Lecture #1

Content

1: **Introduction**

2: *Create, Read, Update, Delete (*CRUD)

3: Schema Design

What is Mongo DB

- MongoDB is a cross-platform, document oriented database that provides

- High performance.
- High availability.
- Easy scalability.

- MongoDB works on concept of collection and document.

Why Mongo DB?

- All the modern applications deals with huge data.
- Development with ease is possible with mongo DB.
- Flexibility in deployment.
- Rich Queries.
- Older database systems may not be compatible with the design.

And it’s a document oriented storage:- Data is stored in the form of JSON Style.

History

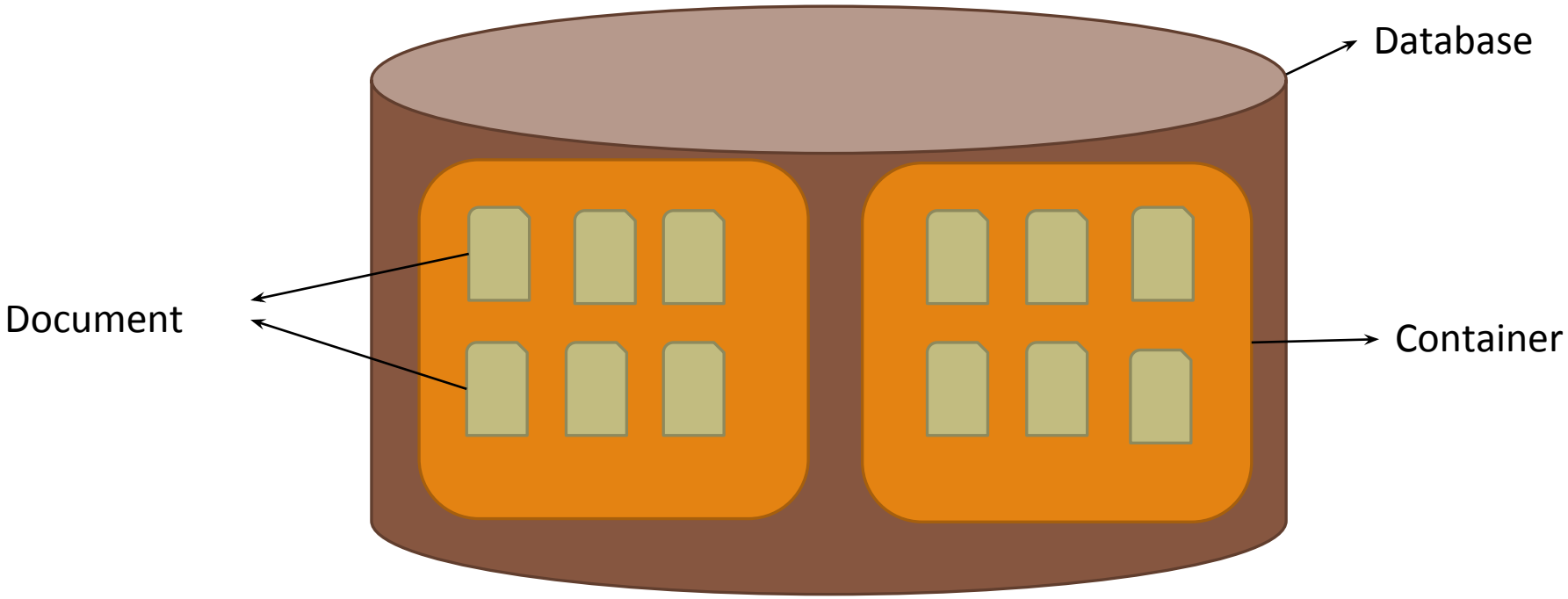
mongoDB = “Hu**mongo**us DB”

- Open-source
- Document-based
- “High performance, high availability”
- Automatic scaling
- C-P on CAP

-blog.mongodb.org/post/475279604/on-distributed-consistency-part-1
-mongodb.org/manual

Mongo DB architecture

Architecture : -



Document(JSON) structure

- The document has simple structure and very easy to understand the content
- JSON is smaller, faster and lightweight compared to XML.
- For data delivery between servers and browsers, JSON is a better choice
- Easy in parsing, processing, validating in all languages
- JSON can be mapped more easily into object oriented system.

```
[
{   "Name": "Tom",
   "Age": 30,
   "Role": "Student",
   "University": "CU",
  }
 {
   "Name": “Sam",
   "Age": 32,
   "Role": "Student",
   "University": “OU",
 }]
```

Company Using mongoDB

-Steve Francia, http://www.slideshare.net/spf13/mongodb-9794741?v=qf1&b=&from_search=13

Data Model

- Document-Based (max 16 MB)
- Documents are in BSON format, consisting of field-value pairs
- Each document stored in a collection
- Collections
  - Have index set in common
  - Like tables of relational db's.
  - Documents do not have to have uniform structure

Ref: -docs.mongodb.org/manual/

JSON

- “JavaScript Object Notation”
- Easy for humans to write/read, easy for computers to parse/generate
- Objects can be nested
- Built on
  - name/value pairs
  - Ordered list of values

Ref: http://json.org/

BSON

- “Binary JSON”
- Binary-encoded serialization of JSON-like docs
- Also allows “referencing”
- Embedded structure reduces need for joins
- Goals
  - Lightweight
  - Traversable
  - Efficient (decoding and encoding)

Ref: http://bsonspec.org/

BSON Example

```
{
"_id" : "37010"
"city" :      "ADAMS",
"pop" :      2660,
"state" :    "TN",
“councilman” : {
                name: “John Smith”
                address: “13 Scenic Way”
              }
}
```

The _id Field

- By default, each document contains an _id field. This field has a number of special characteristics:
  - Value serves as primary key for collection.
  - Value is unique, immutable, and may be any non-array type.
  - Default data type is ObjectId, which is “small, likely unique, fast to generate, and ordered.” Sorting on an ObjectId value is roughly equivalent to sorting on creation time.

Ref: http://docs.mongodb.org/manual/reference/bson-types/

mongoDB vs. SQL

mongoDB	SQL
Document	Tuple
Collection	Table/View
PK: _id Field	PK: Any Attribute(s)
Uniformity not Required	Uniform Relation Schema
Index	Index
Embedded Structure	Joins
Shard	Partition

CRUD

Create, Read, Update, Delete

Getting Started with mongoDB

- To install mongoDB, go to this link and click on the appropriate OS and architecture: http://www.mongodb.org/downloads

- First, extract the files (preferrably to the C drive).

- Finally, create a data directory on C:\ for mongoDB to use
  - i.e. “md data” followed by “md data\db”

  Ref: http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/

Getting Started with mongoDB

- Open your mongodb/bin directory and run mongod.exe to start the database server.

- To establish a connection to the server, open another command prompt window and go to the same directory, entering in mongo.exe. This engages the mongodb shell—it's that easy!

Ref: http://docs.mongodb.org/manual/tutorial/getting-started/

CRUD: Using the Shell

- To check which db you're using — db
- Show all databases — show dbs
- Switch db's/make a new one — use <name>
- See what collections exist — show collections

Note: db's are not actually created until you insert data!

CRUD: Using the Shell (cont.)

To insert documents into a collection/make a new collection:

db.<collection>.insert(<document>)

<=>

INSERT INTO <table>

VALUES(<attributevalues>);

CRUD: Inserting Data

Insert one document

```
db.<collection>.insert({<field>:<value>})
```

Inserting a document with a field name new to the collection is inherently supported by the BSON model.

To insert multiple documents, use an array.

Ref: http://docs.mongodb.org/

CRUD: Querying

- Done on collections.
- Get all docs: `db.<collection>.find()`
  - Returns a cursor, which is iterated over shell to display first 20 results.

```
Add .limit(<number>) to limit results
SELECT * FROM <table>;
```

- Get one doc: `db.<collection>.findOne()`

Ref: http://docs.mongodb.org/

CRUD: Querying

To match a specific value:

```
db.<collection>.find({<field>:<value>})
```

“AND”

```
db.<collection>.find({<field1>:<value1>,
                      <field2>:<value2>
                      })
```

```
SELECT *
FROM <table>
WHERE <field1> = <value1> AND <field2> = <value2>;
```

Ref: http://docs.mongodb.org/

CRUD: Querying

```
OR
db.<collection>.find({ $or: [
<field>:<value1>
<field>:<value2>        ]
})
```

```
SELECT *
FROM <table>
WHERE <field> = <value1> OR <field> = <value2>;
```

Checking for multiple values of same field

```
db.<collection>.find({<field>: {$in [<value>, <value>]}})
```

Ref: http://docs.mongodb.org/

CRUD: Querying

Including/excluding document fields

```
db.<collection>.find({<field1>:<value>}, {<field2>: 0})
```

```
SELECT field1
FROM <table>;
```

```
db.<collection>.find({<field>:<value>}, {<field2>: 1})
```

Find documents with or w/o field

```
db.<collection>.find({<field>: { $exists: true}})
```

Ref: http://docs.mongodb.org/

CRUD: Updating

```
db.<collection>.update(
{<field1>:<value1>},      //all docs in which field = value
{$set: {<field2>:<value2>}},          //set field to value
{multi:true} )          //update multiple docs
```

upsert: if true, creates a new doc when none matches search criteria.

```
UPDATE <table>
SET <field2> = <value2>
WHERE <field1> = <value1>;
```

CRUD: Updating

To remove a field

```
db.<collection>.update({<field>:<value>},
                       { $unset: { <field>: 1}})
```

Replace all field-value pairs

```
db.<collection>.update({<field>:<value>},
                       { <field>:<value>,                    <field>:<value>})
```

*NOTE: This overwrites ALL the contents of a document, even removing fields.

Ref: http://docs.mongodb.org/

CRUD: Removal

Remove all records where field = value

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

```
DELETE FROM <table>
WHERE <field> = <value>;
```

As above, but only remove first document

```
db.<collection>.remove({<field>:<value>}, true)
```

Ref: http://docs.mongodb.org/

CRUD: Isolation

- By default, all writes are atomic **only** on the level of a single document.
- This means that, by default, all writes can be interleaved with other operations.
- You can isolate writes on an **unsharded** collection by adding $isolated:1 in the query area:

```
db.<collection>.remove({<field>:<value>,
                        $isolated: 1})
```

Ref: http://docs.mongodb.org/

MongoDB Videos

https://www.mongodb.com/presentations/introduction-to-mongodb-stitch-drew-dipalma

https://www.slideshare.net/RockeTier/introduction-to-mongo-db-34972479

MongoDB Videos

https://www.mongodb.com/presentations/mongodb-world18--eliot-horowitz-keynote

Schema Design

RDBMS v/s MongoDB

RDBMS		MongoDB
Database	➜	Database
Table	➜	Collection
Row	➜	Document
Index	➜	Index
Join	➜	Embedded Document
Foreign Key	➜	Reference