|  |  |
| --- | --- |
| -  Sindy Saintclair  Tuesday, January 25, 2022  Lesson 2 – Querying Documents | |
| **Learning Objectives and Questions** | **Notes and Answers** |
| **OVERVIEW** | In the last lesson, you spent a small amount of time finding documents after you have inserted them. Now you will explore more about finding documents. |
| **SYNTAX** | The syntax for if you want to find an entire collection without any parameters is find(), as shown below:  db.collectionName.find({})  GO ahead and find all documents within your previously created collection appusers:  db.appusers.find({})  Depending on how many users you have created, you will see the entire document for each user displayed. MongoDB uses a cursor to retrieve the documents for you, which is simply a pointer to the result set. If you have under 20 documents, then you will see the entire set. If your collection exceeds 20 documents, then the first 20 will be shown, and you will need to type it to continue iterating to the next 20 documents. In large collections of millions of documents, you can see how this would be impossible to find the documents you’re look for – and that is where the query operators come in.  When you *don’t* want to return every document within a collection, you can use the syntax below:  db.collectionName.find({query, projection})  The syntax above will select documents and return a cursor to the selected documents. Before continuing it is important you understand a few definitions:   * **Query**: specifies a selection filter using query operators and is optional * **Projection**:specifies certain fields to return in the documents that match the query filter and is optional. To return all fields in the matching documents, omit this parameter * **Cursor**: A pointer to the result set of a query. This is the output of whatever documents you are locating and from which you are selecting.   Great work! Now you will explore the query operators and projections you can add when finding documents. |
| **Query Operators** | To limit your document result set, you will need to use some query operators. Going through all the operators and the different ways you can use them would be another course in itself, so you’ll focus on some fundamental ones here. You will explore different Comparison Query Operators, Logical Query Operators, and Element Query Operators. Make sure you are connected to the appusers collection in your database and that you are running every query explained in this lesson.  ***Comparison Query Operators***  Comparison Operators compare documents against the specified value.  $eq – this query operator will query documents that are equal to the specified value.  db.appusers.find({ \_id : { $eq : 1} })  Above, the query is finding and selecting a document where the \_id field is equal to 1. First, the collection appusers was specified, then the query find(), the field desired, which is the \_idfield, and finally the query operator, which is looking to match a document with an \_id of 1. When using the query operators, they need to go within another document notation which is specified by curly brackets {}.This query could be read as, “find me the document in the collection called appusers that has the unique ObjectId that is equal to the number 1”. If typed correctly, the entire document for the user “Georgina” should be returned.  $ne – matches all documents that are not equal to the specified value.  db.appusers.find({ \_id : { $ne : 1} })  To contradict $eq, the above will return all documents that do NOT have an \_id of 1.  $gt – matches all documents that are *greater than* the specified value.  db.appusers.find({ \_id : { $gt : 1} })  $gte – matches all documents that are *greater than* or *equal* to the specified value.  db.appusers.find({ \_id : { $gte : 2} })  $lt – matches all documents that are *less than* the specified value  db.appusers.find({ \_id : { $lt : 2} })  $lte – matches all documents that are *less than* or *equal* to the specified value  db.appusers.find({ \_id : { $lte : 2} })  ***New Collection***  As you continue learning about the query operators available in MongoDB, you need a bit more info in your collection. Run the following query:  db.inventory.insertMany([  { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A", price: 16.49, sale: true },  { item: "spiral notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A", price: 4.79, sale: false },  { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D", price: 6.99, sale: true },  { item: "day planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D", price: 25.29, sale: true },  { item: "calendar", size: { h: 21.75, w: 17, uom: "in" }, status: "B", price: 5.99, sale: false },  { item: "scissor", qty: 15, size: { h: 8, w: 4.25, uom: "in" }, status: "B", price: 11.59, sale: true },  { item: "tape dispenser", qty: 5, size: { h: 3, w: 7, uom: "in" }, status: "A", price: 3.09, sale: true }  ]);  The above query will create an inventory collection since it does not already exist and will insert seven inventory items.  $in – matches *any* of the values within an array.  db.inventory.find( { qty: { $in: [ 5, 15 ] } } )  The above query will find all documents within an inventory collection where the quantity field (qty) has the value of either 5 or 15.  $nin – matches *none*  of the values within a specified array.  db.inventory.find( { qty: { $nin: [ 5, 15 ] } } )  The above query will locate all documents in the inventory collection where the qty field value does *not* equal 5 or 15. |
| **Logical Operators** | **$and** – performs a logical AND operation on an array of two or more expressions and selects the documents that satisfy all expressions in the array.  db.inventory.find( { $and: [ { price: { $ne: 6.99 } }, { price: { $exists: true } } ] } )  The query above finds all records with a **price** (exists), and the **price** is *not equal* to 6.99.  **$or** – the $or operator performs a logical OR operation on an array of two or more expressions and selects the documents that satisfy at least one of the expressions.  db.inventory.find( { $or: [ { qty: { $lt: 20 } }, { price: 16.49 } ] } )  The above query will locate all documents in the inventory collection where the qty field value is less than 20, or the price field value equal 16.49.  You can combine $and and $or to do something like below:  db.inventory.find( {  $and : [  { $or : [ { price : { $eq : 4.79 } }, { price : { $eq : 3.09 } } ] },  { $or : [ { sale : true }, { qty : { $lt : 20 } } ] }  ]  } )  The query above will return those records with:   * price equal to either 4.79 OR 3.09   AND   * sale is true OR the qty is less than 20.   $not– performs a logical NOT operation on the specified operator-expression. It selects the documents that do not match the operator-expression; this includes documents that do not contain the field.  db.inventory.find( { price: { $not: { $gt: 4.80 } } } )  The above query will locate all documents in the inventory collection with a price field and where the price is not greater than 4.80. So, any item with a price that is 4.80 or less will be returned.  $nor – performs a logical NOR operation on one or more query expressions and selects the documents that fail all the query expressions in the array.  db.inventory.find( { $nor: [ { price: { $gte : 5.99 } }, { sale: { $eq : true } } ] } )  The above query will return all documents that:   * contain the price field whose value is NOT greater than or equal to 5.99 and contain the sale field whose value is NOT equal to true   OR   * contains the price field whose value is NOT greater than or equal to 5.99 but do not contain the sale field   OR   * do NOT contain the price field but contains the sale field with a value that is NOT equal to true   OR   * do not contain the price field and do not contain the sale field. |
| **Element Operators** | $exists – matches documents that contain a certain field, including documents that are null. The syntax is below:  { field: { $exists: <boolean> } }  The value is a Boolean when using $exists, so it can only be true or false.  db.inventory.find( { qty: { $exists: true, $nin: [ 5, 15 ] } } )  The above query finds all documents in the inventory collection where the qty field exists, and its value does not equal 5 or 15.  $type – returns documents where the BSON type of the field matches the BSON type passed to $type.  BSON is a binary serialization format used to store documents and make remote procedure calls in MongoDB. BSON has integers that stand for certain data types. For example, the number 2 defines a type of string. There are also aliases for the BSON types. 2 is a string, and the alias for it is “string”.  The syntax is below:  { field: { $type: <BSON type number> | <String alias> } }  [BSON Types — MongoDB Manual](https://docs.mongodb.com/manual/reference/bson-types/#bson-types) |
| **Array Operators** | $all – this query matches all documents where the value of a field is an array that contains all the specified elements.  { tags: { $all: [ "ssl" , "security" ] } }  The above query is finding all elements int eh tags field that is an array. You have seen something similar to this with $and. The above query is equivalent to:  { $and: [ { tags: "ssl" }, { tags: "security" } ] }  $elemMatch – matches documents that contain an array field with at least one element that matches all specified query criteria. Below, if you have an array of results, you can check the values within that array to return a field with the results that are greater than or equal to 80 and less than 85.  { results: { $elemMatch: { $gte: 80, $lt: 85 } } }  $size – selects documents that have an array with a specified size.  db.collection.find( { field: { $size: 2 } } );  Above will find a specific field within a collection that has a size of 2. That means there are two values within the array. |
| **Project Document** | I’ve done many different queries at this point and introduced quite a few operators that I can mix and combine in various ways to return the data I searched for. But in every case, the queries have always returned the entire document from the collection specified. In some cases, you may want to return only one or a few fields from a document rather than the whole thing. That is where projections come into play.  If you remember, the syntax for the find() query looks like below:  db.collectionName.find({query, projection})  If you do not include projections in your find() operation, you will get the entire document and all its fields as you’ve seen in every example so far. The syntax for projections is as follows, subject to some simple rules:  db.collection.find( { query }, { field: <value>, field : <value>, etc} )  The <value> can be either a 1 or a 0. The 1 indicated that you want to show the field, and the 0 indicates that you do not want to show the field.  When using projections, you cannot have two fields where one includes a field, and the other is excluding fields. For example, if you run the below query:  db.appusers.find( { \_id : 2}, { firstName: 1, lastName: 0 })  You will get an error:  Error: error: {  "ok" : 0,  "errmsg" : "Projection cannot have a mix of inclusion and exclusion.",  "code" : 2,  "codeName" : "BadValue"  }  So, if you wanted to include the firstName only in your query result set and nothing else, it would look like this:  db.appusers.find( { \_id : 25}, { firstName: 1 })  And if you wanted to find only the firstName and lastName fields for all documents and nothing else, you can run the code below:  db.appusers.find({}, {firstName : 1, lastName : 1})  As you can see, the first and last names of all your documents are listed, paired with their ObjectId numbers. If you don’t want to see the \_id but want to see all other fields, you can define that with a projection:  db.appusers.find({}, { \_id : 0})  And there you have it: All fields except for the \_id when documents contain those fields. These are some of the basic querying tools that I will frequently see used in MongoDB databases. |