Exercise - connect to a console

Connect to the console at localhost. Try typing some JavaScript expressions.

- Tell me how many seconds there are in a week
  - print(((7 * 24) * 60) * 60)
- Tell me how many weeks there are in a human lifetime of 80 years.
  - print((80 * 365) / 7)

Exercise - Create a database

- Use the use command to connect to a new database (If it doesn't exist, Mongo will create it when you write to it).
  - use petshop

Exercise - Create a collection

- Use db.createCollection to create a collection. I'll leave the subject up to you.
  - db.createCollection(‘mammals’)
- Run show dbs and show collections to view your database and collections.
  - show dbs
  - show collections

Exercise - Create some documents

- Insert a couple of documents into your collection. I'll leave the subject matter up to you, perhaps cars or hats.
  - db.mammals.insert({name: "Polar Bear"})
  - db.mammals.insert({name: "Star Nosed Mole"})

Exercise - documents

- Use find() to list them out.
  - db.mammals.find()

Exercise

We need to start out by inserting some data which we can work with.

- Add another piranha, and a naked mole rat called Henry.
  - db.pets.insert({name: 'Buddy', species: 'Piranha'})
  - db.pets.insert({name: 'Henry', species: 'Naked Mole Rat'})
- Use find to list all the pets. Find the ID of Mikey the Gerbil.
  - In my case Mikey’s Object ID is 5f5ba0c207146b173f8ec3a7
- Use find to find Mikey by id.
  - db.pets.find(ObjectId("5f5ba0c207146b173f8ec3a7"))
- Use find to find all the gerbils.
  - db.pets.find({ species: "Gerbil" })
- Find all the creatures named Mikey.
  - db.pets.find({ name: "Mikey" })
- Find all the creatures named Mikey who are gerbils.
  - db.pets.find({ name: "Mikey", species: "Gerbil" })
- Find all the creatures with the string "dog" in their species.
  - db.pets.find({ species: "dog" })

Exercise

- Use find to get all the people who are exactly 99 years old
  - db.people.find({ age: '99' })
- Find all the people who are eligible for a bus pass (people older than 65)
  - db.people.find({ age: { $gt: 65 } })
- Find all the teenagers, greater than 12 and less than 20.
  - db.people.find({ age: { $gt: 12, $lt: 20 } })

Exercise - $exists

- Find all the people with cats.
  - db.people.find({ cat: { $exists: true } })
- Find all the pensioners with cats.
  - db.pensioner.find({ cat: { $exists: true } })
- Find all the teenagers with teenage cats.

Exercise - Stockbrokers

- Find all the stocks where the profit is over 0.5
  - db.stocks.find({ "Profit Margin": { $gt: 0.5 } }).pretty()
- Find all the stocks with negative growth
  - db.stocks.find({ "EPS growth this year": { $lt: 0 } }).pretty()

Exercise - $where

- Use $where to find all the people who have a cat.
  - db.people.find({ $where: "this.cat"}).pretty()
- Find all the people who are younger than their cats. Remember, not everyone has a cat, so you will need to use a boolean && to filter out the non-cat owners.
  - db.people.find({ $where: "this.cat && this.age < this.cat.age" }).pretty()
- Does anyone have the same name as their cat? Re-run the insertion script to create more records until someone does.
  - db.people.find({ $where: "this.cat && this.age == this.cat.age" }).pretty()

Exercise - Tidy up your output

- Use projection to format your array of people. We want only the names.
  - db.people.find({}, { name: true })
- Output just the names of the people who are 99 years old
  - db.people.find({ age: 99 }, { name: true })
- Output only the cats
  - db.people.find({ cat: { $exists: true } }, { cat: true }).pretty()

Exercise - remove the ids

- List the cats. Remove the ids from the output.
  - db.people.find({ cat: { $exists: true } }, { cat: true, _id: false }).pretty()

Exercise - count the people

- Find out how many people there are in total.
  - db.people.count()
- Using your collection of people, and $exists, tell me how many people have cats.
  - db.people.find({ cat: { $exists: true } }).count()
  - 396 people have cats
- Use $where to count how many people have cats which are older than them.
  - db.people.find({ $where: "this.cat && this.age < this.cat.age" }).count()
  - 34 people have cats older than them

Exercise - Limit the people

- Give me the first 5 people
  - db.people.find().limit(5)
- Give me the next 5 people
  - db.people.find().limit(5).skip(5)
- Give me the names and ages of the oldest 5 pensioners with piranhas
  - db.people.find({ piranha: {$exists: true} }).sort({age: -1}).limit(5)
- Give me the names and ages of the youngest 5 teenagers with cats, where the cats have the word "Yolanda" in their name.
  - db.people.find({ $where: "this.cat && this.cat.name.includes('Yolanda')" }).sort({age: +1}).limit(5).pretty()

Exercise - Order the people

- Find the youngest 1 person with a cat and a piranha.
  - db.people.find({ $where: "this.cat && this.piranha" }).sort({age: +1}).limit(1)
- Give me just the name of the youngest 1 person with a cat and a piranha.
  - db.people.find({ $where: "this.cat && this.piranha" }, {name: true, _id: false}).sort({age: +1}).limit(1)
- Give me the 5 oldest cats
  - db.people.find({}, {cat: true}).sort({"cat.age": -1}).limit(5).pretty()
- Give me the next 5 oldest cats
  - db.people.find({}, {cat: true}).sort({"cat.age": -1}).limit(5).skip(5).pretty()

Exercise - Stocks

- Find me the top 10 most profitable stocks
  - db.stocks.find({"Profit Margin": {$exists: true}}).sort({"Profit Margin": -1}).limit(10).pretty()
- Add a projection, tell me which sector the most profitable stocks are in.
  - db.stocks.find({"Profit Margin": {$exists: true}}, {Sector: true}).sort({"Profit Margin": -1}).limit(10)
  - Basic Material and Financial
- Which is the least profitable sector.
  - db.stocks.find({"Profit Margin": {$exists: true}}, {Sector: true}).sort({"Profit Margin": +1}).limit(1)
  - Healthcare
- Have a look at the data. Spend a few minutes deciding which stocks you would most like to invest in.
  - No thank you.

Exercise - Cursor methods

- Iterate over each of the people and output them.
  - db.people.find().forEach(person => { print(JSON.stringify(person)) })
- change the find function to find only the people with cats
  - db.people.find({cat: {$exists: true}}).forEach(person => { print(JSON.stringify(person)) })
- Iterate over each of the people, outputting just the cat name and age each time.
  - db.people.find({cat: {$exists: true}}).forEach(person => { print(JSON.stringify({name: person.cat.name, age: person.cat.age})) })
- Use Map to generate an array containing all of the cat names.
  - db.people.find({cat: {$exists: true}}).map(person => {return person.cat.name})

Exercise - Stock ticker

- Sort the stocks by profit
  - db.stocks.find().sort({"Profit Margin": -1}).pretty()
- Now iterate over the cursor and output all of the stocks names and tickers in order of profit.
  - db.stocks.find({}, {Ticker: true, Company: true}).sort({"Profit Margin": -1}).map(stonk => {return {Company: stonk.Company, Ticker: stonk.Ticker}})

Exercise - Create a document

- Refresh your muscle memory. Create a new person now. Ensure that person has a shark.
  - db.people.insert({name: "Jack Black", age: 26, shark: {name: "Sharkie", age: "3"}})

Exercise - Find the shark

- Refresh your muscle memory. Find the person who has a shark.
  - db.people.find({shark: {$exists: true}})
- Use findOne instead of find. This will return only one document.
  - db.people.findOne({name: "Jack Black"})

Exercise - Make everyone older

- A year has gone by. Write a loop that iterates over a cursor and makes everyone one year older.
  - let people = db.people.find()
  - people.forEach(person => { person.age = person.age + 1 db.people.save(person) })
- Remember to make the cats older too. See if you can do both in the same loop.
  - let people = db.people.find()
  - peooople.forEach(person => { if (!person.cat) {return person.age = person.age + 1; db.people.save(person)}; person.age = person.age + 1; person.cat.age = person.cat.age + 1; db.people.save(person) })

Exercise - Pirates

- Find everyone who has the word 'Pirate' in their name. You will need to use a regular expression to do this. {name: /Pirate/}
  - db.people.find({$where: "this.name.includes('Pirate')"}).pretty()
  - db.people.find({name: /Pirate/}).pretty()
- Iterate over the cursor and award each of them a parrot.
  - let pirates = db.find({name: /Pirate/})
  - pirates.forEach(pirate => { pirate.parrot = true; db.people.save(pirate) })

Exercise - remove all the people.

- It's time for a cull. Delete all the 50 year olds.
  - db.people.remove({age: 50})
- We also heard there was some guy running round with a shark. That's a dangerous animal. Take him out, in fact take out anyone with a shark.
  - db.people.remove({shark: {$exists: true}})

Exercise - Create an Empty pipeline

- Try out the aggregate pipeline now. Call aggregate on your people collection. You'll see the result is the same as if you called find.
  - db.people.aggregate()

Exercise - $match

- Use the people dataset. Match all the people who are 10 years old who have ten year old cats.
  - db.people.aggregate({ '$match': {cat: {$exists: true}, age: 10, 'cat.age': 10 }})
- Match all the people who are over 80 years old, and who's cats are over 15 years old.
  - db.people.aggregate({ '$match': {cat: {$exists: true}, age: {$gt: 80}, 'cat.age': {$gt: 15} }}).pretty()

Exercise

- Make a list of cat names.
  - db.people.aggregate({'$project': {'cat.name': true}})
- First $match people with cats, or the output will be a bit sparse.
  - db.people.aggregate([{'$match': {cat: {$exists: true}}},{'$project': {'cat.name': true}}]).pretty()
- Now use $project to pull out only the cat names. You will need to use the dot syntax: '$cat.name'.
  - db.people.aggregate([{'$match': {cat: {$exists: true}}},{'$project': {'cat.name': true, _id: false}}])

Exercise - Stocks

- Use the stocks JSON file.
  - use stocks
- rename "Profit Margin" to simply "Profit". Surpress all other output including the id. I only want to see profit, the company name and the ticker.
  - db.stocks.aggregate({'$project': {Profit: "$Profit Margin"}})

Exercise - String aggregation operators

- Attempt to capitalize all the names. This is useful because Mongo grouping and matching is case sensitive.
  - db.people.aggregate({$project: {name: {$concat: ['$name']}}})

Exercise - Add a hasCat field

- Use projection to add a hasCat field. This might form the basis for a future grouping or counting.
  - db.people.aggregate({$project:{hasCat: {$cond: {if: '$cat', then: true, else: false}}}})

Exercise - Project the stocks

- Modify your stocks. Project the profit as a profit field.
  - db.stocks.aggregate({$project: {Profit: "$Profit Margin"}})
- Add a isProfitable field to show if the profit is greater than 0.
  - db.stocks.aggregate({$project: {isProfitable: {$cond: {if: {$gt: ["$Profit Margin", 0]}, then: true, else: false}}}})
- Add a buyNow field to show if the profit is greater than 0.5
  - db.stocks.aggregate({$project: {buyNow: {$cond: {if: {$gt: ["$Profit Margin", 0.5]}, then: true, else: false}}}})

Exercise

- Try this out on your people data set. You should get a list of distinct names.
  - db.people.aggregate({$group: {_id: '$name'}})
- The output is untidy, each name output in the id field. Add a $project step to the popeline to rename the '_id' field to 'name'.
  - db.people.aggregate([{$group: {_id: '$name'}}, {$project: {"_id": 0, "name": '$_id'}}])

Exercise - Grouping by object

- Try out the above. Notice that the _id field is now an object. Use $project to reformat the data. You now have distinct names and ages.
    - db.people.aggregate([{$group: { _id: {name: '$name', age: '$age'}}}, {$project: {_id: 0, name: '$_id.name', age: '$_id.age'}}])

Exercise - Group and push

- Use $match to select only people with cats
    - db.people.aggregate({$match: {cat: {$exists: true}}})
- Now group by name, and for each person.
    - db.people.aggregate({$group: {_id: '$name'}).pretty()
- Push the cats into the result.
    - db.people.aggregate([{$match: {cat: {$exists: true}}}, {$group: {_id: '$name', entries: {$push: "$$ROOT"}}}]).pretty()

Exercise - Count everything

- Count all the people. How many are there?
    - db.people.aggregate({ $group: { _id: 1, count: { $sum: 1 } } })

Count with $match

- Add a $match step to the start of your pipeline. Count all the people with cats using the aggregate pipeline. How many do you have?
    - db.people.aggregate([{$match: { cat: {$exists: true}}},{ $group: { _id: 1, count: { $sum: 1 } } }])
    - 395

Harder - Count with $project and $cond

- Use project to create a 'hasCat' field. You will need to use $cond to do this: http://docs.mongodb.org/manual/reference/operator/aggregation/cond/. Check that your pipeline now contains the hasCat field.
    - db.people.aggregate([{$project: {hasCat: {$cond: {if: '$cat', then: true, else: false}}}}])
- Now group by hasCat and count.
    - db.people.aggregate([{$project: {hasCat: {$cond: {if: '$cat', then: true, else: false}}}},{$group: {_id: '$hasCat', count: {$sum: 1}}}])
- Finally use $project to clean up your stats. You now have a JSON API call for finding the cat status in your application.
    - ?????

Exercise - Stocks

- group the stocks data by sector.
  - db.stocks.aggregate({$group: {_id: '$Sector'}})
- Use $sum to discover the most profitable sector
  - db.stocks.aggregate({$group: {_id: '$Sector', count: {$sum: 1}}})
- Sort by profitability
  - db.stocks.aggregate({$sort: {"Profit Margin": -1}}).pretty()
- $project the results
  - ?????

Exercise - Enron

- List all the unique senders.
  - db.emails.aggregate({$group: {_id: '$sender'}})
- Count all the unique senders.
  - db.emails.aggregate([{$group: {_id: '$sender'}}, {$group: {_id: 1, count: {$sum: 1}}}])
- group by sender and count to find out which email address sent the most emails.
  - db.emails.aggregate([{$group: {_id: '$sender', count: {$sum: 1}}},{$sort: {count: -1}}, {$limit: 1}])
- Rank the senders in order or emails sent.
  - db.emails.aggregate([{$group: {_id: '$sender', count: {$sum: 1}}},{$sort: {count: -1}}])

Harder

- Rank the senders in order or emails sent + emails received. You will need to use a project stage to do this.
  - ?????

Exercise - Holidays!

- Group the data by year, month and dayOfMonth. Be aware that not every holiday has a date. You may generate the dynamic fields with $project, or directly in $group _id.
  - db.holidays.aggregate({$group: {_id: {year: {$year: '$date'},month: {$month: '$date'},day: {$dayOfMonth: '$date'}}}})
- $push the holiday name into the result set.
  - db.holidays.aggregate([{$group: {_id: {year: {$year: '$date'},month: {$month: '$date'},day: {$dayOfMonth: '$date'}}, name: {$push: '$name'}}}]).pretty()
- Sort by year, month and dayOfMonth.
  - db.holidays.find().sort({date: -1}).pretty()

Exercise - the biggest day

- Add a $count to the $group operator and remove the $push.
  - ?????
- Sort the data by count. Which day has the most holidays?
  - db.holidays.aggregate([{$group: {_id: {$dayOfYear: '$date'}, count: {$sum: 1}}},{$sort: {count: -1}}, {$limit: 1}]).pretty()
  - Day 121
- Which month has the most holidays?
  - db.holidays.aggregate([{$group: {_id: {$month: '$date'}, count: {$sum: 1}}},{$sort: {count: -1}}, {$limit: 1}]).pretty()
  - March

Harder Exercise - bell curve

- Add another group stage. Group your result set by the count field you generated in the previous step. We can say how many days have one holiday, how many have 2, how many have 3, etc.
  - db.holidays.aggregate([{$group: {_id: {$dayOfYear: '$date'}, count: {$sum: 1}}},{$sort: {count: -1}},{$group: {_id: '$count', amountOfDays: {$sum: 1}}},{$sort: {_id: 1}}]).pretty()
  - 1 – 52
  - 2 – 103
  - 3 – 91
- Sort by this new count field. You should see a nice bell curve with the median around 3, and a long tail.
  - db.holidays.aggregate([{$group: {_id: {$dayOfYear: '$date'}, count: {$sum: 1}}},{$sort: {count: -1}},{$group: {_id: '$count', amountOfDays: {$sum: 1}}},{$sort: {amountOfDays: 1}}]).pretty()
- Use $project to tidy your results.
  - ????

Exercise - tag list

unwind the data

```
startupCursor.forEach(startup => { startup.tag_list = startup.tag_list.split(','); db.startups.save(startup) })
```

- Now each company has a tag array, $unwind the tags, then write a $group aggregation to generate a complete list of all available tags without duplicates.
  - db.startups.aggregate([{$unwind: '$tag_list'},{$group: {_id: '$tag_list'}}])
- Add in a count to find the most popular tags.
  - db.startups.aggregate([{$unwind: '$tag_list'},{$group: {_id: '$tag_list', count: {$sum: 1}}}])
- Run the output through $project to generate a nice JSON tag feed.
  - ?????