CS143
Non-Relational Database
(MongoDB)

Professor Junghoo “John” Cho

JSON (JavaScript Object Notation)

- Syntax to represent objects in JavaScript
  - [{ “x”: 3, “y”: “Good”}, { “x”: 4, “y”: “Bad” }]
- One of the most popular data-exchange formats over Internet
  - As JavaScript gained popularity, JSON’s popularity grew
  - Simple and easy to learn
  - Others popular formats include XML, CSV, …

Basic JSON Syntax

- Supports basic data types like numbers and strings, as well as arrays and “objects”
- Double quotes for string: “Best”, “UCLA”, “Worst”, “USC”
- Square brackets for array: [1, 2, 3, “four”, 5]
- Objects: (attribute, name) pairs. Use curly braces
  - { “sid”: 301, “name”: “James Dean” }
- Things can be nested

  -
    ```
    { “sid” : 301,
      “name”: { “first”: “James”, “last”: “Dean” },
       “classes”: [ “CS143”, “CS144” ] }
    ```

RDBMS for JavaScript Object Persistence

- JavaScript applications need a “persistence layer” to store and retrieve JavaScript object
- Traditionally (until mid 2010) this was done with RDBMS
  - RDBMS as massive, safe, efficient, multi-user storage engine
- Q: How can we store JavaScript object in RDB?
- “Impedance mismatch”: Two choices
  1. Store object’s JSON as a string in a column
  2. “Normalize” the object into set of relations
- Q: Pros and cons of each approach?
- Q: Can we just create “native database” for JSON?

MongoDB

- Database for JSON objects
  - Perfect as a simple persistence layer for JavaScript objects
  - “NoSQL database”
- Data is stored as a collection of documents
  - Document: (almost) JSON object
  - Collection: group of “similar” documents
- Analogy
  - Document in MongoDB ~ row in RDB
  - Collection in MongoDB ~ table in RDB

MongoDB “Document”

```
{
   "_id": ObjectId(8df38ad8902c),
   "title": "MongoDB",
   "description": "MongoDB is NoSQL database",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": 100,
   "comments": [
      { "user":"lover", "comment": "Perfect!" },
      { "user":"hater", "comment": "Worst!" }
   ]
}
```

- _id field: primary key
  - May be of any type other than array
  - If not provided, automatically added with a unique ObjectId value
- Stored as BSON (Binary representation of JSON)
  - Supports more data types than JSON
  - Does not require double quotes for field names

MongoDB “Philosophy”

- Adopts JavaScript “laissez faire” philosophy
  - Don’t be too strict! Be accommodating! Handle user request in a “reasonable” way
- Schema-less: no predefined schema
  - Give me anything. I will store it anywhere you want
  - One collection will store documents of *any* kind with no complaint
- No need to “plan ahead”
  - A “database” is created when a first collection is created
  - A “collection” is created when a first document is inserted
- Both blessing and curse

MongoDB Demo

```
show dbs;
use demo;
show collections;
db.books.insertOne({title: "MongoDB", likes: 100});
db.books.find();
show collections;
show dbs;
db.books.insertMany([{title: "a"}, {name: "b"}]);
db.books.find();
db.books.find({likes: 100});
db.books.find({likes: {$gt: 10}});
db.books.updateOne({title: "MongoDB"}, {$set: { likes: 200 }});
db.books.find();
db.books.deleteOne({title: "a"});
db.books.drop();
show collections;
show dbs;
```

Basic MongoDB Commands (1)

- mongo: start MongoDB shell
- use <dbName>: use the database
- show dbs: show list of databases
- show collections: show list of collections
- db.colName.drop(): delete `colName` collection
- db.dropDatabase(): delete current database

Basic MongoDB Commands (2)

- CRUD operations
  - insertOne(), insertMany()
  - findOne(), find()
  - updateOne(), updateMany()
  - deleteOne(), deleteMany()
- Insertion: insertX(doc(s))
  - db.books.insertOne({title: "MongoDB", likes: 100})
  - db.books.insertMany([{title: "a"}, {title: "b"}])

Basic MongoDB Commands (3)

- Retrieval: findX(condition)
    - db.books.findOne({likes: 100})
    - db.books.find({$and: [{likes: {$gte: 10}}, {likes: {$lt: 20}}]})
        - Other Boolean/comaprision operators: $or, $not, $gt, $ne, …
- Update: updateX(condition, update_operation)
    - db.books.updateOne({title: "MongoDB"}, {$set: {title: "MongoDB II"}})
    - db.books.updateMany({title: "MongoDB"}, {$inc: {likes: 1}})
        - Other update operators: $mul (multiply), $unset (remove field), …
- Deletion: deleteX(condition)
    - db.books.deleteOne({title: "MongoDB"})
    - db.books.deleteMany({likes: {$lt: 100}})

MongoDB Aggregates

- MongoDB supports complex queries through “aggregates”
- MongoDB aggregates are very much like SQL SELECT queries
  - stages – SQL SELECT clause
  - pipeline – SQL SELECT statement

MongoDB Aggregates: Example

-
  ```
  { _id: 1, cust_id: "a",
    status: "A", amount: 50 }
  { _id: 2, cust_id: "a",
    status: "A", amount: 100 }
  { _id: 3, cust_id: "c",
    status: "D", amount: 25 }
  { _id: 4, cust_id: "d",
    status: "C", amount: 125 }
  { _id: 5, cust_id: "d",
    status: "A", amount: 25 }
  ```

```
db.orders.aggregate([
  { $match: { status: "A" } },
  { $group: {
       _id: "$cust_id",
       total: { $sum: "$amount" },
       count: { $sum: 1 }
     }
  },
  { $sort: { total: -1 } }
])
```

- Equivalent to SQL SELECT
  - Just $match is fine, for example
  - In $group stage, _id is “group by attributes”

Common Aggregate Stages

- $match ≈ WHERE
- $group ≈ GROUP BY
- $sort ≈ ORDER BY
- $limit ≈ FETCH FIRST
- $project ≈ SELECT
- $unwind: replicate document per every element in the array
  - {$unwind: “y” }: {“x”: 1, “y”: [1, 2] } -> {”x”: 1, “y”: 1}, {“x”: 1, “y”: 2 }
- $lookup: “look up and join” another document based on the attribute value
  - {$lookup: { from: <collection to join>, localField: <local join attr>, foreignField: <remote join attr>, as: <output field name> }}
  - Matching documents are returned as an array in <output field name>

More on MongoDB aggregates

- Short tutorial: https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/
- Reference: https://docs.mongodb.com/manual/reference/method/db.collection.aggregate/

MongoDB vs RDB

- MongoDB document
  - Preserves structure
    - Nested objects
  - Potential redundancy
  - Restructuring or combining data is complex and inefficient
- MongoDB: “laissez faire”
  - No explicit db/collection creation
  - No schema. Anything is fine

- RDB tuple
  - “Flattens” data
    - Set of flat rows
  - Removes redundancy
  - Data can be easily “combined” using relational operators
- RDB: “Straight-jacket”
  - Declare everything before use
  - Reject if not compliant

More on MongoDB

- We learned just the basic
- MongoDB has many more features
  - Transactions
  - Replication
  - (Auto)sharding
  - …
- Read MongoDB documentation and online tutorials to learn more