Update in MongoDB

db.collection.update()

- Modifies an existing document or documents in a collection.
- The method can modify specific fields of an existing document or documents or replace an existing document entirely, depending on the update parameter.
- The syntax is as follows:

```
db.collection.update(query, update, options)
```

db.collection.update()

- This is an example of a document in a books collection:

```
{
  _id: 1,
  item: "TBD",
  stock: 0,
  info: { publisher: "1111", pages: 430 },
  tags: [ "technology", "computer" ],
  ratings: [ { by: "ijk", rating: 4 }, { by: "lmn", rating: 5 } ],
  reorder: false
}
```

db.collection.update()

- The following update statement is made up of two parts.
  - The first part is the query, which finds the document with an _id of 1.
  - The second part is the update itself. This uses the $set operator, and here we are setting the value of the stock property to 5.

```
db.books.update(
   { _id: 1 },
   { $set: { stock: 5 } }
)
```

Updating multiple values

- The following update statement sets several properties to new values.

```
db.books.update(
   { _id: 1 },
   {
     $set: {
       item: "ABC123",
       "info.publisher": "2222",
       tags: [ "software" ],
       "ratings.1": { by: "xyz", rating: 3 }
     }
   }
)
```

Update using the multi option

- Here we set the multi option to true. This means that multiple documents will be changed, if they meet the query criteria.

- You can specify several options by separating them with commas.

```
db.books.update(
    { stock: { $lte: 10 } },
    { $set: { reorder: true } },
    { multi: true }
)
```

Update using the upsert option

- Upsert = update / insert (update the document if it exists, insert the document if it doesn't)

- The following update sets the upsert option to true so that update() creates a new document in the books collection if no document matches the parameter:

```
db.books.update(
   { item: "ZZZ135" },
   {
     item: "ZZZ135",
     stock: 5,
     tags: [ "database" ]
   },
   { upsert: true }
)
```

Updating arrays using arrayFilters

- Consider a collection called students with the following documents:

```
db.students.insert([
   { "_id" : 1, "grades" : [ 95, 92, 90 ] },
   { "_id" : 2, "grades" : [ 98, 100, 102 ] },
   { "_id" : 3, "grades" : [ 95, 110, 100 ] }
])
```

Updating arrays using arrayFilters

- To update all elements that are greater than or equal to 100 in the grades array, use the filtered positional operator $[] with the arrayFilters option:

```
db.students.update(
   { grades: { $gte: 100 } },
   { $set: { "grades.$[element]" : 100 } },
   {
     multi: true,
     arrayFilters: [ { "element": { $gte: 100 } } ]
   }
)
```

Updating an array of documents

- Consider a collection called students2 with the following documents:

```
{
  "_id" : 1,
  "grades" : [
    { "grade" : 80, "mean" : 75, "std" : 6 },
    { "grade" : 85, "mean" : 90, "std" : 4 },
    { "grade" : 85, "mean" : 85, "std" : 6 }
  ]
}
{
  "_id" : 2,
  "grades" : [
    { "grade" : 90, "mean" : 75, "std" : 6 },
    { "grade" : 87, "mean" : 90, "std" : 3 },
    { "grade" : 85, "mean" : 85, "std" : 4 }
  ]
}
```

Updating an array of documents

- To modify the value of the mean field for all elements in the grades array where the grade is greater than or equal to 85, use the filtered positional operator $[] with the arrayFilters:

```
db.students2.update(
   { },
   { $set: { "grades.$[elem].mean" : 100 } },
   {
     multi: true,
     arrayFilters: [ { "elem.grade": { $gte: 85 } } ]
   }
)
```

updateOne(), updateMany()

- MongoDB provides two other methods for updates: updateOne() and updateMany()
- As with insert(), insertOne(), and insertMany(), the methods have similar functionality but aim to provide interoperability with different MongoDB drivers