**Database- Day -4: MongoDB:**

**"basic cursor methods -**

**map, toArray, pretty, forEach, limit, count, sort**

**Aggregation**

**Server-side vs Client-side rendering"**

**Let's assume we have an employees collection with the following fields for each document:**

db.employees.insertMany([

{ employeeId: 1, firstName: "sathvik", lastName: "suresh", age: 08, department: "HR" },

{ employeeId: 2, firstName: "rithik", lastName: "suresh", age: 02, department: "Marketing" },

{ employeeId: 3, firstName: "renu", lastName: "krishnan", age: 33, department: "Finance" },

{ employeeId: 4, firstName: "siva", lastName: "kumar", age: 25, department: "IT" },

{ employeeId: 5, firstName: "rajesh", lastName: "vel", age: 32, department: "Sales" },

]);

**find():**

which **retrieves documents** from a collection.

Eg:

db.employees.find();

### **toArray()**

Converts the cursor to an **array of documents. [{}]**

**Eg:**

db.employees.find().toArray()

### **forEach()**

**Iterates over documents** and applies a function to each one.

**forEach((emp)=>{**

**print(`empname: ${employees.firstName} `)**

**})**

**Eg:**

db.employees.find().forEach((employee)=>{print(`employee name: ${employee.firstName}`)});

### **map()**

Applies a function to each document and **returns an array** of the return values.

**Eg:**

let department = db.employees.find().map((employee)=>employee.department);

print(department);

### **limit()**

Limits the **number of documents returned**.

**Eg:**

db.employees.find().sort({age:-1}).limit(2);

db.employees.find().sort({age:-1}).limit(2).skip(2);

### **countDocuments() -**

Counts the **number of documents that match the query.**

**Eg:**

db.employees.countDocuments();

db.employees.countDocuments({age:{$gt:10}});

### **sort()**

Sorts the documents. Use **1 for ascending order or -1 for descending** order.

**Eg:**

db.employees.find().sort({firstName:1});

db.employees.find().sort({firstName:-1});

db.employees.find().sort(); // default ascending

### **pretty()**

**Improves the readability** of the query output (nothing different in the shell).

**Eg:**

db.employees.find().sort({age:-1}).limit(2).skip(2).pretty()

**Aggregation:**

**Aggregation & Projection:(MongoDB's aggregation framework)**

It allows you to perform **complex data operations**, such as filtering, grouping, sorting, and calculating aggregations, on your documents.

**aggregation stages/pipelines :**

· $match Stage: Filters documents

· $group Stage: Groups documents by a specified key

· $sort Stage: Sorts the documents

· $limit Stage: Limits the number of documents

· $project Stage: Shapes the output of the aggregation

. $lookup stage : joins

Syntax:

db.products.aggregate([{},{},{},{}]);

db.products.aggregate([{$match:},{$group:},{$sort:},{$limit:}]);

**Eg: Users, Tasks (collections & Documents)**

db.users.insertMany([

{ name: "siva", age: 25, email: "siva@gmail.com" },

{ name: "guru", age: 30, email: "guru@gmail.com" },

{ name: "rajesh", age: 22, email: "rajesh@gmail.com"},

{ name: "vel", age: 32, email: "vel@gmail.com" },

]);

db.tasks.insertMany([

{ description: "Day 21 Task", status: "In Progress", user\_id: ObjectId('65e0f8e87a46536ff53de526') },

{ description: "Day 22 Task", status: "Pending", user\_id: ObjectId("65e0f8e87a46536ff53de527") },

{ description: "Day 23 Task", status: "In Progress", user\_id: ObjectId('65e0f8e87a46536ff53de526') },

{ description: "Day 24 Task", status: "Pending", user\_id: ObjectId("65e0f8e87a46536ff53de526") },

{ description: "Day 25 Task", status: "In Progress", user\_id: ObjectId('65e100607a46536ff53de52e') },

{ description: "Day 26 Task", status: "Pending", user\_id: ObjectId("65e100607a46536ff53de52f") },

]);

**//$match - filter**

db.users.aggregate([

{

$match:{age:{$gt:25}}

}

]);

**//$group - same value that contain in a rows**

db.tasks.aggregate([

{

$group:{\_id:'$user\_id', total\_task:{$sum:1}}

}

]);

{$sum:1} -> count + 1

**//$sort (1, -1)**

db.tasks.aggregate([

{ $group:{\_id:'$user\_id', total\_task:{$sum:1}} },

{ $sort:{total\_task:-1} }

]);

**//$limit : pagination**

db.tasks.aggregate([

{ $group:{\_id:'$user\_id', total\_task:{$sum:1}} },

{ $sort:{total\_task:-1} },

{$limit:2}

]);

**//$Skip : (offset)**

db.tasks.aggregate([

{ $group:{\_id:'$user\_id', total\_task:{$sum:1}} },

{ $sort:{total\_task:-1} },

{$skip:2} , // **should write skip first**

{$limit:2}

]);

**//$project (0,1) - exclusion & inclusion**

db.tasks.aggregate([

{ $group:{\_id:'$user\_id', total\_task:{$sum:1}} },

{ $sort:{total\_task:-1} },

{$project:{\_id:0, total\_task:1}}

]);

**//lookUp - join**

db.tasks.aggregate([

{

$lookup: {

from: "users", // Reference the 'users' collection

localField: "user\_id", // Field in the 'tasks' collection

foreignField: "\_id", // Field in the 'users' collection

as: "userDetails" // Field that will contain the output array of the join

}

}

]);

**Eg: 2**

db.tasks.aggregate([

{$lookup:{

from:'users',

localField:'user\_id',

foreignField:'\_id',

as:'user\_details'

}},

{$unwind:'$user\_details'},

{$project:{

\_id:0,

description:1,

status:1,

user\_name:'$user\_details.name',

user\_age:'$user\_details.age',

user\_email:'$user\_details.email'

}}

]);

**Note:**

The $unwind stage is **used to deconstruct this array**. If user\_details contains only one document, $unwind turns **each element of user\_details into a separate document**. This makes it easier to work with the fields of the user\_details document in subsequent stages of the pipeline.

**Server-side vs Client-side rendering:**

| **Aspect** | **Server-Side Rendering (SSR)** | **Client-Side Rendering (CSR)** |
| --- | --- | --- |
| What Happens | The server generates HTML pages with content and sends them to the browser. | The browser loads a minimal HTML page and uses JavaScript to fetch and render content dynamically. |
| Initial Load | Fast initial page load as the server provides fully rendered HTML. | Slower initial load; the browser needs time to download JavaScript and render the page. |
| SEO | Good for SEO as search engines easily index content in the provided HTML. | Requires additional measures for SEO, as search engines may not process JavaScript as effectively. |
| Subsequent Loads | May be slower as each page request involves a round trip to the server. | Faster, as the browser can fetch and update content without full page reloads. |
| User Experience | Immediate display of content; users see a fully formed page quickly. | Initial delay, but smoother interactions once the initial page is loaded. |
| Development | Typically simpler with less emphasis on complex JavaScript. | More complexity in handling client-side rendering logic and state management. |