CS 61:
Database Systems

NoSQL/Mongo CRUD

Adapted from mongodb.com unless otherwise noted

Agenda

1. Why choose NoSQL

2. Mongo CRUD

Relational databases have historically been the “safe bet” for the enterprise

Nobody ever got fired for buying

*

* Or promoted either….
- Pierson

SQL databases are a solid choice for many database scenarios

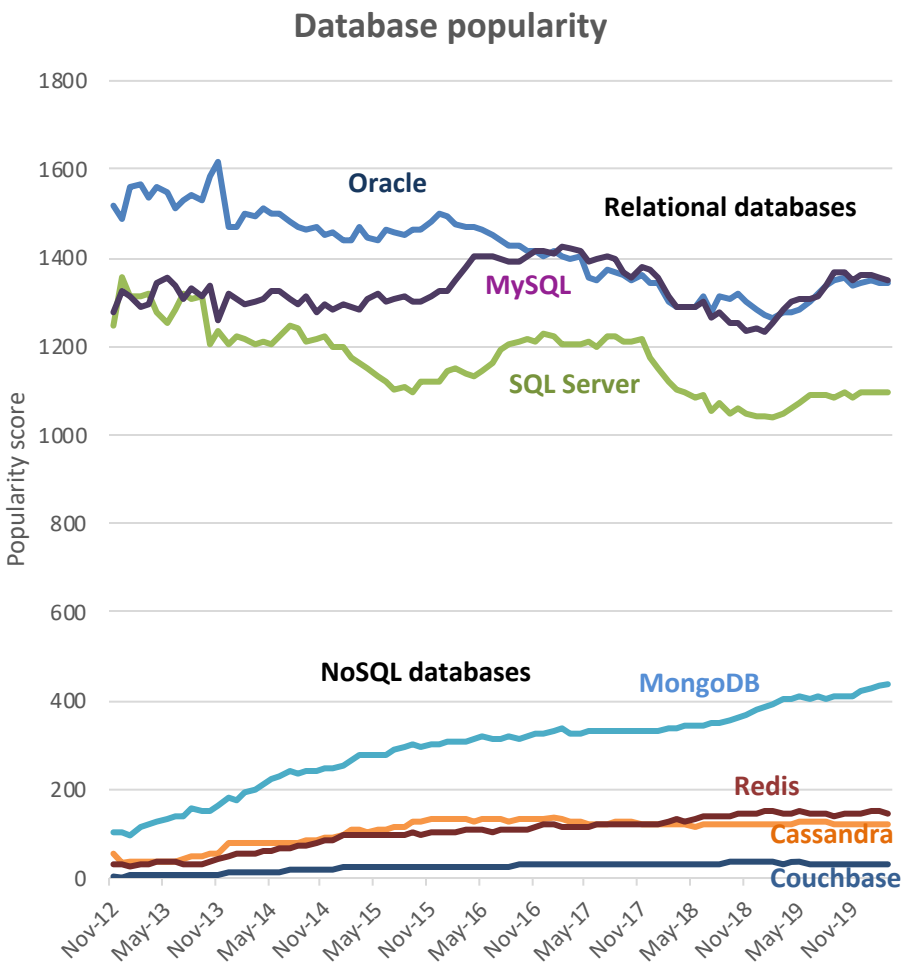
SQL databases scale vertically well (get a bigger box), do not scale horizontally well (get lots of boxes)

NoSQL databases are generally designed for scaling horizontally (sharding)

NoSQL shines with unstructured data

Tech industry saying: https://www.ibm.com/ibm/history/ibm100/us/en/icons/personalcomputer/words/

NoSQL databases have been gaining in popularity



NoSQL (or Not Only SQL) means a non-relational data store

Generally designed to run on clusters of computers (scales horizontally, aka shards) vs. a single server

At least three forces are driving adoption of NoSQL databases

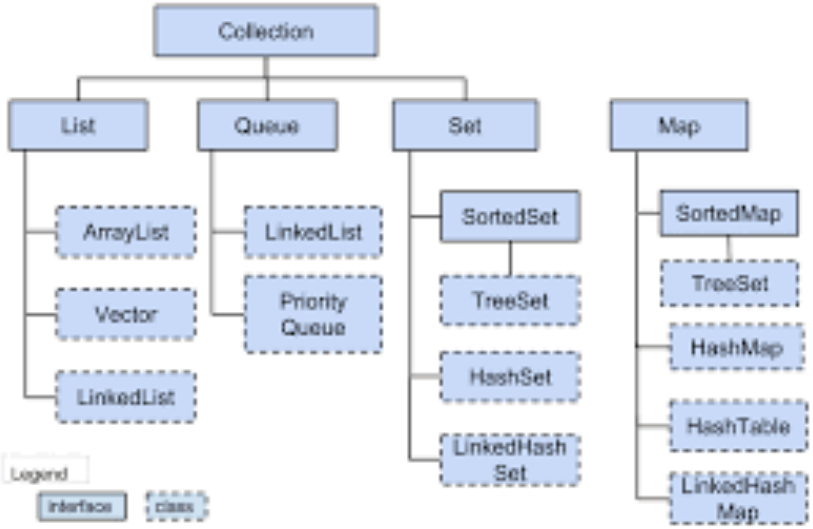
- Programming ease
- The rise of web services
- Big data

Source: https://db-engines.com/en/ranking_trend

Three concepts driving NoSQL: ease, web services, and big data

1. Programming ease

vs.



1. Programming ease

- Developers use complex in-memory data structures to model real world
- RDBMS have one data structure – relations
- Tuple values must be simple for consistency and speed
- Leads to translating between models
- Example: update customer order, must update:
  - Customer table
  - Order table
  - Order items table

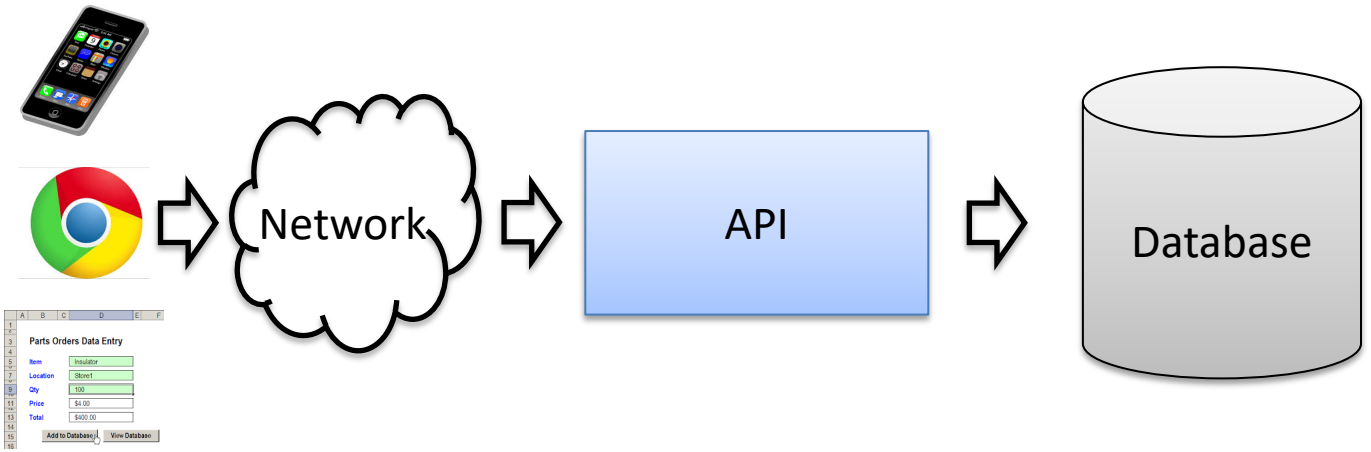
vs

One data structure that stores customer and order data nested in one document

Adapted from: NoSQL Distilled by Sadalage and Fowler

Three concepts driving NoSQL: ease, web services, and big data

2. Web services



2. Rise of web services

- Web services sit between end user applications and database
- Now choice and structure of database not visible to end users (or their applications)
- Easy(ier) to swap out database for a different design or a different type of database
  - Web service API logic adjusted for change
  - Client-side applications may not be affected

Adapted from: NoSQL Distilled by Sadalage and Fowler

Three concepts driving NoSQL: ease, web services, and big data

3. Big data

3. Big data

- 5 V's of big data (volume, velocity, variety, veracity, value)
  - At some point you can not scale vertically (cannot get a bigger box)
  - Variety of data may not fit nicely into pre-defined schema
- Need for 100% up time, cannot rely on a single point of failure
- Global operations need fast data access all over the world
  - Want to distribute data across many machines

Adapted from: NoSQL Distilled by Sadalage and Fowler

A NoSQL distributed system might be the right approach for your needs

1. Programming ease

2. Web services

3. Big data

Consider NoSQL if you are:

- Looking for improved developer productivity by using a more convenient or intuitive data interaction style
- Looking for a highly distributed system running on commodity hardware
- Looking for ability to handle data access with sizes and performance that require a cluster of machines
- Have unstructured data or it is difficult to predict how the application will change over time

Adapted from: NoSQL Distilled by Sadalage and Fowler

Agenda

1. Why choose NoSQL
2. Mongo CRUD

MongoDB stores data in collections comprised of documents

RDBMS	MongoDB
Database	Database
Table	Collection
Row	Document
Column/Attribute	Field (name/value pairs)
Index	Index
JOIN	Linking and embedding

The terms are different, but many of the concepts are similar

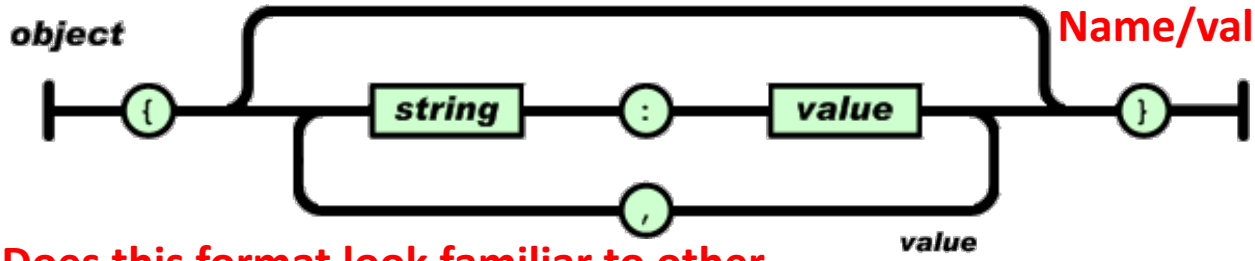
Source: https://www.tutorialspoint.com/mongodb/index.htm

MongoDB uses a variant of JSON to store documents; simple but powerful!

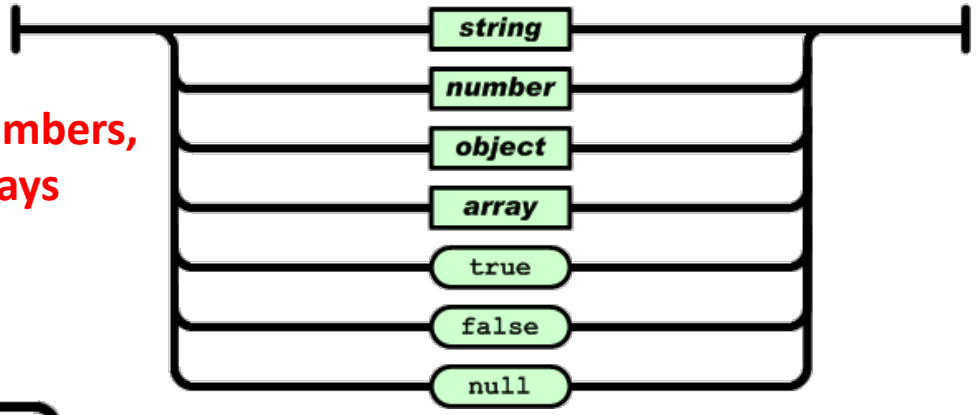
JSON (JavaScript Object Notation) has two high-level structures

1. **Objects: collection of name/value pairs**

Objects are unordered name/value pairs
Begin with { and end with }
Name/value pairs separated by commas

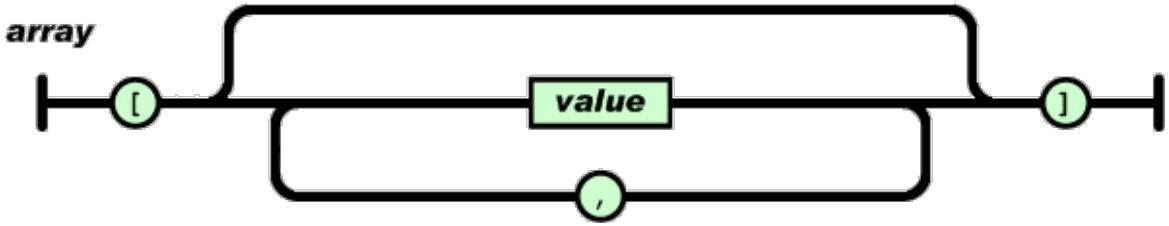


Does this format look familiar to other structures we've seen in CS10?
Finite Automata



- **Values can be strings, numbers, booleans, objects, or arrays**
- **Very powerful "nesting"**

2. **Arrays: ordered list of values**



Arrays are ordered
Begin with [and end with]
Items separated by commas

Source: www.json.org

JSON allows embedded data structures, reducing the need for joins

```
// customer info collection
{ "id":42, "fname":”albert", "mi":"j","lname":"coot",
    "addresses": [
        { "addrType":"billingaddress",
        "streetaddr":"2103 Xenon Way", "city":"Santa Fe", "St":"NM"
        },
        { "addrType": "shippingaddress",
        "streetaddr":"42 Catus Way",   "city":"Taos", "St":"NM"
        }
    ]
}
// shopping cart collection
{ "id":74829312, "customer":42,
    "itemlist":[
    { "UPC":293012429, "price":79.95, "name":"apple 85w power adapter"},
    { "UPC":829381427, "price":59.95, "name":"apple touchpad mouse" }
    ],
"paymentinfo":[
    { "ccard":"1234-5678-9876-5432", "exp":"0115", "ccxact":"111111" }
]
}
```

Can reference other documents, but referential integrity is not enforced like it is in SQL

JSON allows embedded data structures, reducing the need for joins

```
// customer info collection
{ "id":42, "fname":”albert", "mi":"j","lname":"coot",
    "addresses": [
        { "addrType":"billingaddress",
        "streetaddr":"2103 Xenon Way", "city":"Santa Fe", "St":"NM"
        },
        { "addrType": "shippingaddress",
        "streetaddr":"42 Catus Way",   "city":"Taos", "St":"NM"
        }
    ]
}
// shopping cart collection
{ "id":74829312, "customer":42,
    "itemlist":[
    { "UPC":293012429, "price":79.95, "name":"apple 85w power adapter"},
    { "UPC":829381427, "price":59.95, "name":"apple touchpad mouse" }
    ],
"paymentinfo":[
    { "ccard":"1234-5678-9876-5432", "exp":"0115", "ccxact":"111111" }
]
}
```

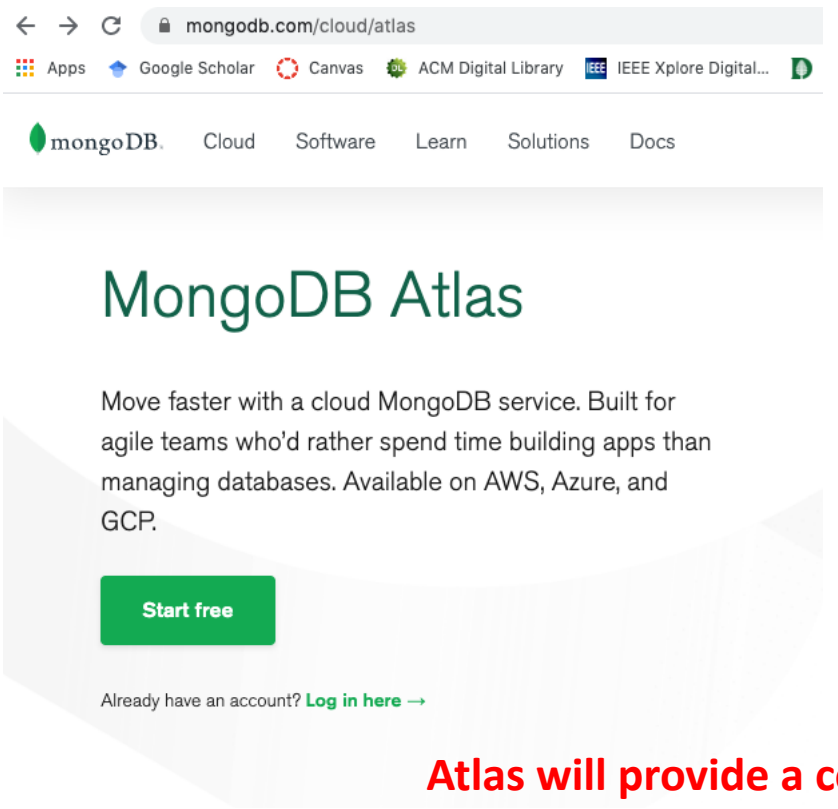
RDMS schema would need:

- **Customer table**
- **Address table (with customer ID as FK)**
- **Shopping cart table (with CustomerID as FK)**
- **Order table (with CustomerID as FK)**

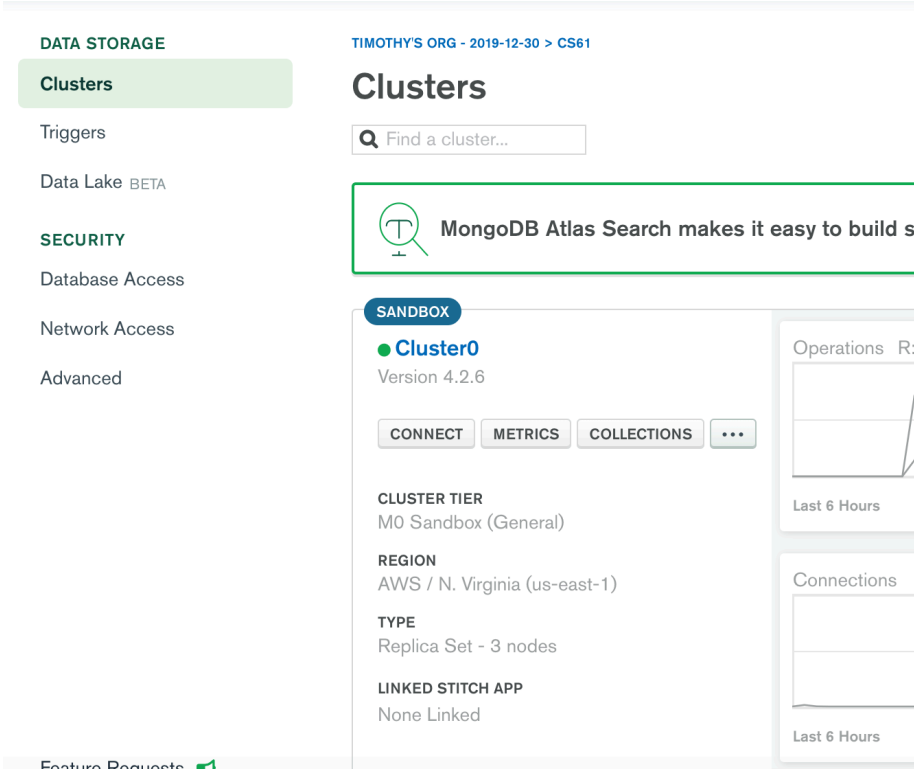
Lots of joins required for RDBMS! Not needed with embedding in JSON

You can get free access to a Mongo installation in the cloud using Atlas

Create free account at
https://www.mongodb.com/cloud/atlas



Compass is GUI like MySQL Workbench



Atlas will provide a connection string
Can install Compass or command line shell to issue commands
I will use the command line

MongoDB documents are a form of JSON and allow document embedding

Blog-style data structure

```
{
      _id: ObjectId(7df78ad8902c)
      title: 'MongoDB Overview',
      by: 'tutorials point',
      tags: ['mongodb', 'database', 'NoSQL'],
      comments: [
            {
                  user:'user1',
                  message: This is user1 comment',
                  dateCreated: new Date(2011,1,20,2,15)
            },
            {
                  user:'user2',
                  message: This is user2  comment',
                  dateCreated: new Date(2011,1,25,7,45)
            }
      ]
}
```

Fields are name/value pairs

You can provide your own _id

If you do not provide one, MongoDB will generate a unique 12-byte value

Comments is an array embedded inside of a document

Source: https://www.tutorialspoint.com/mongodb/index.htm

MongoDB CRUD on a collection is similar to SQL CRUD on a table

CRUD operations

CRUD	SQL	MongoDB
Create	INSERT	insert
Read	SELECT	find
Update	UPDATE	update
Delete	DELETE	remove

The CRUD operations in MongoDB have different names but function similarly on a collection as SQL commands on a table

Embedded and linking documents, however, is different (next class)

First connect to server and select a database to use

Get connection to database

Get connection string from Atlas web site (or use local host)

```
vpn-two-factor-general-230-148-96:~ tim$ mongo "mongodb+srv://cluster0-rq8e6.mongodb.net/test" --username cs61
MongoDB shell version v4.2.0
Enter password:
connecting to: mongodb://cluster0-shard-00-01-rq8e6.mongodb.net:27017,cluster0-shard-00-02-rq8e6.mongodb.net:27017, <snip>
MongoDB server version: 4.2.6
```

Atlas provides a connection string
Provide password when prompted

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> show dbs
admin  0.000GB
cs61   0.000GB
local  3.875GB
```

***show dbs* lists the databases (schemas) available on this server**

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db
test
```

***db* shows the currently selected database (test is default)**

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> use cs61
switched to db cs61
```

***use dbName* switches to *dbName* database**

Database is created if does not already exist, but will not show up in *show dbs* until at least one document is inserted

dropDatabase deletes a database

db.dropDatabase()

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.dropDatabase()
{
"dropped" : "cs61",
"ok" : 1,
"$clusterTime" : {
"clusterTime" : Timestamp(1589633625, 2),
"signature" : {
"hash" : BinData(0,"GjGGdJxmLCiQnjphPBpPqucfiWE="),
"keyId" : NumberLong("6826973194142875651")
}
},
"operationTime" : Timestamp(1589633625, 2)
}
```

dropDatabase() deletes the currently selected database (here cs61)

Database *cs61* is deleted

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> show dbs
admin  0.000GB
local  3.871GB

MongoDB Enterprise Cluster0-shard-0:PRIMARY> db
cs61
```

Currently select database is still *cs61*, but it is empty

createCollections makes a collection, like CREATE TABLE

db.createCollection(“CollectionName”)

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.createCollection("Restaurants")
{
"ok" : 1,
"$clusterTime" : {
"clusterTime" : Timestamp(1589633855, 1),
"signature" : {
"hash" : BinData(0,"4aXY1b0hyVq4XI7esxFogn5IwEM="),
"keyId" : NumberLong("6826973194142875651")
}
},
"operationTime" : Timestamp(1589633855, 1)
}
```

***createCollections* makes a new collection**

***show collections* lists the collections in the database**

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> show collections
Restaurants
```

Insert documents into a collection with *insert,* list documents with *find*

db.Collection.insert({document)

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.insert({name:"Tim's
Tasty Treats",boro:"Manhattan"})
{
"acknowledged" : true,
"insertedId" : ObjectId("5ebfe42a6ead6c5a3b84c554")
}
```

***find* is like SELECT**

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.find()
{ "_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"), "name" : "Tim's Tasty Treats", "boro" : "Manhattan"
}
```

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.find().pretty()
{
"_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"),
"name" : "Tim's Tasty Treats",
"boro" : "Manhattan"
}
```

Adding *pretty* to *find* prints the document with a nice format

Note: _id not in the insert, so MongoDB created one
If insert provides _id, that value is used

Insert several documents at the same time by using a JSON array

db.Collection.insert([{document},{document}…])

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.insert([{name:"Tim's Untasty Treats",boro:"Queens"},{name: "Brooklyn's Best Restaurant",boro:"Brooklyn"},{name:"Bronx diner",boro:"Bronx"}])
BulkWriteResult({
"writeErrors" : [ ],
"writeConcernErrors" : [ ],
"nInserted" : 3,
"nUpserted" : 0,
"nMatched" : 0,
"nModified" : 0,
"nRemoved" : 0,
"upserted" : [ ]
})
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.find()
{ "_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"), "name" : "Tim's Tasty Treats", "boro" : "Manhattan" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c556"), "name" : "Tim's Untasty Treats", "boro" : "Queens" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c557"), "name" : "Brooklyn's Best Restaurant", "boro" : "Brooklyn" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c558"), "name" : "Bronx diner", "boro" : "Bronx" }
```

Insert multiple document as a JSON array of Objects

Three new documents inserted
upsert is a document that overwrites an existing document

Also *insertOne* and *insertMany* commands available

find returns documents matching the provided criteria

db.Collection.find({criteria})

***findOne* returns only one document**

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.find()
{ "_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"), "name" : "Tim's Tasty Treats", "boro" : "Manhattan" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c556"), "name" : "Tim's Untasty Treats", "boro" : "Queens" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c557"), "name" : "Brooklyn's Best Restaurant", "boro" :
"Brooklyn" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c558"), "name" : "Bronx diner", "boro" : "Bronx" }

MongoDB Enterprise Cluster0-shard-0:PRIMARY>
db.Restaurants.find({boro:"Manhattan"}).pretty()
{
"_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"),
"name" : "Tim's Tasty Treats",
"boro" : "Manhattan"
}
```

Find restaurants in Manhattan
Only returns one document in this case

Use *$or, $and, $nor, $not* for multiple criteria
Must put criteria in an array

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.find({$or:
[{boro:"Manhattan"},{boro:"Queens"}]})
{ "_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"), "name" : "Tim's Tasty Treats", "boro" : "Manhattan" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c556"), "name" : "Tim's Untasty Treats", "boro" : "Queens" }
```

find can also project fields and limit the number of documents returned

db.Collection.find({criteria},{field:1, field:0})

1 is returned, 0 is not returned
If project criteria not provided, assume 0 (except for _id)

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.find({},{"name":1})
{ "_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"), "name" : "Tim's Tasty Treats" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c556"), "name" : "Tim's Untasty Treats" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c557"), "name" : "Brooklyn's Best Restaurant" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c558"), "name" : "Bronx diner", "boro" : "Bronx" }

MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.find({},{"name":1,_id:0})
{ "name" : "Tim's Tasty Treats" }
{ "name" : "Tim's Untasty Treats" }
{ "name" : "Brooklyn's Best Restaurant" }
```

Skip _id by setting it to 0

Return only two results

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.find({}).limit(2)
{ "_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"), "name" : "Tim's Tasty Treats", "boro" : "Manhattan"}
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c556"), "name" : "Tim's Untasty Treats", "boro" : "Queens" }
```

Add *sort* to sort results

db.Collection.find({criteria},{field:1, field:0}).sort({key:1, key:-1})

1 sorted in ascending order
-1 sorted in descending order

Ascending sort on boro

MongoDB Enterprise Cluster0-shard-0:PRIMARY> **db.Restaurants.find({}).sort({boro:1})**

{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c557"), "name" : "Brooklyn's Best Restaurant", "boro" : "Brooklyn" }
{ "_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"), "name" : "Tim's Tasty Treats", "boro" : "Manhattan”}
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c556"), "name" : "Tim's Untasty Treats", "boro" : "Queens" }

Descending sort on boro

MongoDB Enterprise Cluster0-shard-0:PRIMARY> **db.Restaurants.find({}).sort({boro:-1})**

{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c556"), "name" : "Tim's Untasty Treats", "boro" : "Queens" }
{ "_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"), "name" : "Tim's Tasty Treats", "boro" : "Manhattan"}
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c557"), "name" : "Brooklyn's Best Restaurant", "boro" : "Brooklyn" }

update can add new fields or update existing field values

db.Collection.update({criteria},{updated data})

MongoDB Enterprise Cluster0-shard-0:PRIMARY> **Add new field for score**
db.Restaurants.update({_id:ObjectId("5ebfe42a6ead6c5a3b84c554")},{$set:{score:5}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

MongoDB Enterprise Cluster0-shard-0:PRIMARY> **db.Restaurants.find({boro:"Manhattan"})**
{ "_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"), "name" : "Tim's Tasty Treats", "boro" : "Manhattan",
"score" : 5 }

MongoDB Enterprise Cluster0-shard-0:PRIMARY> **Can also update field values**
db.Restaurants.update({_id:ObjectId("5ebfe42a6ead6c5a3b84c554")},{$set:{score:6}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

MongoDB Enterprise Cluster0-shard-0:PRIMARY> **db.Restaurants.find({boro:"Manhattan"})**
{ "_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"), "name" : "Tim's Tasty Treats", "boro" : "Manhattan",
"score" : 6 }

Use *remove* to delete documents

Format: db.Collection.remove({criteria})

Add score to Bronx Diner

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.update({_id:ObjectId("5ebfe8c46ead6c5a3b84c558")},{$set:{score:12}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.find()
{ "_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"), "name" : "Tim's Tasty Treats", "boro" : "Manhattan", "score" : 6 }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c556"), "name" : "Tim's Untasty Treats", "boro" : "Queens" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c557"), "name" : "Brooklyn's Best Restaurant", "boro" : "Brooklyn" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c558"), "name" : "Bronx diner", "boro" : "Bronx", "score" : 12 }
```

Remove all documents with score greater than 6

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.remove({score: {$gt: 6}})
WriteResult({ "nRemoved" : 1 })
```

Can use $eq, $lt, $gt, $lte, $gte, $ne, $in, and $nin

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.find()
{ "_id" : ObjectId("5ebfe42a6ead6c5a3b84c554"), "name" : "Tim's Tasty Treats", "boro" : "Manhattan", "score" : 6 }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c556"), "name" : "Tim's Untasty Treats", "boro" : "Queens" }
{ "_id" : ObjectId("5ebfe8c46ead6c5a3b84c557"), "name" : "Brooklyn's Best Restaurant", "boro" : "Brooklyn" }
```

Use db.Collection.remove({}) to remove all documents, like TRUNCATE

Use *createIndex* to add an index on a field for faster lookup like a RDMS

db.Collection.createIndex({field:1})

1 ascending, -1 descending

```
MongoDB Enterprise Cluster0-shard-0:PRIMARY> db.Restaurants.createIndex({boro:1})
{
"createdCollectionAutomatically" : false,
"numIndexesBefore" : 1,
"numIndexesAfter" : 2,
"ok" : 1,
"$clusterTime" : {
"clusterTime" : Timestamp(1589640477, 2),
"signature" : {
"hash" : BinData(0,"xI/A6/Ux5me+MXwxK0Uj+UASLwE="),
"keyId" : NumberLong("6826973194142875651")
}
},
"operationTime" : Timestamp(1589640477, 2)
}
```

Practice

1. Create a collection to hold data about Students where each Student has:
   - Name
   - Year

2. Insert three students
   - Alice 20
   - Bob 21
   - Charlie 22

3. Add GPA attribute to Alice, give her a 3.5

4. Find Students where year > 20