DEPARTMENT OF CSE-H

TOPIC

INTRODUCTION MONGODB

WHAT IS MONGODB?

- Developed by 10gen
  - Founded in 2007
- A document-oriented, NoSQL database
  - Hash-based, *schema-less database*
    - No Data Definition Language
    - In practice, this means you can store hashes with any keys and values that you choose
      - Keys are a basic data type but in reality stored as strings
      - Document Identifiers (_id) will be created for each document, field name reserved by system
    - Application tracks the schema and mapping
    - Uses BSON format
      - Based on JSON – B stands for Binary
- Written in C++
- Supports APIs (drivers) in many computer languages
  - JavaScript, Python, Ruby, Perl, Java, Java Scala, C#, C++, Haskell, Erlang

FUNCTIONALITY OF MONGODB

- Dynamic schema
  - No DDL
- Document-based database
- Secondary indexes
- Query language via an API
- Atomic writes and fully-consistent reads
  - If system configured that way
- Master-slave replication with automated failover (replica sets)
- Built-in horizontal scaling via automated range-based partitioning of data (sharding)
- No joins nor transactions

WHY USE MONGODB?

- Simple queries
- Functionality provided applicable to most web applications
- Easy and fast integration of data
  - No ERD diagram
- Not well suited for heavy and complex transactions systems

RDB CONCEPTSTO NO SQL

RDBMS		MongoDB
Database		Database
Table, View	⇨ ⇨	Collection
Row	⇨	Document (BSON)
Column		Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨ ⇨	Reference
Partition	⇨	Shard

Collection is not strict about what it Stores

Schema-less

Hierarchy is evident in the design

Embedded Document

MONGODB PROCESSES AND CONFIGURATION

- Mongod – Database instance
- Mongos - Sharding processes
  - Analogous to a database router.
  - Processes all requests
  - Decides how many and which *mongod*s should receive the query
  - *Mongos* collates the results, and sends it back to the client.
- Mongo – an interactive shell (a client)
  - Fully functional JavaScript environment for use with a MongoDB
- You can have one *mongos* for the whole system no matter how many mongods you have
- OR you can have one local *mongos* for every client if you wanted to minimize network latency.

DATABASES AND COLLECTIONS

MongoDB stores data records as documents (specifically BSON documents) which are gathered together in collections. A database stores one or more collections of documents.

Create a Database

If a database does not exist, MongoDB creates the database when you first store data for that database.

Syntax

Basic syntax of use DATABASE statement is as follows −

use DATABASE_NAME

The dropDatabase() Method

MongoDB db.dropDatabase() command is used to drop a existing database.

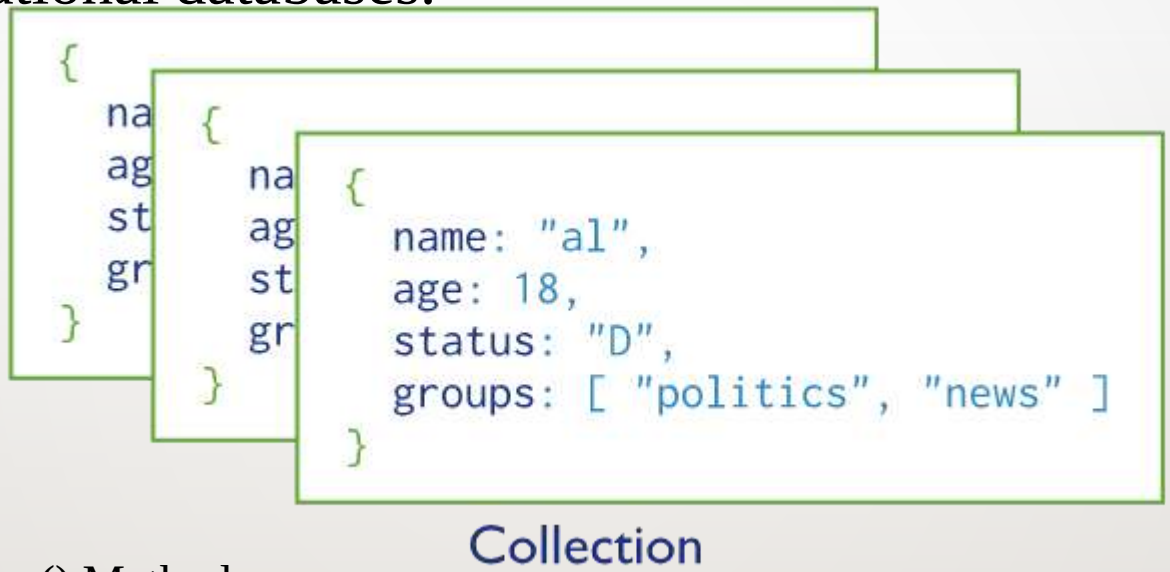
Syntax

Basic syntax of dropDatabase() command is as follows −

db.dropDatabase()

Collections

MongoDB stores documents in collections. Collections are analogous to tables in relational databases.



The createCollection() Method

MongoDB **db.createCollection(name)** is used to create collection.

Syntax

Basic syntax of **createCollection()** command is as follows −

db.createCollection(name)

MongoDB supports many datatypes.

- String: A basic and commonly used data type
- Integer: Used to store numeric values that don't require decimal precision
- Double: Stores numeric numbers with 8 bytes (64-bit IEEE 754 floating point)
- Boolean: Stores true or false values
- Array: Stores arrays
- Object: Stores embedded documents as key-value pairs within other documents
- Date: Stores the current date or time
- ObjectId: A 12-byte identifier that uniquely identifies documents within a collection

BSON FORMAT

- Binary-encoded serialization of JSON-like documents
- Zero or more key/value pairs are stored as a single entity
- Each entry consists of a field name, a data type, and a value
- Large elements in a BSON document are prefixed with a
- length field to facilitate scanning

SCHEMA FREE

- MongoDB does not need any pre-defined data schema
- Every document in a collection could have different data
  - Addresses NULL data fields

```
{name: “will”,
  eyes: “blue”,
  birthplace: “NY”,
  aliases: [“bill”, “la ciacco”],
  loc: [32.7, 63.4],
  boss: ”ben”}
```

```
{name: “jeff”,
  eyes: “blue”,
  loc: [40.7, 73.4],
  boss: “ben”}
```

```
{name: “brendan”,
  aliases [“el diablo”]}
```

```
{name: “ben”,
  hat: ”yes”}
```

```
{name: “matt”,
  pizza: “DiGiorno”,
  height: 72,
  loc: [44.6, 71.3]}
```

mongoDB

JSON FORMAT

- Data is in name / value pairs
- A name/value pair consists of a field name followed by a colon, followed by a value:
  - Example: “name”: “R2-D2”
- Data is separated by commas
  - Example: “name”: “R2-D2”, race : “Droid”
- Curly braces hold objects
  - Example: {“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”}
- An array is stored in brackets []
  - Example [{“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”},
  - {“name”: “Yoda”, affiliation: “rebels”}]

MONGODB FEATURES

- Document-Oriented storage
- Full Index Support
- Replication & High Availability
- Auto-Sharding
- Querying
- Fast In-Place Updates
- Map/Reduce functionality

Agile

Scalable

CRUD OPERATIONS

- Create
  - db.collection.insert(<document>)
  - db.collection.insertOne(<document>)
  - db.collection.insertMany([<document>, [<document>,..])
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
  - db.collection.updateOne(<query>, <update>, { upsert: true })
  - db.collection.updateMany(<query>, <update>, { upsert: true })
- Delete
  - db.collection.deleteMany(<query>)
  - db.collection.deleteOne(<query>)

Collection specifies the collection or the ‘table’ to store the document

MONGODB - INSERT DOCUMENT

Insert Documents
There are 2 methods to insert documents into a MongoDB database.
insertOne()
To insert a single document, use the insertOne() method.
This method inserts a single object into the database.

Syntax
- The basic syntax of insert() command is as follows –
- >db.COLLECTION_NAME.insert(document)

Db.collection specifies the collection or the 'table' to store the document

- db.collection_name.insert(<document>)
  - Omit the _id field to have MongoDB generate a unique key
  - Example db.**parts**.insert({{type: "screwdriver", quantity: 15 })
  - db.**parts**.insert({_id: 10, type: "hammer", quantity: 1 })

insertMany()

To insert multiple documents at once, use the insertMany() method.
This method inserts an array of objects into the database.

Syntax:

```
db.Collection_name.insertMany(
[{<document 1>, <document 2>, ...}])
```

MONGODB – FIND OPERATIONS

MongoDB provides powerful methods for retrieving documents from its collections with **find()** and **findOne()** methods. The find() method supports **complex queries** with various **operators** and allows specifying which fields to include or exclude and optimizing performance by using indexes.

Find() Method

•It is a **primary** method for retrieving documents from a collection in **MongoDB**.

•It Supports a wide range of query operators, including **comparison**, **logical** and **element** operators also allowing for complex queries.

•It allows us to specify which fields to include or exclude in the result set and reduce the amount of data transferred.

•It automatically uses **indexes** to optimize query performance also improving speed and efficiency.

Syntax:

```
db.Collection_name.find(selection_criteria, projection,options)
```

findOne() Method

- **findOne()** in MongoDB is a method used to retrieve a single document from a collection that matches a specified criteria.
- It returns **only one document**, even if multiple documents match the criteria.
- If no matching document is found, it returns null.
- You can optionally provide a **query criteria** to specify which document to find.

MONGODB – UPDATE() METHOD

MongoDB update operations allow us to modify documents in a collection. These operations can update a single document or multiple documents based on specified criteria.
MongoDB offers various update operators to perform specific actions like **setting a value, incrementing a value** or **updating elements within arrays**.
MongoDB update()
The MongoDB update() method is a method that is used to update a single document or multiple documents in the collection. When the document is updated the _id field remains unchanged.
The db.collection.update() method updates a single document by default. To update all documents that match the given query, use the multi: true option. Include the option "multi: true".

Syntax

```
db.COLLECTION_NAME.update({SELECTI
ON_CRITERIA},
{$set:{UPDATED_DATA}}, {
upsert: <boolean>,
multi: <boolean>,
writeConcern: <document>,
collation: <document>,
arrayFilters: [ <filterdocument1>,
... ],
hint: <document|string>
})
```

MONGODB – UPDATE() METHOD

MongoDB Update() Method

The update() method updates the values in the existing document.

Syntax

The basic syntax of **update()** method is as follows −

>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)

- db.collection_name.update(<query>, <update>, { upsert: true })
  - Will update 1 or more records in a collection satisfying query
- db.collection_name.findAndModify(<query>, <sort>, <update>,<new>, <fields>,<upsert>)
  - Modify existing record(s) – retrieve old or new version of the record

DELETE OPERATIONS

db.collection.deleteOne()

db.collection.deleteOne() deletes the first document that matches the filter. Use a field that is part of a unique index such as _id for precise deletions.

db.collection.deleteMany() operation successfully deletes one or more documents

- db.collection_name.remove(<query>, <justone>)
  - Delete all records from a collection or matching a criterion
  - <justone> - specifies to delete only 1 record matching the criterion
  - Example: db.parts.remove(type: /^h/ }) - remove all parts starting with h
  - Db.parts.remove() – delete all documents in the parts collections

MONGODB PROJECTION

- In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

Syntax

The basic syntax of **find()** method with projection is as follows

```
>db.COLLECTION_NAME.find({},{KEY:1})
```

Limit() Method

To limit the records in MongoDB, you need to use **limit()** method. The method accepts one number type argument, which is the number of documents that you want to be displayed.

Syntax

The basic syntax of **limit()** method is as follows −

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

MongoDB Skip() Method

Apart from limit() method, there is one more method **skip()** which also accepts number type argument and is used to skip the number of documents.

Syntax

The basic syntax of **skip()** method is as follows −

>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)

Example

Following example will display only the second document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(1).skip(1)
{"title":"NoSQL Overview"} >
```

MongoDB – sort() Method

- To sort documents in MongoDB, you need to use **sort()** method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

Syntax

The basic syntax of **sort()** method is as follows –

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

- MongoDB count()
- The count() method in MongoDB is a simple and effective way to count the number of documents that meet specific criteria in a collection. It can take an optional query parameter to filter the documents before counting. The count() method is available in both the db.collection and cursor objects in MongoDB.
- db.collection.count() or db.collection.countDocuments()
- Returns the count of documents that would match a `find()` query for the collection or view. The `db.collection.count()` method does not perform the `find()` operation but instead counts and returns the number of results that match a query.

MONGODB – FINDANDMODIFY() METHOD

The **MongoDB** findAndModify method is used to modify and return a **single document** that matches the given criteria. It can be used to **update**, **remove** or **insert** documents atomically.

```
db.collection.findOneAndDelete( filter, options )
```

MONGODB – FINDONE AND DELETE() METHOD

The **MongoDB findOneAndDelete()** is a powerful tool for deleting documents from a collection based on specific criteria. It allows for the deletion of a single document that matches the query filter.

Syntax:

```
db.Collection_name.findOneAndDelete(
 Selection_criteria,
{
    projection: <document>,
    sort: <document>,
    maxTimeMS: <number>,
    collation: <document>
})
```

MONGODB FINDONE AND REPLACE() METHOD

- The **findOneAndReplace()** method replaces the first matched document based on the selection criteria.
  replacing
- By default, this method returns the original document. To return the replacement document, set the value of the returnNewDocument option to true.
- **`db.collection.findOneAndReplace( filter, replacement, options )`**

QUERY OPERATORS

Name	Description
$eq	Matches value that are equal to a specified value
$gt, $gte	Matches values that are greater than (or equal to a specified value
$lt, $lte	Matches values less than or (equal to) a specified value
$ne	Matches values that are not equal to a specified value
$in	Matches any of the values specified in an array
$nin	Matches none of the values specified in an array
$or	Joins query clauses with a logical OR returns all
$and	Join query clauses with a loginal AND
$not	Inverts the effect of a query expression
$nor	Join query clauses with a logical NOR
$exists	Matches documents that have a specified field

https://docs.mongodb.org/manual/reference/operator/query/

CRUD EXAMPLES

```
> db.user.insert({
      first: "John",
      last : "Doe",
      age: 39
})
```

```
> db.user.find ()
{ "_id" : ObjectId("51"),
      "first" : "John",
      "last" : "Doe",
      "age" : 39
}
```

```
> db.user.update(
      {"_id" : ObjectId("51")},
      {
            $set: {
                  age: 40,
                  salary: 7000}
      }
)
```

```
> db.user.remove({
      "first": /^J/
})
```

SQL VS. MONGO DB ENTITIES

My SQL	Mongo DB
START TRANSACTION; INSERT INTO **contacts** VALUES (NULL, 'joeblow'); INSERT INTO **contact_emails** VALUES (NULL, "joe@blow.com", LAST_INSERT_ID()), (NULL, "joseph@blow.com", LAST_INSERT_ID()); COMMIT;	db.contacts.save({ userName: "joeblow", emailAddresses: ["joe@blow.com", "joseph@blow.com"] });

Similar to IDS from the 70's
Bachman's brainchild

DIFFERENCE:
MongoDB separates physical structure from logical structure

Designed to deal with large &distributed

Reference Books:

1.NoSQL with MongoDB in 24 Hours, Sams Teach Yourself 1st Edition by Brad Dayley

2. MongoDB Basics 1st Edition by Peter Membrey, David Hows, and Eelco Plugge

Sites and Web links:

1. https://www.mongodb.com/docs/manual/faq/