g able

MongoDB 3.0

Read / Write Operations

Topics

- JSON
- Write Operation (Insert, Update, Delete)
- Read Operation (Query)
- Labs

MongoDB CRUD Introduction

- MongoDB stores data in the form of documents, which are JSON-like field and value pairs.
- Documents are analogous to structures in programming languages that associate keys with values (e.g. dictionaries, hashes, maps, and associative arrays).
- Formally, MongoDB documents are BSON documents. BSON is a binary representation of JSON with additional type information.
- In the documents, the value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents.

JSON Document

```
{
  name: "sue",                    ← field: value
  age: 26,                        ← field: value
  status: "A",                    ← field: value
  groups: [ "news", "sports" ]    ← field: value
}
```

JavaScript Object Notation (JSON)

- Example

```
{
    “firstname” : “Barack”,
    “lastname” : “Obama”,
    “born” : 1961
}
```

- JSON vs. XML

```
<person>
    <firstname>Barack</firstname>
    <lastname>Obama</lastname>
    <born>1961</born>
</person>
```

JSON Examples

- Name & Value

  { “name” : “value” }

- Array

  { “Regions” : [“North”, “South”, “Middle”, “Northeastern”] }

- Anonymous Object

  { “FirstName” : “Steve”, “LastName” : “Jobs” }

- Object

  {“Company” : {“Name” : “G-ABLE”, “Email” : “mail@g-able.com”}}

- Array of Objects

```
{ “Contacts” : [
  {“name” : “Steve Jobs”, “Email” : “stevejobs@apple.com”},
  {“name” : “Mark Zuckerbergs”, “Email” : “zuckerbergs@facebook.com”} ]
}
```

JSON vs. Class

```
{ “Customer” :
    { “FirstName” : “Steve”,
      “YearOfBirth”: 1950
    }
}
```

```
class Customer {
  string FirstName;
  int YearOfBirth;
}
```

```
{ “Regions” : [
    { “Name” : “Northern” },
    { “Name” : “Southern” } ]
}
```

```
class Region {
  string [] Name;
}
```

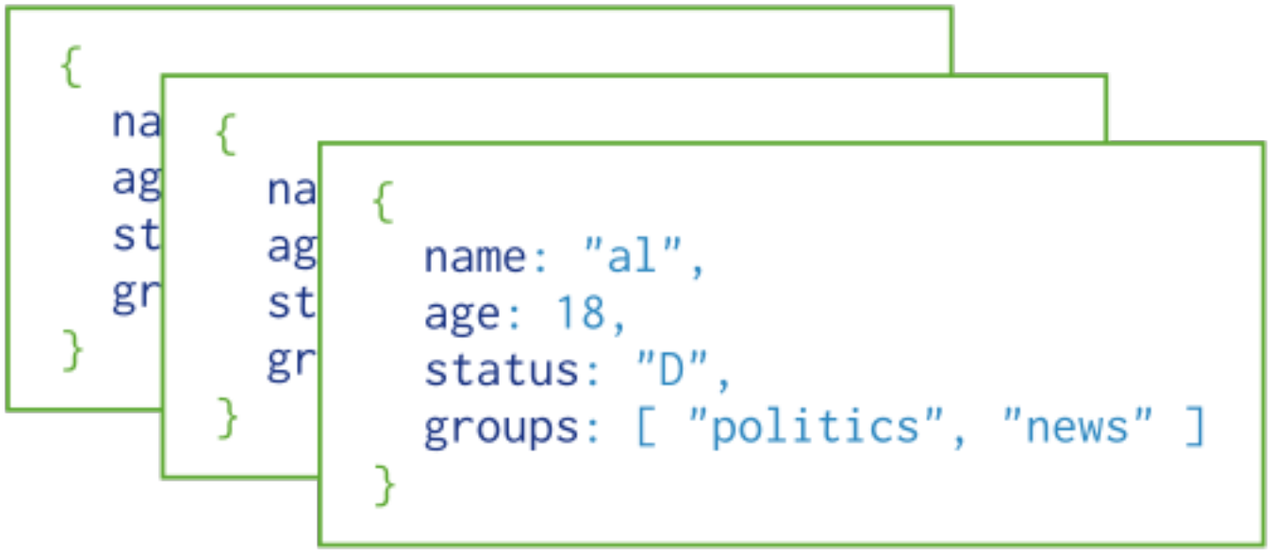
```
{ “Customers” :
  { “Contact” :
    { “Name” : “Steve Jobs”,
      “Company” : “Apple” }
    },
    “Address” :
    {  “AddressDetail” : “”,
       “ZipCode” : “12345”}
    }
  }
}
```

```
class Customers {
  Contact contact;
  Address address;
}
class Contact {
  string Name;
  string Company;
}
class Address {
  string AddressDetail;
  string ZipCode;
}
```

C# :- http://json2csharp.com
Java :- http://www.jsonschema2pojo.org

Collections

- MongoDB stores all documents in collections. A collection is a group of related documents that have a set of shared common indexes. Collections are analogous to a table in relational databases.



Collection

Write Operations

- A write operation is any operation that creates or modifies data in the MongoDB instance.
- In MongoDB, write operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.
- There are three classes of write operations in MongoDB: insert, update, and remove.
- For the update and remove operations, you can specify criteria, or conditions, that identify the documents to update or remove. These operations use the same query syntax to specify the criteria as read operations.

Insert

- In MongoDB, the db.collection.insert() method adds new documents to a collection.

- The following diagram highlights the components of a MongoDB insert operation.

Insert Behaviour

- If you add a new document without the _id field, the client library or the **mongod** instance adds an _id field and populates the field with a unique ObjectId.
- If you specify the _id field, the value must be unique within the collection.
- For operations with write concern, if you try to create a document with a duplicate _id value, **mongod** returns a duplicate key exception.
- Example:-

```
db.users.insert (
    { _id: "001234",
      name: "Steve, Jobs",
      company: "Apple Inc."
    }
)
```

Read Operations

- Read operations, or queries, retrieve data stored in the database.
- In MongoDB a query targets a specific collection of documents.
- Queries specify criteria, or conditions, that identify the documents that MongoDB returns to the clients.
- A query may include a projection that specifies the fields from the matching documents to return. You can optionally modify queries to impose limits, skips, and sort orders.

Query Interfaces

- For query operations, MongoDB provides a db.collection.find() method. The method accepts both the query criteria and projections and returns a cursor to the matching documents. You can optionally modify the query to impose limits, skips, and sort orders.

- The following diagram highlights the components of a MongoDB query operation:

```
db.users.find(                          ← collection
   { age: { $gt: 18 } },                ← query criteria
   { name: 1, address: 1 }              ← projection
).limit(5)                              ← cursor modifier
```

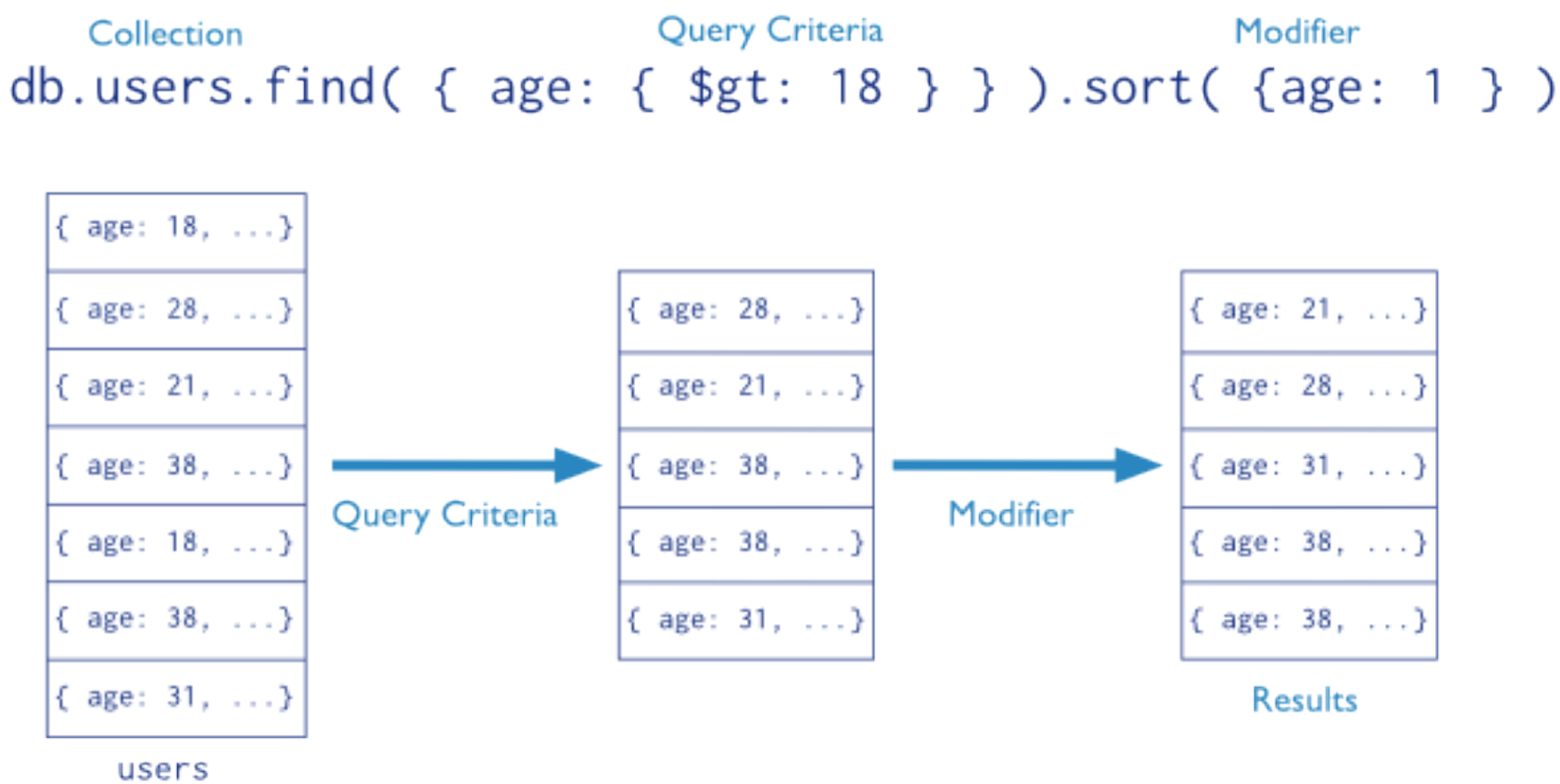
Query Behaviour

- All queries in MongoDB address a single collection.
- You can modify the query to impose limits, skips, and sort orders.
- The order of documents returned by a query is not defined unless you specify a sort().
- Operations that modify existing documents (i.e. updates) use the same query syntax as queries to select documents to update.
- In aggregation pipeline, the $match pipeline stage provides access to MongoDB queries.
- MongoDB provides a db.collection.findOne() method as a special case of find() that returns a single document.

Tips: .findOne() method will print JSON in friendly format.

Query Statements

- Consider the following diagram of the query process that specifies a query criteria and a sort modifier:



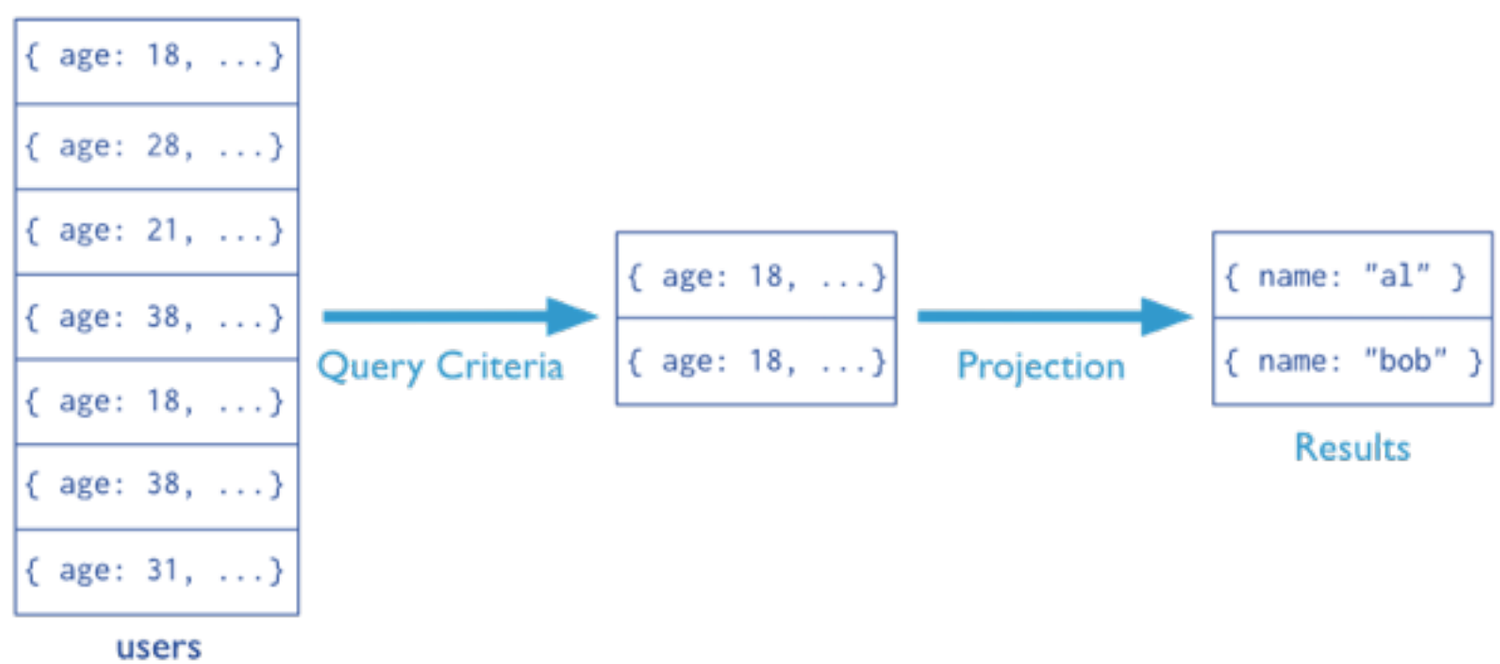
- In the diagram, the query selects documents from the users collection. Using a query selection operator to define the conditions for matching documents, the query selects documents that have age greater than (i.e. $gt) 18. Then the sort() modifier sorts the results by age in ascending order.

Projections

- Queries in MongoDB return all fields in all matching documents by default. To limit the amount of data that MongoDB sends to applications, include a projection in the queries.
- By projecting results with a subset of fields, applications reduce their network overhead and processing requirements.
- Projections, which are the second argument to the find() method, may either specify a list of fields to return or list fields to exclude in the result documents.

Collection Query Criteria Projection

```
db.users.find( { age: 18 }, { name: 1, _id: 0 } )
```



Modify Documents

- MongoDB provides the update() method to update the documents of a collection. The method accepts as its parameters:
  - an update conditions document to match the documents to update,
  - an update operations document to specify the modification to perform, and an options document.
- To specify the update condition, use the same structure and syntax as the query conditions.

Modify Documents

- Use update operators to change field values.

```
db.collection.update(
    {key: “value”},
    {
        $set: {
          field: “value”,
          ...
       }
    }
)
```

Query block

Update block

Note: If the updating fields doesn’t exist in the documents, it will be added.

- Update an embedded field.

```
db.test.update (
    {key: “value”},
    {
        $set: {
            “outerField.innerField”:”value”
        }
    }
)
```

Modify Documents

- By default, the update() method updates a single document.
- To update multiple documents, use the `multi` option in the update() method.

```
db.test.update (
    {key: “value”},
    {
        $set: {
            “field”:”value”
        }
    },
    {multi: true}
)
```

Modify Documents

- By default, if no document matches the update query, the update() method does nothing.
- However, by specifying `upsert: true`, the update() method either updates matching document or documents, or inserts a new document using the update specification if no matching document exists.

```
db.test.update (
    {key: “value”},
    {
        $set: {
            “outerField.innerField”:”value”
        }
    },
    {multi: true, upsert: true}
)
```

- Another way to modify documents is .findAndModify() method and .save() method. For more details, see..

  http://docs.mongodb.org/manual/reference/method/db.collection.findAndModify/#db.collection.findAndModify and http://docs.mongodb.org/manual/reference/method/db.collection.save/#db.collection.save

Remove Documents

- In MongoDB, the `db.collection.remove()` method removes documents from a collection.
- You can remove all documents from a collection, remove all documents that match a condition, or limit the operation to remove just a single document.
- Remove All Documents

  ```
  db.collection.remove({})
  ```

- Remove Documents that Match a Condition.

  ```
  db.collection.remove({field:”value”})
  ```

- Remove a Single Document that Matches a Condition

  ```
  db.collection.remove({field:”value”}, 1)
  ```

Lab1: Insert Documents

1. Open Terminal.
2. Run MongoDB shell by type…

```
mongo
```

3. Insert item data into inventory by type…

```
use store

db.inventory.insert(
   {
     item: "ABC1",
     details: {
        model: "14Q3",
        manufacturer: "XYZ Company"
     },
     stock: [ { size: "S", qty: 25 }, { size: "M", qty: 50 } ],
     category: "clothing"
   }
)
```

4. Verify the insertion by querying the collection.

```
db.inventory.find()
```

Lab1: Create an array of documents

5. Define a variable `mydocuments` that holds an array of documents to insert.

```
var mydocuments =
    [
      {
        item: "ABC2",
        details: { model: "14Q3", manufacturer: "M1 Corporation" },
        stock: [ { size: "M", qty: 50 } ],
        category: "clothing"
      },
      {
        item: "MNO2",
        details: { model: "14Q3", manufacturer: "ABC Company" },
        stock: [ { size: "S", qty: 5 }, { size: "M", qty: 5 }, { size: "L", qty: 1 } ],
        category: "clothing"
      },
      {
        item: "IJK2",
        details: { model: "14Q2", manufacturer: "M5 Corporation" },
        stock: [ { size: "S", qty: 5 }, { size: "L", qty: 1 } ],
        category: "houseware"
      }
    ];
```

6. Pass the `mydocuments` array to the db.collection.insert() to perform a bulk insert.

```
db.inventory.insert( mydocuments );
```

Lab1: Insert Multiple Documents with Bulk

7. Initialize a Bulk operations builder for the collection inventory.

```
var bulk = db.inventory.initializeUnorderedBulkOp();
```

8. Add two insert operations to the bulk object using the Bulk.insert() method.

```
bulk.insert(
   {
     item: "FD119",
     details: { model: "1203", manufacturer: "XYZ Company" },
     stock: [ { size: "L", qty: 15 } ],
     category: "food"
   }
);
bulk.insert(
   {
     item: "ZYT1",
     details: { model: "14Q1", manufacturer: "ABC Company"  },
     stock: [ { size: "S", qty: 5 }, { size: "M", qty: 5 } ],
     category: "houseware"
   }
);
```

9. Call the execute() method on the bulk object to execute the operations in its list.

```
bulk.execute();
```

Lab2: Query Documents

1. An empty query document ({}) selects all documents in the collection.

   ```
   db.inventory.find()
   ```

2. To specify equality condition, use the query document { <field>: <value> } to select all documents that contain the <field> with the specified <value>.

   ```
   db.inventory.find( {category: "clothing"} )
   ```

3. A query document can use the query operators to specify conditions in a MongoDB query.

   ```
   db.inventory.find( {category: { $in: [ 'houseware', 'food'] } } )
   ```

Lab2: Query Documents

4. Equality Match on Fields within an Embedded Document.

```
db.inventory.find(
  { 'stock.qty': { $lt: 10 } }
)
```

5. Specify AND Conditions

```
db.inventory.find(
  { category: "clothing",
    'stock.qty': { $lt: 10 }
  }
)
```

6. Specify AND Conditions

```
db.inventory.find(
  {
    $or: [
        {category: "clothing"},
        {'stock.qty': { $gt: 10 } }
    ]
  }
)
```

Lab2: Query Documents

7. Specify AND as well as OR Conditions

```
db.inventory.find(
  {
    category: "clothing",
    $or: [
        {'size': "L"},
        {'stock.qty': { $gt: 10 } }
    ]
  }
)
```

Lab3: Projection

1. Return all documents with specific fields and _id field only.

   ```
   db.inventory.find( {}, { item:1 } )
   ```

2. Return two fields and _id field.

   ```
   db.inventory.find( {}, { item:1, category:1 } )
   ```

3. Return all documents with specific fields, exclude _id.

   ```
   db.inventory.find( {}, { item:1, _id:0 } )
   ```

   Note: You cannot combine inclusion and exclusion semantics in a single projection with the exception of the _id field.

4. Return documents with category value is “clothing” and exclude only “stock” and “category”.

   ```
   db.inventory.find(
     {category:"clothing"},
     {stock:0, category:0}
   )
   ```

Lab4: Modify Documents

1. Update document typed “food” to “foods & drinks”

```
db.inventory.update(
    {category:"food"},
    {
        $set:{
            category:"foods & drinks"}
    },
    {multi:true}
)
```

2. Use `upsert` to insert or update documents.

```
db.inventory.update(
  {item:"ABC3"},
  {$set: {
    "details" : {
        "model" : "15Q1",
        "manufacturer" : "QQ Company"},
    "stock" : [
        {"size" : "S", "qty" : 30 },
        { "size" : "M", "qty" : 30 },
        { "size" : "L", "qty" : 30 },
        { "size" : "XL", "qty" : 30 } ],
    "category" : "clothing"
  }},
  {upsert: true}
)
```

Lab4: Modify Documents

3. Update embedded documents

```
db.inventory.update(
    {"stock.size":"S"},
    {$set:{
        "stock":[
            {"size":"S",  “qty”:0},
            {"size":"M",  “qty":0},
            {"size":"L",  “qty":0},
            {"size":"XL", "qty":0}
        ]
    }},
    {multi:true}
)
```

Note: this command will update all documents that contains stock size “S” and replace “stock” documents with values in $set block.

References

- MongoDB CRUD Introduction
  http://docs.mongodb.org/manual/core/crud-introduction/

- Read Operations Overview
  http://docs.mongodb.org/manual/core/read-operations-introduction/