CS190 (CS122D): Beyond SQL Data Management

– Lecture #8 –

Mike Carey
UC Irvine
mjcarey@ics.uci.edu
Announcements

• Hopefully all’s going smoothly up in Astra-land!

<table>
<thead>
<tr>
<th>HW</th>
<th>Available</th>
<th>Due Date/Time</th>
<th>HW Topic</th>
<th>Setup Info</th>
<th>Details</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW1</td>
<td>Mo 4/05</td>
<td>Th 4/15 (11:59 PM)</td>
<td>SQL Review</td>
<td>HW1 Setup</td>
<td>HW1 Details</td>
<td>HW1 Solution</td>
</tr>
<tr>
<td>HW2</td>
<td>Th 4/15</td>
<td>Mo 4/26 (11:59 PM)</td>
<td>Cassandra</td>
<td>HW2 Setup</td>
<td>HW2 Details</td>
<td>HW2 Solution</td>
</tr>
</tbody>
</table>

• Let’s check on where we are, course-wise...

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>HW</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>W 4/14</td>
<td>Column-family stores: BigTable, Cassandra</td>
<td>Ch. 10 NoSQL Distilled (old!)</td>
<td></td>
</tr>
<tr>
<td>M 4/19</td>
<td>Column-family stores: Cassandra (cont.)</td>
<td>Cassandra materials (as needed)</td>
<td></td>
</tr>
<tr>
<td>W 4/21</td>
<td>Document stores: JSON and MongoDB</td>
<td>Ch. 9 NoSQL Distilled (old!)</td>
<td></td>
</tr>
<tr>
<td>M 4/26</td>
<td>Document stores: MongoDB (cont.)</td>
<td>MongoDB materials (as needed)</td>
<td></td>
</tr>
<tr>
<td>W 4/28</td>
<td>NoSQL DB design principles</td>
<td>Ch. 3 NoSQL Distilled</td>
<td></td>
</tr>
<tr>
<td>M 5/03</td>
<td><strong>Midterm Exam (Checkpoint)</strong></td>
<td><strong>3:30-4:50 PM -- be there!!!</strong></td>
<td></td>
</tr>
</tbody>
</table>

• **Today**: *JSON and MongoDB (I)*
  • Starting our document DB sub-tour...!
JSON (JavaScript Object Notation)

• Lightweight text format for *data interchange*

• Popular due to its simplicity and ease of use
  • Humans can read/write it
  • Machines can parse/generate it
  • See [https://www.json.org/](https://www.json.org/) for the spec

• Built on two structures found in most PL’s
  • A collection of name/value pairs
    • *a.k.a.* object, record, struct, dictionary, hash table, keyed list, associative array, ...
  • An ordered list of values
    • *a.k.a.* array, vector, list, sequence, ...

Q: Why ordered and not unordered...?
JSON Objects

object

{ whitespace
  field name
  whitespace
  string
  whitespace
  : value

}
JSON Arrays
JSON Values

![JSON Value Diagram]

- value
  - whitespace
  - string
  - number
  - object
  - array
  - true
  - false
  - null

vs. missing
JSON Numbers
JSON Whitespace

![Diagram of JSON whitespace with categories: space, linefeed, carriage return, horizontal tab]
Simple JSON Object Example

```json
{
"id": 1,
"name": "Foo",
"price": 123,
"tags": [
"Bar",
"Eek"
],
"stock": {
"warehouse": 300,
"retail": 20
}
}
```
Some Things to Notice

• Each JSON object is *self-describing*
  • Everything you need to know is contained therein

• There is no notion, in JSON, of “defining” a type
  • Every object is built from “first principles”

• The JSON world can be highly heterogeneous
  • May use a given field name differently in different objects
  • Each element in a given array can be of any type
  • “Any structural or type similarity from one JSON object to the next is purely coincidental.” (😊)

• In practice, of course, an application may have an unspoken (or spoken) intended schema for an object
JSON Schema

• An optional schema “add on” to the JSON world
  • Can be used as technical documentation
  • Can be used to validate data being exchanged
  • Expressed in JSON (of course!)
  • See https://json-schema.org/ for the spec

• Gaining in popularity
  • There are quite a few schema validators available
  • Some NoSQL databases support it (including MongoDB)
  • Both basic JSON and JSON Schema validators exist online, if you are interested
Simple JSON Schema Example

```json
{
  "$schema": "http://json-schema.org/schema#",
  "title": "Product",
  "description": "Some product data",
  "type": "object",
  "required": ["id", "name", "price"],
  "properties": {
    "id": {
      "type": "number",
      "description": "Product identifier"
    },
    "name": {
      "type": "string",
      "description": "Name of the product"
    },
    "price": {
      "type": "number",
      "minimum": 0
    }
  },
  ...}
```

```json
...
  "tags": {
    "type": "array",
    "items": {
      "type": "string"
    }
  },
  "stock": {
    "type": "object",
    "properties": {
      "warehouse": {
        "type": "number"
      },
      "retail": {
        "type": "number"
      }
    }
  }
```

M. Carey, Spring 2021: CS122D
JSON Schema Example (I)

```json
{
  "$schema": "http://json-schema.org/schema#",
  "title": "Product",
  "description": "Some product data",
  "type": "object",
  "required": ["id", "name", "price"],
  "properties": {
    "id": {
      "type": "number",
      "description": "Product identifier"
    },
    "name": {
      "type": "string",
      "description": "Name of the product"
    },
    "price": {
      "type": "number",
      "minimum": 0
    },
    ...
  }
}
```

- **Version of Schema standard**
- **Title for this schema (doc)**
- **Description of this schema (doc)**
- **JSON type of target data**
- **Mandatory object fields**
- **Type details of each field**
- **Constraint on price values**
JSON Schema Example (II)

```
{ "id": 1,  
  "name": "Foo",  
  "price": 123,  
  "tags": [  
    "Bar",  
    "Eek"
  ],  
  "stock": {  
    "warehouse": 300,  
    "retail": 20
  }
}

PASS

...  
"tags": {  
  "type": "array",  
  "items": {  
    "type": "string"
  },  
  
  "stock": {  
    "type": "object",  
    "properties": {  
      "warehouse": {  
        "type": "number"
      },  
      "retail": {  
        "type": "number"
      }
    }
  }
}```
Document Stores

• “A **document-oriented database**, or **document store**, is a **computer program** designed for storing, retrieving and managing document-oriented information, also known as **semi-structured data**.” *(from Wikipedia)*

• Common characteristics include
  • “Come as you are” for data (no schema)
  • JSON-based document data model
  • Richer functionality than a simple KV store
    • Some degree of query capabilities
    • Secondary indexes (and other goodies)

• MongoDB is our first example...
MongoDB Architecture

1. Dynamic Document Schema
   ```json
   {  
      name: "John Smith" ,  
      date: "2013-08-01" ,  
      address: "10 3rd St. " ,  
      phone: [  
         { home: 1234567890 } ,  
         { mobile: 1234568138 } ]  
   }
   ```

2. Native language drivers
   ```javascript
   db.customer.insert({...})
   db.customer.find({  
      name: "John Smith"  
   })
   ```

3. High availability
   - Replica sets

4. High performance
   - Data locality
   - Rich Indexes
   - RAM

5. Horizontal scalability
   - Sharding
Proper document schema design yields more entity data per document than found in a relational database row.
A (Very) Simple Example

customers:

{ "custid": "C13",  
  "name": "T. Cruise",  
  "address": {  
    "street": "201 Main St.",  
    "city": "St. Louis, MO",  
    "zipcode": "63101" },  
  "rating": 750
}, ...

orders:

{ "orderno": 1002,  
  "custid": "C13",  
  "order_date": "2017-05-01",  
  "ship_date": "2017-05-03",  
  "items": [  
    { "itemno": 460,  
      "qty": 95,  
      "price": 100.99 },  
    { "itemno": 680,  
      "qty": 150,  
      "price": 8.75} 
  ]
}, ...

M. Carey, Spring 2021: CS122D
Mongo Shell

For more information on usage: https://docs.mongodb.com/manual/reference/method

```javascript
Enterprise atlas-yulqg91-shard-0 [primary]> show dbs
demodb 41 kB
admin 332 kB
local 4.28 GB
```

```javascript
Enterprise atlas-yulqg91-shard-0 [primary]> db.customers.find({name: "T. Cruise"})
```
Compass

MongoDB Compass - cs122d.6qf6u.mongodb.net/demodb.customers

Documents

 документов 7

TOTAL SIZE 1.0KB
AVG. SIZE 148B

INDEXES 1
TOTAL SIZE 20.0KB
AVG. SIZE 20.0KB

FIND
RESET

FILTER
{name: "T. Cruise"}

Displaying documents 1 - 1 of 1

One document found:

_id: ObjectId("607f371a9599923cac33de0b")
custid: "C13"
name: "T. Cruise"
address: {
  street: "201 Main St."
city: "St. Louis, MO"
zipcode: "63101"
}

rating: 750
Jupyter + PyMongo

This will give you programmatic access experience (plus a nice lab notebook)!
A Quick Preview

• Excellent “out of box” user experience
  • Can get something up and running quickly
• MQL = Mongo Query Language  \(\text{Not SQL!}\)
  • `db.mycoll.find( filter )`
  • `db.mycoll.aggregate( pipeline )`
  • (Limitations w.r.t. joins)
• All the expected CRUD functionality
  • `create( insert ), retrieve( find/aggregate ), update, delete`
  • `One and Many` variants of most CRUD operations
  • Some interesting bells & whistles
• Multiple data formats
  • JSON and BSON (Binary JSON)
• Critical feature warning...!
  • Unable to connect to Atlas while on Starbucks WiFi... (😭)
```
In [1]:
import pymongo
from bson.json_util import dumps

# uri = "your-Atlas-instance-url-goes-here"
uri = "mongodb+srv://dbabss:CS122dDemo789?!@cs122d.6q6u.mongodb.net/demodb?retryWrites=true&w=majority"
client = pymongo.MongoClient(uri)
demodb = client.demodb

In [2]:
customers = [
    {"custid": "C13", "name": "T. Cruise", "address": { "street": "201 Main St.", "city": "St. Louis, MO", "zipcode": "63101" }},
    {"custid": "C25", "name": "M. Streep", "address": { "street": "690 River St.", "city": "Hanover, MA", "zipcode": "02339" }},
    {"custid": "C31", "name": "B. Pitt", "address": { "street": "360 Mountain Ave.", "city": "St. Louis, MO", "zipcode": "63101" }},
    {"custid": "C35", "name": "J. Roberts", "address": { "street": "420 Green St.", "city": "Boston, MA", "zipcode": "02339" }},
    {"custid": "C37", "name": "T. Hanks", "address": { "street": "120 Harbor Blvd.", "city": "Boston, MA", "zipcode": "02339" }},
    {"custid": "C41", "name": "B. Duvall", "address": { "street": "150 Market St.", "city": "St. Louis, MO", "zipcode": "63101" }},
    {"custid": "C47", "name": "S. Loren", "address": { "street": "Via del Corso.", "city": "Rome, Italy" }, "rating": 625}
]

In [3]:
demodb.customers.insert_many(customers)
Out[3]: <pymongo.results.InsertManyResult at 0x7ff8409fed80>

In [4]:
# count the number of customers
demodb.customers.count_documents( {} )
Out[4]: 7

In [6]:
# SELECT * FROM customers LIMIT 1

cursor = demodb.customers.find_one( {} )
dumps(cursor, indent=2)
{
    "_id": {
        "$oid": "607faff62e40c6bdeb5a1a3"
    },
    "custid": "C13",
    "name": "T. Cruise",
    "address": {
        "street": "201 Main St.",
        "city": "St. Louis, MO",
        "zipcode": "63101"
    },
    "rating": 750
}
```

Note: Every object has an _id
Note: Creates the collection if needed!
Note: Simple operations are simple
(Live demo?)
Questions?

{JSON} + mongoDB