Array in MongoDB

Array in MongoDB

- Array is a list of values that can take many different forms.
- Create a document:

db.Collection_name.operation({field: [“Val1”,“Val2”]})

```
>>> db.CourseRepository.insertMany([
. . . {item: “Diary”, qty:25, tags: [“blue”, “red”], dim_cm:[14,12]},
. . . {item: “Register”, qty:50, tags: [“blue, “brown”], dim_cm:[21,16]},
. . . {item: “Chart”, qty:100, tags: [“red”, “blue, “white”], dim_cm:[60,30]},
. . . {item: “Planner”, qty:75, tags: [“blank”, “red”], dim_cm:[30, 24]},
. . . {item: “PostCard”, qty:45, tags: [“blue”], dim_cm:[18,15]},
. . . ])
```

Querying on Array

- The find method can be used to return arrays inside a document.

Match an Array:

- To specify equality condition on an array, use the query document { <field>: <value> } .

  where <value> is the exact array to match, including the order of the elements.

Exa

```
>>> db.CourseRepository.insertMany([
. . . {item: "Diary", qty:25, tags: ["blue", "red"], dim_cm:[14,12]},
. . . {item: "Register", qty:50, tags: ["blue, "brown"], dim_cm:[21,16]},
. . . {item: "Chart", qty:100, tags: ["red", "blue, "white"], dim_cm:[60,30]},
. . . {item: "Planner", qty:75, tags: ["blank", "red"], dim_cm:[30, 24]},
. . . {item: "PostCard", qty:45, tags: ["blue"], dim_cm:[18,15]},
. . . ])
```

Query 1: db.CourseRepository.find({ tags: [“blue", “red"] })

Query 2: db.CourseRepository.find({ tags: { $all: [“blue", “red"] } })

- This query would return the document having tags “blue" and ”red", in the specified order.

```
{
>>> db.CourseRepository.find( {tags: ["blue", "red"] } )
{"id": ObjectID("5d7f6d1ba9479814c0cb7172"), "item":"Diary", "qty":25, "tags":
["blue", "red" ], "dim_cm":[14,12]}
```

- To find an array that contains both the elements “blue" and ”red", without regard to order or other elements in the array, use the $all operator:

```
>>> db.Courserepository.find( {tags: {$all: ["blue", "red"]}})
{"id": ObjectID("5d7f6d1ba9479814c0cb7172"), "item":"Diary", "qty":25, "tags":
["blue", "red" ], "dim_cm":[14,12]}
{"id": ObjectID("5d7f6d1ba9479814c0cb7174"), "item":"Chart", "qty":100, "tags":
[ "red" ,"blue", "white" ], "dim_cm":[60,30]}
```

```
>>> db.CourseRepository.insertMany([
. . . {item: "Diary", qty:25, tags: ["blue", "red"], dim_cm:[14,12]},
. . . {item: "Register", qty:50, tags: ["blue, "brown"], dim_cm:[21,16]},
. . . {item: "Chart", qty:100, tags: ["red", "blue, "white"], dim_cm:[60,30]},
. . . {item: "Planner", qty:75, tags: ["blank", "red"], dim_cm:[30, 24]},
. . . {item: "PostCard", qty:45, tags: ["blue"], dim_cm:[18,15]},
. . . ])
```

Query 1: db.CourseRepository.find({ tags: [“red", “blue"] })

Query 2: db.CourseRepository.find({ tags: { $all: [“red", “blue"] } })

Query 3: db.CourseRepository.find({ tags: [“red"] })

Query 4: db.CourseRepository.find({ tags: {$all: [“red"] }})

Query 5: db.CourseRepository.find({ tags: [“red", “white"] })

Query 6: db.CourseRepository.find({ tags: { $all: [“red", “white"] } })

Query 2:
db.CourseRepository.find
({ tags: { $all: [“red", “blue"] } })

```
[
  {
    _id: ObjectId('65c4800e1c6a904c63c416fb'),
    item: 'Diary',
    qty: 25,
    tags: [ 'blue', 'red' ],
    dim_cm: [ 14, 12 ]
  },
  {
    _id: ObjectId('65c4800e1c6a904c63c416fd'),
    item: 'Chart',
    qty: 100,
    tags: [ 'red', 'blue', 'white' ],
    dim_cm: [ 60, 30 ]
  }
]
```

Query 4:
db.CourseRepository.find
({ tags: {$all: [“red"] }})

```
[
  {
    _id: ObjectId('65c4800e1c6a904c63c416fb'),
    item: 'Diary',
    qty: 25,
    tags: [ 'blue', 'red' ],
    dim_cm: [ 14, 12 ]
  },
  {
    _id: ObjectId('65c4800e1c6a904c63c416fd'),
    item: 'Chart',
    qty: 100,
    tags: [ 'red', 'blue', 'white' ],
    dim_cm: [ 60, 30 ]
  },
  {
    _id: ObjectId('65c4800e1c6a904c63c416fe'),
    item: 'Planner',
    qty: 75,
    tags: [ 'black', 'red' ],
    dim_cm: [ 30, 24 ]
  }
]
```

Query 6:
db.CourseRepository.find
({ tags: { $all: [“red", “white"] } })

```
[
  {
    _id: ObjectId('65c4800e1c6a904c63c416fd'),
    item: 'Chart',
    qty: 100,
    tags: [ 'red', 'blue', 'white' ],
    dim_cm: [ 60, 30 ]
  }
]
```

Query an Array for an Element:

- To query if the array field contains at least *one* element with the specified value, use the filter { <field>: <value> } .

  Example: db.CourseRepository.find({ tags: "red" })

- To specify conditions on the elements in the array field, use query operators in the query filter document:

  Syntax: { <array field>: { <operator1>: <value1>, ... } }

  Example: db.CourseRepository.find({ dim_cm: { $gt: 25 } })

- This operation queries for all documents where the array dim_cm contains at least one element whose value is greater than 25.

```
>>> db.CourseRepository.insertMany([
. . . {item: "Diary", qty:25, tags: ["blue", "red"], dim_cm:[14,12]},
. . . {item: "Register", qty:50, tags: ["blue, "brown"], dim_cm:[21,16]},
. . . {item: "Chart", qty:100, tags: ["red", "blue, "white"], dim_cm:[60,30]},
. . . {item: "Planner", qty:75, tags: ["blank", "red"], dim_cm:[30, 24]},
. . . {item: "PostCard", qty:45, tags: ["blue"], dim_cm:[18,15]},
. . . ])
```

Specify Multiple Conditions for Array Elements

- When specifying **compound conditions** on array elements, the query can be specified such that either a **single array element meets these condition or any combination of array elements** meets the conditions.

❑ **Query an Array with Compound Filter Conditions on the Array Elements:** db.CourseRepository.find({ dim_cm: { $gt: 15, $lt: 20 } })

- This example queries for documents where the *dim_cm* array contains elements that in some combination satisfy the query conditions; e.g., one element can satisfy the greater than 15 condition and another element can satisfy the less than 20

```
>>> db.CourseRepository.insertMany([
. . . {item: "Diary", qty:25, tags: ["blue", "red"], dim_cm:[14,12]},
. . . {item: "Register", qty:50, tags: ["blue, "brown"], dim_cm:[21,16]},
. . . {item: "Chart", qty:100, tags: ["red", "blue, "white"], dim_cm:[60,30]},
. . . {item: "Planner", qty:75, tags: ["blank", "red"], dim_cm:[30, 24]},
. . . {item: "PostCard", qty:45, tags: ["blue"], dim_cm:[18,15]},
. . . ])
```

```
>>> db.CourseRepository.insertMany([
. . . {item: "Diary", qty:25, tags: ["blue", "red"], dim_cm:[14,12]},
. . . {item: "Register", qty:50, tags: ["blue, "brown"], dim_cm:[21,16]},
. . . {item: "Chart", qty:100, tags: ["red", "blue, "white"], dim_cm:[60,30]},
. . . {item: "Planner", qty:75, tags: ["blank", "red"], dim_cm:[30, 24]},
. . . {item: "PostCard", qty:45, tags: ["blue"], dim_cm:[18,15]},
. . . ])
```

- db.CourseRepository.find({ dim_cm: { $gt: 15, $lt: 20 } })

```
[
  {
    _id: ObjectId('65c4800e1c6a904c63c416fc'),
    item: 'Register',
    qty: 50,
    tags: [ 'blue', 'brown' ],
    dim_cm: [ 21, 16 ]
  },
  {
    _id: ObjectId('65c4800e1c6a904c63c416ff'),
    item: 'Postcard',
    qty: 45,
    tags: [ 'blue' ],
    dim_cm: [ 18, 15 ]
  }
]
```

❑ Query for an Array Element that Meets Multiple Criteria:

- Use $elemMatch operator to specify multiple criteria on the elements of an array such that at least one array element satisfies all the specified criteria.

  db.CourseRepository.find({ dim_cm: { $elemMatch: { $gt: 22, $lt: 30 } } })

- This example queries for documents where the dim_cm array contains

```
>>> db.CourseRepository.insertMany([
. . . {item: "Diary", qty:25, tags: ["blue", "red"], dim_cm:[14,12]},
. . . {item: "Register", qty:50, tags: ["blue, "brown"], dim_cm:[21,16]},
. . . {item: "Chart", qty:100, tags: ["red", "blue, "white"], dim_cm:[60,30]},
. . . {item: "Planner", qty:75, tags: ["blank", "red"], dim_cm:[30, 24]},
. . . {item: "PostCard", qty:45, tags: ["blue"], dim_cm:[18,15]},
. . . ])
```

gt) 22 and less than

❑ Query for an Element by the Array Index Position

- Using the dot notation, you can specify query conditions for an element at a particular index or position of the array. The array uses zero-based indexing.

  db. CourseRepository.find({ "dim_cm.1": { $gt: 25 } })

Query an Array by Array Length

- Use the $size operator to query for arrays by number of elements.

```
db.CourseRepository.find( { "tags": { $size: 3 } } )
```

- This query selects documents where array tags has 3 elements.

❖ **Nested array in MongoDB**

when a set of field values is bundled inside a particular

Syntax: `({field: [{ sub_field_1a: "valX1", sub_field_1b: "valY1}, {sub_field_2a: "valX2", sub_field_2b: "valY2"} ] })`

```
>>> >>> db.CourseRepository.insertMany([
. . . {item: "Diary", stock: [{storehouse: "X", qty:15}, { warehouse:"Z", qty:25}],_id:"BSI-001"},
. . . {item: "Register", stock: [{storehouse: "X", qty:10}],_id:"BSI-002"},
. . . {item: "Chartbook", stock: [{storehouse: "X", qty:40}, { storehouse:"Y", qty:25}],_id:"BSI-003"},
. . . {item: "Planner", stock: [{storehouse: "X", qty:30}, { storehouse:"Y", qty:5}],_id:"BSI-004"},
. . . {item: "Postcard", stock: [{storehouse: "Y", qty:15}, { storehouse:"Z", qty:35}],_id:"BSI-005"},
. . . ]);
. . .
```

Array Update Operators

Operator	Description
$push	Appends an element to the end of the array
$pop	Removes one element from the end of the array.
$pull	Removes all elements in the array that match a specified value.
$pullAll	Removes all elements in the array that match any of the specified values
$addToSet	Appends an element to the array if not already present

Array operator: $pop

- The $pop operator removes the first or last element of an array.

- Pass $pop a value of -1 to remove the first element of an array and 1 to remove the last element in an array.

- The $pop operator has the form:

  { $pop: { <field>: <-1 | 1>, ... } }

- The $pop operation fails if the <field> is not an array.

- If the $pop operator removes the last item in the <field>, the <field> will then hold an empty array.

Examples

Remove the First Item of an Array:

Create the students collection:

```
db.students.insertOne( { _id: 1, scores: [ 8, 9, 10 ] } )
```

Query: Remove *first* element, 8, from the scores array:

```
db.students.updateOne( { _id: 1 }, { $pop: { scores: -1 } } )
```

Output: The first element, 8, has been removed from the scores array

```
{ _id: 1, scores: [ 9, 10 ] }
```

Remove the Last Item of an Array

```
db.students.insertOne( { _id: 10, scores: [ 9, 10 ] } )
```

Remove the *last* element, 10, from the scores array by sp

```
db.students.updateOne( { _id: 10 }, { $pop: { scores: 1 } } )
```

The last elem

```
{ _id: 10, scores: [ 9 ] }
```

oved from the scores array:

Array operator: $pull

- The $pull operator removes from an existing array all instances of a value or values that match a specified condition.

- Syntax:

  { $pull: { <field1>: <value|condition>, <field2>: <value|condition>, ... } }

  To specify a <field> in an embedded document or in an array, use dot notation.

- If you specify a <condition> and the array elements are embedded documents, $pull operator applies the <condition> as if each array element were a document in a collection.

- If the specified <value> to remove is a document, $pull removes only the elements in the array that have the exact same fields and values. The ordering of the fields can differ.

Array operator: $pull Examples

Remove All Items That Equal a Specified Value

1) Create the stores collection:

```
db.stores.insertMany( [
   {
      _id: 1,
      fruits: [ "apples", "pears", "oranges", "grapes", "bananas" ],
      vegetables: [ "carrots", "celery", "squash", "carrots" ]
   },
   {
      _id: 2,
      fruits: [ "plums", "kiwis", "oranges", "bananas", "apples" ],
      vegetables: [ "broccoli", "zucchini", "carrots", "onions" ]
   }
] )
```

Output:

3)

```
{
  _id: 1,
  fruits: [ 'pears', 'grapes', 'bananas' ],
  vegetables: [ 'celery', 'squash' ]
},
{
  _id: 2,
  fruits: [ 'plums', 'kiwis', 'bananas' ],
  vegetables: [ 'broccoli', 'zucchini', 'onions' ]
}
```

2) Query:

```
db.stores.updateMany(
    { },
    { $pull: { fruits: { $in: [ "apples", "oranges" ] }, vegetables: "carrots" } }
)
```

This operation removes
(i) "apples" and "oranges" from the fruits array
(ii) "carrots" from the vegetables array

Remove All Items That Match a Specified $pull Condition

Create the *profiles* collection:

```
db.profiles.insertOne( { _id: 1, votes: [ 3, 5, 6, 7, 7, 8 ] } )
```

Remove all items from the votes array that are greater than or equal to

```
db.profiles.updateOne( { _id: 1 }, { $pull: { votes: { $gte: 6 } } } )
```

```
{ _id: 1, votes: [  3,  5 ] }
```

e document only has values less than 6:

Remove Items from an Array of Documents

Create the survey collectio

```
db.survey.insertMany([
   {
      _id: 1,
      results: [
         { item: "A", score: 5 },
         { item: "B", score: 8 }
      ]
   },
   {
      _id: 2,
      results: [
         { item: "C", score: 8 },
         { item: "B", score: 4 }
      ]
   }
] )
```

Removes all elements from the results array that contain both a score field equal to 8 and an i

```
db.survey.updateMany(
   { },
   { $pull: { results: { score: 8 , item: "B" } } }
)
```

- The $pull expression applies the condition to each element of the results array as though it were a top-level document.
- After the operation, the results array contains no documents that contain both a score field equal to 8 and an item field equal to "B".
- Create

```
db.survey.insertMany([
   {
      _id: 1,
      results: [
         { item: "A", score: 5 },
         { item: "B", score: 8 }
      ]
   },
   {
      _id: 2,
      results: [
         { item: "C", score: 8 },
         { item: "B", score: 4 }
      ]
   }
] )
```

- Removes all elements from the results array that contain both a score field equa
- Query:

```
db.survey.updateMany(
   { },
   { $pull: { results: { score: 8 , item: "B" } } }
)
```

- Result:

```
{ _id: 1, results: [ { item: 'A', score: 5 } ] },
{
  _id: 2,
  results: [ { item: 'C', score: 8 }, { item: 'B', score: 4 } ]
}
```

Remove Documents from Nested Arrays

Create a new survey collection with documents that are embedded in nested arrays.

```
db.survey.insertMany( [
   {
      _id: 1,
      results: [
         {
            item: "A",
            score: 5,
            answers: [ { q: 1, a: 4 }, { q: 2, a: 6 } ]
         },
         {
            item: "B",
            score: 8,
            answers: [ { q: 1, a: 8 }, { q: 2, a: 9 } ]
         }
      ]
   },
   {
      _id: 2,
      results: [
         {
            item: "C",
            score: 8,
            answers: [ { q: 1, a: 8 }, { q: 2, a: 7 } ]
         },
         {
            item: "B",
            score: 4,
            answers: [ { q: 1, a: 0 }, { q: 2, a: 8 } ]
         }
      ]
   }
] )
```

Specify multiple conditions on the elements of the answers array with $elemMatch:

```
db.survey.updateMany(
   { },
   {
      $pull:
         {
            results:
               {
                  answers: { $elemMatch: { q: 2, a: { $gte: 8 } } }
               }
         }
   }
)
```

```
{
  _id: 1,
  results: [
    {
      item: 'A',
      score: 5,
      answers: [ { q: 1, a: 4 }, { q: 2, a: 6 } ]
    }
  ]
},
{
  _id: 2,
  results: [
    {
      item: 'C',
      score: 8,
      answers: [ { q: 1, a: 8 }, { q: 2, a: 7 } ]
    }
  ]
}
```

Array operator: $pullall

- The *$pullAll* operator removes all instances of the specified values from an existing array.

- Unlike the $pull operator that removes elements by specifying a query, *$pullAll* removes elements that match the listed values.

- Syntax: { $pullAll: { <field1>: [<value1>, <value2> ...], .. } }
  - ❖ To specify a <field> in an embedded document or in an array, use dot notation.

Example: Create the survey collection:

```
db.survey.insertOne( { _id: 1, scores: [ 0, 2, 5, 5, 1, 0 ] } )
```

Remove all instances of the values "0" and "5" from the scores array:

```
db.survey.updateOne( { _id: 1 }, { $pullAll: { scores: [ 0, 5 ] } } )
```

Output:

```
{ "_id" : 1, "scores" : [ 2, 1 ] }
```

Array Update Operator: $push

- The $push operator appends a specified value to an array.

  Syntax: { $push: { <field1>: <value1>, ... } }

  - To specify a <field> in an embedded document or in an array, use dot notation.

- If the field is absent in the document to update, $push adds the array field with the value as its element.

- If the field is **not** an array, the operation will fail.

- If the value is an array, $push appends the whole array as a *single* element. To add each element of the value separately, use the $each modifier with $push.

$push operator: Append a Value to an Array

Create the students collection:

```
db.students.insertOne( { _id: 1, scores: [ 44, 78, 38, 80 ] } )
```

The following example appends 89 to the scores array:

```
db.students.updateOne(
   { _id: 1 },
   { $push: { scores: 89 } }
)
```

Output:

```
{ _id: 1, scores: [ 44, 78, 38, 80, 89 ] }
```

$push operator: Append a Value to Arrays in Multiple Documents

- Add the following documents to the students collection:

```
db.students.insertMany( [
   { _id: 2, scores: [ 45, 78, 38, 80, 89 ] } ,
   { _id: 3, scores: [ 46, 78, 38, 80, 89 ] } ,
   { _id: 4, scores: [ 47, 78, 38, 80, 89 ] }
] )
```

- The following $push operation appends 95 to the scores array in each document:

```
db.students.updateMany(
   { },
   { $push: { scores: 95 } }
)
```

- To confirm that each scores array includes 95, run the following ope

```
db.students.find()
```

- Output

```
[
   { _id: 1, scores: [ 44, 78, 38, 80, 89, 95 ] },
   { _id: 2, scores: [ 45, 78, 38, 80, 89, 95 ] },
   { _id: 3, scores: [ 46, 78, 38, 80, 89, 95 ] },
   { _id: 4, scores: [ 47, 78, 38, 80, 89, 95 ] }
]
```

$push operator: Append Multiple Values to an Array

- Use $push with the $each modifier to append multiple values to the array field.

- The following example appends each element of [90, 92, 85] to the scores array for the document where the name field equals joe:

```
db.students.updateOne(
   { name: "joe" },
   { $push: { scores: { $each: [ 90, 92, 85 ] } } }
)
```

Array operator: $addToSet

- The $addToSet operator adds a value to an array unless the value is already present, in which case $addToSet does nothing to that array.

- Syntax:

  { $addToSet: { <field1>: <value1>, ... } }

- To specify a <field> in an embedded document or in an array, use dot notation.

- *$addToSet* only ensures that there are no duplicate items *added* to the set and does not affect existing duplicate elements.
- *$addToSet* does not guarantee a particular ordering of elements in the modified set.

Missing Field:

- If $addToSet used on a field that is absent from the document to update, $addToSet creates the array field with the specified value as its element.

Field is Not an Array:

- If you use $addToSet on a field that is **not** an array, the operation will fail.
- For example, create the pigments collection:

```
db.pigments.insertOne( { _id: 1, colors: "blue, green, red" } )
```

- The colors field is not an array. The following $addToSet operation fails:

```
db.pigments.updateOne(
   { _id: 1 },
   { $addToSet: { colors: "mauve" } }
)
```

$addToSet: Value to Add is An Array

- If the value is an array, $addToSet appends the whole array as a *single* element.

Create the *alphabet* collection:

```
db.alphabet.insertOne( { _id: 1, letters: ["a", "b"] } )
```

- The following operation appends the array ["c", "d"] to the letters field:

```
db.alphabet.updateOne(
   { _id: 1 },
   { $addToSet: { letters: [ "c", "d" ] } }
)
```

- The array ["c", "d"] is added as a single element:

```
{ _id: 1, letters: [ 'a', 'b', [ 'c', 'd' ] ] }
```

- To add each element of the value **separately**, use the $each modifier with $addToSet.

$addToSet: Value to Add is a Document

- If the value is a document, MongoDB determines that the document is a duplicate if an existing document in the array matches the to-be-added document exactly; i.e. the existing document has the exact same fields and values *and* the fields are in the same order.

```
db.inventory.insertOne(
   { _id: 1, item: "polarizing_filter", tags: [ "electronics", "camera" ] }
)
```

- **Add to Array:** Add the element "accessories" to the tags array since "accessories" does not exist in the array.

```
db.inventory.updateOne(
   { _id: 1 },
   { $addToSet: { tags: "accessories" } }
)
```

- **Value Already Exists:** $addToSet operation has no effect because "camera" is already an element of the tags array:

```
db.inventory.updateOne(
   { _id: 1 },
   { $addToSet: { tags: "camera"  } }
)
```

Modifiers

Modifier	Description
$each	Appends multiple values to the array field.
$slice	Limits the number of array elements. Requires the use of the *$each* modifier.
$sort	Orders elements of the array. Requires the use of the *$each* modifier.
$position	Specifies the location in the array at which to insert the new elements. Requires the use of the *$each* modifier. Without the *$position* modifier, the *$push* appends the elements to the end of the array.

$each modifier

- The *$each* modifier is available for use with the *$addToSet* operator and the *$push* operator.

- Use with the *$addToSet* operator to add multiple values to an array <field> if the values do not exist in the <field>.

```
{ $addToSet: { <field>: { $each: [ <value1>, <value2> ... ] } } }
```

- Use with the $push operator to append multiple values to an array <field>.

```
{ $push: { <field>: { $each: [ <value1>, <value2> ... ] } } }
```

- The $push operator can use $each modifier with other modifiers.

Use $each with $push Operator

- The following example appends each element of [90, 92, 85] to the scores array for the document where the name fie

```
db.students.updateOne(
   { name: "joe" },
   { $push: { scores: { $each: [ 90, 92, 85 ] } } }
)
```

$each Modifier

- You can use the $addToSet operator with the $each modifier. The $each modifier allows the $addToSet operator to add multiple values to the array field.
- A collection inventory has the following document:

```
{ _id: 2, item: "cable", tags: [ "electronics", "supplies" ] }
```

- Then the following operation uses the $addToSet operator with the $each modifier to add multiple elements to

```
db.inventory.updateOne(
   { _id: 2 },
   { $addToSet: { tags: { $each: [ "camera", "electronics", "accessories" ] } } }
)
```

- The operation only adds "camera" and "accessories" to the tag y:

```
{
   _id: 2,
   item: "cable",
   tags: [ "electronics", "supplies", "camera", "accessories" ]
}
```

$sort modifier

- The $sort modifier orders the elements of an array during a $push operation.
- To use the $sort modifier, it **must** appear with the $each modifier.
- An empty array [] can pass to the $each modifier such that only the $sort modifier has an effect.
- The $sort modifier can sort array elements that are not documents.
- If the array elements are documents, the modifier can sort by either the whole document or by a specific field in the documents.
- Trying to use the $sort modifier without the $each modifier results in an error.
- The $sort no longer requires the $slice modifier.

```
{
  $push: {
     <field>: {
       $each: [ <value1>, <value2>, ... ],
       $sort: <sort specification>
     }
  }
}
```

For <sort specification>:

- To sort array elements that are not documents, or if the array elements are documents, to sort by the whole documents, specify 1 for ascending or -1 for descending.
- If the array elements are documents, to sort by a field in the documents, specify a sort document with the field and the direction, i.e. { field: 1 } or { field: -1 }. Do **not** reference the containing array field in the sort specification.
  (e.g. { "arrayField.field": 1 } is incorrect).

Sort Array of Documents by a Field in the Documents

- 1. Create the students collection:

```
db.students.insertOne(
   {
     "_id": 1,
     "quizzes": [
       { "id" : 1, "score" : 6 },
       { "id" : 2, "score" : 9 }
     ]
   }
)
```

3. Output:

```
{
  "_id" : 1,
  "quizzes" : [
     { "id" : 1, "score" : 6 },
     { "id" : 5, "score" : 6 },
     { "id" : 4, "score" : 7 },
     { "id" : 3, "score" : 8 },
     { "id" : 2, "score" : 9 }
  ]
}
```

- 2. Query: Append additional documents to the quizzes array and then sorts all the elements of the array by the ascending score field.

```
db.students.updateOne(
   { _id: 1 },
   {
     $push: {
       quizzes: {
         $each: [ { id: 3, score: 8 }, { id: 4, score: 7 }, { id: 5, score: 6 } ],
         $sort: { score: 1 }
       }
     }
   }
)
```

- The sort document refers directly to the field in the documents and does not reference the containing array field quizzes; i.e. { score: 1 } and **not** { "quizzes.score": 1}

Sort Array Elements That Are Not Documents

- Add the following document to the students collection:

```
db.students.insertOne( { "_id" : 2, "tests" : [  89,  70,  89,  50 ] } )
```

- Query: Add two more elements to the tests array and sorts the elements.

```
db.students.updateOne(
   { _id: 2 },
   { $push: { tests: { $each: [ 40, 60 ], $sort: 1 } } }
)
```

- The updated document has the elements of the tests array in ascending order:

```
{ "_id" : 2, "tests" : [  40,  50,  60,  70,  89,  89 ] }
```

Update Array Using Sort Only

- Add the following document to the students collection:

```
db.students.insertOne( { "_id" : 3, "tests" : [  89,  70,  100,  20 ] } )
```

- To update the tests field to sort its elements in descending order, specify the { $sort: -1 } and specify an empty array [] for the $each modifier. For example:

```
db.students.updateOne(
   { _id: 3 },
   { $push: { tests: { $each: [ ], $sort: -1 } } }
)
```

- The example sorts the tests field values in descending order:

```
{ "_id" : 3, "tests" : [ 100,  89,  70,  20 ] }
```

Use $sort with Other $push Modifiers

- Add the following document to the students collection:

```
db.students.insertOne(
   {
      "_id" : 5,
      "quizzes" : [
         { "wk": 1, "score" : 10 },
         { "wk": 2, "score" : 8 },
         { "wk": 3, "score" : 5 },
         { "wk": 4, "score" : 6 }
      ]
   }
)
```

```
db.students.updateOne(
   { _id: 5 },
   {
     $push: {
       quizzes: {
          $each: [ { wk: 5, score: 8 }, { wk: 6, score: 7 }, { wk: 7, score: 6 } ]
          $sort: { score: -1 },
          $slice: 3
       }
     }
   }
)
```

- The $push operation uses:
- the $each modifier to add multiple documents to the quizzes array,
- the $sort modifier to sort all the elements of the modified quizzes array by the score field in descending order, and
- the $slice modifier to keep only the **first** three sorted elements of the quizzes array.

- After the ope … nest scoring quizzes are in the array:

```
{
  "_id" : 5,
  "quizzes" : [
     { "wk" : 1, "score" : 10 },
     { "wk" : 2, "score" : 8 },
     { "wk" : 5, "score" : 8 }
  ]
}
```

The order of the modifiers in the query does not change the order that the modifiers are applied.

$slice modifier

- The $slice modifier limits the number of array elements during a $push operation.

- To use the $slice modifier, it **must** appear with the $each modifier. You can pass an empty array [] to the $each modifier such that only the $slice modifier has an effect.
- Syntax:

```
{
  $push: {
     <field>: {
       $each: [ <value1>, <value2>, ... ],
       $slice: <num>
     }
  }
}
```

- The <num> can be:
  - **Zero:** To update the array <field> to an empty array.
  - **Negative:** To update the array <field> to contain only the last <num> elements.
  - **Positive:** To update the array <field> contain only the first <num> elements.

Slice from the end of the array

A collection students contains the following document:

```
{ "_id" : 1, "scores" : [ 40, 50, 60 ] }
```

- Adds new elements to the scores array, and then uses the $slice modifier to trim the array to the last five elements:

```
db.students.updateOne(
   { _id: 1 },
   {
     $push: {
       scores: {
         $each: [ 80, 78, 86 ],
         $slice: -5
       }
     }
   }
)
```

- The result of the operation is slice the elements of the updated scores array to the last five elements:

```
{ "_id" : 1, "scores" : [  50,  60,  80,  78,  86 ] }
```

Slice from the Front of the Array

A collection students contains the following document:

```
{ "_id" : 2, "scores" : [ 89, 90 ] }
```

Adds new elements to the scores array, and then uses the $slice modifier to trim the array to the first three elements.

```
db.students.updateOne(
   { _id: 2 },
   {
     $push: {
       scores: {
         $each: [ 100, 20 ],
         $slice: 3
       }
     }
   }
)
```

The result of the operation is to slice the elements of the updated scores array to the first three elements:

```
{ "_id" : 2, "scores" : [  89,  90,  100 ] }
```

Update Array Using Slice Only

A collection students contains the following document:

```
{ "_id" : 3, "scores" : [  89,  70,  100,  20 ] }
```

To update the scores field with just the effects of the $slice modifier, specify the number of elements to slice (e.g. -3) for the $slice modifier and an empty array [] for the $each modifier, as in the following:

```
db.students.updateOne(
  { _id: 3 },
  {
    $push: {
      scores: {
         $each: [ ],
         $slice: -3
      }
    }
  }
)
```

The result of the operation is to slice the elements of the scores array to the last three elements:

```
{ "_id" : 3, "scores" : [  70,  100,  20 ] }
```

Use $slice with Other $push Modifiers

Add the following document to the students collection:

```
db.students.insertOne(
   {
      "_id" : 5,
      "quizzes" : [
         { "wk": 1, "score" : 10 },
         { "wk": 2, "score" : 8 },
         { "wk": 3, "score" : 5 },
         { "wk": 4, "score" : 6 }
      ]
   }
)
```

- Perform the following $push operation uses:
- the $each modifier to add multiple documents to the quizzes array,
- the $sort modifier to sort all the elements of the modified quizzes array by the score field in descending order, and
- the $slice modifier to keep only the **first** three sorted elements of the quizzes array.

```
db.students.updateOne(
   { _id: 5 },
   {
     $push: {
       quizzes: {
       $each: [ { wk: 5, score: 8 }, { wk: 6, score: 7 }, { wk: 7, score: 6 } ],
       $sort: { score: -1 },
       $slice: 3
    }
  }
}
)
```

- After the operation only the three highest scoring quizzes are in the array:

```
{
  "_id" : 5,
  "quizzes" : [
     { "wk" : 1, "score" : 10 },
     { "wk" : 2, "score" : 8 },
     { "wk" : 5, "score" : 8 }
  ]
}
```

- The order of the modifiers is immaterial to the order in which the modifiers are processed.

$position modifier

- The $position modifier specifies the location in the array at which the $push operator inserts elements.

- Without the $position modifier, the $push operator inserts elements to the end of the array.

- To use the $position modifier, it **must** appear with the $each modifier.
- Syntax:

```
{
  $push: {
    <field>: {
       $each: [ <value1>, <value2>, ... ],
       $position: <num>
    }
  }
}
```

- <num> indicates the position in the array, based on a zero-based array index (position).

- A non-negative number corresponds to the position in the array, starting from the beginning of the array.

- If the value of <num> is greater or equal to the length of the array, the $position modifier has no effect and $push adds elements to the end of the array.

- A negative number corresponds to the position in the array, counting from (but not including) the last element of the array.
  - For example, -1 indicates the position just before the last element in the array.
  - If you specify multiple elements in the $each array, the last added element is in the specified position from the end.
  - If the absolute value of <num> is greater than or equal to the length of the array, the $push adds elements to the beginning of the array.

Examples

Add Elements at the Start of the Array

Create the students collection:

```
db.students.insertOne( { "_id" : 1, "scores" : [ 100 ] } )
```

Update scores field to add the elements 50, 60 and 70 to the beginning of the array:

```
db.students.updateOne(
   { _id: 1 },
   {
     $push: {
        scores: {
           $each: [ 50, 60, 70 ],
           $position: 0
        }
     }
   }
)
```

The operation results in the following updated document:

```
{ "_id" : 1, "scores" : [  50,  60,  70,  100 ] }
```

Add Elements to the Middle of the Array

Add a document to the *students* collection:

```
db.students.insertOne( { "_id" : 2, "scores" : [  50,  60,  70,  100 ] } )
```

Update scores field to add the elements 20 and 30 at the array index (pos

```
db.students.updateOne(
   { _id: 2 },
   {
     $push: {
        scores: {
           $each: [ 20, 30 ],
           $position: 2
        }
     }
   }
)
```

The operation results in the following updated document:

```
{ "_id" : 2, "scores" : [  50,  60,  20,  30,  70,  100 ] }
```

Use a Negative Array Index (Position) to Add Elements to the Array

- $position can accept a negative array index (position) value to indicate the position starting from the end, counting from (but not including) the last element of the array. For example, e the last element in

```
db.students.insertOne(
   { "_id" : 3, "scores" : [  50,  60,  20,  30,  70,  100 ] }
)
```

- Example
- Query:

```
db.students.updateOne(
   { _id: 3 },
   {
     $push: {
        scores: {
           $each: [ 90, 80 ],
           $position: -2
        }
     }
   }
)
```

This operation specifies -2 for the $position to add 90 at the position two places before the last element, and then 80 at the position two places before the last element.

❖ With a negative array index (position), if you specify multiple elements in the $ea n from the end.

```
{ "_id" : 3, "scores" : [ 50, 60, 20, 30, 90, 80, 70, 100 ] }
```

Use $push Operator with Multiple Modifiers

```
db.students.insertOne(
   {
      "_id" : 5,
      "quizzes" : [
         { "wk": 1, "score" : 10 },
         { "wk": 2, "score" : 8 },
         { "wk": 3, "score" : 5 },
         { "wk": 4, "score" : 6 }
      ]
   }
)
```

```
db.students.updateOne(
   { _id: 5 },
   {
     $push: {
       quizzes: {
          $each: [ { wk: 5, score: 8 }, { wk: 6, score: 7 }, { wk: 7, score: 6 } ],
          $sort: { score: -1 },
          $slice: 3
       }
     }
   }
)
```

- the *$each* modifier to add multiple documents to the quizzes array,
- the *$sort* modifier to sort all the elements of the modified quizzes array by the score field in descending order,
- the *$slice* modifier to keep only the first three sorted elements of the quizzes

Output

```
{
  "_id" : 5,
  "quizzes" : [
     { "wk" : 1, "score" : 10 },
     { "wk" : 2, "score" : 8 },
     { "wk" : 5, "score" : 8 }
  ]
}
```

- When used with modifiers, the $push operator has the form:

  { $push: { <field1>: { <modifier1>: <value1>, ... }, ... } }

- The processing of the $push operation with modifiers occur in the following order, regardless of the order in which the modifiers appear:

1. Update array to add elements in the correct position.
2. Apply sort, if specified.
3. Slice the array, if specified.
4. Store the array.