MongoDB

By
Bharath Subramanyam
Relational Databases

- Relational Databases started to become popular in the 80s and have been widely used since then

- Properties of Relational Databases-
  - Fixed Schema
  - High Level Query Language (SQL)
  - ACID properties
  - Primitive Data Partitioning Technology
Problems with Relational Databases

• Was not designed to run on clusters. Difficult to scale horizontally

• Impedance mismatch problem with Relational Databases
Impedance Mismatch
NoSQL

- Johan Oskarsson – twitter #nosql

- Characteristics
  - Non-relational
  - Cluster Friendly
  - Schema Less
  - Mostly Opensource
  - Simple APIs and no joins
Data Model

• Key Value Store (like a hashmap which is persistent)

• Document Models (MongoDB) (Can group things into natural aggregates)

• Column Family (Get the data with the row key and column family name)

• Graph Models
Document Data model

- Database
  - Collections (Tables)
    - Document (Row)
    - Fields (Columns)
    - _id field (Primary Key)
- JSON or XML
- MongoDB uses JSON format
JSON Format

```
{
    "id": 1200,
    "customerName": "Brad",
    "lineItems": [
        {
            "productId": 501, "qty": 5},
        {
            "productId": 553, "qty": 2}
    ]
}
```

Basic Constructs

Base Value = Boolean, int, String...
Object = {}
Array = []
JSON

- JSON object: set of unordered elements
- elements: key/value pairs
- keys must be unique within an object
- values can contain objects
- empty value: null, [] (or simply omit element)
• MongoDB documents in a collection must have unique identifier

• Documents can be referenced using unique identifier
Mapping Relational Data to JSON

Natural mapping:

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Jane</td>
<td>3634</td>
</tr>
<tr>
<td>12</td>
<td>Sarah</td>
<td>6343</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Cust_Id</th>
<th>Date</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>505</td>
<td>10</td>
<td>04-20-15</td>
<td>Apple Watch</td>
</tr>
<tr>
<td>500</td>
<td>10</td>
<td>04-19-15</td>
<td>iPhone6</td>
</tr>
<tr>
<td>100</td>
<td>12</td>
<td>04-01-14</td>
<td>MacBook</td>
</tr>
</tbody>
</table>

```json
Customers
{
  "id": 10,
  "name": "Jane",
  "phone": 3634,
  "orders": [
    {
      "id": 505,
      "date": "04-20-15",
      "product": "Apple Watch"
    },
    {
      "id": 500,
      "date": "04-19-15",
      "product": "iPhone6"
    }
  ]
}

{
  "id": 12,
  "name": "Sarah",
  "phone": 6343,
  "orders": [
    {
      "id": 100,
      "date": "04-01-14",
      "product": "MacBook"
    }
  ]
}
```
Mapping JSON to Relational DB

- **Missing elements:**

  ```json
  { "name" : "Jane",
    "phone" : 3634
  }
  { "name" : "Jim"
  }
  ```

  - no phone

- **Could represent in a table with nulls:**

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>1234</td>
</tr>
<tr>
<td>Jim</td>
<td>-</td>
</tr>
</tbody>
</table>
Mapping JSON to Relational DB

- Repeated elements:

```json
{
   "name" : "Jane",
   "phones" : [3634, 2345]
}
```

- Difficult with tables:

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>3456</td>
</tr>
<tr>
<td></td>
<td>2345</td>
</tr>
</tbody>
</table>

Two phones
Aggregates

• In OOP Orders and Line Items are created as different Classes
• However, Orders and Line Items can be considered as one unit
• In Relational Databases, the values are splattered across different tables
• However, Document databases save this data in terms of a single unit
• It is easier to move back and forth this single unit (You get to store your aggregate at a single instead of it being spread across clusters)
A Problem with the Document Database

• You want to query based on product as the aggregate
• Would have to run a Map Reduce job
• Problematic when you have to slice and dice your data
Replicas

• Why Replication?
  • High Availability of Data and no Downtime
  • Disaster Recovery

• Replica set is a group of two or more nodes

• In a replica set, one node is primary node and remaining nodes are secondary.

• All data replicates from primary to secondary node.

• At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.

• After the recovery of failed node, it again join the replica set and works as a secondary node.
Replicas
Sharding

• Sharding is the process of breaking the data into pieces and storing them across multiple machines.
Sharding

• Shard: This is where the collection data is actually stored. A shard is a replica set.

• Config-Server- Config-servers track state about which servers contain what parts of a sharded collection. Sharded clusters have exactly 3 config servers.

• Query Routers-The query router processes and targets the operations to shards and then returns results to the clients.
Consistency

• Relational Databases – ACID
  (Atomic, Consistent, Isolation, Durable)

• Aggregate Databases- Transaction within an aggregate is ACID.

• Two types of Consistency issues-
  • Logical Consistency
  • Replication Consistency
Logical Consistency

- User → Server → DB → Server → User
  - get
  - Post

v101

Post v102
Replication Consistency

• 2 people booking a hotel room example.
CAP Theorem

• Choose only 2
  • Consistency
  • Availability
  • Partition Tolerance

• There are levels of Availability and Consistency.
Generic MongoDB query

- `db.collection.find({query}, {projection})`

- Eg. `db.posts.find({"author" : "Dan Sullivan"}, {"title" : 1})`
Thank You!