# CRUD in MongoDB

https://docs.mongodb.com/guides/

## CRUD

**C**reate

```
db.collection.insert( <document> )
db.collection.save( <document> )
db.collection.update( <query>, <update>, { upsert: true } )
```

**R**ead

```
db.collection.find( <query>, <projection> )
db.collection.findOne( <query>, <projection> )
```

**U**pdate

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

**D**elete

```
db.collection.remove( <query>, <justOne> )
```

# Examples

## In RDBMS

```
CREATE TABLE users (
    id MEDIUMINT NOT NULL
        AUTO_INCREMENT,
    user_id Varchar(30),
    age Number,
    status char(1),
    PRIMARY KEY (id)
)
```

```
DROP TABLE users
```

## In MongoDB

**Either insert the 1st docuement**

```
db.users.insert( {
    user_id: "abc123",
    age: 55,
    status: "A"
 } )
```

**Or create “Users” collection explicitly**

```
db.createCollection("users")
```

```
db.users.drop()
```

https://docs.mongodb.com/manual/core/schema-validation/#schema-validation-json

## Insert one document

- From the mongo shell

- Switch to the moderndb database
  - use moderndb

```
db.inventory.insertOne(
   { "item" : "canvas",
     "qty" : 100,
     "tags" : ["cotton"],
     "size" : { "h" : 28, "w" : 35.5, "uom" : "cm" }
   }
)
```

# Update

```
db.users.update(                              ←  collection
    { age: { $gt: 18 } },                     ←  update criteria
    { $set: { status: "A" } },                ←  update action
    { multi: true }                           ←  update option
)
```

Otherwise, it will update only the 1st matching document

**Equivalent to in SQL:**

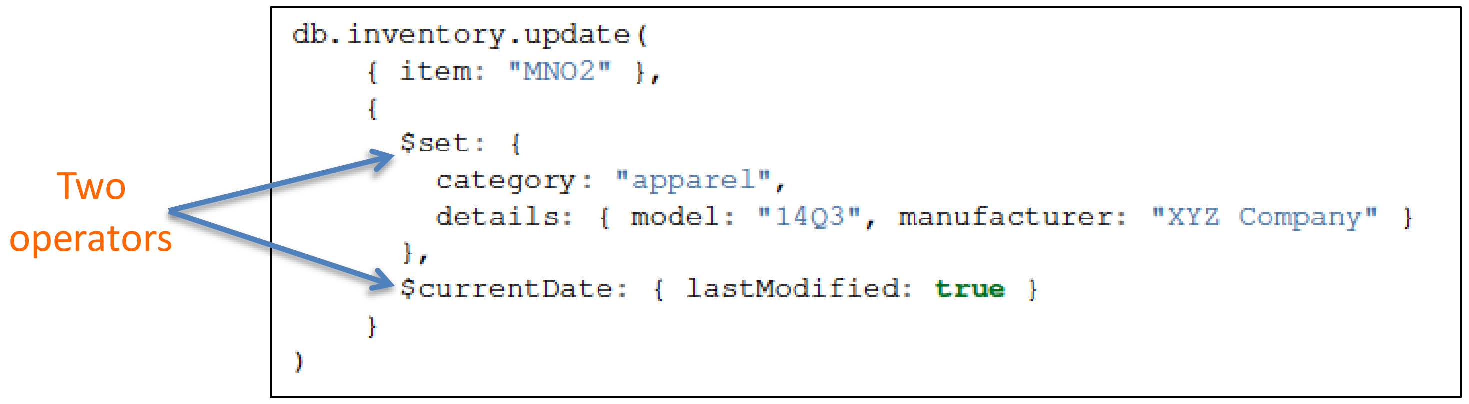
```
UPDATE users                ←  table
SET    status = 'A'         ←  update action
WHERE  age > 18             ←  update criteria
```

## UpdateOne - UpdateMany

```
db.inventory.updateOne(
   { "item" : "paper" }, // specifies the document to update
   {
     $set: {  "size.uom" : "cm",  "status" : "P" },
     $currentDate: { "lastModified": true }
   }
)


db.inventory.updateMany(
    { "qty" : { $lt: 50 } }, // specifies the documents to update
    {
      $set: { "size.uom" : "cm", "status": "P" },
      $currentDate : { "lastModified": true }
    }
)
```

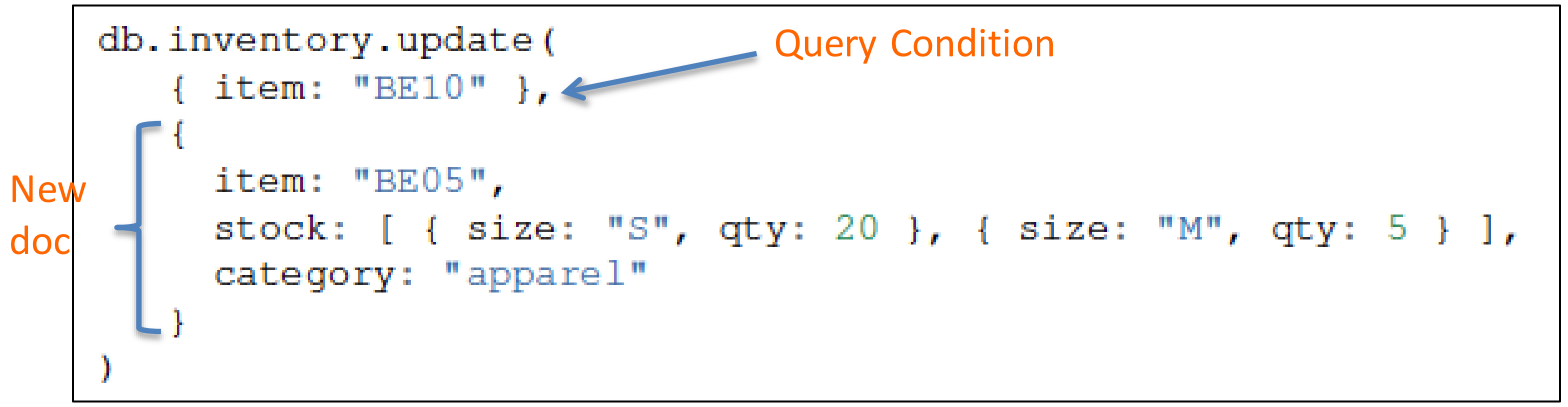
# Update (Cont'd)



```
db.inventory.update(
    { item: "MNO2" },
    {
      $set: {
        category: "apparel",
        details: { model: "14Q3", manufacturer: "XYZ Company" }
      },
      $currentDate: { lastModified: true }
    }
)
```

For the document with `item` equal to `"MNO2"`, use the `$set` operator to update the `category` field and the `details` field to the specified values and the `$currentDate` operator to update the field `lastModified` with the current date.

# Replace a document



```
db.inventory.update(
    { item: "BE10" },
    {
      item: "BE05",
      stock: [ { size: "S", qty: 20 }, { size: "M", qty: 5 } ],
      category: "apparel"
    }
)
```

For the document having item = “BE10”, replace it with the given document

# Insert or Replace

```
db.inventory.update(
   { item: "TBD1" },
   {
     item: "TBD1",
     details: { "model" : "14Q4", "manufacturer" : "ABC Company" },
     stock: [ { "size" : "S", "qty" : 25 } ],
     category: "houseware"
   },
   { upsert: true }
)
```

The ***upsert*** option

If the document having item = “TBD1” is in the DB, it will be replaced
Otherwise, it will be inserted.

## Delete

- Deletes the first document that matches the condition

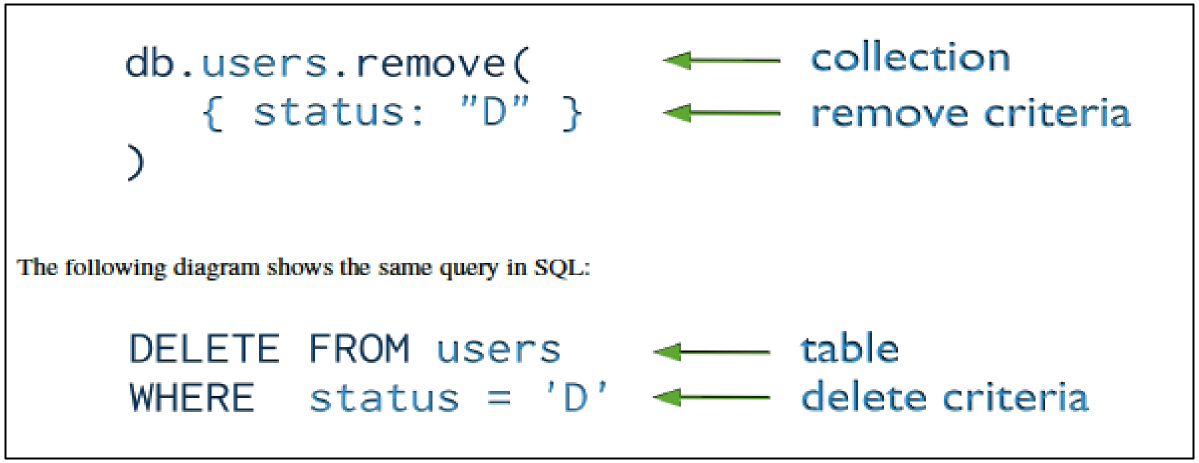
```
db.inventory.deleteOne(
    { "status": "D" } // specifies the document to delete
)
```

- Deletes ALL documents that match the condition

```
db.inventory.deleteMany(
    { "status" : "A" } // specifies the documents to 
delete
)
```

## Remove (also delete)

You can put condition on any field in the document (even *_id*)



```
db.users.remove(
   { status: "D" }
)
```

The following diagram shows the same query in SQL:

```
DELETE FROM users
WHERE  status = 'D'
```

db.users.remove ( ) → Removes all documents from *users* collection

# Import json file to MongoDB

https://docs.mongodb.com/guides/server/import/

Download the file:
https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/inventory.crud.json

```
mongoimport --db moderndb --collection inventory
        --drop --file ~\downloads\inventory.crud.json
```

Or if you enabled authentication:

```
mongoimport --db moderndb --collection inventory
        --authenticationDatabase admin --username <user>
        --password <password>  --drop
        --file ~\downloads\inventory.crud.json
```

## References in Mongo

- Manual references is the practice of including one document’s `_id` field in another document. The application can then issue a second query to resolve the referenced fields as needed

- DBRefs are references from one document to another using the value of the first document’s _id field, collection name, and, optionally, its database name.

```
db.inventory.update(
{ item : “paper" },
{ $set : { country: { $ref: "countries", $id: "us" } } }
)
```

## Retrieving references

```
var paper = db. inventory.findOne({ item : “paper" })
```

Retrieve country, to query the countries collection using the stored $id.

```
db.countries.findOne({ _id: paper.country.$id })
```

Better yet, in JavaScript, you can ask the document the name of the collection stored in the fields reference.

```
var paperCountryRef = paper.country.$ref;
db[paperCountryRef].findOne({ _id: paper.country.$id })
```

The last two queries are equivalent; the second is just a bit more data-driven.

## Querying with code

- You can request that MongoDB run a decision function across your documents
- Should be a last resort, this queries cannot be indexed, Mongo do not optimize them

```
db.inventory.find(function() {
    return this.qty > 50 && this.qty < 100;
})
```

- You can also use the $where clause

```
db.inventory.find({$where: “this.qty > 50 && this.qty < 100”})
```

## The _id index

- Mongo automatically creates an index by the _id

```
 db.inventory.getIndexes()
```

```
db.getCollectionNames().forEach(function(collection) {
    print("Indexes for the " + collection + " collection:");
    printjson(db[collection].getIndexes());
});
```

Let’s import the city_inspections.json collection from ICON into the moderndb database, on a new collection called city_inspections

```
db.city_inspections.find({certificate_number:
10003581}).explain("executionStats").executionStats
```

# Profiler

- *System profiler* allows to profile queries in a normal test or production environment
  - Level 1 stores only slower queries greater than 100 milliseconds
  - Level 2 stores all queries

```
db.setProfilingLevel(2)
db.city_inspections.find({certificate_number: 10003581})
```

This will create a new object in the system.profile collection, which you can read as any other table to get information about the query, such as a timestamp for when it took place and performance information (such as executionTimeMillis-Estimate as shown). You can fetch documents from that collection like any other:

```
db.system.profile.find()
```

## For today..

- Create a new database named blogger with a collection named articles. Insert a new article with an author name and email, creation date, and text.
- Update the article with an array of comments, containing a comment with an author and text.
- Summit the two statements to ICON.