CS122D: Beyond SQL Data Management
   —Lecture #9 —

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

• Hopefully HW 2’s wrapping/wrapped up smoothly!

<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>
<tr>
<td>HW3</td>
<td>Mo 4/26</td>
<td>Th 5/06 (11:59 PM)</td>
<td>MongoDB</td>
<td>HW3 Setup</td>
<td>HW3 Details</td>
<td>HW3 Solution</td>
</tr>
</tbody>
</table>

• Let’s check in again on where we are:

<table>
<thead>
<tr>
<th>Date</th>
<th>Lecture Topic</th>
<th>Resource/Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>W 4/21</td>
<td>Document stores: JSON and MongoDB</td>
<td>Ch. 9 NoSQL Distilled (old!)</td>
</tr>
<tr>
<td>M 4/26</td>
<td>Document stores: MongoDB (cont.)</td>
<td>MongoDB materials (as needed)</td>
</tr>
<tr>
<td>W 4/28</td>
<td>NoSQL DB design principles</td>
<td>Ch. 3 NoSQL Distilled</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>
</tr>
</tbody>
</table>

• Today: **MongoDB (II)**
  • **Mongo Query Language (MQL) deep dive (plus odds & ends)...**
Basic MongoDB Functionality

• “CRUD”: Create, Retrieve, Update, Delete
  • SQL statements: INSERT, SELECT, UPDATE, DELETE
• MQL = Mongo Query Language
  • db.mycoll.find( filter, projection )
  • db.mycoll.aggregate ( pipeline )
  • ...
• Provides all the expected CRUD functionality
  • create (insert), retrieve (find/aggregate), update, delete
  • _one and _many variants of most CRUD operations
  • Some interesting bells & whistles (e.g., $graphLookup)
• Multiple data formats
  • JSON and BSON (Binary JSON)

We’ll focus on these...!
# MongoDB vs. SQL Concepts (Review)

<table>
<thead>
<tr>
<th>SQL DBMS</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Database</td>
</tr>
<tr>
<td>Table</td>
<td>Collection</td>
</tr>
<tr>
<td>Row</td>
<td>Document</td>
</tr>
<tr>
<td>Column</td>
<td>Field</td>
</tr>
<tr>
<td>Parent/Child Table</td>
<td>Embedded Sub-Document or Array</td>
</tr>
<tr>
<td>Join</td>
<td>Linking (vs. Embedding)</td>
</tr>
<tr>
<td>Index</td>
<td>Index</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Example Data (for Compass/Jupyter demos)

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}  
  ]  
}, ...
```
MongoDB Compass

• Database browsing
  • Databases and their collections
  • Works for both local and Atlas MongoDB instances

• Interactive filtering
  • Also helpful for filter debugging (😊)

• Schema ‘stat’-ing
  • To understand your data’s characteristics

• Other features (tabs) you should explore
  • Indexes, Explain Plan, Aggregations

• Okay, let’s take a quick look:
MQL (PyMongo+Jupyter Demo)

• Database and collection creation
• Querying using `db.collection.find(filter, projection)`
  • SQL SELECT use cases and their MQL equivalents
• Basic MQL CRUD operations
  • SQL INSERT, UPDATE, and DELETE use cases
• Querying using `db.collection.aggregate(pipeline)`
  • MongoDB pipeline principles and operators
  • SQL aggregation and GROUP BY use cases
  • See next slide for some important mapping info
• Joins in MQL
  • `$lookup` operator in pipelines (and its limitations)
• Okay, let’s start with find: [PyMongo+Jupyter demo](#)
# MongoDB Pipelines vs. SQL

<table>
<thead>
<tr>
<th>SQL Terms/Functions/Concepts</th>
<th>MongoDB Aggregation Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE</td>
<td>$match (same <em>filter</em> language)</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>$group</td>
</tr>
<tr>
<td>HAVING</td>
<td>$match (after $group)</td>
</tr>
<tr>
<td>SELECT</td>
<td>$project</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>$sort</td>
</tr>
<tr>
<td>LIMIT</td>
<td>$limit</td>
</tr>
<tr>
<td>SUM</td>
<td>$sum</td>
</tr>
<tr>
<td>COUNT</td>
<td>$sum (of 1’s)</td>
</tr>
<tr>
<td>OUTER JOIN</td>
<td>$lookup (only if unsharded)</td>
</tr>
</tbody>
</table>

https://docs.mongodb.com/manual/reference/sql-aggregation-comparison/
MongoDB Availability and Scaling

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

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

3. High availability
   - Replica sets
     - Shard 1
       - Primary
       - Secondary
       - Secondary
     - Shard 2
       - Primary
       - Secondary
       - Secondary
     - Shard N
       - Primary
       - Secondary
       - Secondary

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

5. Horizontal scalability
   - Sharding
MongoDB Replica Sets

• Groups of *mongod* instances that maintain the same data set

• Primary and secondary nodes
  • Primary receives all writes and *asynchronously* replicates its operations to secondaries
  • Primary acknowledges writes with \{ w: "majority" \} write concern
  • Clients may specify a *read preference* to send their read operations to secondaries
MongoDB Consistency Options

• **Read concern** options
  • *local*: returned data could still be rolled back
  • *majority*: returned data was written to a *majority*
  • *linearizable*: see all majority writes as of start of read
  • See documentation for these and a few other options

• **Write concern** (acknowledgement) options
  • `{ w: <value>, j: <boolean>, wtimeout: <number> }`
    • *w*: write has been propagated to this many copies
    • *j*: write has been journaled (i.e., recorded in the log)
    • *wtimeout*: to avoid indefinite blocking of write operation

• Support for ACID transactions
  • Pretty recently added (both intra- and inter-shard)

• See MongoDB docs for more info on all of these
Questions?