MongoDB

Kevin Swingler

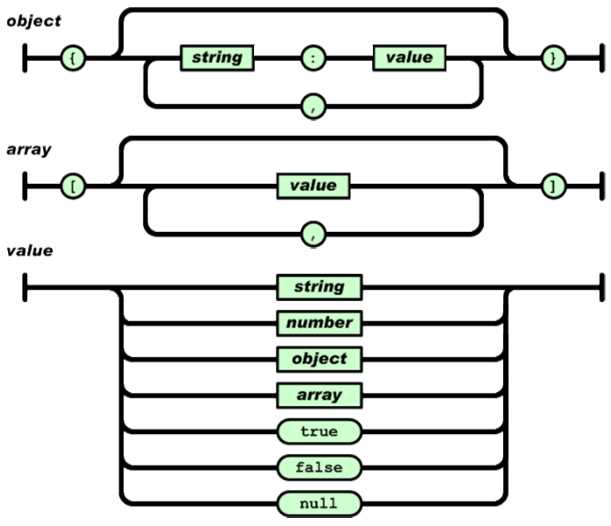
Introduction

- MongoDB is a *document database*
- It stores data in Binary JSON (BSON) but you interact with it in JSON
- Documents are JSON objects and are organised into collections
- Each document has an ID

Example Document

```
{
Id:847543,
Name:iPhone5,
Features:[GPS,Retina Display,Siri],
Reviews:[
{Reviewer:458743,Date:12.4.1013,Speed:
  Slow},
{Reviewer:636534,Date:2.5.1013,Camera:
  Great},
]
}
```

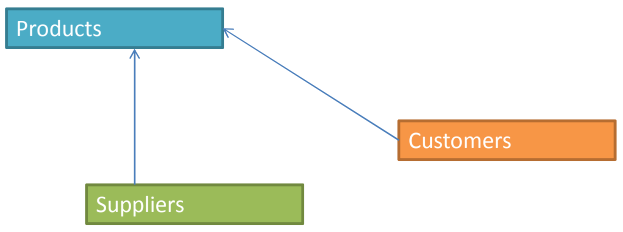
An Aside: JSON



Database Design

- Not a relational database, but can support relationships
- In the previous example, the reviewer ID refers to another document – the one with an ID of `458743`, for example
- Note that ANY document can refer to ANY other, so there is not the same kind of ER structure you get in relational databases

However ...



There may still be relationships in the real world that you want to reflect.

Primary Keys

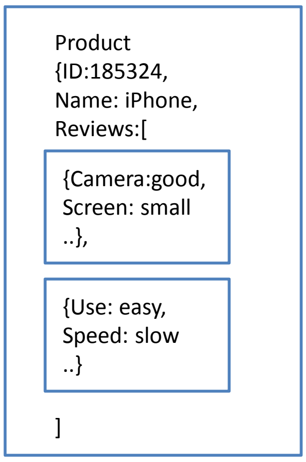
- MongoDB automatically assigns each document a unique key
  - Key is unique as generated from timestamp, machine ID, process ID and a counter
- You can, if you want, override that and provide your own keys, but for large automatically generated data sets, the auto option is probably best

Foreign Keys

- As there are no relational schema, there are no fields defined in a document, (other than the PK) and so nowhere to define foreign keys
- Foreign keys can be implemented by simply including the ID of one document another
- Integrity is not enforced by the DB – you must do it at the application level

Documents in Documents

- A document in MongoDB can contain other documents (just as in JSON)



```
Product
{ID:185324,
Name: iPhone,
Reviews:[
  {Camera:good,
  Screen: small
  ..},

  {Use: easy,
  Speed: slow
  ..}
]
```

Query Language

- CRUD (yes, really)

  Create, Read, Update, Delete

Create

- Create a collection explicitly:
```
db.creatCollection(“books”)
```

- See what collections your database has
```
show collections
```

Insert

- Insert into a collection
```
db.books.insert(doc)
```
- Where doc is defined as a JSON object
```
doc={Name:"On the Road",
Author: "Jack Kerouac"}
```
- Short cut:
```
b=db.books
b.insert(...)
```

Read

- Find everything in a collection

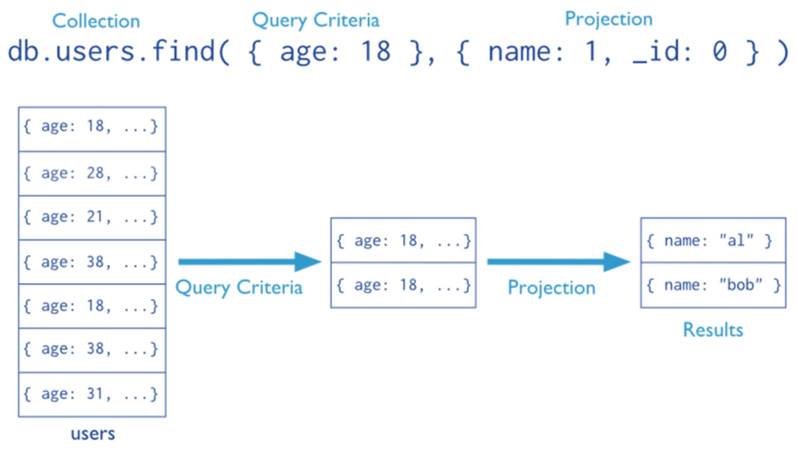
```
> db.books.find()

  { "_id" : ObjectId("52658e0a84b47fef69ebab5f"), "Name" :
  "On the Road", "Author" : "Jack Kerouac" }
  { "_id" : ObjectId("52658e4984b47fef69ebab60"), "Name" :
  "Collected Poems", "Editor" : "J.E Bowles" }
```

Read

```
db.col.find({JSON query},{Projection})
```

- The JSON query tells the DB what to search for
- The projection tells the DB which fields to return

Example



http://docs.mongodb.org/manual/core/read-operations-introduction/

Read

- Find a specific thing

```
b.find({Name : "Collected Poems"})
```

- Or all documents with a specific key

```
db.books.find( { Editor:{ "$exists":"true"}})
```

Operators

- Note the `$Exists()`
- Operators in MongoDB are words with a `$` in front
- You will see more presently ...

Numbers

- Numeric data is inserted into MongoDB without quotes, just as we'd treat a number in any programming language

```
db.eg.insert({Name : "Bill", Age : 18})
db.eg.insert({Name : "Ted", Age : 17})

db.eg.find({Age:17})
```

Ranges

- Find numbers in a range

```
db.eg.find({Age: {$lt : 18}})
```

Other useful operators include:

`$lte` - Less than or equal
`$gt` - Greater than
`$gte` - Greater than or equal
`$ne` - Not equal

Modifiers

- Sort

```
db.eg.find({Age: {$lt : 18}}).sort({Age:1})
```

- Limit

```
db.eg.find({Age: {$lt : 18}}).limit(5)
```

Projection

- Retrieve only some of the fields in a document

```
db.eg.find({Age: : 18} , {name:1,email:1}
```

- Or exclude some

```
db.eg.find({Age: : 18} , {address:0}
```

Cursors

- A cursor is used to iterate through the results of a find()

```
var myCursor = db.inventory.find();
var myFirstDocument = myCursor.hasNext() ?
  myCursor.next() : null;
```

Javascript

- The programming language that is native to MongoDB is Javascript
- You can type Javascript into the MongoDB shell, just as you can type CRUD commands

Javascript

- Set a variable and insert
```
var a=1
db.c.insert({val:a})
```
- Search
```
var b=2
db.c.find({a:b-1})
```
- JSON
```
var doc={Name:”Kevin”, Age:21}
db.c.insert(doc)
```

Arrays

- To search for each and any of an array of possible matches on the same field, you can specify an array of target values. This is better than a string of `$or` operators.

```
db.eg.find({Name : { $in : ["Bill" , "Ted"]}})
```

- Note the `[ ]` square brackets to denote the array.

Arrays

- You can insert an array as a value in a document:

```
db.eg.insert({Likes: ["Ice cream","Apples","Chocolate"]})
```

- and search for one or more items in it:

```
db.eg.find({Likes: { $in: ["Apples","Gin"]}})
```

- If you want to match a single term to the array, the single term still needs to be in an array of length one:

```
db.eg.find({Likes: { $in: ["Apples"]}})
```

- The opposite of `$in` is `$nin`, which means not in.

Searching an Array in the DB

- If a database field is an array, you can search the array in exactly the same way as matching a single field

```
{“Entries”:[“a”, “b”, “c”]}
```

Is found, for example with

```
find({“Entries” : “a”})
```

Manipulating Arrays

- You can add and remove elements from an array in a document using:
- `$addToSet` Adds elements to an array only if they do not already exist in the set.
- `$pop` Removes the first or last item of an array.
- `$pullAll` Removes all matching values from an **array**.
- `$pull` Removes all array elements that match a specified **query**.
- `$push` Adds an item to an array.

- See the practical session on this for more detail

- http://docs.mongodb.org/manual/reference/operator/update-array/

Regular Expressions

- You can build some reasonably complex searches with a mixture of AND, OR, and NOT, but logical expressions are less useful if you want to match certain classes of string. For example, a search to find all the names that include a number (Like Joe90 or Ben10). For this, we need regular expressions.
- Enclose the expression in /.../o
- o is a set of options

Update

- Start with

```
{   Country: France,
    Capital:
    { Name:Paris,
      Population:5000000
    }
}
```

Add a Field

```
db.colc.update(
{ "Country" : "France“},
{ $set: {“Language”: “French”}
)
```

Update a Field

```
db.colc.update(
{ "Country" : "France“},
  {
     $set: { “Capital.Population”: 20
  }
}
)
```

Regular Expressions

Find names that contain "ev" anywhere

```
find({Name: /ev/})
```

Find names that start with "K", ignoring case

```
find({Name: /^K/i})
```

Find names that contain "a" or "b" or "c" anywhere

```
find({Name: /[abc]/})
```

Find names that end with some numbers

```
find({Name: /\w{1,}\d{1,}/})
```

Delete

- Remove documents with a certain value

```
db.books.remove({Editor:"J.E. Bowles"})
```

Child Objects

- Take this entry as an example:

```
{   Country: France,
    Capital:
    { Name:Paris,
      Population:5000000
    }
}
```

Child Objects

- How to query the population of the capital?

```
db.cities.find({},{“Capital.Population”:1})
```

- Only returns the population of the capital

```
db.cities.find({“Capital.Population”:”$lt”:
  1000000})
```

- Returns cities with a capital population of less than 1000000

Practicalities

- MongoDB is Free download and install it and have a go

  www.mongodb.org/downloads

  Runs in two command line windows – one for the server (Mongod) and one for the client (mongo)

Web Links

Home
  www.mongodb.org/

Install
  www.mongodb.org/downloads

CRUD
  docs.mongodb.org/manual/crud/

Regular Expressions
  docs.mongodb.org/manual/reference/operator/query/regex/
  www.w3schools.com/jsref/jsref_obj_regexp.asp

Try it live
  try.mongodb.org/