CRUD operations in MongoDB

CRUD operations in MongoDB refer to the basic database actions: **Create**, **Read**, **Update**, and **Delete**. Below is a detailed explanation with examples for each operation.

1. Create (Insert) Operation

This involves inserting documents (records) into a MongoDB collection.

Example:

```
// Insert a single document into a collection called "users"
db.users.insertOne({
    name: "Alice",
    age: 25,
    email: "alice@example.com"
});

// Insert multiple documents into a collection
db.users.insertMany([
    { name: "Bob", age: 30, email: "bob@example.com" },
    { name: "Charlie", age: 35, email: "charlie@example.com" }
]);
```

2. Read (Find) Operation

This operation is used to retrieve data from a collection. MongoDB's `find` function is used to perform a read operation.

Example:

```
// Find all documents in the "users" collection
db.users.find();

// Find documents with specific criteria (e.g., age > 30)
db.users.find({ age: { $gt: 30 } });

// Find a single document
db.users.findOne({ name: "Alice" });

// Find documents with a projection (only select specific fields)
db.users.find({}, { name: 1, email: 1, _id: 0 });
```

3. Update Operation

The `update` operation allows you to modify existing documents in a collection. Common update functions include `updateOne`, `updateMany`, and `replaceOne`.

Example:

```
// Update a single document
db.users.updateOne(
    { name: "Alice" },             // Filter
    { $set: { age: 26 } }          // Update action
);

// Update multiple documents
db.users.updateMany(
    { age: { $gt: 30 } },          // Filter
    { $set: { emailVerified: true } }  // Update action
);
```

```
// Replace a single document
db.users.replaceOne(
    { name: "Bob" },               // Filter
    { name: "Bob", age: 31, email: "bob_new@example.com" }  // New document
);
```

4. Delete Operation

The `delete` operation allows you to remove documents from a collection. Common functions are `deleteOne` and `deleteMany`.

Example:

```
// Delete a single document
db.users.deleteOne({ name: "Charlie" });

// Delete multiple documents
db.users.deleteMany({ age: { $lt: 30 } });
```

Complete Example in JavaScript using Node.js and MongoDB

```
const { MongoClient } = require('mongodb');

async function main() {
  const uri = "mongodb+srv://<username>:<password>@cluster.mongodb.net/myFirstDatabase";
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const database = client.db("testdb");
    const users = database.collection("users");
    // CREATE Operation
    await users.insertOne({ name: "David", age: 22, email: "david@example.com" });
    await users.insertMany([
      { name: "Eva", age: 28, email: "eva@example.com" },
      { name: "Frank", age: 32, email: "frank@example.com" }
    ]);

    // READ Operation
    const allUsers = await users.find().toArray();
    console.log("All Users:", allUsers);

    const user = await users.findOne({ name: "David" });
    console.log("User Found:", user);

    // UPDATE Operation
    await users.updateOne(
      { name: "David" },
      { $set: { age: 23 } }
    );

    // DELETE Operation
    await users.deleteOne({ name: "Eva" });

  } finally {
    await client.close();
  }
}

main().catch(console.error);
```

Key Points:

- **Inserting Documents**: `insertOne()` and `insertMany()`.
- **Reading Documents**: `find()` (for multiple), `findOne()` (for single).
- **Updating Documents**: `updateOne()`, `updateMany()`, and `replaceOne()`.
- **Deleting Documents**: `deleteOne()` and `deleteMany()`.

This covers all the basic CRUD operations with MongoDB and how to use them in practice.

MongoDB query operators and projection operators

In MongoDB, **query operators** and **projection operators** are powerful tools for refining the data you search for and the fields you display. Here's a breakdown of each:

1. Query Operators

Query operators are used in MongoDB to filter the documents based on specific conditions. They allow for more complex queries beyond exact matches. The most common query operators are:

Comparison Operators

- **`$eq`**: Matches values that are equal to a specified value.
- **`$gt`**: Matches values greater than a specified value.
- **`$gte`**: Matches values greater than or equal to a specified value.
- **`$lt`**: Matches values less than a specified value.
- **`$lte`**: Matches values less than or equal to a specified value.
- **`$ne`**: Matches values that are not equal to a specified value.
- **`$in`**: Matches any of the values specified in an array.
- **`$nin`**: Matches none of the values specified in an array.

Example:

```
// Find users whose age is greater than 25
db.users.find({ age: { $gt: 25 } });

// Find users whose name is either "Alice" or "Bob"
db.users.find({ name: { $in: ["Alice", "Bob"] } });
```

Logical Operators

- `$and` : Joins query clauses with a logical AND.
- `$or` : Joins query clauses with a logical OR.
- `$not` : Inverts the effect of a query expression.
- `$nor` : Joins query clauses with a logical NOR.

Example:

```
// Find users whose age is greater than 25 AND emailVerified is true
db.users.find({ $and: [{ age: { $gt: 25 } }, { emailVerified: true }] });

// Find users whose age is less than 30 OR email contains "example.com"
db.users.find({ $or: [{ age: { $lt: 30 } }, { email: /example\.com/ }] });
```

Element Operators

- `$exists` : Checks if a field exists or not.
- `$type` : Selects documents where the field is of a specific type.

Example:

```
// Find users who have the field "email"
db.users.find({ email: { $exists: true } });

// Find users where "age" is a number
db.users.find({ age: { $type: "number" } });
```

Evaluation Operators

- **`$regex`** : Allows for matching strings using regular expressions.
- **`$expr`** : Allows for the use of aggregation expressions within the query.
- **`$text`** : Performs text searches on text-indexed fields.
- **`$where`** : Runs a JavaScript expression on each document.

Example:

```
// Find users whose name starts with "A"
db.users.find({ name: { $regex: /^A/ } });

// Find users using a JavaScript expression to check for age > 25
db.users.find({ $where: "this.age > 25" });
```

2. Projection Operators

Projection operators in MongoDB are used to include or exclude specific fields when retrieving documents. They allow you to focus only on the fields you need.

Basic Projections

- **`1`** : Include the field.
- **`0`** : Exclude the field.

Example:

```
// Include only "name" and "email" fields (exclude "_id" by setting _id to 0)
db.users.find({}, { name: 1, email: 1, _id: 0 });
```

Projection with Conditional Operators

- **`$elemMatch`**: Projects the first matching element in an array based on a query condition.
- **`$slice`**: Limits the number of items in an array displayed in the results.
- **`$meta`**: Projects metadata associated with the document (often used with text searches).

Example:

```
// Project only the first matching element in the "addresses" array where "city" is "New York"
db.users.find({}, { addresses: { $elemMatch: { city: "New York" } } });

// Project the first 2 items from the "addresses" array
db.users.find({}, { addresses: { $slice: 2 } });
```

Use Case: Combining Query and Projection

You can combine both query and projection operators to create precise, refined queries.

Example:

```
// Find users with age greater than 25 and display only "name" and "email" fields
db.users.find(
  { age: { $gt: 25 } },  // Query
  { name: 1, email: 1, _id: 0 }  // Projection
);
```

```
// Find users with "Alice" or "Bob" in their name and display their name and age only
db.users.find(
  { name: { $in: ["Alice", "Bob"] } },
  { name: 1, age: 1, _id: 0 }
);
```

Summary of Key Query Operators

Operator	Description	Example
`$eq`	Equal to	`{ age: { $eq: 25 } }`
`$gt`	Greater than	`{ age: { $gt: 25 } }`
`$lt`	Less than	`{ age: { $lt: 25 } }`
`$and`	Logical AND	`{ $and: [{ age: { $gt: 25 } }, { email: /example/ }] }`
`$or`	Logical OR	`{ $or: [{ age: { $gt: 25 } }, { email: /example/ }] }`
`$exists`	Field exists	`{ email: { $exists: true } }`
`$regex`	Regular expression matching	`{ name: { $regex: /^A/ } }`

Summary of Key Projection Operators

Operator	Description	Example
`1` / `0`	Include/Exclude specific fields	`{ name: 1, age: 1, _id: 0 }`
`$elemMatch`	Project first matching element in an array	`{ addresses: { $elemMatch: { city: "New York" } } }`
`$slice`	Project a subset of an array	`{ addresses: { $slice: 2 } }`
`$meta`	Project metadata (e.g., text score in searches)	`{ score: { $meta: "textScore" } }`

This covers the basics of **query** and **projection operators** in MongoDB. By combining these, you can fine-tune your data searches and retrievals.