Update Operators

updateOne() and updateMany()

- The .updateOne() method updates a single document that satisfies a given filter.
- db.users.updateOne({ name: “Bob"}, { $set: { course: “CSE"}, $currentDate: {lastModified:true } })
- The .updateMany() method updates all documents that satisfy a specific filter criteria.
- db.employees.updateMany({ salary: 70000 }, { $set: { salary: 85000 }})

Update operators

- To update a document, MongoDB has many in-built update operators. These operators are to be used for passing the updated form of the document to the update methods.

Operator	Description
Operator	Description
$currentDate	Sets the value of a field to current date, either as a Date or a Timestamp
$inc	Increments the value of the field by the specified amount
$min	Only updates the field if the specified value is less than the existing field value
$max	Only updates the field if the specified value is greater than the existing field value
$mul	Multiplies the value of the field by the specified amount
$rename	Renames a field
$set	Sets the value of a field in a document

$inc operator

- The $inc operator increments a field by a specified value.

Syntax: { $inc: { <field1>: <amount1>, <field2>: <amount2>, ... } }

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

- The $inc operator accepts positive and negative values.

- If the field does not exist, $inc creates the field and sets the field to the specified value.

- Use of the $inc operator on a field with a null value will generate an error.

```
db.updateopr.insertMany(
    {_id: 1, qty: 20, wt:25},
    {_id: 2, qty: 15, wt: 45},
    {_id: 3, qty: 40, wt:30},
    {_id: 4, qty: 25, wt: 35}
    )
```

Write the queries for the following:

Question1: Increase the value of field *qty* by 2 for _id:1

Question2: Decrease the value of field *qty* by 2 and increase the *wt* by 5 for _id:2

Question3: Increase the value of field qty by 5 and wt by 5 for all documents

Question4: Increase the value of field qty by 5 and wt by 5 for documents with _id:2, _id:4.

```
db.updateopr.insertMany(
    {_id: 1, qty: 20, wt:25},
    {_id: 2, qty: 15, wt: 45},
    {_id: 3, qty: 40, wt:30},
    {_id: 4, qty: 25, wt: 35}
    )
```

Write the queries for the following:

Question1: Increase the value of field *qty* by 2 for _id:1

Question2: Decrease the value of field *qty* by 2 and increase the *wt* by 5 for _id:2

Question3: Increase the value of field qty by 5 and wt by 5 for all documents

Question4: Increase the value of field qty by 5 and wt by 5 for documents with _id:2, _id:4.

Answers:

Query1: db.updateopr.updateOne({_id:1}, {$inc:{qty:2}})

Query2: db.updateopr.updateOne({_id:2}, {$inc:{qty:-2, wt:3}})

Query3: db.updateopr.updateMany({}, {$inc:{qty:5, wt:5}})

Query4: db.updateopr.updateMany({$or:[{_id2},{_id:4}]}, {$inc:{qty:5, wt:5}})

Question: (For Embedded Document)

```
db.products.insertOne(
   {
     _id: 1,
     sku: "abc123",
     quantity: 10,
     metrics: { orders: 2, ratings: 3.5 }
   }
)
```

Update the following fields:

- increase the value of “orders” field by 1
- increase the value of “quantity” field by -2

Example: (For Embedded Document)

```
db.products.insertOne(
   {
     _id: 1,
     sku: "abc123",
     quantity: 10,
     metrics: { orders: 2, ratings: 3.5 }
   }
)
```

In this example updateOne() operation uses the $inc operator to:

- increase the "metrics.orders" field by 1
- increase quantity)
- Query:

```
db.products.updateOne(
   { sku: "abc123" },
   { $inc: { quantity: -2, "metrics.orders": 1 } }
)
```

- Output:

```
{
  _id: 1,
  sku: 'abc123',
  quantity: 8,
  metrics: { orders: 3, ratings: 3.5 }
}
```

$rename operator

- The $rename operator updates the name of a field.

Syntax:

{$rename: { <field1>: <newName1>, <field2>: <newName2>, ... } }

- The new field name must differ from the existing field name.

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

Example:

```
db.students.updateOne(
   { _id: 1 },
   { $rename: { 'nickname': 'alias', 'cell': 'mobile' } }
)
```

- This operation renames the field *nickname* to *alias*, and the field *cell* to *mobile*.

- The *$rename* operator logically performs an *$unset* of both the old name and the new name, and then performs a *$set* operation with the new name.

- As such, the operation may not preserve the order of the fields in the document; i.e. the renamed field may move within the document.

- If the document already has a field with the <newName>, the $rename operator removes that field and renames the specified <field> to <newName>.

- If the field to rename does not exist in a document, *$rename* does nothing (i.e. no operation).

- For fields in embedded documents, the *$rename* operator can rename these fields by using dot operator. *$rename* does not work if these fields are in array elements.

Rename a Field

```
db.students.insertMany( [
   {
     "_id": 1,
     "alias": [ "The American Cincinnatus", "The American Fabius" ],
     "mobile": "555-555-5555",
     "nmae": { "first" : "george", "last" : "washington" }
   },
   {
     "_id": 2,
     "alias": [ "My dearest friend" ],
     "mobile": "222-222-2222",
     "nmae": { "first" : "abigail", "last" : "adams" }
   },
   {
     "_id": 3,
     "alias": [ "Amazing grace" ],
     "mobile": "111-111-1111",
     "nmae": { "first" : "grace", "last" : "hopper" }
   }
] )
```

```
db.students.updateMany( {}, { $rename: { "nmae": "name" } } )
```

Rename a Field That Does Not Exist

```
db.students.updateOne( { _id: 1 }, { $rename: { 'wife': 'spouse' } } )
```

Rename a Field in an Embedded Document

```
{
  "_id": 1,
  "alias": [ "The American Cincinnatus", "The American Fabius" ],
  "mobile": "555-555-5555",
  "name": { "first" : "george", "last" : "washington" }
}

{
   "_id" : 2,
   "alias" : [ "My dearest friend" ],
   "mobile" : "222-222-2222",
   "name" : { "first" : "abigail", "last" : "adams" }
}

{ "_id" : 3,
  "alias" : [ "Amazing grace" ],
  "mobile" : "111-111-1111",
  "name" : { "first" : "grace", "last" : "hopper" } }
```

```
db.students.updateOne( { _id: 1 }, { $rename: { "name.first": "name.fname" } } )
```

```
{
  _id: 1,
  alias: [ 'The American Cincinnatus', 'The American Fabius' ],
  mobile: '555-555-5555',
  name: { last: 'washington', fname: 'george' }
}
```

$mul operator

- Multiply the value of a field by a number. To specify a $mul expression, use the following prototype:

Syntax: { $mul: { <field1>: <number1>, ... } }

- The field to update must contain a numeric value.

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

- If the field does not exist in a document, *$mul* creates the field and sets the value to zero of the same numeric type as the multiplier.

```
db.products.insertOne(
   { "_id" : 1, "item" : "Hats", "price" : Decimal128("10.99"), "quantity" : 25 }
)
```

Query: Multiply the price field by 1.25 and the quantity field by 2.

Multiply the Value of a Field

```
db.products.insertOne(
   { "_id" : 1, "item" : "Hats", "price" : Decimal128("10.99"), "quantity" : 25 }
)
```

Query: Multiply the price field by 1.25 and the quantity field by 2.

```
db.products.updateOne(
   { _id: 1 },
   { $mul:
      {
         price: Decimal128( "1.25" ),
         quantity: 2
      }
   }
)
```

In the updated document:

- price: original value, 10.99, multiplied by 1.25
- quantity: original value, 25, multiplied by 2

```
{ _id: 1, item: 'Hats', price: Decimal128("13.7375"), quantity: 50 }
```

Apply $mul Operator to a Non-existing Field

```
db.products.insertOne( { _id: 2,  item: "Unknown" } )
```

```
db.products.updateOne(
   { _id: 2 },
   { $mul: { price: Decimal128("100") } }
)
```

- This query attempts to apply the $mul operator to a field that is not in the document.
- The db.collection.updateOne() operation results:

➢ inserts the price field

```
{ _id: 2, item: 'Unknown', price: Decimal128('0') }
```

➢ sets Decimal128("0").

Multiply Mixed Numeric Types

```
db.products.insertOne( { _id: 3,  item: "Scarf", price: Decimal128("10") } )
```

Query: uses the $mul operator to multiply the value in the price field Decimal128(10) by Int32(5):

```
db.products.updateOne(
   { _id: 3 },
   { $mul: { price: Int32(5) } }
)
```

Output:

```
{ _id: 3, item: 'Scarf', price: Decimal128("50") }
```

- The value in the price field is of type Decimal128.

Mixed Type: Multiplication with values of mixed numeric types (32-bit integer, 64-bit integer, float) may result in conversion of numeric type. For multiplication with values of mixed numeric types, the following type conversion rules apply:

	32-bit Integer	64-bit Integer	Float
32-bit Integer	32-bit or 64-bit Integer	64-bit Integer	Float
64-bit Integer	64-bit Integer	64-bit Integer	Float
Float	Float	Float	Float

$set operator

- The $set operator replaces the value of a field with the specified value.

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

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

Examples:

- db.users.updateOne({ name: “Bob"}, { $set: { course: “CSE"}})
- db.employees.updateMany({ salary: 70000 }, { $set: { salary: 85000 }})

- If the field does not exist, $set will add a new field with the specified value.
- If specifying multiple field-value pairs, $set will update or create each field.

Set Top-Level Fields

```
db.products.insertOne(
   {
     _id: 100,
     quantity: 250,
     instock: true,
     reorder: false,
     details: { model: "14QQ", make: "Clothes Corp" },
     tags: [ "apparel", "clothing" ],
     ratings: [ { by: "Customer007", rating: 4 } ]
   }
)
```

Query: For the document matching the criteria _id equal to 100, use the $set operator to

- update the value of the *quantity* field to 500
- *details* field with new embedded document containing field *model* with value 2600 and field *make* with value *Fashionaires*
- the tags field with new array elements: coats, outerwear and clothing.

- Query:

```
db.products.updateOne(
   { _id: 100 },
   { $set:
      {
        quantity: 500,
        details: { model: "2600", make: "Fashionaires" },
        tags: [ "coats", "outerwear", "clothing" ]
      }
   }
)
```

- Output:

```
{
  _id: 100,
  quantity: 500,
  instock: true,
  reorder: false,
  details: { model: '2600', make: 'Fashionaires' },
  tags: [ 'coats', 'outerwear', 'clothing' ],
  ratings: [ { by: 'Customer007', rating: 4 } ]
}
```

Set Fields in Embedded Documents

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

```
{
  _id: 100,
  quantity: 500,
  instock: true,
  reorder: false,
  details: { model: '2600', make: 'Fashionaires' },
  tags: [ 'coats', 'outerwear', 'clothing' ],
  ratings: [ { by: 'Customer007', rating: 4 } ]
}
```

Query: For the document matching the criteria _id equal to 100, updates the make field in the *details* document.

```
db.products.updateOne(
   { _id: 100 },
   { $set: { "details.make": "Kustom Kidz" } }
)
```

Output

```
{
   _id: 100,
   quantity: 500,
   instock: true,
   reorder: false,
   details: { model: '2600', make: 'Kustom Kidz' },
   tags: [ 'coats', 'outerwear', 'clothing' ],
   ratings: [ { by: 'Customer007', rating: 4 } ]
}
```

```
{
  _id: 100,
  quantity: 500,
  instock: true,
  reorder: false,
  details: { model: '2600', make: 'Fashionaires' },
  tags: [ 'coats', 'outerwear', 'clothing' ],
  ratings: [ { by: 'Customer007', rating: 4 } ]
}
```

```
db.products.updateOne(
   { _id: 100 },
   { $set: { "details.make": "Kustom Kidz" } }
)
```

```
{
   _id: 100,
   quantity: 500,
   instock: true,
   reorder: false,
   details: { model: '2600', make: 'Kustom Kidz' },
   tags: [ 'coats', 'outerwear', 'clothing' ],
   ratings: [ { by: 'Customer007', rating: 4 } ]
}
```

```
db.products.updateOne(
   { _id: 100 },
   { $set: { details:
      {make: "Kustom Kidz"}
      }
   })
```

Replaces the entire embedded document, removing all other fields in the embedded details document.

```
details: { make: 'Kustom Kidz' }
```

Set Elements in Arrays

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

```
db.products.insertOne(
   {
     _id: 100,
     quantity: 250,
     instock: true,
     reorder: false,
     details: { model: "14QQ", make: "Clothes Corp" },
     tags: [ "apparel", "clothing" ],
     ratings: [ { by: "Customer007", rating: 4 } ]
   }
)
```

Query: For the document matching the criteria _id equal to 100, update the value of second element of the *tags* field and the first element of the *rating* field in the ratings array.

```
{
  _id: 100,
  quantity: 500,
  instock: true,
  reorder: false,
  details: { model: '2600', make: 'Fashionaires' },
  tags: [ 'coats', 'outerwear', 'clothing' ],
  ratings: [ { by: 'Customer007', rating: 4 } ]
}
```

Query: For the document matching the criteria _id equal to 100, update the value of second element of the *tags* field and the first element of the *rating* field in the ratings array.

```
db.products.updateOne(
   { _id: 100 },
   { $set:
      {
        "tags.1": "rain gear",
        "ratings.0.rating": 2
      }
   }
)
```

Output

```
{
  _id: 100,
  quantity: 500,
  instock: true,
  reorder: false,
  details: { model: '2600', make: 'Kustom Kidz' },
  tags: [ 'coats', 'rain gear', 'clothing' ],
  ratings: [ { by: 'Customer007', rating: 2 } ]
}
```

$unset Operator

- The $unset operator deletes a particular field.

Syntax: { $unset: { <field1>: "", …} }

- The specified value in the $unset expression (i.e. "") does not impact the operation.
- To specify a <field> in an embedded document or in an array, use dot notation.
- If the field does not exist, then $unset does nothing (i.e. no operation).

```
db.products.insertMany( [
   { "item": "chisel", "sku": "C001", "quantity": 4, "instock": true },
   { "item": "hammer", "sku": "unknown", "quantity": 3, "instock": true },
   { "item": "nails", "sku": "unknown", "quantity": 100, "instock": true }
] )
```

- Update the *first* document in the products collection where the value of sku is unknown.

➢ remove the quantity field

➢ remove the instock field

```
db.products.insertMany( [
   { "item": "chisel", "sku": "C001", "quantity": 4, "instock": true },
   { "item": "hammer", "sku": "unknown", "quantity": 3, "instock": true },
   { "item": "nails", "sku": "unknown", "quantity": 100, "instock": true }
] )
```

Update the *first* document in the products collection where the value of sku is unknown.

- remove the quantity field
- remove the instock field

```
db.products.updateOne(
   { sku: "unknown" },
   { $unset: { quantity: "", instock: "" } }
)
```

```
{
  item: 'chisel',
  sku: 'C001',
  quantity: 4,
  instock: true
},
{
  item: 'hammer',
  sku: 'unknown'
},
{
  item: 'nails',
  sku: 'unknown',
  quantity: 100,
  instock: true
}
```

$currentDate

- The *$currentDate* operator sets the value of a field to the current date, either as a Date or a timestamp. The default type is Date.

Syntax: { $currentDate: { <field1>: <typeSpecification1>, ... } }

➢ <typeSpecification> can be either:

- a boolean true to set the field value to the current date as a Date, or
- a document { $type: "timestamp" } or { $type: "date" } which explicitly specifies the type. The operator is *case-sensitive* and accepts only the lowercase "timestamp" or the lowercase "date".

```
demo> db.customers.insertOne({ _id: 1, status: "a", lastModified: ISODate("2013-10-02T01:11:18.965Z")})
{ acknowledged: true, insertedId: 1 }
demo> db.customers.find()
[
  {
    _id: 1,
    status: 'a',
    lastModified: ISODate('2013-10-02T01:11:18.965Z')
  }
]
```

The following operation updates the lastModified field to the current date, the "cancellation.date" field to the current timestamp as well as updating the status field to "D" and the "cancellation.reason" to "user request"

```
demo> db.customers.updateOne({ _id: 1}, {$currentDate: {lastModified:true,
"cancellation.date":{$type:"timestamp"}},$set:{"cancellation.reason":"user
request",status:"D"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
demo> db.customers.find()
[
  {
    _id: 1,
    status: 'D',
    lastModified: ISODate('2024-02-13T16:07:02.322Z'),
    cancellation: {
      date: Timestamp({ t: 1707840422, i: 1 }),
      reason: 'user request'
demo>
  }
]
```

$min operator:

- The $min updates the value of the field to a specified value *if* the specified value is **less than** the current value of the field.

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

- If the field does not exist, the $min operator sets the field to the specified value.

Example 1: Use $min to Compare Numbers

```
db.scores.insertOne( { _id: 1, highScore: 800, lowScore: 200 } )
```

- The lowScore of the document with _id: 1 is updated to the specified value 200.

```
db.scores.updateOne( { _id: 1 }, { $min: { lowScore: 150 } } )
```

- This operation uses $min to compare 200 to the specified value 150 and updates the value of lowScore to 150.

```
{ _id: 1, highScore: 800, lowScore: 150 }
```

Example 2:

```
{ _id: 1, highScore: 800, lowScore: 150 }
```

Query:

```
db.scores.updateOne( { _id: 1 }, { $min: { lowScore: 250 } } )
```

- This operation has no effect since the current value of the field lowScore, i.e 200, is less than 250.
- Output:

```
{ _id: 1, highScore: 800, lowScore: 150 }
```

Use $min to Compare Dates

Create the tags collection:

```
db.tags.insertOne(
   {
     _id: 1,
     desc: "crafts",
     dateEntered: ISODate("2013-10-01T05:00:00Z"),
     dateExpired: ISODate("2013-10-01T16:38:16Z")
   }
)
```

```
db.tags.updateOne(
   { _id: 1 },
   { $min: { dateEntered: new Date("2013-09-25") } }
)
```

- This operation compares the current value of the dateEntered field, i.e. ISODate("2013-10-01T05:00:00Z"), with the specified date new Date("2013-09-25") to determine whether to update t

```
{
  _id: 1,
  desc: "crafts",
  dateEntered: ISODate("2013-09-25T00:00:00Z"),
  dateExpired: ISODate("2013-10-01T16:38:16Z")
}
```

$max Operator:

- The $max operator updates the value of the field to a specified value *if* the specified value is **greater than** the current value of the field.
- Syntax: { $max: { <field1>: <value1>, ... } }
- If the field does not exists, the $max operator sets the field to the specified value.

Use $max to Compare Numbers

```
db.scores.insertOne( { _id: 1, highScore: 800, lowScore: 200 } )
```

- The highScore for the document currently has the value 800. The following operation:

➢ Compares the highscore, 800, to the specified value, 950.

➢ Updates highScore to 950 since 950 is greater than 800.

```
db.scores.updateOne( { _id: 1 }, { $max: { highScore: 950 } } )
```

```
{ _id: 1, highScore: 950, lowScore: 200 }
```

Use $max to Compare Dates

Create the tags collection:

```
db.tags.insertOne(
   {
     _id: 1,
     desc: "crafts",
     dateEntered: ISODate("2013-10-01T05:00:00Z"),
     dateExpired: ISODate("2013-10-01T16:38:16.163Z")
   }
)
```

```
db.tags.updateOne(
   { _id: 1 },
   { $max: { dateExpired: new Date("2013-09-30") } }
)
```

- This operation compares the current value of the dateExpired field, ISODate("2013-10-01T16:38:16.163Z"), with the specified date new Date("2013-09-30") to determine whether to update the field.

```
{
   _id: 1,
   desc: "decorative arts",
   dateEntered: ISODate("2013-10-01T05:00:00Z"),
   dateExpired: ISODate("2013-10-01T16:38:16.163Z")
}
```

- new Date("2013-09-30") is not the newest date, so the operation does *not* update the dateExpired field.