CS122D: Beyond SQL Data Management — Lecture #10 —

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

• Hopefully HW3 is going well so far!

<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:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<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: **NoSQL DB design!**  *(after finishing up from last time)*
  • **Depending on time** – Syllabus-driven brain dump; early Q&A
  • **Note** – HW3 would be a useful pre-midterm study exercise!
# 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><code>$match</code> (same <em>filter</em> language)</td>
</tr>
<tr>
<td>GROUP BY</td>
<td><code>$group</code></td>
</tr>
<tr>
<td>HAVING</td>
<td><code>$match</code> (after <code>$group</code>)</td>
</tr>
<tr>
<td>SELECT</td>
<td><code>$project</code></td>
</tr>
<tr>
<td>ORDER BY</td>
<td><code>$sort</code></td>
</tr>
<tr>
<td>LIMIT</td>
<td><code>$limit</code></td>
</tr>
<tr>
<td>SUM</td>
<td><code>$sum</code></td>
</tr>
<tr>
<td>COUNT</td>
<td><code>$sum</code> (of 1’s)</td>
</tr>
<tr>
<td>OUTER JOIN</td>
<td><code>$lookup</code> (only if unsharded)</td>
</tr>
</tbody>
</table>

MongoDB Availability and Scaling

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

3. High availability
   - Replica sets

   2 shards:
   - 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
Example Collections

**customers:**

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

**orders:**

```json
{ "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}  
] }
```

**products:**

```json
{ "itemno": 460,  
"category": "music",  
"name": "Fender Bender Flight Case",  
"descr": "Sturdy flight case for Fender Bender guitars",  
"manuf": "Fender Bender",  
"listprice": 109.99 }
```

Q: Is this a “good” document DB design for NoSQL...?
Some Possible Relational Designs

There’s a wide spectrum of options – but what’s “best”?

Customer(custid, name, addrid, rating)
Address(addrid, street, city, zipcode)
Order(orderno, custid, order_date, ship_date)
Lineitems(orderno, itemno, qty, price)
Products(itemno, category, name, descrip, manuf, listprice)

Customer1(custid, name)
... 
Customer3(custid, rating)
Address1(addrid, street)
... 
Order1(orderno, custid)
... 
Lineitems1(orderno, itemno, qty)
... 
Products1(itemno, category)
... 
Products5(itemno, listprice)

Universe(custid, name, rating, street, city, zipcode, orderno, order_date, ship_date, itemno, qty, price, category, name, descrip, manuf, listprice)
Relational DB Design Theory

<table>
<thead>
<tr>
<th>Normal Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1NF</td>
<td>All attributes must be atomic (<em>a.k.a.</em> scalar)</td>
</tr>
<tr>
<td>2NF</td>
<td>Eliminates partial dependencies on PK</td>
</tr>
<tr>
<td>3NF</td>
<td>Eliminates transitive dependencies on PK</td>
</tr>
<tr>
<td>BCNF</td>
<td>Eliminates “all” remaining redundancy</td>
</tr>
</tbody>
</table>

- **Bottom line normalization objectives**
  - “The world is flat”
  - “One fact, one place”

- **Other considerations**
  - Query performance, joins, space, ....
ER-Driven Relational DB Design

<table>
<thead>
<tr>
<th>ER Concept</th>
<th>Relational Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>Table with entity’s attributes and PK</td>
</tr>
<tr>
<td>Relationship (M:N)</td>
<td>Table with relationship’s attributes and FKs</td>
</tr>
<tr>
<td>Relationship (1:N)</td>
<td>Merge relationship table with N-side entity table</td>
</tr>
<tr>
<td>Composite attribute</td>
<td>Use flattened column naming convention</td>
</tr>
<tr>
<td>Multivalued attribute</td>
<td>Add separate side table with entity’s PK as FK</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Delta tables or mashup table</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Same bottom line objectives as before
  - “The world is flat”
  - “One fact, one place”
Document DB Design Principles?

• Revisit our previous design objectives
  • “The world is flat” 📈
  • “One fact, one place” 📈

• Central issue: aggregate object design
  • Often follows from the application
  • Unit of read/write/modify operations
  • Unit of ACID behavior
  • Unit of storage (contiguous bytes)
  • Object size
    • Initial expected size
    • Eventual expected size (‼️)
  • ...

M. Carey, Spring 2021: CS122D
Example Collections *(from slide 2)*

**customers:**

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

**orders:**

```json
{ "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}  
  ] 
}, ...
```

**products:**

```json
{ "itemno": 460,  
  "category": "music",  
  "name": "Fender Bender Flight Case",  
  "description": "Sturdy flight case for Fender Bender guitars",  
  "manuf": "Fender Bender",  
  "listprice": 109.99 
},  
{ "itemno": 680,  
  "category": "essentials",  
  "name": "Automatic Beer Opener",  
  "description": "Robotic beer bottle opener",  
  "manuf": "Robo Brew",  
  "listprice": 29.95 
}, ...
```

Q: Is this a “good” document DB design for NoSQL...?
Some Possible Nesting Designs

• Our previous example (from slide 2)

Q: Is this a “good” design?

customer

| address |

order

| items |

| item |

| item |

| ... |

product
Some Possible Nesting Designs (II)

• Or we could nest things more heavily...

Caution: Is this a customer’s whole order history...?!
Some Possible Nesting Designs (II)

• Or even *more* heavily...
Some Possible Nesting Designs (III)

• Or, we could *unnest* things entirely...!
  (JSON equivalent of a normalized relational design)
## ER-Driven Document DB Design

<table>
<thead>
<tr>
<th>ER Concept</th>
<th>Document Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>Collection with entity’s attributes and PK</td>
</tr>
<tr>
<td>Relationship (M:N)</td>
<td>Collection with relationship’s attributes and Fks <em>(linking)</em></td>
</tr>
<tr>
<td>Relationship (1:N)</td>
<td>Merge relationship collection with N-side entity collection <em>(linking)</em></td>
</tr>
<tr>
<td>Composite attribute</td>
<td>Use nested object <em>(embedding)</em></td>
</tr>
<tr>
<td>Weak entity</td>
<td>Use nested object <em>(embedding)</em></td>
</tr>
<tr>
<td>Multivalued attribute</td>
<td>Use nested array <em>(embedding)</em></td>
</tr>
<tr>
<td>Inheritance</td>
<td>Celebrate diversity: Use entity collection with type flag(s)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Bottom line objectives**
  - Natural “entry points” for queries
  - “One fact, one place”

*NOTE:* Might consider using a nested array of FK “links” on each side to “cut out the middleman” *iff* their sizes were small...
Cutting Out the M:N Middleman

- The aforementioned nested array approach
  - Based on arrays of keys
  - An appropriate option \textit{iff} both arrays are small
  - For example....
Object Size (I/O) Considerations

Q: Do most order queries also want the line items?
Query (and Indexing) Considerations

SELECT * FROM customers c, orders o WHERE c.custid = o.custid AND ...
SELECT * FROM customers WHERE ...
SELECT * FROM products WHERE ...
SELECT * FROM orders o, o.items i WHERE ...

CREATE INDEX
ON customers (name)
CREATE INDEX
ON customers (address.city)
CREATE INDEX ON orders (custid)
CREATE INDEX ON orders (items.itemno)
CREATE INDEX ON orders (shipdate)
CREATE INDEX ON products (listprice)
Modeling Inheritance

Note: An object of a particular subtype is free to contain any/all fields that it needs in the JSON world...!

Possible ways to indicate the subtype(s) of an object include:

- **Disjoint subtypes**: a “type” field containing a value of “order”, “phone_order”, or “web_order”

- **Non-disjoint subtypes**: a “types” field containing an array of values, or a set of boolean type indicator fields, such as “is_order”, “is_phone_order”, and “is_web_order”
Anti-Pattern 1: Fields vs. Values

Fields

Fish are friends ...

NOT FOOD!

VALUES

VALUES

...{  
"storeid": 288,
"2020-01-01": 1258.77
}

...  
SELECT * FROM daily_sales
WHERE day >= "2020-01-01"
AND day < "2020-02-01"

...{  
"storeid": 288,
"day": "2020-01-01",
"sales": 1258.77
}

...  
CREATE INDEX ON daily_sales (day)
Anti-Pattern 2: Unbounded Arrays

“Those who nest history are bound to regret it...”
Anti-Pattern 3: Heterogeneous Values

... { "itemno": 347,
  "name": "Beer Cooler Backpack",
  "colors": ["black"],
  "price": 29.95 },
{ "itemno": 375,
  "name": "Stratuscaster Guitar",
  "colors": ["sunburst", "black", "cherry"],
  "price": 1499.99 }, ...

VS.

... { "itemno": 347,
  "name": "Beer Cooler Backpack",
  "colors": ["black"],
  "price": 29.95 },
{ "itemno": 375,
  "name": "Stratuscaster Guitar",
  "colors": ["sunburst", "black", "cherry"],
  "price": 1499.99 }, ...

M. Carey, Spring 2021: CS122D
Document DB Design Principles!

• Rethink the relational DB design objectives
  • “The world is flat” 👎
  • “One fact, one place” 👍

• Central issue: *aggregate object design*
  • Follows from the application (E-R)
  • Unit of read/write/modify ops and ACID
  • Unit of storage (contiguous bytes)
  • Object size (both initial and eventual!)
  • Consider queryability (and indexability)
  • Avoid anti-patterns...!
Questions?