CRUD Operations in MongoDB

Creating New Documents

a) Insert a single document using `insertOne()`:

```javascript
db.collectionName.insertOne({ field1: value1, field2: value2, ... });
```

Example:

```javascript
db.products.insertOne({ name: "Product 1", price: 10.99, category: "Electronics" });
```

b) Insert multiple documents using `insertMany()`:

```javascript
db.collectionName.insertMany([{ field1: value1, field2: value2 }, { field1: value3, field2: value4 }, ... ]);
```

Example:

```javascript
db.products.insertMany([
{ name: "Product 2", price: 19.99, category: "Electronics" },
{ name: "Product 3", price: 5.99, category: "Clothing" },
{ name: "Product 4", price: 7.99, category: "Clothing" }
]);
```

Retrieving and Finding Documents

a) Basic querying using `find()`:

- Retrieve all documents in a collection:

```javascript
db.collectionName.find();
```

- Retrieve documents based on a specific condition:

```javascript
db.collectionName.find({ field: value });
```
Example:
```javascript
db.products.find({ category: "Electronics" });
```

b) Querying with comparison operators:
- Retrieve documents using comparison operators like `$gt`, `$lt`, etc.
```javascript
db.collectionName.find({ field: { $gt: value } });
```
Example:
```javascript
db.products.find({ price: { $gt: 10 } });
```

c) Querying with logical operators:
- Combine multiple conditions using logical operators like `$and`, `$or`, etc.
```javascript
db.collectionName.find({ $and: [{ condition1 }, { condition2 }] });
```
Example:
```javascript
db.products.find({ $and: [{ price: { $gt: 5 } }, { category: "Electronics" }] });
```

d) Querying arrays using array operators:
- Use array operators like `$in`, `$nin`, etc., to query array fields.
```javascript
db.collectionName.find({ field: { $in: [value1, value2] } });
```
Example:
```javascript
db.products.find({ category: { $in: ["Electronics", "Clothing"] } });
```

e) Projections:
- Specify which fields to include or exclude in the query result.

```javascript
db.collectionName.find({}, { field1: 1, field2: 1 });
```

Example:

```javascript
db.products.find({}, { name: 1, price: 1 });
```

Sorting and Limiting Data in Query Results

a) Sorting results:

- Sort documents based on a specific field in ascending order (`1`) or descending order (`-1`).

```javascript
db.collectionName.find().sort({ field: 1 });
```

Example:

```javascript
db.products.find().sort({ price: 1 });
```

b) Limiting results:

- Limit the number of returned documents.

```javascript
db.collectionName.find().limit(limitValue);
```

Example:

```javascript
db.products.find().limit(5);
```

c) Combining sorting and limiting:

- Retrieve a specific number of documents in a sorted order.

```javascript
db.collectionName.find().sort({ field: 1 }).limit(limitValue);
```

Example:

```javascript
db.products.find().sort({ price: -1 }).limit(5);
```

Working with Nested Documents and Arrays

a) Accessing nested documents:

- Access fields within nested documents using dot notation.

```javascript
documentName.nestedDocument.field;
```

Example:

```javascript
db.orders.find({ "customer.name": "John Doe" });
```

b) Manipulating nested documents:

- Update fields within nested documents using dot notation and `$set`.

```javascript
db.collectionName.updateOne({ _id: ObjectId("documentId") }, { $set: {
"nestedDocument.field": value } });
```

Example:

```javascript
db.orders.updateOne({ _id: ObjectId("6154e4cbe8d52b02409ae048") }, { $set: {
"customer.address": "456 Elm St" } });
```

c) Accessing and manipulating arrays:

- Access elements within an array using the array index.

```javascript
documentName.arrayName[index];
```

Example:

```javascript
db.products.find({ "reviews.0.rating": { $gt: 4 } });
```

d) Combining nested documents and arrays:

- Access and manipulate nested documents within arrays using dot notation and array indexes.

```javascript
documentName.arrayName[index].nestedDocument.field;
```

Example:

```javascript
db.orders.find({ "orderItems.0.productId": ObjectId("6154e510e8d52b02409ae049") });
```

Updating Documents Using Update Operators

a) $set operator:

- Update specific fields or add new fields within a document.

```javascript
db.collectionName.updateOne({ _id: ObjectId("documentId") }, { $set: { field: value } });
```

Example:

```javascript
db.products.updateOne({ _id: ObjectId("6154e4cbe8d52b02409ae048") }, { $set: { price: 11.99 }
});
```

b) $unset operator:

- Remove specified fields from a document.

```javascript
db.collectionName.updateOne({ _id: ObjectId("documentId") }, { $unset: { field: "" } });
```

Example:

```javascript
db.products.updateOne({ _id: ObjectId("6154e4cbe8d52b02409ae048") }, { $unset: { price: "" }
});
```

c) $inc operator:

- Increment or decrement the value of a numeric field.

```javascript
db.collectionName.updateOne({ _id: ObjectId("documentId") }, { $inc: { field: incrementValue
} });
```

Example:

```javascript
db.products.updateOne({ _id: ObjectId("6154e4cbe8d52b02409ae048") }, { $inc: { quantity: 1 }
});
```

d) $push operator:
- Append elements to an array field within a document.
```javascript
db.collectionName.updateOne({ _id: ObjectId("documentId") }, { $push: { arrayField: element } });
```
Example:
```javascript
db.products.updateOne({ _id: ObjectId("6154e4cbe8d52b02409ae048") }, { $push: { tags: "new" } });
```

e) $pull operator:
- Remove elements from an array field that match a specified condition.
```javascript
db.collectionName.updateOne({ _id: ObjectId("documentId") }, { $pull: { arrayField: { condition } } });
```
Example:
```javascript
db.products.updateOne({ _id: ObjectId("6154e4cbe8d52b02409ae048") }, { $pull: { tags: "old" } });
```

Deleting Documents from a Collection

a) deleteOne():
- Remove a single document based on the filter criteria.
```javascript
db.collectionName.deleteOne({ field: value });
```
Example:
```javascript
db.products.deleteOne({ name: "Product 1" });
```

b) deleteMany():
- Remove multiple documents based on the filter criteria.

```javascript
db.collectionName.deleteMany({ field: value });
```

Example:

```javascript
db.products.deleteMany({ category: "Electronics" });
```

Ensure to adapt the collection name, field names, values, and filter criteria as per your specific MongoDB setup and data requirements. These examples serve as a starting point to understand the concepts and perform the mentioned operations in MongoDB.