MongoDB Basics",
  isbn: "1234567890",
  pageCount: 250,
  publishedDate: new Date(),
  status: "DRAFT",
  authors: ["John Doe"],
  categories: ["Databases"]
});
```

2. Insert two new books into the collection in a single command.

- `title: "Mastering React", isbn: "9876543210", pageCount: 350, status: "PUBLISH", authors: ["Jane Smith"], categories: ["Frontend", "JavaScript"]`
- `title: "Node.js Essentials", isbn: "1234987650", pageCount: 420, status: "PUBLISH", authors: ["Alex Brown"], categories: ["Backend", "JavaScript"]`

```
JSON
db.books.insertMany([
  {
    title: "Mastering React",
    isbn: "9876543210",
    pageCount: 350,
    status: "PUBLISH",
    authors: ["Jane Smith"],
    categories: ["Frontend", "JavaScript"]
  },
  {
    title: "Node.js Essentials",
    isbn: "1234987650",
    pageCount: 420,
    status: "PUBLISH",
    authors: ["Alex Brown"],
    categories: ["Backend", "JavaScript"]
  }
]);
```

3. Find and return the complete document(s) where the `authors` array contains the exact value `"Gojko Adzic"`.

```
JSON
db.books.find({ authors: "Gojko Adzic" });
```

4. Retrieve the `title` and `publishedDate` of all books published before January 1, 2010. Try searching how you can print only selected fields in the output.

```
JSON
db.books.find(
  { publishedDate: { $lt: new Date("2010-01-01") } },
  { title: 1, publishedDate: 1, _id: 0 }
);
```

5. Update the `status` of the book whose `title` is "Specification by Example" from "PUBLISH" to "ARCHIVED".

```
JSON
        db.books.updateOne(
          { title: "Specification by Example" },
          { $set: { status: "ARCHIVED" } }
        );
```

6. Increment the `pageCount` of the book with the `title` "Flex 3 in Action" by 20.

```
JSON
        db.books.updateOne(
          { title: "Flex 3 in Action" },
          { $inc: { pageCount: 20 } }
        );
```

7. Rename the field `shortDescription` to `summary` for all books that have this field present. Look up the `$exists` update operator for this.

```
JSON
        db.books.updateMany(
          { shortDescription: { $exists: true } },
          { $rename: { "shortDescription": "summary" } }
        );
```

8. Remove the field `thumbnailUrl` from all books where `pageCount` is equal to 0.

```
JSON
        db.books.updateMany(
          { pageCount: 0 },
          { $unset: { thumbnailUrl: "" } }
        );
```

9. Delete the book with the `title` equal to "Flex 4 in Action".

```
JSON
        db.books.deleteOne({ title: "Flex 4 in Action" });
```

10. Delete all books that include "Internet" in their `categories` array.

```
JSON
        db.books.deleteMany({ categories: "Internet" });
```

Section 2: Operators & Regex

11. Find and return the number of books where `pageCount` is greater than or equal to 400 and less than or equal to 600, and `status` is "PUBLISH".

```
JSON
        db.books.countDocuments(
          {
            pageCount: { $gte: 400, $lte: 600 },
            status: "PUBLISH"
          });
```

12. Return the `title` and `categories` of all books where `categories` include either "Java" or "Mobile".

```
JSON
        db.books.find(
          { categories: { $in: ["Java", "Mobile"] } },
          { title: 1, categories: 1, _id: 0 }
        );
```

13. Using a regular expression, find all books whose title starts with the word "Android" (case-sensitive).

```json
db.books.find(
  { title: { $regex: /^Android/ } }
);
```

14. Return the title and publishedDate of all books, sorted by publishedDate in descending order.

```json
db.books.find(
  {},
  { title: 1, publishedDate: 1, _id: 0 }
).sort({ publishedDate: -1 });
```

15. Find the number of books that either have a pageCount greater than 500 **or** a publishedDate after January 1, 2010, and must also include the category "Software Engineering".

```json
db.books.countDocuments({
  $and: [
    {
      $or: [
        { pageCount: { $gt: 500 } },
        { publishedDate: { $gt: new Date("2010-01-01") } }
      ]
    },
    { categories: "Software Engineering" }
  ]});
```

Section 3: Aggregation Pipeline

16. Using an aggregation pipeline, group the books by `status` and count how many books belong to each status value.

```JSON
db.books.aggregate([
  { $group: { _id: "$status", count: { $sum: 1 } } }
]);
```

17. Calculate the average `pageCount` across all books using an aggregation pipeline.

```JSON
db.books.aggregate([
  { $group: { _id: null, avgPages: { $avg: "$pageCount" } } }
]);
```

18. Group all books by each unique category (assume a book can belong to multiple categories) and calculate the maximum `pageCount` for each category. To solve this question, read about [`$unwind` (aggregation)]() here.

```JSON
db.books.aggregate([
  { $unwind: "$categories" },
  { $group: { _id: "$categories", maxPages: { $max: "$pageCount" } } }
]);
```

19. First filter books that have a `publishedDate` after January 1, 2010, and then group them by `status`, returning the count of books per status.

```
JSON
      db.books.aggregate([
        { $match: { publishedDate: { $gt: new Date("2010-01-01") } } },
        { $group: { _id: "$status", count: { $sum: 1 } } }
      ]);
```

20. Group all books by `status` and count how many books exist in each group. Then, filter the groups to return only those where the count is greater than 2.

```
JSON
      db.books.aggregate([
        { $group: { _id: "$status", count: { $sum: 1 } } },
        { $match: { count: { $gt: 2 } } }
      ]);
```