MongoDB Connection

```
const { MongoClient } = require('mongodb');
const uri = 'mongodb://localhost:27017/?
appName=MongoDB+Compass&directConnection=true&serverSelectionTimeoutMS=2000';

let db;

MongoClient.connect(uri)
  .then((client) => {
    db = client.db('bookstore'); // Connect to the 'bookstore' database
    return db.collection('books'); // Access the 'books' collection
  })
  .then((booksCollection) => {
    return booksCollection.find().toArray(); // Retrieve all documents as an array
  })
  .then((books) => {
    console.log('Books:', books);
    db.client.close() // Print the retrieved documents
  })
  .catch((err) => {
    console.error('Error:', err);
  });
```

The `.find()` method in MongoDB is used to query documents from a collection. It supports various parameters to define the filter criteria, projection, and options for the query. Below is a breakdown of its parameters:

Syntax

```
db.collection('collectionName').find(query, projection)
```

- `query` (optional): Specifies the filter criteria to match documents.
- `projection` (optional): Specifies which fields to include or exclude in the result.

Parameters in Detail

1. Query Parameter (`query`)

Defines the criteria for matching documents. It is an object where you can specify conditions.

Examples:

- Match documents with specific fields:

  ```
  { field: value }
  ```

  Example:

  ```
  { age: 25 }
  ```

  Matches documents where the `age` is `25`.
- Use comparison operators:

  ```
  { field: { $operator: value } }
  ```

  Example:

  ```
  { age: { $gt: 25 } }
  ```

  Matches documents where `age` is greater than `25`.
- Logical operators:

  ```
  { $and: [{ condition1 }, { condition2 }] }
  ```

  Example:

  ```
  { $or: [{ age: { $lt: 18 } }, { age: { $gt: 60 } }] }
  ```

  Matches documents where `age` is less than `18` or greater than `60`.
- Match arrays or subdocuments: Example:

  ```
  { tags: { $in: ["nodejs", "mongodb"] } }
  ```

  Matches documents where the `tags` array contains `"nodejs"` or `"mongodb"`.

---

2. Projection Parameter (`projection`)

Defines which fields to include or exclude in the result. It is an object where you specify field names and their inclusion or exclusion.

Examples:

- Include specific fields:

  ```
  { field1: 1, field2: 1 }
  ```

Example:

```
{ name: 1, age: 1 }
```

Includes only `name` and `age` in the result.

- Exclude specific fields:

```
{ field: 0 }
```

Example:

```
{ password: 0 }
```

Excludes the `password` field from the result.

> **Note**: You cannot mix inclusion and exclusion in the same projection (except for the `_id` field).

---

3. Options (using `.find()` and `.find().toArray()`)

In addition to `query` and `projection`, the `.find()` method can be combined with options like `.limit()`, `.skip()`, and `.sort()`.

- `limit`: Restricts the number of documents returned.

```
db.collection('books').find({}, { title: 1 }).limit(5)
```

Returns only the first 5 documents with the `title` field.

- `skip`: Skips a number of documents.

```
db.collection('books').find({}, { title: 1 }).skip(5)
```

Skips the first 5 documents and returns the rest.

- `sort`: Sorts documents by fields.

```
db.collection('books').find({}, { title: 1 }).sort({ title: 1 })
```

Sorts the documents by `title` in ascending order. Use `-1` for descending order.

Examples of `.find()` Usage

1. Query All Documents

```
db.collection('books').find()
```

2. Query with Filter

Find books with the author "JK Rowling":

```
db.collection('books').find({ author: "JK Rowling" })
```

3. Query with Projection

Find books but only return the `title` and `author` fields:

```
db.collection('books').find({}, { title: 1, author: 1, _id: 0 })
```

4. Query with Conditions

Find books published after 2000:

```
db.collection('books').find({ year: { $gt: 2000 } })
```

5. Query with Sort and Limit

Find the first 3 books sorted by `title` in descending order:

```
db.collection('books').find().sort({ title: -1 }).limit(3)
```

Notes on `find()` vs `findOne()`

- `.find()` returns a **cursor** that you can iterate over, and you can call `.toArray()` to get an array of results.
- `.findOne()` directly returns the first matching document.

Example:

```
db.collection('books').findOne({ title: "Harry Potter" })
```

Returns:

```
{ _id: ObjectId("..."), title: "Harry Potter", author: "JK Rowling", ... }
```

Event Emitter (Custom Events)

```
const EventEmitter = require('events');

class MyEmitter extends EventEmitter {}

const myEmitter = new MyEmitter();

myEmitter.on('customEvent', () => {
  console.log('Custom event emitted!');
});

myEmitter.emit('customEvent');
```

```
var events = require('events')

em = new events.EventEmitter()

em.addListener('Event 1', (data)=>{
        console.log("1"*10+"\n"+data)
})

em.on('Event 2', (data)=>{
        console.log("2"*10+"\n"+data)
})

em.emit('Event 1', "this is event 1 data")
em.emit('Event 2', 'this is event 2 data')
```

Method	Description	Syntax	Returns	Exampl
`addListener()`	Adds a listener function for the specified event. Equivalent to `on()`.	`addListener(event, listener)`	The `EventEmitter` instance (for chaining).	`emitter console`
`on()`	Registers a listener for the specified event.	`on(event, listener)`	The `EventEmitter` instance (for chaining).	`emitter receive`
`once()`	Adds a one-time listener for the specified event. The listener is removed	`once(event, listener)`	The `EventEmitter` instance (for chaining).	`emitter data or`

Method	Description	Syntax	Returns	Exampl
	after the first call.			
`removeListener()`	Removes a specific listener from the specified event.	`removeListener(event, listener)`	The `EventEmitter` instance (for chaining).	`emitter`
`removeAllListeners()`	Removes all listeners for a specific event, or all events if none specified.	`removeAllListeners([event])`	The `EventEmitter` instance (for chaining).	`emitter`
`setMaxListeners()`	Sets the maximum number of listeners allowed for a single event.	`setMaxListeners(n)`	The `EventEmitter` instance (for chaining).	`emitter`
`listeners()`	Returns an array of registered listeners for a specific event.	`listeners(event)`	Array of listener functions for the specified event.	`console`
`emit()`	Triggers an event, executing all registered listeners.	`emit(event, [arg1], [arg2], [...])`	`true` if the event had listeners, `false` otherwise.	`emitter`
`listenerCount()`	Returns the number of listeners for a given event.	`listenerCount(emitter, eventName)`	Number of registered listeners for the event.	`console 'data'`

Event	Description	Syntax	Example
`newListener`	Triggered when a listener is added.	Emitted: `(event, listener)`	`emitter.on('newListener', (event, listener) => console.log('Listener added:', event));`
`removeListener`	Triggered when a listener is	Emitted: `(event,`	`emitter.on('removeListener', (event, listener) => console.log('Listener`

Event	Description	Syntax	Example
	removed.	`listener)`	`removed:', event));`

React Router

```
// styles.js
import styled from ‘styled-components’;
import {NavLink} from ‘react-router-dom’;

export const StyledLink = styled(NavLink)`
        margin-right: 5px;
`;
export default StyledLink;

// App.js
import {BrowserRouter as Router, Route, Routes, Link} from ‘react-router-dom’;
import {StyledLink} from ‘./styles.js’;

<Router>
        <nav>
                <StyledLink to=“/”>Home</StyledLink>
                <Link to=“/contact”>Contact</Link>
                <Link to=“/about”>About Us</Link>
        </nav>
  <Routes>
    <Route path="/" element={<Home/>}/>
    <Route path="/about" element={<About/>} />
    <Route path="/contact" element={<Contact/>} />
  </Routes>
</Router>

// index.js
import App from './App'

let root = ReactDOM.createRoot(document.querySelector("#root"));

root.render(
        <App/>
)
```