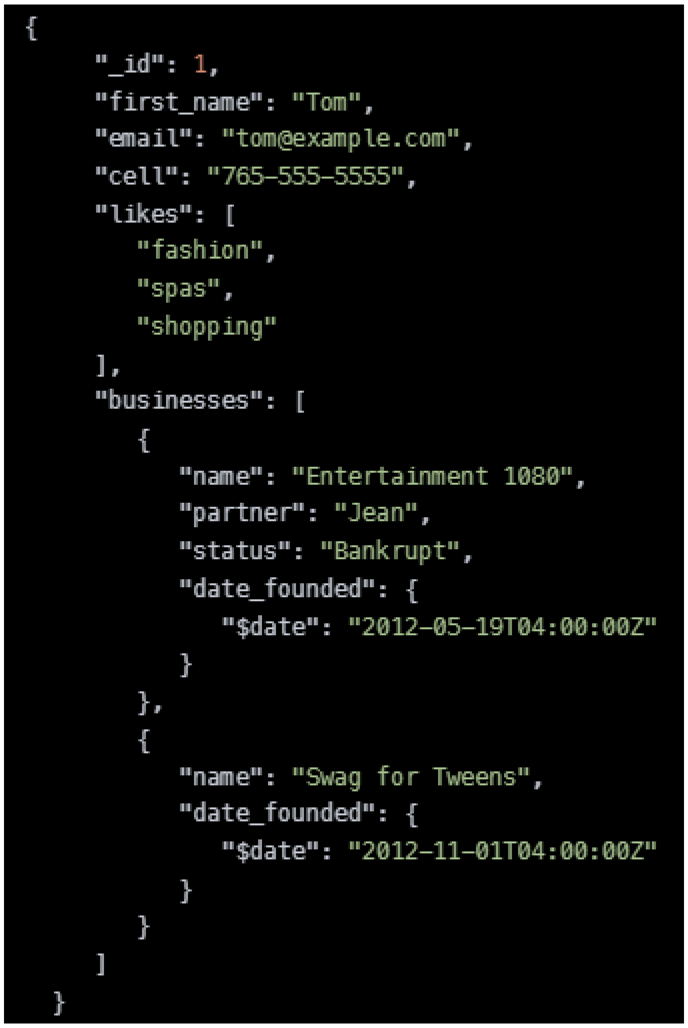
Mongo db Basics:

Basics terms:

- ***Document:****A document is a record in a document database. A document typically stores information about one object and any of its related metadata*
  - *Sample:*
    - *Below is a JSON document that stores information about a user named Tom.*



```
{
     "_id": 1,
     "first_name": "Tom",
     "email": "tom@example.com",
     "cell": "765-555-5555",
     "likes": [
        "fashion",
        "spas",
        "shopping"
     ],
     "businesses": [
        {
           "name": "Entertainment 1080",
           "partner": "Jean",
           "status": "Bankrupt",
           "date_founded": {
              "$date": "2012-05-19T04:00:00Z"
           }
        },
        {
           "name": "Swag for Tweens",
           "date_founded": {
              "$date": "2012-11-01T04:00:00Z"
           }
        }
     ]
  }
```

- ***Collection****:A collection is a group of documents. Collections typically store documents that have similar contents.*

Not all documents in a collection are required to have the same fields, because document databases have a flexible schema. Note that some document databases provide schema validation, so the schema can optionally be locked down when needed.

- ***Replica Set*** - a few connected machines that store the same data to ensure that if something happens to one of the machines the data will remain intact. Comes from the word replicate - to copy something.
- ***Instance*** - a single machine locally or in the cloud, running a certain software, in our case it is the MongoDB database.
- ***Cluster*** - group of servers that store your data.

-BSON : **Binary JSON**

-JSON vs BSON

	JSON	BSON
Encoding	UTF-8 String	Binary
Data Support	String, Boolean, Number, Array	String, Boolean, Number (Integer, Float, Long, Decimal128...), Array, Date, Raw Binary
Readability	Human and Machine	Machine Only

-MongoDB stores data in ***BSON*** format both internally, and over the network

1.Import and export:

-2 types:

- JSON Import and export
    - **mongoexport** --uri="mongodb+srv://<your username>:<your password>@<your cluster>.mongodb.net/dbname" --collection=sales **--out=sales.json**
    - **mongoimport** --uri="mongodb+srv://<your username>:<your password>@<your cluster>.mongodb.net/dbname" --drop **sales.json**
        - –drop : removes existing databases

- ***BSON Import and export:***
    - **mongodump** --uri "mongodb+srv://<your username>:<your password>@<your cluster>.mongodb.net/dbname"
    - **mongorestore** --uri "mongodb+srv://<your username>:<your password>@<your cluster>.mongodb.net/dbname" --**drop** nameOfDump
        - –drop : removes existing databases
- Export mongo query to csv:

  Refer :

    - https://database.guide/how-to-export-mongodb-query-results-to-a-csv-file/
    - https://stackoverflow.com/questions/36319052/use-mongoexport-with-a-query-for-isodate

Command: Execute outside mongo db

- mongoexport --authenticationDatabase camelot_svc_leads -d camelot_svc_leads -c lead_details --type=csv --fields pan,leadSource,leadNo,source --out mongo_report.csv

- mongoexport --host=<ip> --port=8000 --username=axismfleads --authenticationDatabase=camelot_svc_leads --db=camelot_svc_leads --collection=lead_details --type=csv --fields=leadNo,leadSource,isAuthFT --query '{

  "createdDate": {

"$gte": { "$date": "2022-06-16T18:30:00.001Z" },

"$lte": { "$date": "2022-06-17T23:59:59.000Z" }

```
  }
}'   --out=mongo_report.csv
```

2.*Data explore in atlas:*

- **Namespace -** The concatenation of the database name and collection name is called a namespace.
- We looked at the sample_training.zips collection and issued the following queries:
  - {"state": "NY"}
  - {"state": "NY", "city": "ALBANY"}

3. Find command

- show dbs
- use sample_training
- show collections
- db.zips.find({"state": "NY"})
  - This gives **1st 20 record**
  - To iterate next use **it.**
  - Different forms of find:
    - db.zips.find({"state": "NY"}).**count()**
    - db.zips.find(**{"state": "NY", "city": "ALBANY"}**)
    - db.zips.find({"state": "NY", "city": "ALBANY"}).**pretty() : to beautify the result.**

4.Insert command:

- *Insert three test documents:*
  - *db.collectionName.insert([{ "test": 1 }, { "test": 2 }, { "test": 3 }])*
- *Insert three test documents but specify the _id values:*
  - *db.inspections.insert([{ "_id": 1, "test": 1 },{ "_id": 1, "test": 2 }, { "_id": 3, "test": 3 }])*
    - ***Only** { "_id": 1, "test": 1 } is inserted and the rest will be discarded since id is duplicated.*

    - *Insert multiple documents specifying the _id values, and using the **"ordered": false** option.*
    - *db.inspections.insert([{ "_id": 1, "test": 1 },{ "_id": 1, "test": 2 },{ "_id": 3, "test": 2 }, { "_id": 3, "test": 3 }],{ "ordered": false })*
    - *db.inspections.find({ "_id": 3 })*
        - *{ "_id": 3, "test": 2 }*

5. Update command:

- **updateOne**
- **updateMany**
- **Update all documents** *in the* zips *collection where the* city *field is equal to* "HUDSON" *by* ***adding* 10 *to the curren****t value of the* "pop" *field.*
    - *db.zips.**updateMany**({ "city": "HUDSON" }, { "**$inc**": { "pop": 10 } })*
        - *({**wherecondition**}, {update operation})*
- ***Update a single document** in the zips collection where the zip field is equal to "12534" by **setting the value** of the "pop" field to 17630.*
    - *db.zips.**updateOne**({ "zip": "12534" }, { "**$set**": { "pop": 17630 } })*
- *Update one document in the grades collection where the student_id is ``250`` *, and the class_id field is 339 , by adding a document element to the "scores" array.*
    - *db.grades.updateOne({ "student_id": 250, "class_id": 339 }, { "$**push**": { "scores": { "type": "extra credit", "score": 100 } } })*

6. Delete command:

- **deleteOne**
- **deleteMany**
- db.inspections.deleteMany({ "test": 1 })
- db.inspections.deleteOne({ "test": 3 })
- *Drop the* inspection ***collection**:*
    - *db.inspection.**drop**()*
- ***remove column in mongo***
    - *db.example.update({}, {$unset: {words:1}}, false, true);*
    - *Refer this:*
    - *http://www.mongodb.org/display/DOCS/Updating#Updating-%24unset*

7. Comparison operator:

Syntax: db.find(<field name>: {<comp operator>:<value>}})
Ex : db.find{“pop”:{$lt:1000}})

Find all documents where the tripduration *was less than or equal to* 70 *seconds and the* usertype *was not* Subscriber:

db.trips.find({ "tripduration": { **"$lte"** : 70 },
"usertype": { **"$ne":** "Subscriber" } }).pretty()

Find all documents where the tripduration *was less than or equal to* 70 *seconds and the* usertype *was* Customer *using a redundant equality operator:*

db.trips.find({ "tripduration": { "$lte" : 70 },
"usertype": { ***"$eq"****: "Customer" }}).pretty()*

Find all documents where the tripduration *was less than or equal to* 70 *seconds and the* usertype *was* Customer *using the implicit equality operator:*

db.trips.find({ "tripduration": { "$lte" : 70 },
"usertype": "Customer" }).pretty()

8.Logical operator :

$and
$or
$nor :fails to match both clause
Syntax: {<operator>:[{statements},{statements}]}
$not:{$not:{statement}}

$and is implicitly present in a query
When we want to use a same operator twice, the $and must used explicitly

Same field;
{“f1‘:{“$lt”:2000, “$gt” : 500}}   **Implicit $and**

Ex:Explicit $and

```
db.routes.find({ "$and": [ { "$or" :[ { "dst_airport": "KZN" },
                                      { "src_airport": "KZN" }
                                    ] },
                           { "$or" :[ { "airplane": "CR2" },
                                      { "airplane": "A81" } ] }
                 ]}).pretty()
```

9.$expr: compare between fields within a document.

Find all documents where the trip started and ended at the same station:

*db.trips.find({ **"$expr"**: { "$eq": [**"$end station id", "$start station id"]** }*
}).*count()*

Find all documents where the trip lasted longer than 1200 seconds, and started and ended at the same station:

```
db.trips.find({ "$expr": { "$and": [ { "$gt": [ "$tripduration", 1200 ]},
                        { "$eq": [ "$end station id", "$start station id" ]}
                      ]}}).count()
```

10.Array operator:

https://docs.mongodb.com/manual/tutorial/query-arrays/

$all: to skip the order.
$size: to select a array which has exactly the specified size
EX: 1.{arrayname : arrayelement1} just to select the array which includes arrayelement1.
2.{arrayname : [arrayelement1,arrayelement2]} to select the array which includes arrayelement1 and arrayelement2 only.
3.{arrayname :{“$all”: [arrayelement2,arrayelement1]}} to select the array which includes arrayelement1 and arrayelement2 irrespective of their order and other elements .
4.{arrayname :{“$size”:20}} to select the array which has exactly 20 elements .

5.Query for an Element by the Array Index Position
db.inventory.find({ "dim_cm.1": { $gt: 25 } })

Find all documents with exactly 20 *amenities which include all the amenities listed in the query array:*

```
db.listingsAndReviews.find({ "amenities": {
                              "$size": 20,
                              "$all": [ "Internet", "Wifi",  "Kitchen",
                                      "Heating", "Family/kid friendly",
                                      "Washer", "Dryer", "Essentials",
                                      "Shampoo", "Hangers",
                                      "Hair dryer", "Iron",
                                      "Laptop friendly workspace" ]
                                    }
                         }).pretty()
```

Using the sample_airbnb.listingsAndReviews collection find out how many documents have the "property_type" "House", and include "Changing table" as one of the "amenities"?

```
db.listingsAndReviews.find({ "property_type": "House",
                          "amenities": "Changing table" }).count()
```

11.Projection: To select the field that needs to be displayed in the result set.

To include field use 1
To exclude use 0
Note: We cannot mix 0 and 1. Except for the id field.

```
db.listingsAndReviews.find({ "amenities": "Wifi" },
                       { "price": 1, "address": 1, "_id": 0 }).pretty()
```

$elemMatch: The `$elemMatch` operator matches documents that contain an ***array field with at least one element that matches all the specified query criteria***

```
{ <field>: { $elemMatch: { <query1>, <query2>, ... } } }
db.grades.find({ "scores": { "$elemMatch": { "type": "extra credit" } }
}).pretty()
```