**Schemas & Relations – How to structure your documents**

* **Command to drop database:**

Get list of DB first -> show dbs

Choose & move to that db -> use flights

Then execute -> db.dropDatabase()

Use this query to remove the single collection in a database.

db.myCollection.drop();

* **Module Introduction:**

**A screenshot of a cell phone

Description automatically generated**

* **Why do we use schema?**

**A screenshot of a cell phone

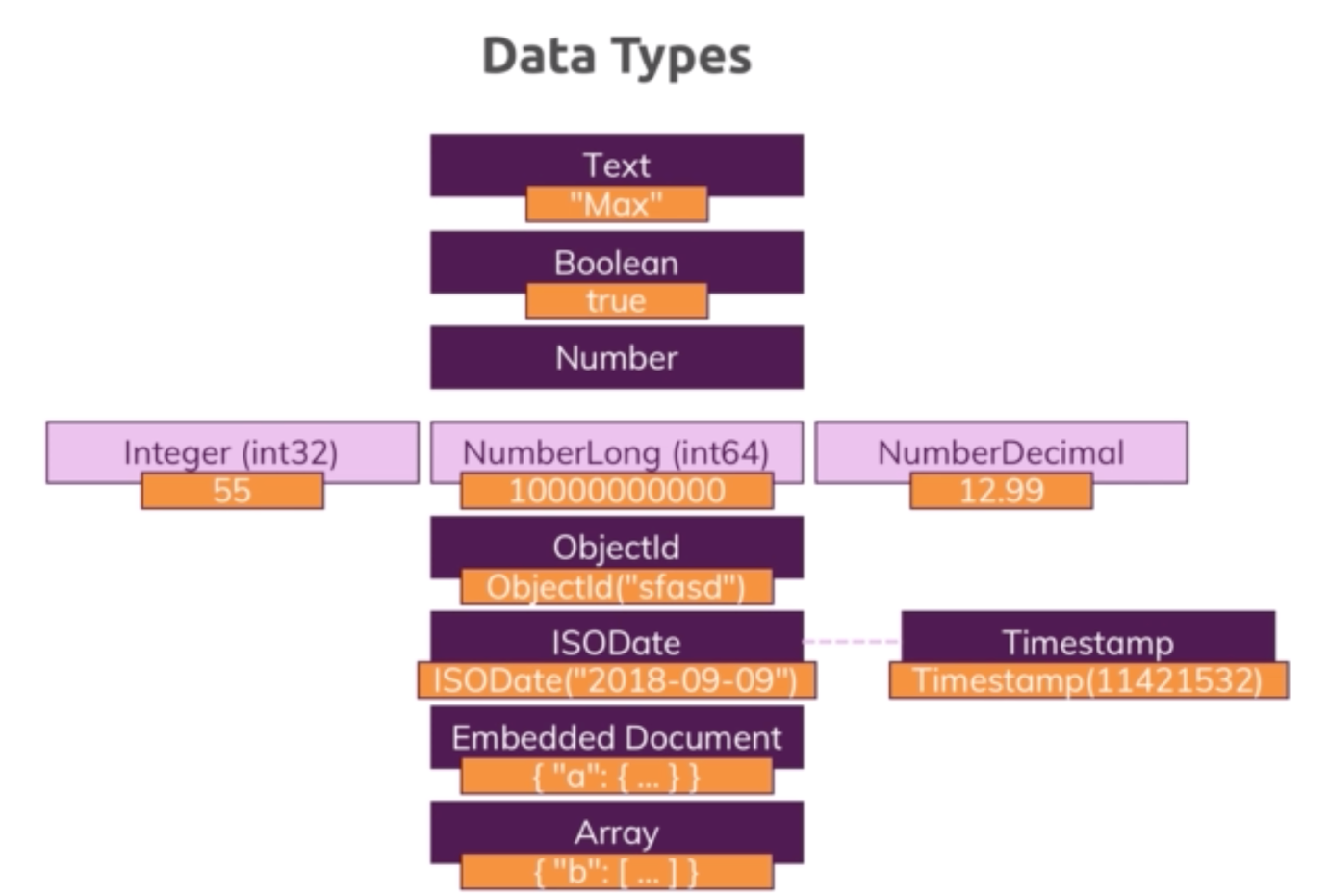
Description automatically generated**

* **Structuring Documents:**

A screenshot of a cell phone

Description automatically generated

* **Data types overview:**

****

* **Data types in action:**

Query contains all values:

**db.companies.insertOne({"name": "Fresh Apples Inc", isStartup: true, employees: 33, funding: 1234567890123456789, details: {ceo: "Mark Super"}, tags: [{title: "super"}, {title:"perfect"}], foundDate: new Date(), insertedAt: new Timestamp()})**

Use db.stats() method to get the details of the DB.

{

"db" : "companyData",

"collections" : 1,

"views" : 0,

"objects" : 1,

"avgObjSize" : 233,

"dataSize" : 233,

"storageSize" : 16384,

"numExtents" : 0,

"indexes" : 1,

"indexSize" : 16384,

"fsUsedSize" : 320059699200,

"fsTotalSize" : 499963170816,

"ok" : 1

}

Note: Always try to use the appropriate data type to save the memory usage, please find the below comparison when we use plain integer and

NumberInt().

**db.numbers.insertOne({a: 1})**

{

"db" : "companyData",

"collections" : 1,

"views" : 0,

"objects" : 1,

"avgObjSize" : 33,

**"dataSize" : 33,**

"storageSize" : 4096,

"numExtents" : 0,

"indexes" : 1,

"indexSize" : 4096,

"fsUsedSize" : 320081477632,

"fsTotalSize" : 499963170816,

"ok" : 1

}

After using NumberInt:

**db.numbers.insertOne({a: NumberInt(1)})**

{

"db" : "companyData",

"collections" : 1,

"views" : 0,

"objects" : 1,

"avgObjSize" : 29,

**"dataSize" : 29,**

"storageSize" : 20480,

"numExtents" : 0,

"indexes" : 1,

"indexSize" : 20480,

"fsUsedSize" : 320085192704,

"fsTotalSize" : 499963170816,

"ok" : 1

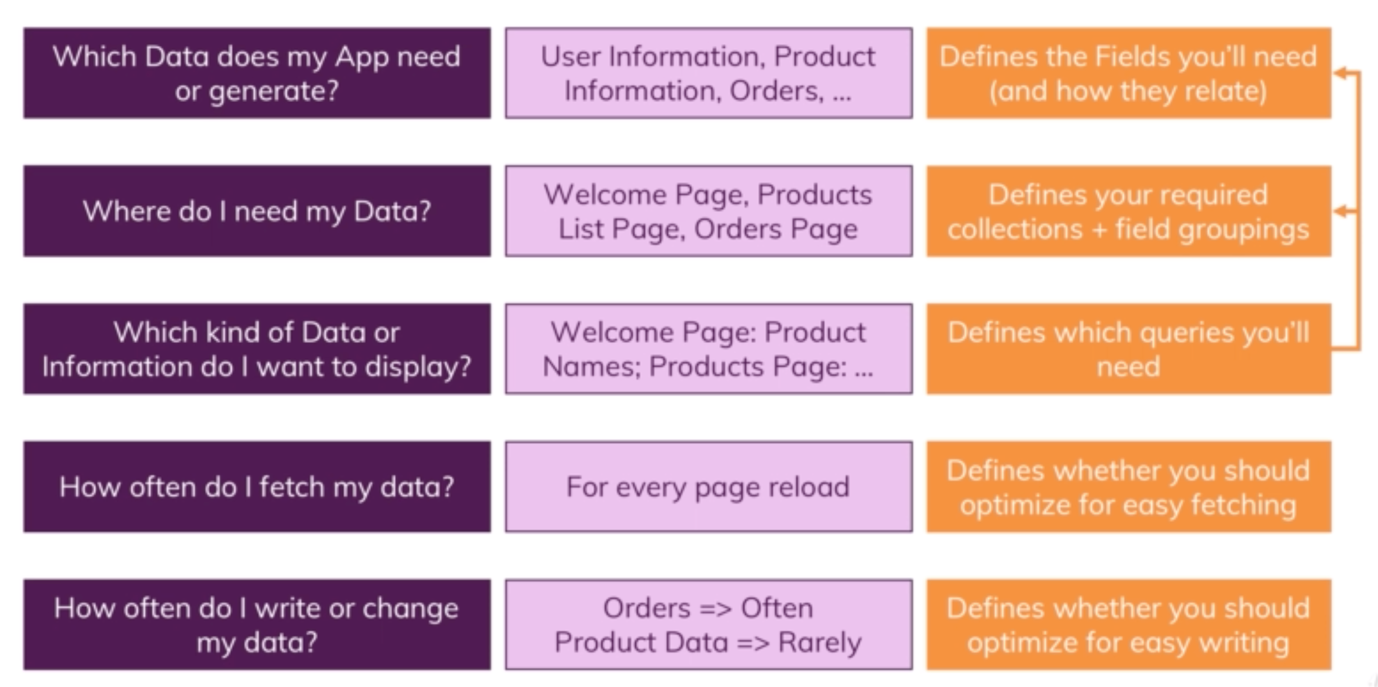
}

Query to get the data type of a column:

typeof db.numbers.findOne().a

Result: **number**

* **Data schemas & data Modelling:**



* **Understanding Relations:**

A screenshot of a cell phone

Description automatically generated

* **One To One – Embedded:**

**db.patients.insertOne({name: "Saravana", age: 31, diseaseSummary: "summary-saravana-1"})**

{

"\_id" : ObjectId("5f1d1106f4ade3fbde32a56d"),

"name" : "Saravana",

"age" : 31,

"diseaseSummary" : "summary-saravana-1"

}

**var disd = db.patients.findOne().diseaseSummary**

**db.diseaseSummaries.findOne({\_id: disd})**

{ "\_id" : "summary-saravana-1", "diseases" : [ "Cold", "Cough" ] }

Note: In case of strong one to one relationship, it is better to keep it as an embedded document.

**db.patients.insertOne({name: "Saravana", age: 31, diseaseSummary: {disease: ["Cold", "Cough"]}})**

{

"\_id" : ObjectId("5f1db7acf4ade3fbde32a56e"),

"name" : "Saravana",

"age" : 31,

"diseaseSummary" : {

"disease" : [

"Cold",

"Cough"

]

}

}

* **One to One using Reference:**

**A screenshot of a cell phone

Description automatically generated**

Note: Keeping the one to one relation as an embedded document and a separate collection depends on the use case.

Let’s say if we have a web application that needs to print the car details separately, in this case it will be better to go for a separate collection.

* **One To Many – Embedded:**

A screenshot of a cell phone

Description automatically generated

**db.questionThreads.insertOne({creator: "Saravana", question: "How does that work?", answers: ["q1a1", "q1a2"]})**

{

"\_id" : ObjectId("5f1dbb3cf4ade3fbde32a56f"),

"creator" : "Saravana",

"question" : "How does that work?",

"answers" : [

"q1a1",

"q1a2"

]

}

**db.answers.insertMany([{\_id: "q1a1", text: "It works like that"}, {\_id: "q1a2", text: "It also works this way"}])**

{ "\_id" : "q1a1", "text" : "It works like that" }

{ "\_id" : "q1a2", "text" : "It also works this way" }

**Note: Even for this scenario it will be better to go for an embedded document.**

**db.questionThreads.insertOne({creator: "Saravana", question: "How does that work?", answers: [{text: "It works this way"}, {text: "It works that way also"}]})**

{

"\_id" : ObjectId("5f1dbd9df4ade3fbde32a570"),

"creator" : "Saravana",

"question" : "How does that work?",

"answers" : [

{

"text" : "It works this way"

},

{

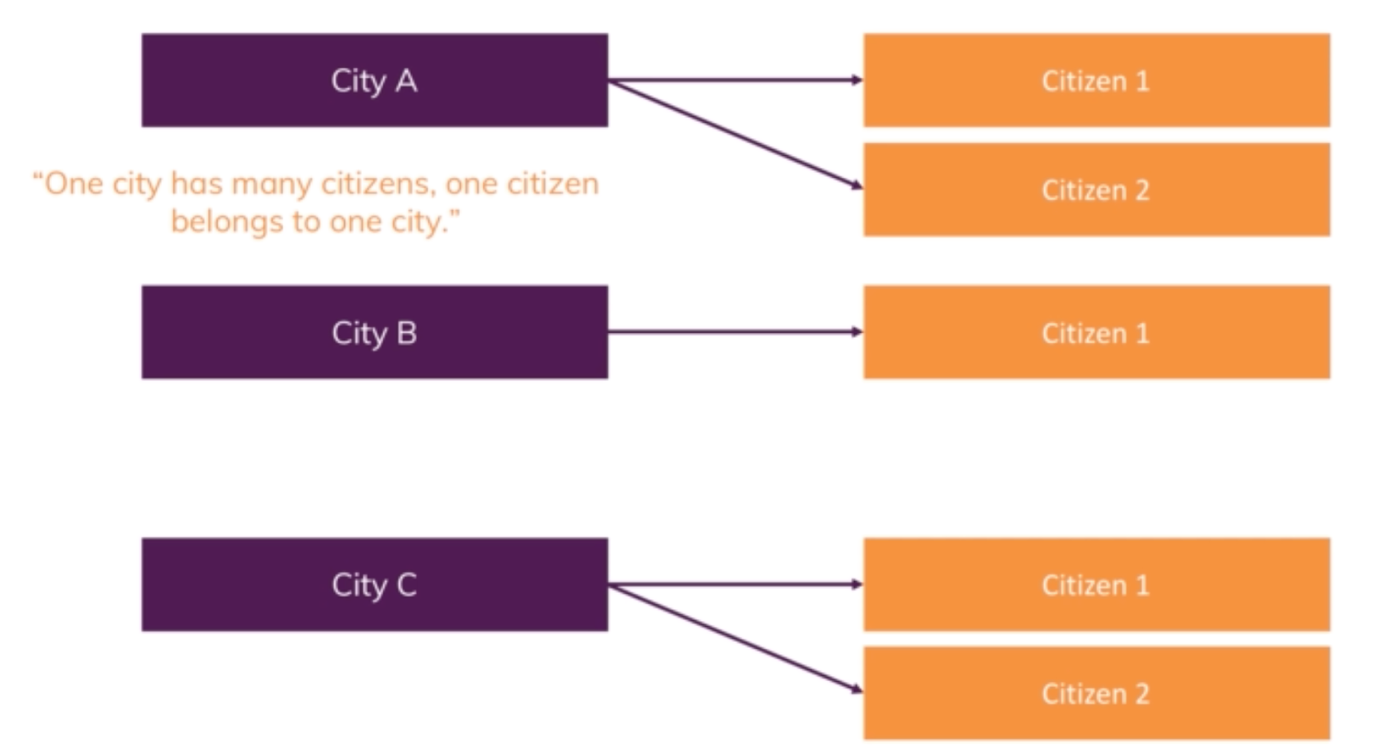
"text" : "It works that way also"

}

]

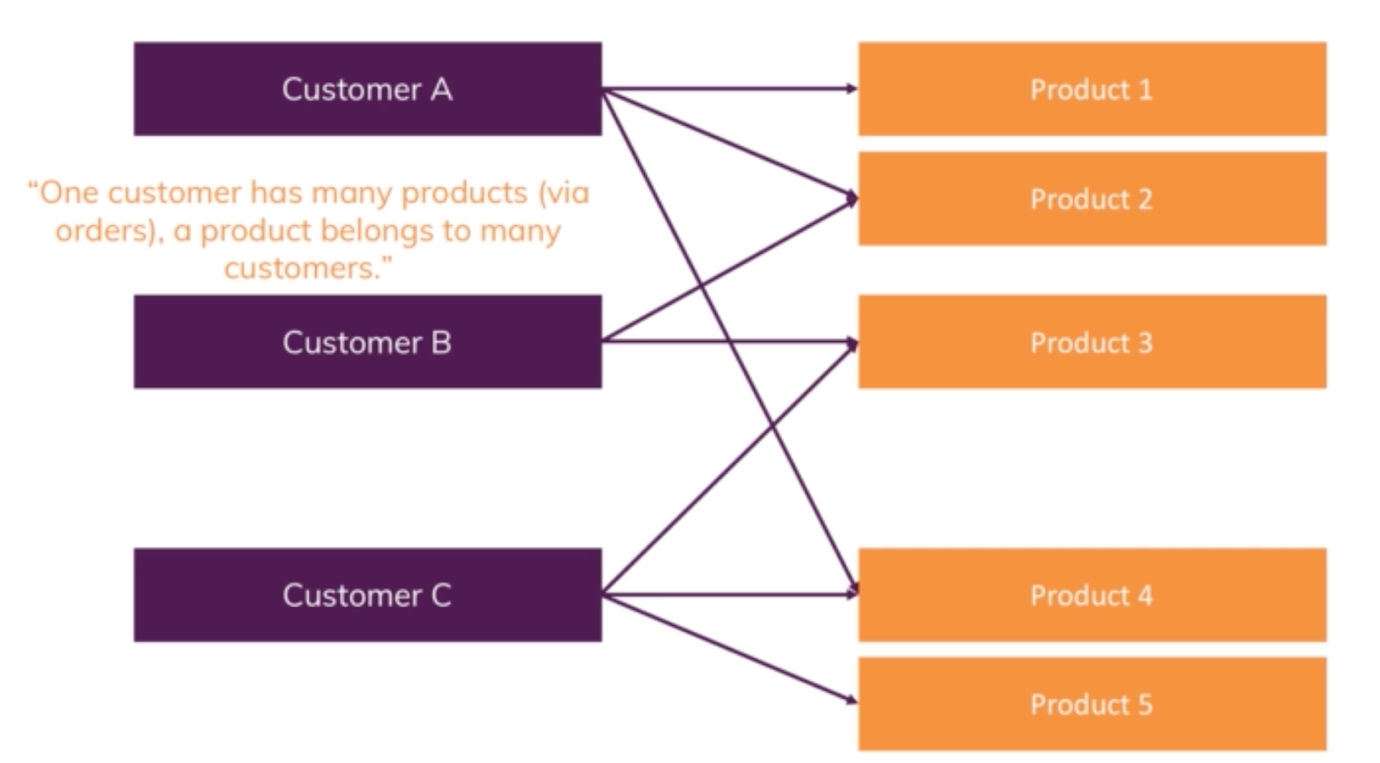
}

* **One To Many – Using Reference:**

****

In case of One to Many with above scenario, it will be better to go for reference as we many need the city or citizen details alone to display somewhere.

* **Many to Many:**

****

SQL approach:

db.products.insertOne({title: "Spring Microservices in Action", price: 20.00})

db.customers.insertOne({name: "Saravana", age: 31})

db.orders.insertOne({productId: ObjectId("5f22fa8f6e949f15db33fab3"), customerId: ObjectId("5f22fab16e949f15db33fab4")})

In NoSQL we don’t need a third (orders) table, we can achieve it by two tables itself.

db.orders.drop();

db.customers.updateOne({}, {$set: {orders: [{prodcutId: ObjectId("5f22fa8f6e949f15db33fab3"), quantity: 2}]}})

{

"\_id" : ObjectId("5f22fab16e949f15db33fab4"),

"name" : "Saravana",

"age" : 31,

"orders" : [

{

"prodcutId" : ObjectId("5f22fa8f6e949f15db33fab3"),

"quantity" : 2

}

]

}

We can use embedded approach also, but it will end up in duplication of product details.

Apart from duplication issue, in case of change in product details, we have to update the details everywhere in the collection.

db.customers.updateOne({}, {$set: {orders: [{title: "Spring Microservices in Action", price: 20.00, quantity: 2}]}})

{

"\_id" : ObjectId("5f22fab16e949f15db33fab4"),

"name" : "Saravana",

"age" : 31,

"orders" : [

{

"title" : "Spring Microservices in Action",

"price" : 20,

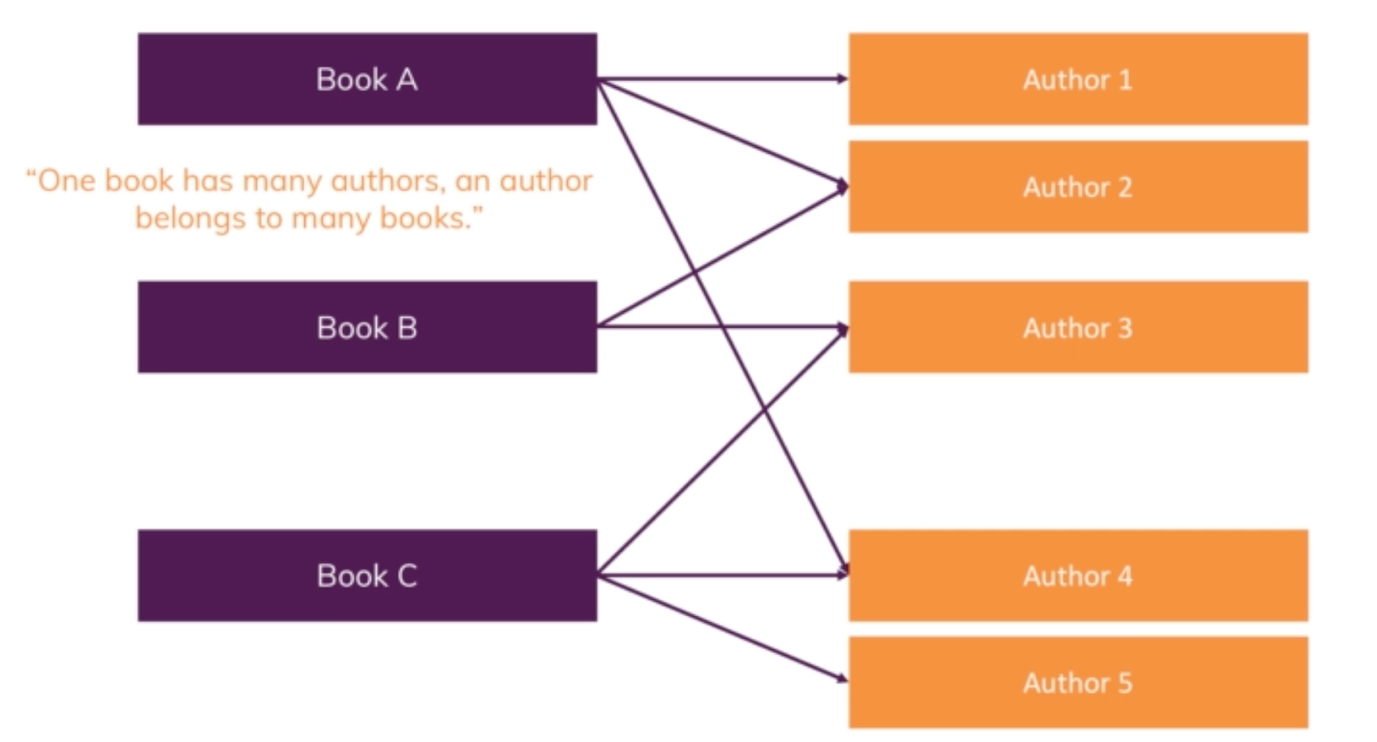
"quantity" : 2

}

]

}

* **Many to Many – Using reference:**

****