

Lecture 6: MongoDB Manipulating Data (Intermediate Query)

Big Data Systems Design

In the last lecture

❖ Query Document

- find() method
 - *db.COLLECTION_NAME.find(query, projection)*
- Query (document)
 - Provide ways to locate data within the database
 - Comparison
 - Logical
- Projection (document)
 - Determines which fields are returned in the matching documents

❖ Update Document

- update() method

❖ Delete Document

- remove()

In the last lecture

- ❖ Query operator
 - Comparison operators

Name	Description
<code>\$eq</code>	Matches values that are equal to a specified value.
<code>\$gt</code>	Matches values that are greater than a specified value.
<code>\$gte</code>	Matches values that are greater than or equal to a specified value.
<code>\$in</code>	Matches any of the values specified in an array.
<code>\$lt</code>	Matches values that are less than a specified value.
<code>\$lte</code>	Matches values that are less than or equal to a specified value.
<code>\$ne</code>	Matches all values that are not equal to a specified value.
<code>\$nin</code>	Matches none of the values specified in an array.

In the last lecture

- ❖ Query operator
 - Logical operators

Name	Description
<code>\$and</code>	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
<code>\$not</code>	Inverts the effect of a query expression and returns documents that do <i>not</i> match the query expression.
<code>\$nor</code>	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
<code>\$or</code>	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

Table of Contents

❖ Part 1.

- Intermediate query

❖ Part 2.

- Element Operators

❖ Part 3.

- Regular expressions

Part 1

INTERMEDIATE QUERY

Intermediate query

❖ Query embedded document

```
1  {
2    "address": {
3      "building": "1007",
4      "coord": [ -73.856077, 40.848447 ],
5      "street": "Morris Park Ave",
6      "zipcode": "10462"
7    },
8    "borough": "Bronx",
9    "cuisine": "Bakery",
10   "grades": [
11     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
12     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
13     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
14     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
15     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
16   ],
17   "name": "Morris Park Bake Shop",
18   "restaurant_id": "30075445"
19 }
```

The diagram illustrates a JSON document structure. Three red boxes highlight specific parts of the document:

- A box around the `"coord"` array in the `"address"` object.
- A box around the `"grades"` array.
- A box around the `"name"` field.

Red arrows point from the right towards each of these highlighted sections.

Intermediate query

❖ Query embedded document

```
1 {
2   "address": {
3     "building": "1007",
4     "coord": [ -73.856077, 40.848447 ],
5     "street": "Morris Park Ave",
6     "zipcode": "10462"
7   },
8   "borough": "Bronx",
9   "cuisine": "Bakery",
10  "grades": [
11    { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
12    { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
13    { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
14    { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
15    { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
16  ],
17  "name": "Morris Park Bake Shop",
18  "restaurant_id": "30075445"
19 }
```



Embedded documents

Intermediate query

❖ Query embedded document

- We will use the inventory collection for query embedded document

```
db.inventory.insert([
```

```
{ item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "D" },  
{ item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },  
{ item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },  
{ item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },  
{ item: "paper", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }  
]);
```

```
BulkWriteResult({  
  "nInserted" : 5,  
  ...  
})
```

Intermediate query

❖ Query embedded document

- To specify an equality condition on a field that is an embedded document, use the following syntax:
 - `{ <field>: <value> }`
 - <value> is the document to match
- The following query selects all documents where the field size equals the document
 - `db.inventory.find({ size: { h: 14, w: 21, uom: "cm" } })`

See the result

Intermediate query

❖ Query embedded document

- Equality matches on the whole embedded document require an **EXACT** match of the specified <value> document including the field order
- The following query does not match any documents in the inventory collection

- **db.inventory.find({ size: { w: 21, h: 14, uom: "cm" } })**

See the result

- **db.inventory.find({ size: { h: 14, w: 21} })**

See the result

Intermediate query

❖ Query embedded document

- To specify a query condition on fields, use **dot notation**
 - *("field.nestedField")*
 - *Don't forget quotation*
- All documents where the field uom nested in the size field equals "in"
 - **db.inventory.find({ "size.uom": "in" })**

See the result

- Try to re-write the following query using dot notation
 - **db.inventory.find({ size: { h: 14, w: 21} })**

- Result
 - **db.inventory.find({ "size.h": 14, "size.w": 21})**

See the result

Intermediate query

❖ Query an array

```
1 {
2   "address": {
3     "building": "1007",
4     "coord": [ -73.856077, 40.848447 ],
5     "street": "Morris Park Ave",
6     "zipcode": "10462"
7   },
8   "borough": "Bronx",
9   "cuisine": "Bakery",
10  "grades": [
11    { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
12    { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
13    { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
14    { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
15    { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
16  ],
17  "name": "Morris Park Bake Shop",
18  "restaurant_id": "30075445"
19 }
```



Array of Values

Array of Documents

Embedded documents

Intermediate query

❖ Query an array

- We will use the inventory collection for querying an array

- `db.inventory2.insert([`
 `{ item: "journal", qty: 25, tags: ["blank", "red"], dim_cm: [14, 21] },`
 `{ item: "notebook", qty: 50, tags: ["red", "blank"], dim_cm: [14, 21] },`
 `{ item: "paper", qty: 100, tags: ["red", "blank", "plain"], dim_cm: [14, 21] },`
 `{ item: "planner", qty: 75, tags: ["blank", "red"], dim_cm: [22.85, 30] },`
 `{ item: "postcard", qty: 45, tags: ["blue"], dim_cm: [10, 15.25] }`
 `]);`

```
BulkWriteResult({  
  "nInserted" : 5,  
  ...  
})
```

Intermediate query

❖ Query an array

- To specify equality condition on an array, use the query document
 - `{ <field>: <value> }`
 - <value> is the EXACT array to match, including the order of the elements
- All documents where the field tags value is an array with exactly two elements, "red" and "blank", in the specified order
 - **`db.inventory2.find({ tags: ["red", "blank"] })`**

See the result

Intermediate query

❖ Query an array

- To query if the array field contains at least one element with the specified value
 - $\{ <field>: <value> \}$
 - $<value>$ is the element value
- All documents where tags is an array that contains the string "red" as one of its elements
 - **`db.inventory2.find({ tags: "red" })`**

See the result

Intermediate query

❖ Query for an Element by the Array Index Position

- Using dot notation
 - The array uses **zero-based indexing**
- All documents where the second element in the array dim_cm is greater than 25
 - **db.inventory2.find({ "dim_cm.1": { \$gt: 25 } })**

See the result

- Task
 - All documents where the first element of tags is blank and second element of dim_cm is greater than 20
 - **db.inventory2.find({"tags.0": "blank", "dim_cm.1": { \$gt: 20 } })**

See the result

Intermediate query

❖ Query an array of documents

```
1 {
2   "address": {
3     "building": "1007",
4     "coord": [ -73.856077, 40.848447 ],
5     "street": "Morris Park Ave",
6     "zipcode": "10462"
7   },
8   "borough": "Bronx",
9   "cuisine": "Bakery",
10  "grades": [
11    { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
12    { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
13    { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
14    { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
15    { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
16  ],
17  "name": "Morris Park Bake Shop",
18  "restaurant_id": "30075445"
19 }
```

Array of Values

Array of Documents

Embedded documents

Intermediate query

❖ Query an array of documents

- We will use the inventory collection for the rest of the lecture

- `db.inventory3.insert([`
 `{ item: "NORWAY", instock: [{ warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 }] },`
 `{ item: "notebook", instock: [{ warehouse: "C", qty: 5 }] },`
 `{ item: "paper", instock: [{ warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 }] },`
 `{instock: [{ warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 }] },`
 `{ item: null, instock: [{ warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 }] }`
 `]);`
• `BulkWriteResult({`
 `"nInserted" : 5,`
 `...`
 `})`

Intermediate query

❖ Query an array of documents

- The following example selects all documents where an element in the instock array matches the specified document

- **db.inventory3.find({ "instock": { warehouse: "A", qty: 5 } })**

See the result

- Equality matches on the whole embedded document require an exact match of the specified document

- **db.inventory3.find({ "instock": { qty: 5, warehouse: "A" } })**

See the result

Intermediate query

❖ Query an array of documents

- If you do not know the index position of the document nested in the array
 - Concatenate the name of the array field with a dot (.) and the name of the field in the nested document
- Example
 - Selects all documents where the “instock” array has at least one embedded document that contains the field “qty” whose value is greater than or equal to 20
 - **db.inventory3.find({ 'instock.qty': { \$gte: 20 } })**

Intermediate query

❖ Query an array of documents

- Using dot notation, you can specify query conditions for field in a document at a particular index or position of the array
- The instock array has as its first element a document that contains the field qty whose value is 5
 - **db.inventory3.find({"instock.0.qty": 5})**

See the result

- Task
 - The instock array has as its first element a document that contains the field qty whose value is less than or equal to 20
 - **db.inventory3.find({ 'instock.0.qty': { \$lte: 20 } })**

See the result

Part 2

ELEMENT OPERATORS

Element Operators

- ❖ Query operator
 - Element Operators

Name	Description
<code>\$exists</code>	Matches documents that have the specified field.
<code>\$type</code>	Selects documents if a field is of the specified type.

Element Operators

❖ Query an array of documents

- We will use the inventory collection for the rest of the lecture

- `db.inventory3.insert([`
 `{ item: "NORWAY", instock: [{ warehouse: "A", qty: 5 }, { warehouse: "C", qty: 15 }] },`
 `{ item: "notebook", instock: [{ warehouse: "C", qty: 5 }] },`
 `{ item: "paper", instock: [{ warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 }] },`
 `{instock: [{ warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 }] },`
 `{ item: null, instock: [{ warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 }] }`
 `]);`
• `BulkWriteResult({`
 `"nInserted" : 5,`
 `...`
 `})`

Element Operators

❖ Equality filter

- Query matches documents that either contain the item field whose value is null or that do not contain the item field

- *{ item : null }*

- Example

- **db.inventory3.find({ item: null })**

See the result

- Task

- Find the documents in the item field that does not contain null values
 - **db.inventory3.find({item:{\$ne:null}})**

See the result

Element Operators

❖ Existence check

- The query matches documents that do not contain the item field
 - *{ item : { \$exists: false } }*

- Example

- **db.inventory3.find({ item : { \$exists: false } })**

See the result

- Task

- Select item field where a record exists including null (show only item field)
 - **db.inventory3.find({ item : { \$exists: true } }, { _id: 0, item: 1 })**

See the result

Element Operators

❖ \$type

- Selects documents where the value of the field is an instance of the specified BSON type(s)
- Querying by data type is useful when dealing with highly unstructured data where data types are not predictable
- Syntax
 - *{ field: { \$type: <BSON type> } }*

Element Operators

❖ \$type

- BSON types

Type	Number	Alias	Notes
Double	1	"double"	
String	2	"string"	
Object	3	"object"	
Array	4	"array"	
Binary Data	5	"binData"	
Undefined	6	"undefined"	Deprecated
ObjectId	7	"objectId"	
Boolean	8	"bool"	
Date	9	"date"	
Null	10	"null"	
Regular Expression	11	"regex"	
DBPointer	12	"dbPointer"	Deprecated
JavaScript	13	"javascript"	
Symbol	14	"symbol"	Deprecated
Javascript (with scope)	15	"javascriptWithScope"	
32-bit integer	16	"int"	
Timestamp	17	"timestamp"	
64-bit Integer	18	"long"	
Decimal128	19	"decimal"	New in Version 3.4
Min Key	-1	"minKey"	
Max Key	127	"maxKey"	

Element Operators

❖ \$type

- Examples

```
db.grades.insertMany(  
  [  
    { "_id" : 1, name : "Alice King" , classAverage : 87.33333333333333 },  
    { "_id" : 2, name : "Bob Jenkins", classAverage : "83.52" },  
    { "_id" : 3, name : "Cathy Hart", classAverage: "94.06" },  
    { "_id" : 4, name : "Drew Williams" , classAverage : NumberInt("93") }  
  ]  
)  
• BulkWriteResult({  
  "nInserted" : 4,  
  ...  
})
```

Element Operators

❖ \$type

▪ Examples

- The following queries return all documents where classAverage is the BSON type string

- **db.grades.find({ "classAverage" : { \$type : "string" } });**

- **db.grades.find({ "classAverage" : { \$type : 2 } });**

See the result

- \$type supports the number alias, which will match against the following BSON types: double, 32-bit integer, 64-bit integer, decimal

- **db.grades.find({ "classAverage" : { \$type : "number" } });**

See the result

Element Operators

❖ \$type

- The \$type expression can also accept an array of BSON types and has the following syntax:

- *{ field: { \$type: [<BSON type1> , <BSON type2>, ...] } }*

▪ Example

- Return all documents where classAverage is the BSON type string or double or is an array containing an element of the specified types

- **db.grades.find({ "classAverage" : { \$type : [2 , 1] } });**

- **db.grades.find({ "classAverage" : { \$type : ["string" , "double"] } });**

See the result

Part 3

REGULAR EXPRESSIONS

Query Document

❖ Query operator

- Evaluation Operators

Name	Description
<code>\$expr</code>	Allows use of aggregation expressions within the query language.
<code>\$jsonSchema</code>	Validate documents against the given JSON Schema.
<code>\$mod</code>	Performs a modulo operation on the value of a field and selects documents with a specified result.
<code>\$regex</code>	Selects documents where values match a specified regular expression.
<code>\$text</code>	Performs text search.
<code>\$where</code>	Matches documents that satisfy a JavaScript expression.

Intermediate query

❖ Query embedded document

- We will use the inventory collection for query embedded document

```
db.inventory.insert([
```

```
  { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "D" },  
  { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },  
  { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },  
  { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },  
  { item: "paper", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }  
]);
```

```
BulkWriteResult({  
  "nInserted" : 5,  
  ...  
})
```

Regular Expressions

❖ Query operator

- Logical operators: `$regex`
 - Provides regular expression capabilities for pattern matching strings in queries
 - To use `$regex`, use one of the following syntaxes
 - `{ <field>: { $regex: /pattern/, $options: '<options>' } }`
 - `{ <field>: { $regex: 'pattern', $options: '<options>' } }`
 - `{ <field>: { $regex: /pattern/<options> } }`
 - Select documents that contain paper keyword in their item name
 - **`db.inventory.find({item: {$regex: "paper"}})`**

See the result

Regular Expressions

❖ Query operator

- Logical operators: \$regex
 - Adding the ^ and \$ character will do the exact text matches
 - The ^ is used to make sure that the string starts with a certain character
 - \$ is used to ensure that the string ends with a certain character
 - Find documents, where item name starts with “note” keyword
 - **db.inventory.find({item: {\$regex: "^note"}})**

See the result

- Find documents, where item name ends with “nal” keyword
 - **db.inventory.find({item: {\$regex: "nal\$"}})**

See the result

Regular Expressions

❖ Query operator

- Logical operators: \$regex

- You can use 'options: i' to overcome case sensitivity

- **db.inventory.find({item: {\$regex: "PAPER", \$options: 'i'}})**

See the result

- Task

- Find the documents where the status equals A and either quantity is less than 30 or item starts with the character p

- **db.inventory.find({
status: "A",
\$or: [{ qty: { \$lt: 30 } }, { item: { \$regex: "^p" } }]
})**

Summary and Discussions

- ❖ Intermediate queries
 - Query embedded document
 - Query array
 - Query array of documents
- ❖ Element Operators
- ❖ Regular expressions

Questions?

SEE YOU NEXT TIME!