**MongoDB**

NoSQL database, manage a humongous amount of data

Not only structured query language

Instead of storing data in rows and columns, data is stored in documents

Data in these documents is stored as field value pairs, similar to Json

Document format:

{

Name: ‘Bob’,

Age: 30,

gpa: 3.0,

fullTime: false,

}

A collection is a group of one or more documents

A Database is a group of one or more collections

Makes scaling the database very easy

**Mongosh**

Show dbs: shows all databases

Use (db name) to use a database, will also create a db if it does not already exist

**CREATION**

db.createCollection("students")

db.dropDatabase() within db location

db.(collection name).insertOne({name: “Bob})

One will be created if it does not exist

Db.(collection name).find()

**INSERTING DATA/DATA TYPES**

db.students.insertMany([])

school> db.students.insertMany([{name:"Joe", age:25, gpa:2.3}, {name:"Harry", age:38, gpa:1.2}, {name:"George", age:23, gpa:3.5}])

new Date() – takes current time and date unless given information

UTC timezone

**DATA TYPES**

db.students.insertOne({name: “Larry Johnson”,

age:43, Integer

gpa: 2.7, Double

fullTime: true, Boolean

registerDate: new Date(), Date

graduationDate: null, Null-placeholder/no value

courses: [“Biology”, “Chemistry”, “Programming”], one field that has more than one value

address: {street:”123 St.”, address = nested documents

city:”Fake City”,

zip: 12345}})

**SORTING AND LIMITING**

db.students.find().sort({name:1})

1 for alphabetical, -1 for reverse alphabetical

db.students.find().sort({gpa:1})

1 for ascending, -1 for descending

db.students.find().limit(1)

returns x documents

db.students.find().sort({gpa:-1}).limit(1)

find highest/lowest gpa, limited to 1 document

**FIND**

db.students.find({name:"Bob"})

school> db.students.find({gpa:2.7})

school> db.students.find({gpa:2.7, fullTime:true})

.find({query}, {projection parameter])

school> db.students.find({}, {name:true})

Returns all documents & the projection parameters, in this case it is name

school> db.students.find({}, {\_id:false, name:false})

school> db.students.find({}, {name:false})

Returns all documents without the projection parameters

school> db.students.find({}, {\_id:false, name:true, gpa:true})

Want to see a certain field? Label it true – name:true

Don’t want to see a certain field? Label it false – name:false

**UPDATE**

db.students.updateOne(filter, update)

filter = what you’re looking for, update = what you are changing it to

db.students.updateOne({name:”Bob”}, {$set:{new Date()}})

school> db.students.updateOne({name:"Bob"}, {$set:{fullTime:true}})

school> db.students.updateOne({\_id: ObjectId("649df7a17437779c62d22100")}, {$unset:{fullTime:""}})

Unset operator removes a field

db.students.updateOne({name:"Bob"}, {$unset:{fullTime:""}})

school> db.students.updateMany({fullTime:{$exists:false}}, {$set:{fullTime:true}})

Selection criteria – if fulltime field does not exist (false)

If they do not have this field, give them one (true)

**DELETE**

db.students.deleteOne({name:”Bob”})

db.students.deleteMany({fulltime:false})

db.students.deleteMany({registerDate:{$exists:false}})

**COMPARAISON OPERATORS**

db.students.find({name:{$ne:"Bob"}})

Shows all data that does not contain “Bob”

db.students.find({age:{$lt:20}})

Shows all data of age that is less than ($lt) 20

$lte = less than equal to

db.students.find({age:{$gt:30}})

shows all data of age that is more than 30

$gt = greater than

$gte = greater than equals to

db.students.find({gpa:{$gte:3, $lte:4}})

Multiple operations can be used at once

This example is looking for gpas greater than or equal to 3 but less than or equal to 4

**LOGICAL OPERATORS**

$and

$not

$nor

$or

db.students.find({$and: [{fullTime:true}, {age:{$gte:22}}]})

Logical operator $and allows for two conditions when finding something

This will find any students that are fulltime and are older than or equal to 22 years old

db.students.find({$or: [{fullTime:true}, {age:{$gte:22}}]})

Logical operator $or allows for either of the two conditions to be true

db.students.find({$nor: [{fullTime:true}, {age:{$lte:22}}]})

Logical operator $nor requires both conditions to be false

This will find any students that are not fullTime and older than 22

db.students.find({age:{$not:{$gte:30}}})

Logical operator $not returns data that is not the condition

Return any data that does not meet the condition

**INDEXES**

Indexes cut down on search time

db.students.createIndex({name: 1})

Creates an alphabetical index

db.students.getIndexes()

db.students.dropIndex("name\_1")

Applying an index to a data field allows for a quick lookup for the field, but takes more memory and slows insertion update and remove operations because the binary search tree needs to be updated

**COLLECTIONS-P2**

show collections

db.createCollection(“teachers”, {capped:true, size: 10000000, max:100} ,{autoIndexId:false})

Size is in bytes, 10000000 = 10 mb

Max = max number of documents

db.(collection name).drop()