NOSQL - MongoDB

Mirunalini.P

SSNCE

April 10, 2020

Table of Contents

Session Outcome

At the end of this session, participants will be able to

- Understand MongoDB Basics
- Understand MongoDB Data Model
- Understand MongoDB distributed system Characteristics

MongoDB

- MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability.
- MongoDB works on concept of collection and document
- **Database**:Database is a physical container for collections. A single MongoDB server typically has multiple databases.
- **Collection:**
  - Group of MongoDB documents is called as collection, which is equivalent to an RDBMS table.
  - A collection exists within a single database.
  - Collections do not enforce a schema.
  - Documents within a collection can have different fields.
  - All documents in a collection are of similar or related purpose.

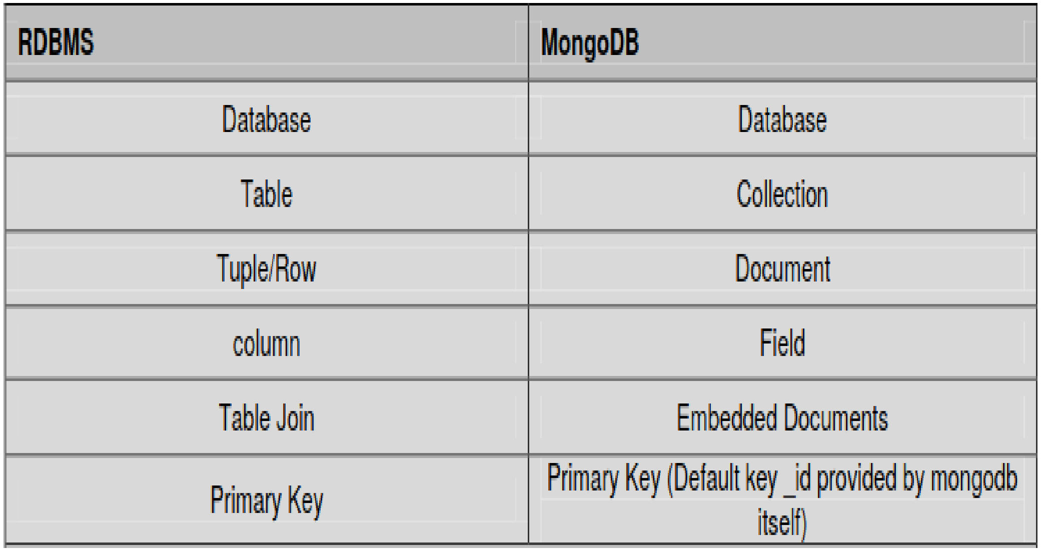
MongoDB

Document:

- A document is a set of key-value pairs.
- Documents have dynamic schema.
- The documents in the same collection do not need to have the same set of fields or structure
- Common fields in a collection's documents may hold different types of data.

Relationship of RDBMS terminology with MongoDB

Table shows the relationship of RDBMS terminology with MongoDB



RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)

Figure 1: RDBS vs MongoDB

The Basics

- A MongoDB instance may have zero or more **databases**
- A database may have zero or more **collections**.
- A collection may have zero or more **documents**.
- Docs in the same collection dont even need to have the same **fields**
- Docs can embed other documents
- Documents are addressed in the database via a unique key
- A document may have one or more fields.

Example: Mongo Document

```
var mydoc = {
                _id: ObjectId("5099803df3f4948bd2f98391")
                  name: { first: "Alan", last: "Turing" }
                  birth: new Date('Jun 23, 1912'),
                  death: new Date('Jun 07, 1954'),
                  contribs: [ "Turing machine",
                  "Turing test", "Turingery" ],
                  views : NumberLong(1250000)
              }
```

- _id holds an ObjectId.
- name holds an embedded document that contains the fields first and last.
- birth and death hold values of the Date type.
- contribs holds an array of strings.
- views holds a value of the NumberLong type.

Example: Mongo Collection

```
{
"_id": ObjectId("4efa8d2b7d284dad101e4bc9"),
   "Last Name": "DUMONT",
   "First Name": "Jean",
   "Date of Birth": "01-22-1963"
},
{
   "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
   "Last Name": "PELLERIN",
    "First Name": "Franck",
    "Date of Birth": "09-19-1983",
    "Address": "1 chemindes Loges",
     "City": "VERSAILLES"
}
```

MongoDB Data Model

- Documents stored in binary JSON (BSON) format
- Individual documents stored in a collection
- The operation **createCollection** used to create each collection.
- Capping parameter used to choose the storage options for each collection

To create project collection to hold project objects from COMPANY database

```
db.createCollection( project , { capped : true,
size : 1310720, max : 500 } )
```

Object_id

- Each document in collection has unique ObjectID field called _id automatically indexed in the collection
- The objectId can be specified by the user or it can be system generated.
- System-generated ObjectIds have specific format which is 16-byte id value.
    - Timestamp (4 bytes)
    - Node id (3 bytes)
    - Processid (2 bytes)
    - Counter (3 bytes)

MongoDB Data Model

- A collection does not have a schema
- Structure of the data fields in documents choosen based on how documents will be accessed
- User can choose normalized or denormalized design

Project document with an array of embedded workers:

The workers information is embedded in the project document; so there is no need for the worker collection. This is known as the denormalized pattern.

```
{ _id                    :  ''P1'',
Pname        :  ''ProductX'',
Plocation:  ''Bellaire'',
Workers:  [
                {
                        Ename:  ''John Smith'',
                        Hours:  32.5
                },
                {
                         Ename:  ''Joyce English'',
                         Hours:  20.0
                }
             ]
```

Project document with an embedded array of worker ids:

```
{
_id:      P1    ,
Pname:    ProductX   ,
Plocation:    Bellaire    ,
WorkerIds:   [    W1    ,    W2    ]
}
{
_id:      W1    ,
Ename:    John Smith   ,
Hours:  32.5
}

{ _id:     W2    ,
Ename:    Joyce English   ,
Hours:  20.0
```

Normalized project and worker documents (not a fully normalized design for M:N relationships)

```
{
_id: ‘‘P1’’,
Pname: ‘‘ProductX’’,
Plocation: ‘‘Bellaire’’
}
{
_id: ‘‘W1’’,
Ename: ‘‘John Smith’’,
ProjectId: ‘‘P1’’,
Hours:32.5
}
{
_id: ‘‘W2’’,
Ename: ‘‘Joyce English’’,
ProjectId: ‘‘P1’’,
```

MongoDB CRUD Operations

Documents can be created and inserted into their collections using the insert operation

```
db.<collection_name>.insert(<document(s)>)
```

```
db.mycol.insert({
_id: ObjectId(7df78ad8902c),
title: 'MongoDB Overview',
description: 'MongoDB is no sql database',
by: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
})
```

mycol is collection name, if the collection doesn't exist in the database MongoDB will create thie collection and then insert document in it.
To insert multiple documents in single query, you can pass an array of documents in insert() command.

MongoDB Delete Document

The remove() Method:Used to remove document from the collection.

```
db.<collection_name>.remove(<condition>)
Consider the mycol collection has following data:
{ "_id" : ObjectId(5983548781331adf45ec5),
 "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6),
 "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7),
"title":" Tutorials Point Overview"}

db.mycol.remove({'title':'MongoDB Overview'})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec6),
 "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7),
 "title":" Tutorials Point Overview"}
```

MongoDB Distributed Systems Characteristics

- **Two-phase commit method:**Used to ensure atomicity and consistency of multidocument transactions
- The concept of **replica set** is used to create multiple copies on different nodes.
- It uses a variation of the **master-slave** approach
- **Primary copy, Secondary copy, and Arbiter**
- Arbiter does not hold a replica but participates in elections to choose a primary if primary fails.
- Write operations must be applied to the primary and then propogated to the secondaries.
- For read operations, default read preference are primary node.
- Secondary copies are mainly to make sure the system continues operation if the primary fails.

Sharding in MongoDB

- Horizontal partitioning divides the documents into disjoint partitions (shards)
- Allows adding more nodes as needed
- Shards stored on different nodes to achieve load balancing
- shards posses different characteristics
    - Partitioning field (shard key) must exist in every document in the collection
    - It must have an index
- The values of the shardkey are divided into chunks either through **range partitioning or hash partitioning.**

Sharding in MongoDB

- **Range partitioning** creates the chunks by specifying a range of key values and works best with range queries.
- **Hash Partitioning** Partitioning based on the hash values of each shard key

Reference

Fundamentals of Database systems 7^{th} Edition by Ramez Elmasri.