Course code : **CSE3009**
Course title : **No SQL Data Bases**
Module : **4**
Topic : **1**

Document Oriented Database

Objectives

This session will give the knowledge about

- Document
- Collection
- CRUD Operations

Document Oriented Databases

Document databases, also called document-oriented databases, use a key-value approach to storing data but with important differences from key-value databases.

A document database stores values as documents. In this case, documents are semi structured entities, typically in a standard format such as JavaScript Object Notation (JSON) or Extensible Markup Language (XML).

It refers to data structures that are stored as strings or binary representations of strings.

Document Oriented Databases

Instead of storing each attribute of an entity with a separate key, document databases store multiple attributes in a single document. Here is a simple example of a document in JSON format:

```
{
“firstName”: “Alice”,
“lastName”: “Johnson”,
“position”: “CFO”,
“officeNumber”: “2-120”,
“officePhone”: “555-222-3456”,
}
```

Document Oriented Databases

One of the most important characteristics of document databases is you do not have to define a fixed schema before you add data to the database.

Simply adding a document to the database creates the underlying data structures needed to support the document.

The lack of a fixed schema gives developers more flexibility with document databases than they have with relational databases.

Document databases provide application programming interfaces (APIs) or query languages that enable you to retrieve documents based on attribute values.

Document

Documents and collections are the basic data structures of a document database. They are somewhat analogous to rows and tables in relational databases.

A document is a set of ordered key-value pairs. A key value is a data structure that consists of two parts called, the key and the value.

A key is a unique identifier used to look up a value. A value is an instance of any supported data type, such as a string, number, array, or list.

Document: Ordered Sets of Key-Value Pairs

Because a document is a set, it has one instance of each member. Members are key-value pairs. For example,

{ ‘foo’: ‘a’, ‘bar’: ‘b’, ‘baz’: ‘c’} is also equivalent to

{‘baz’: ‘c’, ‘foo’: ‘a’, ‘bar’: ‘b’}

A slight change turns this set into a nonset (also known as a bag):

{‘foo’: ‘a’, ‘bar’: ‘b’, ‘baz’: ‘c’, ‘foo’: ‘a’}

Document: Ordered Sets of Key-Value Pairs

However, for the purposes of designing document databases, these are different documents. The order of key-value pairs matters in determining the identity of a document.

The document { 'foo': 'a', 'bar': 'b', 'baz': 'c'} is not the same document as {'baz': 'c', 'foo': 'a', 'bar': 'b'}.

Document

Keys are generally strings. Some key-value databases support a more extensive set of key data types, so document databases could, in principle, support multiple data types as well.

Values can be a variety of data types. As you might expect, document databases support values of numbers and strings. They also support more structured data types, such as arrays and other documents.

Arrays are useful when you want to track multiple instances of a value and the values are all of one type.

Document

{
‘employeeName’ : ‘Janice Collins’,
‘department’ : ‘Software engineering’
‘startDate’ : ‘10-Feb-2010’,
‘pastProjectCodes’ : [189847, 187731, 176533, 154812]
}

The key pastProjectCodes is a list of project code numbers. All project codes are numbers, so it is appropriate to use an array.

Document

```
{  'employeeName' :  'Janice Collins',
    'department' : 'Software engineering',
    'startDate' : '10-Feb-2010',
    'pastProjects' : {
{'projectCode' : 189847,
      'projectName' : 'Product Recommendation System',
      'projectManager' : 'Jennifer Delwiney' },
{'projectCode' : 187731,
  'projectName' :  'Finance Data Mart version 3',
  'projectManager' : 'James Ross'},
{'projectCode': 176533,
 'projectName' : 'Customer Authentication',
    'projectManager' : 'Nick Clacksworth'},
{'projectCode': 154812,
 'projectName' : 'Monthly Sales Report',
 'projectManager': 'Bonnie Rendell'}
}
```

Collection

A collection is a group of documents. The documents within a collection are usually related to the same subject entity, such as employees, products, logged events, or customer profiles.

It is possible to store unrelated documents in a collection, but this is not advised. At the most basic level, collections allow you to operate on groups of related documents.

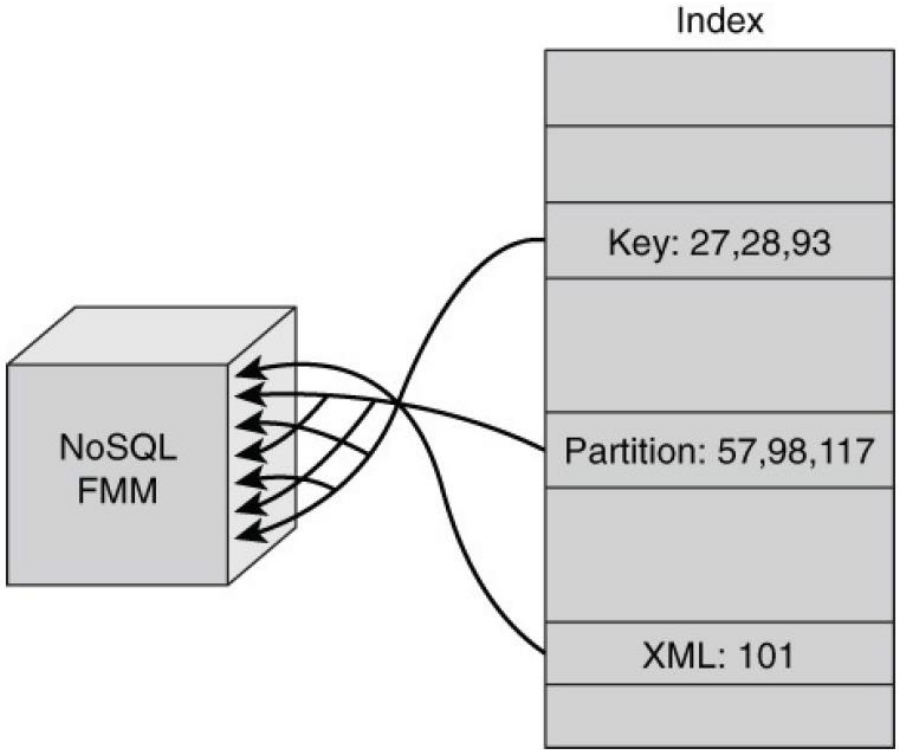
In addition to allowing you to easily operate on groups of documents, collections support additional data structures that make such operations more efficient.

Collection - Index

A more efficient approach to scanning all documents in a collection is to use an index.

Indexes on collections are like indexes in the back of book: a structured set of information that maps from one attribute, such as a key term, to related information, such as a list of page numbers.

Indexes map attributes, such as key terms, to related information, such as page numbers. Using an index is faster than scanning an entire book for key terms.



Embedded Document

One of the advantages of document databases is that they allow developers to store related data in more flexible ways than typically done in relational databases.

Relational data models separate data about different entities into separate tables. This requires looking up information in both tables using a process known as joining.

Joining two large tables can be potentially time consuming and require a significant number of read operations from disk.

Embedded Document

An embedded document enables document database users to store related data in a single document.

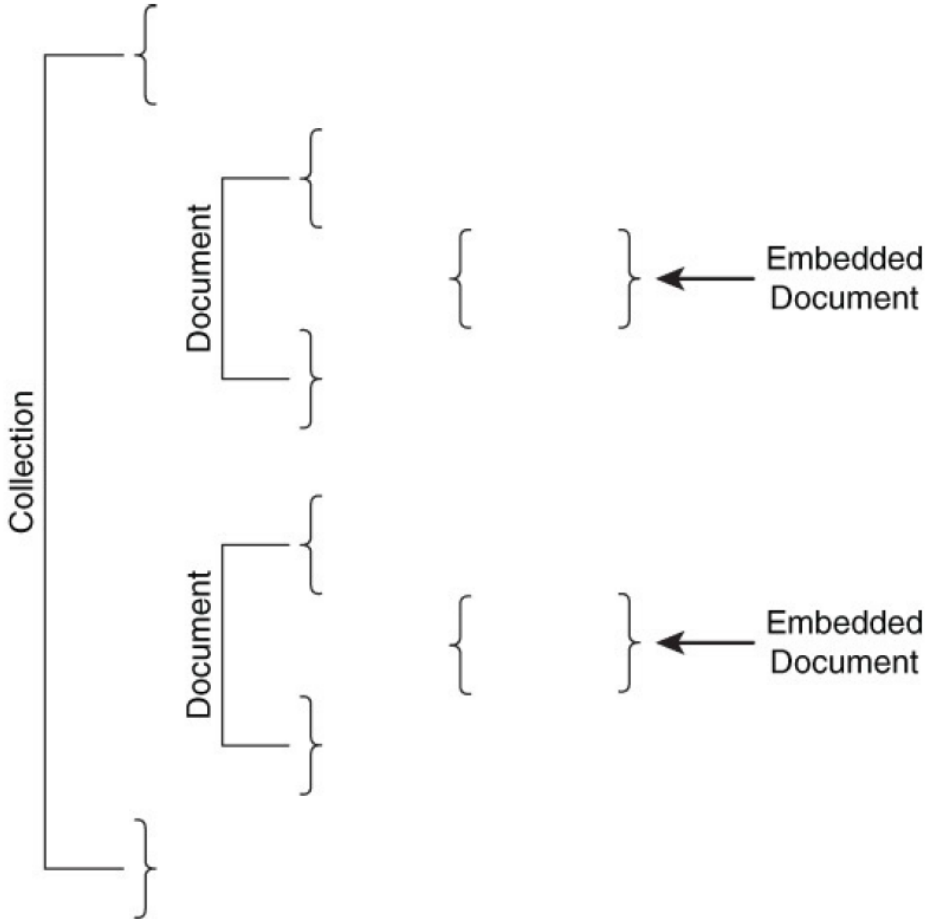
This allows the document database to avoid a process called joining in which data from one table, called the foreign key, is used to look up data in another table.

Embedded documents allow related data to be stored together. When the document is read from disk, both the primary and the related information are read without the need for a join operation.

Embedded Document

Embedded documents are documents within a document.

Embedding is used to efficiently store and retrieve data that is frequently used together.



Mongo DB

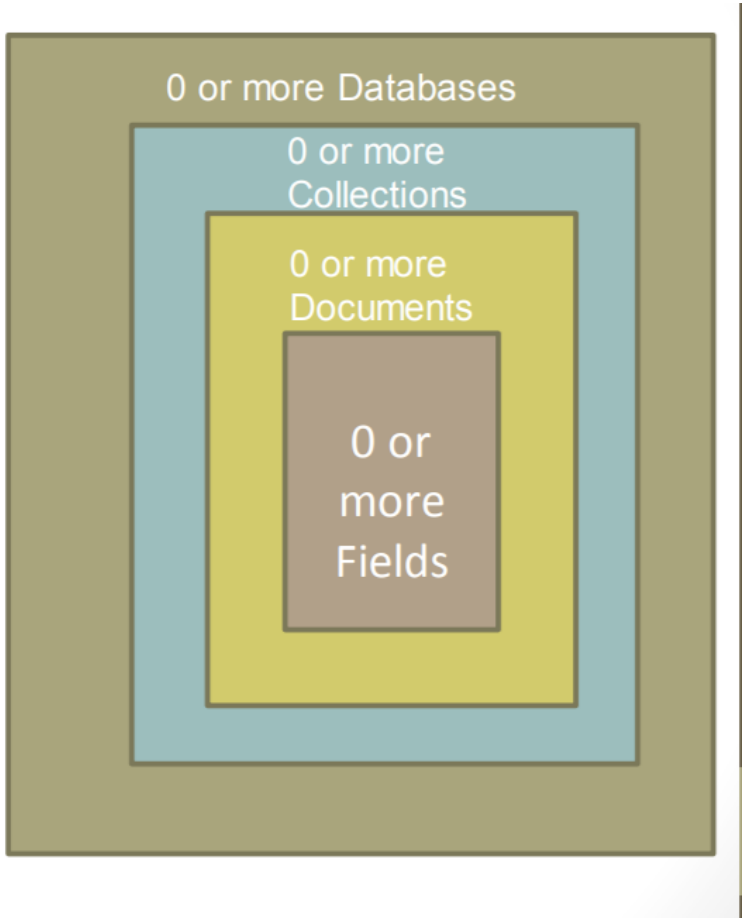
- A document-oriented, NoSQL database developed by 10gen (Founded Year : 2007)
- Hash-based, schema-less database
- No Data Definition Language - you can store hashes with any keys and values that you choose
- Uses BSON format - Based on JSON – B stands for Binary
- Written in C++
- Supports APIs (drivers) in many computer languages - JavaScript, Python, Ruby, Perl, Java, Java Scala, C#, C++, Haskell, Erlang

MongoDB Terminology

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

MongoDB: Hierarchical Objects

- A database may have zero or more ‘collections’.
- A collection may have zero or more ‘documents’.
- A document may have one or more ‘fields’.



The use Command

MongoDB use DATABASE_NAME is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax: **use DATABASE_NAME**

Example

Create a database called ‘movies’ and write a MongoDB query to select database as ‘movies’

>use movies

>switched to db movies

show dbs command

To check your currently selected database, use the command db

```
>db
movies
```

If you want to check your databases list, use the command show dbs.

```
>show dbs
local     0.78125GB
test      0.23012GB
```

Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

```
>db.movie.insert({"name":"tutorials point"})
>show dbs
```

db.dropDatabase() command

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

Syntax

Basic syntax of dropDatabase() command is as follows:

```
db.dropDatabase()
```

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

Example

```
>use mydb
switched to db mydb
>db.dropDatabase()
>{ "dropped" : "mydb", "ok" : 1 }
```

createCollection() Method

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

Syntax

Basic syntax of createCollection() command is as follows:

```
db.createCollection(name, options)
```

In the command, name is name of collection to be created. Options is a document and is used to specify configuration of collection.

Examples

```
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 }
```

createCollection() Method

Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.**
autoIndexId	Boolean	(Optional) If true, automatically create index on _id field.s Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.**
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

drop() Method

MongoDB's db.collection.drop() is used to drop a collection from the database.

Syntax

Basic syntax of drop() command is as follows:

```
db.COLLECTION_NAME.drop()
```

Example

```
>db.mycollection.drop()
true
```

Mongo DB Datatypes

- String − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- Integer − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- Boolean − This type is used to store a boolean (true/ false) value.
- Double − This type is used to store floating point values.
- Min/ Max keys − This type is used to compare a value against the lowest and highest BSON elements.

Mongo DB Datatypes

- Arrays − This type is used to store arrays or list or multiple values into one key.
- Timestamp − ctimestamp. This can be handy for recording when a document has been modified or added.
- Object − This datatype is used for embedded documents.
- Null − This type is used to store a Null value.
- Symbol − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.

Mongo DB Datatypes

- Date − This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- Object ID − This datatype is used to store the document's ID.
- Binary data − This datatype is used to store binary data.
- Code − This datatype is used to store JavaScript code into the document.
- Regular expression − This datatype is used to store regular expression.

insert() Method

To insert data into MongoDB collection, you need to use MongoDB's insert() or save() method.

Syntax

The basic syntax of insert() command is as follows:

```
>db.COLLECTION_NAME.insert(document)
```

Example

insert() Method

```
>db.mycol.insert({
_id: ObjectId(7df78ad8902c),
eid: '12e123',
name: 'Krishnan',
Salary: 50000,
tags: ['faculty', 'database', 'NoSQL'],
likes: 100
})
```

_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows −

_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id,
3 bytes incrementer)

insert() Method

To insert multiple documents in a single query

```
>db.mycol.insert([
{
eid: '12e123',
name: 'Krishnan',
Salary: 50000,
tags: ['faculty',
      'database',
      'NoSQL'],
likes: 100
},
```

```
{
eid: '12e143',
name: 'Madhan',
Salary: 40000,
tags: ['faculty', 'database', 'NoSQL'],
address:[ { street: 'kk nagar',
            city: 'chennai',
            pin: 600045 } ]
}
])
```

find() Method

To query data from MongoDB collection, you need to use MongoDB's find() method.

Syntax
The basic syntax of find() method is as follows:

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

find() method will display all the documents in a non-structured way.

The pretty() Method
To display the results in a formatted way, you can use pretty() method.

Syntax

```
>db.mycol.find().pretty()
```

Where Clause in MongoDB

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50

AND, OR in MongoDB

```
>db.mycol.find(
   {
      $and: [
         {key1: value1}, {key2:value2}
      ]
   }
).pretty()
```

```
>db.mycol.find(
   {
      $or: [
         {key1: value1}, {key2:value2}
      ]
   }
).pretty()
```

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)
```

Example

```
>db.mycol.update({'title':'MongoDB Overview'},
        {$set:{'title':'New MongoDB Tutorial'}},{multi:true})
```

Save() Method

The save() method replaces the existing document with the new document passed in the save() method.

Syntax

The basic syntax of MongoDB save() method is shown below −

```
>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

Example

```
>db.mycol.save( {
 "_id" : ObjectId(5983548781331adf45ec5), "title":"Tutorials Point New
Topic", "by":"Tutorials Point"
} )
```

remove() Method

MongoDB's remove() method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

- **deletion criteria** − (Optional) deletion criteria according to documents will be removed.
- **justOne** − (Optional) if set to true or 1, then remove only one document.

Syntax

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

remove() Method

```
>db.mycol.remove({'title':'MongoDB Overview'})
```

Remove Only One

If there are multiple records and you want to delete only the first record, then set justOne parameter in remove() method.

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Remove All Documents

If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. This is equivalent of SQL's truncate command.

```
>db.mycol.remove({})
```

Summary

This session will give the knowledge about

- Document
- Collection
- CRUD Operations