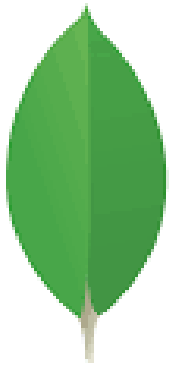
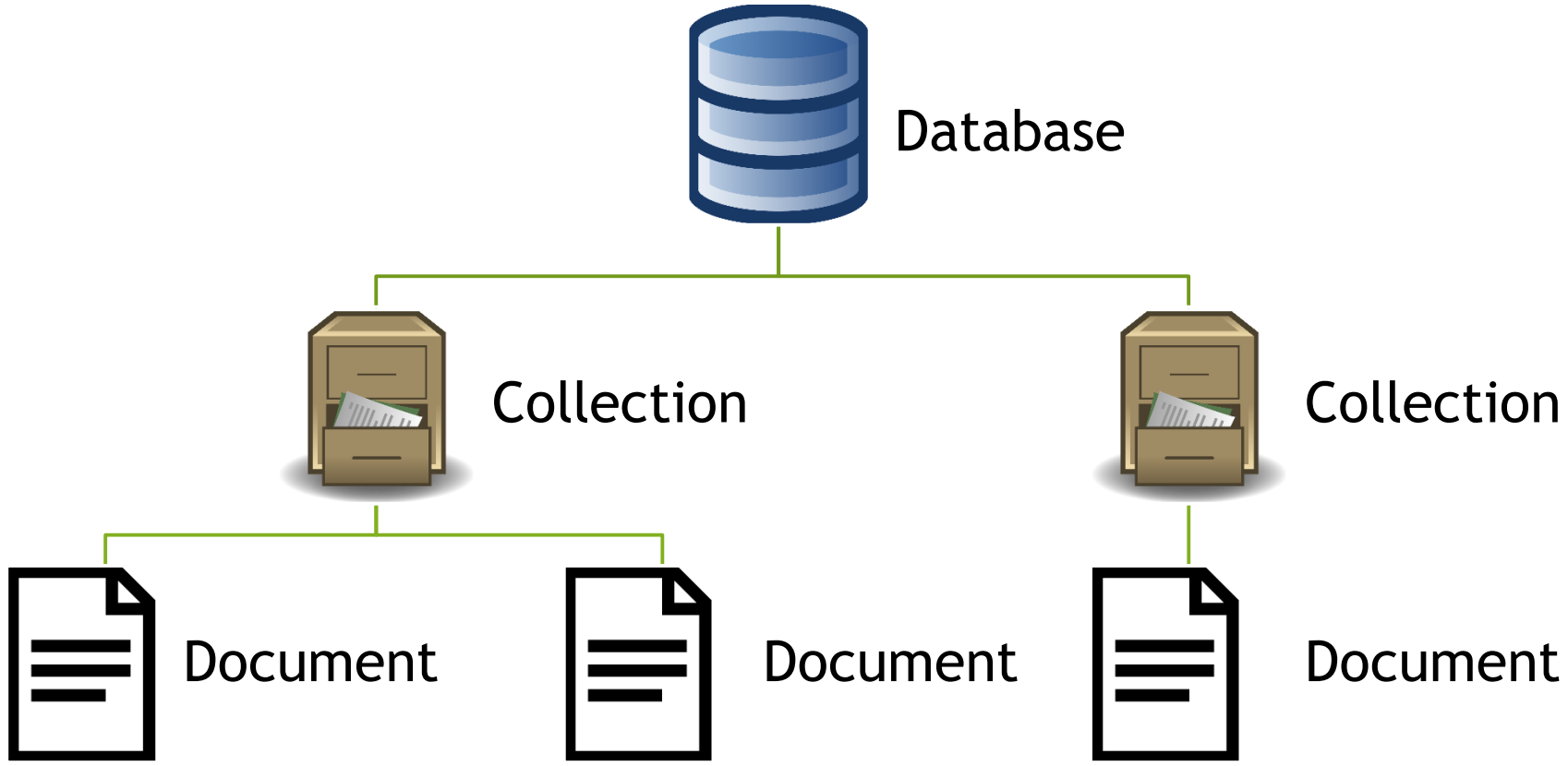


mongoDB®

C9b-Basic NoSQL

CRUD operations

MongoDB: Document-based NoSQL Database



SQL Server	MongoDB
Database	Database
Table	Collection
Index	Index
Row	Document
Column	Field

Collection

- Collections are sets of (usually) related documents.
- You can have as many collections as you like.
- Because Mongo has no joins, a Mongo query can pull data from only one collection at a time.

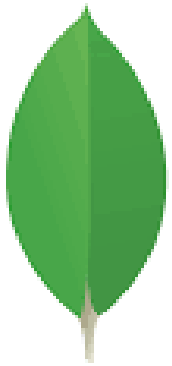
Document

- Documents are JSON objects that live inside a collection.
- Each document contains key-value pairs (similar to Dictionary in Python) where the keys are fields, and values are data associated with the fields.
- The size limit for a document is 16MB which is more than ample for most use cases.

```
{
    "_id": ObjectID("5e9a568b12de6ebf44ce8ffe"),
    "ID": 1,
    "Name": "John",
    "Age": 25
}
```

Syntax: {
field1: data1,
field2: data2,
…
}

- Each document is identified using a unique key.
- The _id field is automatically indexed when document is inserted into the database.
- IDs are 12 byte BSON objects, not Strings which is why we need the ObjectID function.



mongoDB®

Create

CRUD operations

Create new database or access an existing database

1. use:

- Select a particular database to access, e.g. class_info. This will create class_info if it does not already exist:

  Syntax: use DATABASE_NAME

  - use class_info
  - show dbs

```
> use class_info
switched to db class_info
> show dbs
admin   0.000GB
local   0.000GB
```

Note: Since `class_info` is empty, it does not show up

Create a new collection (1st method):

2. createCollection()

- After selecting a particular database, create a collection using createCollection() method:

  Syntax: db.createCollection(COLLECTION_NAME)

  - db.createCollection('students')
  - show collections ##display all collections associated with current db
  - show dbs

```
> db
class_info
> db.createCollection('students')
{ "ok" : 1 }
> show collections
students
```

```
> show dbs
admin          0.000GB
class_info     0.000GB
local          0.000GB
```

now `class_info` db will show up as it is no longer empty

Create a new collection (2nd method):

3. insert()

- Create a new collection by using insert method
- If the specified collection name does not exist, then it will be created.

Syntax: db.COLLECTION_NAME.insert(document)

- db.teachers.insert({name: “Miss Tan”, subject: “Math”})
- show collections ##display all collections associated with current db

```
>  db.teachers.insert({name:'Miss Tan',subject:'Math'})
WriteResult({ "nInserted" : 1 })
> show collections
students
teachers
```

Insert documents

4. insert()

- Documents can be inserted into a collection using insert() method, similar to previous.

```
> db.students.insert(
    {
        name: ‘Lynn’,
        gender: ‘F’,
        age: 17,
        hobbies:[‘singing’, ‘dancing’, ‘gaming’]
    }
)
```

```
> db
class_info
> db.students.insert({name:'Lynn',gender:'F',age:17,hobbies:['singing','dancing','gaming']})
WriteResult({ "nInserted" : 1 })
```

Insert documents

5. insertOne()

Use insertOne to insert only one record

Syntax: db.COLLECTION_NAME.insertOne(document)

```
db.students.insertOne(
        {
            name: ‘John’,
            gender: ‘M’,
            age: 18,
            hobbies: [‘soccer’, ‘gaming’]
        }
)
```

```
> db.students.insertOne({name:'John',gender:'M',age:18,hobbies:['gaming','soccer']})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5e844639c6285bb5290c2b16")
}
```

Insert documents

6. insertMany()

- Use insertMany to insert multiple records

  Syntax: db.COLLECTION_NAME.insertMany(ARRAY : document)

```
> db.teachers.insertMany([{name:'Mr Lim',subject:'Phy'},{name:'Mr Lee',subject:'Chem'}
])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5e9aaa9712de6ebf44ce9007"),
                ObjectId("5e9aaa9712de6ebf44ce9008")
        ]
}
```

```
db.teachers.insertMany(
        [
                {
                        name: 'Mr Lim',
                        subject: 'Phy'
                },
                {
                        name: 'Mr Lee',
                        subject: 'Chem'
                }
        ]
)
```

Exercise 1

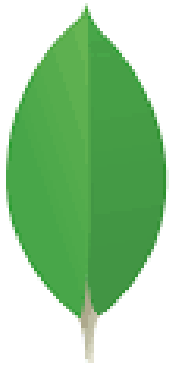
Create a petshop NoSQL database

- Containing the collection “pets”, “customers”
- Insert these details into pets and customers collections respectively:

name	species
Mikey	Piranha
Davey	Goldfish
Suzy	Cat
Mikey	Dog
Terry	Dog
Mimi	Cat

name	email
John	john@gmail.com
Mary	mary@gmail.com

- Run show dbs and show collections to view your database and collections.
- Submit your screenshot



mongoDB®

Read

CRUD operations

Read all documents in a collection

1. find():

- Display all documents in a collection.
- Similar to SELECT * FROM <table> in SQL.

  Syntax: db.COLLECTION_NAME.find( )

  - db.students.find( )

```
> db.students.find()
{ "_id" : ObjectId("5e9aa99212de6ebf44ce9005"), "name" : "Lynn", "gender" : "F", "age" : 17,
"hobbies" : [ "singing", "dancing", "gaming" ] }
{ "_id" : ObjectId("5e9aa9cb12de6ebf44ce9006"), "name" : "John", "gender" : "M", "age" : 18,
"hobbies" : [ "soccer", "gaming" ] }
```

```
> db.students.find(ObjectId("5e9aa99212de6ebf44ce9005"))
{ "_id" : ObjectId("5e9aa99212de6ebf44ce9005"), "name" : "Lynn", "gender" : "F", "age" : 17,
"hobbies" : [ "singing", "dancing", "gaming" ] }
```

Read document by ObjectID

2. find(<ObjectID>):

- Display the document in a collection based on unique key.
- Syntax: db.COLLECTION_NAME.find(<ObjectID>)
  - db.students.find(ObjectId("5e9aa99212de6ebf44ce9005"))

Replace this with the ObjectID generated by your Mongo Server

```
> db.students.find(ObjectId("5e9aa99212de6ebf44ce9005"))
{ "_id" : ObjectId("5e9aa99212de6ebf44ce9005"), "name" : "Lynn", "gender" : "F", "age" : 17,
"hobbies" : [ "singing", "dancing", "gaming" ] }
```

Read documents that fulfils a criterion

3. Search and display documents in a collection that fulfils a criterion:

- Similar to SELECT * FROM <table> WHERE <criteria> in SQL.
- Note that the criteria is expressed in JSON format as well
- The following is an example to query for documents that fulfil specific equality condition where name equals to Lynn.

  Syntax: db.COLLECTION_NAME.find(criteria)

  - db.students.find({name: 'Lynn'})

```
> db.students.find({name:'Lynn'})
{ "_id" : ObjectId("5e9aa99212de6ebf44ce9005"), "name" : "Lynn", "gender" : "F", "age" : 17,
"hobbies" : [ "singing", "dancing", "gaming" ] }
```

Read documents that fulfils > 1 criteria

4. Display documents in a collection that fulfils certain criteria:

- Similar to SELECT * FROM <table> WHERE <criteria1> AND <criteria2> AND … in SQL.
- The following is an example to query for documents that fulfil specific equality condition where name equals to Lynn and age equals to 17.

  Syntax: db.COLLECTION_NAME.find(criteria)

  - db.students.find({name: 'Lynn', age: 17})

```
> db.students.find({name:'Lynn', age:17})
{ "_id" : ObjectId("5e9aa99212de6ebf44ce9005"), "name" : "Lynn", "gender" : "F", "age" : 17,
"hobbies" : [ "singing", "dancing", "gaming" ] }
```

Read only one documents that fulfils certain criteria

5. Display only the first document in a collection that fulfils certain criteria:

- Similar to SELECT * FROM <table> WHERE <criteria1> AND <criteria2> LIMIT 1 in SQL.

  Syntax: db.COLLECTION_NAME.findOne(criteria)

  - db.students.findOne({name: 'Lynn', age: 17})

```
> db.students.findOne({name:'Lynn', age:17})
{
        "_id" : ObjectId("5e9aa99212de6ebf44ce9005"),
        "name" : "Lynn",
        "gender" : "F",
        "age" : 17,
        "hobbies" : [
                "singing",
                "dancing",
                "gaming"
        ]
}
```

Notice that this is a "prettier" display compared to the usual

Read all documents in a collection pretty

6. pretty():

- Display documents in a pretty format.
- Follows after a find({…}) operation
- Syntax:

db.COLLECTION_NAME.find( ).pretty()

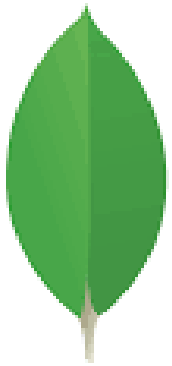
- db.students.find( ).pretty( )

```
> db.students.find().pretty()
{
        "_id" : ObjectId("5e9aa99212de6ebf44ce9005"),
        "name" : "Lynn",
        "gender" : "F",
        "age" : 17,
        "hobbies" : [
                "singing",
                "dancing",
                "gaming"
        ]
}
{
        "_id" : ObjectId("5e9aa9cb12de6ebf44ce9006"),
        "name" : "John",
        "gender" : "M",
        "age" : 18,
        "hobbies" : [
                "soccer",
                "gaming"
        ]
}
```

Exercise 2

Using the petshop db created in Exercise 1,

- Add another piranha called Henry.
- List all the pets. Find the ID of Mikey the Dog.
- Use find to find Mikey by id.
- Use find to find all the cats.
- Find all the creatures named Mikey.
- Find all the creatures named Mikey who are piranha.
- Submit your screenshot



mongoDB®

Query operators

CRUD operations

Insert documents

- Insert these 5 records to students collection, building on the class_info db
- Note that the documents from the same collection can have a different schema from one another

1. { name: “Kate”, gender: ‘F’, age: 16, cca: “tennis” }
2. { name: “Ernest”, gender: ‘M’, age: 17, cca: “choir”, hobbies: [‘singing’] }
3. { name: ‘Sam’, gender: ‘M’, age: 16, hobbies: [‘running’, ‘bowling’] }
4. { name: ‘Amy’, gender: ‘F’, hobbies: [‘drawing’, ‘painting’] }
5. { name: ‘Raul’, gender: ‘M’}

Comparison query operators table

No	Name	Description	example syntax
1	$eq	Matches values that are equal to a specified value.	.find({age: { $eq : 17 } })
2	$gt	Matches values that are greater than a specified value.	.find({age: { $gt : 17 } })
3	$gte	Matches values that are greater than or equal to a specified value.	.find({age: { $gte : 17 } })
4	$in	Matches any of the values specified in an array.	.find({age: { $in : [15, 16, 17] } })
5	$lt	Matches values that are less than a specified value.	.find({age: { $lt : 17 } })
6	$lte	Matches values that are less than or equal to a specified value.	.find({age: { $lte : 17 } })
7	$ne	Matches all values that are not equal to a specified value.	.find({age: { $ne : 17 } })
8	$nin	Matches none of the values specified in an array, i.e. not in the array	.find({age: { $nin : [18,17] } })

Comparison Query Operators

1. Comparison query operators:

- For **all** the comparison operators, write a query in MongoDB to test on the class_info db
- e.g: db.students.find({hobbies: {$in: ['singing', 'gaming']}})

```
> db.students.find({hobbies:{$in:['singing','gaming']}}).pretty()
{
        "_id" : ObjectId("5e9aa99212de6ebf44ce9005"),
        "name" : "Lynn",
        "gender" : "F",
        "age" : 17,
        "hobbies" : [
                "singing",
                "dancing",
                "gaming"
        ]
}
{
        "_id" : ObjectId("5e9aa9cb12de6ebf44ce9006"),
        "name" : "John",
        "gender" : "M",
        "age" : 18,
        "hobbies" : [
                "soccer",
                "gaming"
        ]
}
```

```
{
        "_id" : ObjectId("5e9af79e12de6ebf44ce900a"),
        "name" : "Ernest",
        "gender" : "M",
        "age" : 17,
        "cca" : "choir",
        "hobbies" : [
                "singing"
        ]
}
```

Logical query operators

No	Name	Description	example syntax
1	$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.	.find({ $and: [{age: { $gte : 17 } } , {gender: “M” }] })
2	$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.	.find({ $or: [{age: { $lt : 18 } } , {gender: “M” }] })
3	$not	Inverts the effect of a query expression and returns documents that do *not* match the query expression.	.find({ name: { $not: { $eq: “Lynn” } } })

Logical Query Operators

2. Logical query operators:

- For **all** the logical operators, write a query in MongoDB to test on the class_info db
- e.g: db.students.find({ $and: [{age: { $gte : 17 } } , {gender: “M” }])

```
> db.students.find({$and:[{age:{$gte:17}},{gender:'M'}]}).pretty()
{
        "_id" : ObjectId("5e9aa9cb12de6ebf44ce9006"),
        "name" : "John",
        "gender" : "M",
        "age" : 18,
        "hobbies" : [
                "soccer",
                "gaming"
        ]
}
{
        "_id" : ObjectId("5e9af79e12de6ebf44ce900a"),
        "name" : "Ernest",
        "gender" : "M",
        "age" : 17,
        "cca" : "choir",
        "hobbies" : [
                "singing"
        ]
}
```

Element query operators

No	Name	Description	example syntax
1	$exists	Matches documents that have the specified field.	.find({ cca: { $exists: true } })
		Finds all documents where the specified field does not exist.	.find({ cca: { $exists: false } })
2	$type	Selects documents if a field is of the specified type. $type is useful when querying highly unstructured data where data types are not predictable. The standard types in MongoDB are: “string”, “array”, “double” and “object”.	.find({ hobbies: { $type: “array” } })

Element Query Operator

3. Element query operators:

- For all the element operators, write queries in MongoDB to test on the class_info db,
- Combine its use with the other query operators
- e.g: db.students.find({ $and :[{ cca: { $exists: true}}, {age: { $gte : 17 } }] })

```
> db.students.find({$and:[{cca:{$exists:true}},{age:{$gte:17}}]}).pretty()
{
        "_id" : ObjectId("5e9af79e12de6ebf44ce900a"),
        "name" : "Ernest",
        "gender" : "M",
        "age" : 17,
        "cca" : "choir",
        "hobbies" : [
                "singing"
        ]
}
```

Insert documents

- Create a new collection results
- Insert these 5 records to results collection
- Note that the documents from the same collection can have a different schema from one another

1. { name: “Kate”, scores: [{ phy : 62 } , { math : 54 } , { chem : 71}] }
2. { name: “Ernest”, scores: [{ bio : 58 } , { math : 76} , { chem : 45}] }
3. { name: ‘Sam’, scores: [{ comp : 67 } , { math : 68 } , { chem : 46}] }
4. { name: ‘Amy’, scores: [{ math : 66 } , { hist : 74} , { art : 75}] }
5. { name: ‘Raul’, scores: [{ phy : 35 } , { math : 44} , { chem : 49}] }

Array query operators

No	Name	Description	example syntax
1	$elemMatch	Matches documents that contain an array field with at least one element that matches all the specified query criteria.	.find({ scores: { $elemMatch: { math: {$gte:60 } } } })
	nested attribute		.find({ “scores.math”: {$gte:60 } })

Array Query Operator

4. Array query operator:

- For the $elemMatch array operator, write queries in MongoDB to test on the class_info db,
- Combine its use with the other query operators

e.g: db.results.find({ scores: { $elemMatch: { math: {$gte:60 } } } })

```
> db.results.find({scores:{$elemMatch:{math:{$gte:60}}}})
{ "_id" : ObjectId("5e9bb1e5f82ddbb0af0bae7f"), "name" : "Ernest"
, "scores" : [ { "phy" : 58 }, { "math" : 76 }, { "chem" : 45 } ]
 }
{ "_id" : ObjectId("5e9bb1e5f82ddbb0af0bae80"), "name" : "Sam", "
scores" : [ { "comp" : 67 }, { "math" : 68 }, { "chem" : 46 } ] }

{ "_id" : ObjectId("5e9bb1e5f82ddbb0af0bae81"), "name" : "Amy", "
scores" : [ { "math" : 66 }, { "hist" : 74 }, { "art" : 75 } ] }
```

Exercise 3

For the class_info database, write queries to find students who are:

- Female
- Female or sing as a hobby
- Male and without cca
- cca-less and hobby-less
- Male and above the age of 16 and has a cca
- 17 years old and above and does not like gaming.
- Scoring less than 50 marks in Chemistry
- Submit your screenshot

Query methods

No	Name	Description	example syntax	Outcome
1	limit()	displays only the number of documents specified within the limit	.find().limit(1)	Only the first document displayed
2	skip()	Skips the specified number of documents to be displayed	.find().limit(3).skip(1)	First record is skipped, display 2nd, 3rd and 4th document
3	sort()	Displays documents in sorted order as specified by the field(s) and sort order: 1 for ascending, -1 for descending	.find().sort({age:1})	documents sorted according to age, in ascending order.
4	count()	Count the number of documents	.find().count()	Returns the number of documents

Query methods

5. Limit, Skip, Sort, Count methods:

- For the query methods, write queries in MongoDB to test on the class_info db,
- Combine its use with the other query operators or methods

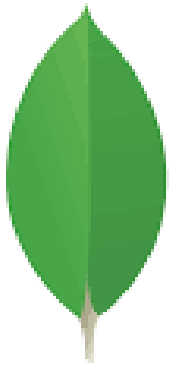
e.g: db.results.find({ scores: { $elemMatch: { math: {$gte:60 } } } }).sort({name:1})

```
> db.results.find({scores:{$elemMatch:{math:{$gte:60}}}}).sort({name:1})
{ "_id" : ObjectId("5e9bb1e5f82ddbb0af0bae81"), "name" : "Amy", "scores" : [ { "math" : 66 }, { "hist" : 74 }, { "art" : 75 } ] }
{ "_id" : ObjectId("5e9bb1e5f82ddbb0af0bae7f"), "name" : "Ernest", "scores" : [ { "phy" : 58 }, { "math" : 76 }, { "chem" : 45 } ] }
{ "_id" : ObjectId("5e9bb1e5f82ddbb0af0bae80"), "name" : "Sam", "scores" : [ { "comp" : 67 }, { "math" : 68 }, { "chem" : 46 } ] }
```

Exercise 4

For the class_info database, write queries to find :

- Female students, sorted in ascending order of age, skipping the first female
- Male students, sorted in alphabetical order of name, limited to 2
- Students who failed chemistry (<50 marks) scores sorted in descending order
- Number of students who scored more than 50 for mathematics
- Top 2 students in mathematics
- Submit your screenshot



mongoDB.

Update operators

CRUD operations

Update operations

1. update():

- Update an existing record/document

  Syntax: db.COLLECTION_NAME.update(criteria, update, options)

  - db.teachers.update({name: 'Mr Lee'}, { {name: 'Mr Lee'}, {subject: 'Comp'} })

```
> db.teachers.find()
{ "_id" : ObjectId("5e9aa90112de6ebf44ce9004"), "name" : "Miss Tan", "subject" : "Math" }
{ "_id" : ObjectId("5e9aaa9712de6ebf44ce9007"), "name" : "Mr Lim", "subject" : "Phy" }
{ "_id" : ObjectId("5e9aaa9712de6ebf44ce9008"), "name" : "Mr Lee", "subject" : "Chem" }
> db.teachers.update({name:"Mr Lee"},{name:"Mr Lee",subject:"Comp"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.teachers.find()
{ "_id" : ObjectId("5e9aa90112de6ebf44ce9004"), "name" : "Miss Tan", "subject" : "Math" }
{ "_id" : ObjectId("5e9aaa9712de6ebf44ce9007"), "name" : "Mr Lim", "subject" : "Phy" }
{ "_id" : ObjectId("5e9aaa9712de6ebf44ce9008"), "name" : "Mr Lee", "subject" : "Comp" }
```

Update operations

2. update() with upsert:

- If upsert set to true, creates a new document when no document matches the query criteria. The default value is false, which does not insert a new document when no match is found.
- When multiple clients issue the following same update, setting upsert:true prevents the same document from being inserted more than once.
- db.teachers.update({name: 'Mr Chua'}, { {name: 'Mr Chua'}, {subject: 'Chem' } , {upsert: true} })

```
> db.teachers.update({name:'Mr Chua'},{name:'Mr Chua',subject:"Chem"},{upsert:true})
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("5e9b10dd687985a29f75eb50")
})
> db.teachers.find()
{ "_id" : ObjectId("5e9aa90112de6ebf44ce9004"), "name" : "Miss Tan", "subject" : "Math" }
{ "_id" : ObjectId("5e9aaa9712de6ebf44ce9007"), "name" : "Mr Lim", "subject" : "Phy" }
{ "_id" : ObjectId("5e9aaa9712de6ebf44ce9008"), "name" : "Mr Lee", "subject" : "Comp" }
{ "_id" : ObjectId("5e9b10dd687985a29f75eb50"), "name" : "Mr Chua", "subject" : "Chem" }
```

Update operations

3. $set operator:

- Using update and $set to update specified fields of an existing document
- db.teachers.update({name: ‘Mr Chua’}, {$set : {subject: ‘Bio’ } })

```
> db.teachers.update({name:'Mr Chua'},{$set:{subject:'Bio'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.teachers.find({name:'Mr Chua'})
{ "_id" : ObjectId("5e9b10dd687985a29f75eb50"), "name" : "Mr Chua", "subject" : "Bio" }
```

Update operations

4. updateOne():

- If the query finds more than one record that fulfil the criteria, only update the first occurrence
- Similar to setting multi:false as option when using update()

Syntax: db.COLLECTION_NAME.updateOne(criteria, update)

db.students.updateOne({age: {$exists:false} }, {$set: {age: 17} })

```
> db.students.updateOne({age:{$exists:false}},{$set:{age:17}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.students.find()
{ "_id" : ObjectId("5e9aa99212de6ebf44ce9005"), "name" : "Lynn", "gender" : "F", "age" : 17, "hobbies" : [ "singing",
"dancing", "gaming" ] }
{ "_id" : ObjectId("5e9aa9cb12de6ebf44ce9006"), "name" : "John", "gender" : "M", "age" : 18, "hobbies" : [ "soccer", "
gaming" ] }
{ "_id" : ObjectId("5e9af79e12de6ebf44ce9009"), "name" : "Kate", "gender" : "F", "age" : 16, "cca" : "tennis" }
{ "_id" : ObjectId("5e9af79e12de6ebf44ce900a"), "name" : "Ernest", "gender" : "M", "age" : 17, "cca" : "choir", "hobbi
es" : [ "singing" ] }
{ "_id" : ObjectId("5e9af79e12de6ebf44ce900b"), "name" : "Sam", "gender" : "M", "age" : 16, "hobbies" : [ "running", "
bowling" ] }
{ "_id" : ObjectId("5e9af79e12de6ebf44ce900c"), "name" : "Amy", "gender" : "F", "hobbies" : [ "drawing", "painting" ],
 "age" : 17 }
{ "_id" : ObjectId("5e9af79e12de6ebf44ce900d"), "name" : "Raul", "gender" : "M" }
```

Update operations

5. updateMany():

- Updates all matching records
- Similar to setting multi:false as option when using update()

Syntax: db.COLLECTION_NAME.updateMany(criteria, update)

- db.teachers.update({name: { $exists : true } }, { $set: { position: 'HOD'})

```
> db.teachers.updateMany({name:{$exists:true}},{$set:{position:'HOD'}})
{ "acknowledged" : true, "matchedCount" : 4, "modifiedCount" : 4 }
> db.teachers.find()
{ "_id" : ObjectId("5e9aa90112de6ebf44ce9004"), "name" : "Miss Tan", "subject" : "Math", "position" : "HOD" }
{ "_id" : ObjectId("5e9aaa9712de6ebf44ce9007"), "name" : "Mr Lim", "subject" : "Phy", "position" : "HOD" }
{ "_id" : ObjectId("5e9aaa9712de6ebf44ce9008"), "name" : "Mr Lee", "subject" : "Comp", "position" : "HOD" }
{ "_id" : ObjectId("5e9b10dd687985a29f75eb50"), "name" : "Mr Chua", "subject" : "Bio", "position" : "HOD" }
```

Update operations

6. $unset:

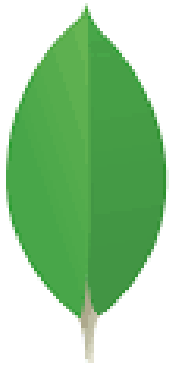
- The $unset operator deletes or removes a particular field.
- The values of that field has no effect, so use an empty string for convenience.
- db.teachers.update({name: 'Mr Lee' }, { $unset: { position: ''})

```
> db.teachers.find()
{ "_id" : ObjectId("5e9aa90112de6ebf44ce9004"), "name" : "Miss Tan", "subject" : "Math", "position" : "HOD" }
{ "_id" : ObjectId("5e9aaa9712de6ebf44ce9007"), "name" : "Mr Lim", "subject" : "Phy", "position" : "HOD" }
{ "_id" : ObjectId("5e9aaa9712de6ebf44ce9008"), "name" : "Mr Lee", "subject" : "Comp", "position" : "HOD" }
{ "_id" : ObjectId("5e9b10dd687985a29f75eb50"), "name" : "Mr Chua", "subject" : "Bio", "position" : "HOD" }
> db.teachers.update({name:'Mr Lee'},{$unset:{position:''}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.teachers.find()
{ "_id" : ObjectId("5e9aa90112de6ebf44ce9004"), "name" : "Miss Tan", "subject" : "Math", "position" : "HOD" }
{ "_id" : ObjectId("5e9aaa9712de6ebf44ce9007"), "name" : "Mr Lim", "subject" : "Phy", "position" : "HOD" }
{ "_id" : ObjectId("5e9aaa9712de6ebf44ce9008"), "name" : "Mr Lee", "subject" : "Comp" }
{ "_id" : ObjectId("5e9b10dd687985a29f75eb50"), "name" : "Mr Chua", "subject" : "Bio", "position" : "HOD" }
```

Exercise 5

Write statements to make the following changes to class_info:

- Add Raul's age as 19
- Add on to Amy's hobbies by including knitting
- For all males age 18 and above, include a new field enlist, and set it to false
- For those without cca field, include it and set it to empty string
- Change position of Physics teacher to subject head
- Remove the position field for all teachers except for Miss Tan
- Change the chemistry score for Raul to 39
- Submit your screenshot



mongoDB®

Delete operators

CRUD operations

Delete a document

1. remove( ):

- Delete/remove all documents matching the query criteria
- To delete all documents, pass { } as the argument
- The <justOne> option is false by default, hence all documents that matches the query criteria will be removed.
  - Syntax: db.COLLECTION_NAME.remove(criteria, <justOne>)
  - db.teachers.remove({name: 'Mr Lim'})

```
> db.teachers.remove({name:'Mr Lim'})
WriteResult({ "nRemoved" : 1 })
```

Delete a document

2. deleteOne( ):

- Delete/remove the first document that matches the query criteria.
- Similar to setting <justOne> option as 1 or true
- Syntax: db.COLLECTION_NAME.deleteOne(criteria)
  - db.students.deleteOne({gender: 'F'})

```
> db.students.deleteOne({gender:'F'})
{ "acknowledged" : true, "deletedCount" : 1 }
```

Delete a document

3. deleteMany( ):

- Delete/remove all documents that matching the query criteria.
- Similar to remove( )
- Syntax: db.COLLECTION_NAME.deleteMany(criteria)
  - db.students.deleteMany({age: 16})

```
> db.students.deleteMany({age:16})
{ "acknowledged" : true, "deletedCount" : 2 }
```

Drop a collection

4. drop():

- Drop the entire collection
  - Syntax: db.COLLECTION_NAME.drop ()
  - db.teachers.drop()
  - show collections

```
> db.teachers.drop()
true
> show collections
students
```

Exercise 6

Write statements to make the following changes to class_info:

- Remove all female students
- Remove the youngest male student
- Remove the result with the worst math score
- Drop results collection
- Submit your screenshot

Drop a MongoDB Database

5. dropDatabase():

- Select a particular database to access, e.g. class_info.

  Syntax: db.dropDatabase()

- Steps to drop a database
  - show dbs #show all databases
  - use class_info #select the database which you want to drop
  - db #ensure that the current db is the one you want to drop
  - db.dropDatabase() #drop the database
  - show dbs #show all databases and the dropped db will not show

```
> db.dropDatabase()
{ "dropped" : "class_info", "ok" : 1 }
```

Optional: MongoDB Aggregation Operators

SQL Concepts	MongoDB Aggregation Operators
WHERE	$match
GROUP BY	$group
HAVING	$match
SELECT	$project
ORDER BY	$sort
LIMIT	$limit
SUM()	$sum
COUNT()	$sum
JOIN	$lookup

Resources

- https://docs.mongodb.com/manual/crud/ (official manual)
- https://www.tutorialspoint.com/mongodb/index.htm (tutorial)
- https://www.w3schools.com/python/python_mongodb_getstarted.asp (guided tutorial)
- http://nicholasjohnson.com/mongo/course/workbook/ (more exercises)
- https://www.w3resource.com/mongodb-exercises/ (more exercises)
- https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb (pymongo)