**Birla Institute of Technology and Science, Pilani.**

**Database Systems**

**Lab No #9**



What is NoSQL?

NoSQL is a non-relational Database Management System, that does not require a fixed schema, avoids joins, and is easy to scale. NoSQL database is used for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps. For example companies like Twitter, Facebook, Google that collect terabytes of user data every single day. NoSQL database stands for "Not Only SQL" or "Not SQL."

Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data.

Why NoSQL**?**

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.

The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out." NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.

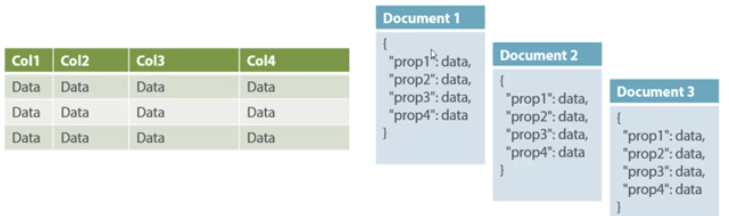
There are mainly four categories of NoSQL databases. Each of these categories has its unique attributes and limitations. No specific database is better to solve all problems. You should select a database based on your product needs.These are:

* Key-value Pair Based
* Column-oriented Graph
* Graphs based
* Document-oriented

In this labsheet, we are going to look at MongoDB, which is a document based NoSQL database.

Document-Oriented NoSQL Databases:

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.



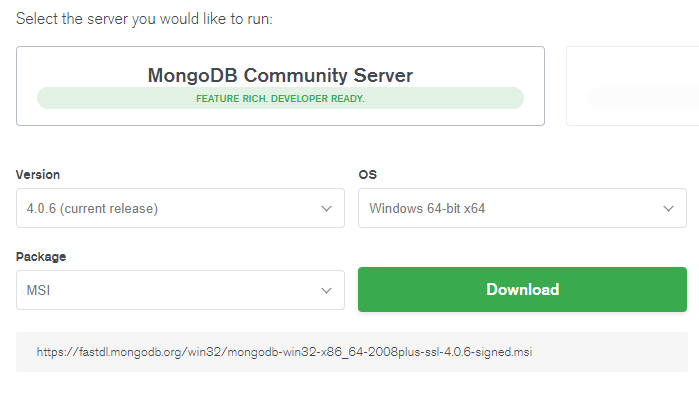
Relational vs Document based Databases

In this diagram on your left you can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON. Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object. You do not require to define which make it flexible. The document type is mostly used for CMS systems, blogging platforms, real-time analytics & e-commerce applications. It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.

* Note: JSON stands for Java Script Object Notation. JSON objects are surrounded by curly braces {}. JSON objects are written in key/value pairs. Keys must be strings, and values must be a valid JSON data type (string, number, object, array, boolean or null). Keys and values are separated by a colon. Each key/value pair is separated by a comma. Example-{ "name":"John", "age":30, "car":null } is a JSON object where “name”, ”age” and “car” are keys and “John”, 30 and null are their values respectively.

Installing MongoDB on Windows

We require that MongoDB be successfully installed on your laptop before the start of your labsheet. Please click the following link – [Downloads Page](http://www.mongodb.org/downloads), to go to the downloads page as given below.



Choose the following options if your laptop is 64-bit (choose the 32-bit version if your laptop is 32-bit). Now press download.

Your package should start downloading. Locate the downloaded MongoDB .msi file, which typically is located in the default Downloads folder. Double-click the .msifile. A set of screens will appear to guide you through the installation process.

You may specify an installation directory if you choose the “Custom” installation option. However these instructions assume that you have installed MongoDB to

*C:\Program Files\MongoDB\Server\3.2\*

1. **Set up the MongoDB environment**: MongoDB requires a data directory to store all data. MongoDB’s default data directory path is \data\db. Create this folder using the following commands from a commandprompt:

* Md \data\db

1. **Start MongoDB**: To start MongoDB, run [mongod.exe](https://docs.mongodb.com/v3.2/reference/program/mongod.exe/#bin.mongod.exe). For example, from the command prompt. This will start the main MongoDB database process. The waiting for connections message in the console output indicates that the [mongod.exe](https://docs.mongodb.com/v3.2/reference/program/mongod.exe/#bin.mongod.exe) process is running successfully.

* “C:\Program Files\MongoDB\Server\3.2\bin\mongod.exe”

1. **Connect to MongoDB**: To connect to MongoDB through the ~bin.mongo.exe shell, open another command prompt.

* “C:\Program Files\MongoDB\Server\3.2\bin\mongo.exe”

1. **Now begin using 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. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection - Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document - A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

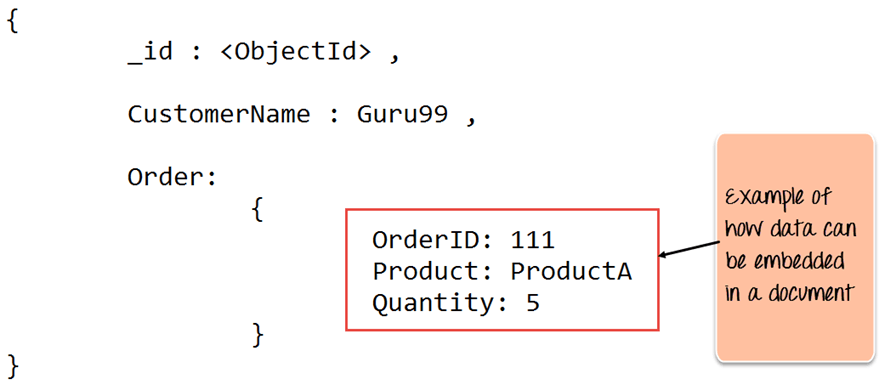
The following table shows the relationship of RDBMS terminology with MongoDB.

|  |  |
| --- | --- |
| **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) |
| **Database Server and Client** | |
| Mysqld/Oracle | mongod |
| mysql/sqlplus | mongo |

**\_id** is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide \_id while inserting the document. If you don’t provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

Any relational database has a typical schema design that shows number of tables and the relationship between these tables. While in MongoDB, there is no concept of relationship.

Embedded Documents in MongoDB



From the example in the snippet, you can note that the Order Data (OrderID, Product, and Quantity ) which in RDBMS will normally be stored in a separate table, is actually stored as an embedded document in the collection itself in MongoDB. This is one of the key differences in how data is modeled in MongoDB.

Advantages of MongoDB over RDBMS

* **Schema less** − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
* Structure of a single object is clear.
* No complex joins.
* Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
* Tuning.
* **Ease of scale-out** − MongoDB is easy to scale.
* Conversion/mapping of application objects to database objects not needed.
* Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Data in MongoDB has a flexible schema.documents in the same collection. They do not need to have the same set of fields or structure, and common fields in a collection’s documents may hold different types of data.

How to create a database in MongoDB

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.

**> use DATABASE\_NAME**

If you want to use a database with name **<mydb>,** type“use mydb” in the prompt.

>db

mydb

>show dbs

>db.movie.insert({"name":"dbms"})

>show dbs

local 0.78125GB

mydb 0.23012GB

test 0.23012GB

How to drop a database using MongoDB command

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

**db.dropDatabase()**

>show dbs

local 0.78125GB

mydb 0.23012GB

test 0.23012GB

If you want to delete new database **<mydb>**

>use mydb

(switched to db mydb)

>db.dropDatabase()

>{ "dropped" : "mydb", "ok" : 1 }

>show dbs

local 0.78125GB

test 0.23012GB

How to create a collection using MongoDB

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

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

**name** is name of collection to be created. **Options** is a document and is used to specify configuration of collection.

>use test

switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }

>show collections

mycollection

system.indexes

>db.createCollection("mycol", { capped : true, autoIndexId : true, size :

6142800, max : 10000 } )

{ "ok" : 1 }

In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

>db. MongoDBTutorial.insert({"name" : " MongoDBTutorial"})

>show collections

mycol

mycollection

system.indexes

MongoDBTutorial

How to drop a collection using MongoDB

**db.collection.drop()** is used to drop a collection from the database.

**db.COLLECTION\_NAME.drop()**

>use mydb

(switched to db mydb)

>show collections

mycol

mycollection

system.indexes

MongoDBTutorial

>db.mycollection.drop()

true

>show collections

mycol

system.indexes

MongoDBTutorial

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

MongoDB supports many datatypes. Some of them are −

* **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.
* **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.
* **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.

How to insert document in MongoDB collection

**>db.COLLECTION\_NAME.insert(document)**

>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 our collection name

To insert multiple documents in a single query, you can pass an array of documents in insert() command.

>db.post.insert([

{

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 100

},{

title: 'NoSQL Database',

description: "NoSQL database doesn't have tables",

tags: ['mongodb', 'database', 'NoSQL'],

likes: 20,

comments: [

{

user:'user1',

message: 'My first comment',

dateCreated: new Date(2013,11,10,2,35),

like: 0

}

]

}

])

To insert the document you can use **db.post.save(document)** also.

How to query document from MongoDB collection

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

**>db.COLLECTION\_NAME.find()**

**find()** method will display all the documents in a non-structured way.To display the results in a formatted way, you can use **pretty()** method.

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

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

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

In the **find()** method, if you pass multiple keys by separating them by ',' then MongoDB treats it as **AND** condition

>db.mycol.find(

{ $and: [

{key1: value1}, {key2:value2}

]

}

).pretty()

>db.mycol.find({$and:[{"description": "MongoDB is no sql database" },{"title": "MongoDB Overview"}]}).pretty() {

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

Query documents based on the OR condition, you need to use **$or** keyword.

>db.mycol.find(

{ $or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

>db.mycol.find({$or:[{"description": "MongoDB is no sql database" },{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

Limit Query

This modifier is used to limit the number of documents which are returned in the result set for a query. The following example shows how this can be done.

**db.Employee.find().limit(2).forEach(printjson);**

The above code takes the find function which returns all of the documents in the collection but then uses the limit clause to limit the number of documents being returned to just 2.

Sort Query

One can specify the order of documents to be returned based on ascending or descending order of any key in the collection. The following example shows how this can be done.

What is Primary Key in MongoDB?

In MongoDB, \_id field as the primary key for the collection so that each document can be uniquely identified in the collection. The \_id field contains a unique ObjectID value. By default when inserting documents in the collection, if you don't add a field name with the \_id in the field name, then MongoDB will automatically add an Object id field. When you query the documents in a collection, you can see the ObjectId for each document in the collection.

If you want to ensure that MongoDB does not create the \_id Field when the collection is created and if you want to specify your own id as the \_id of the collection, then you need to explicitly define this while creating the collection.

When explicitly creating an id field, it needs to be created with \_id in its name.

**db.Employee.insert({\_id:10, "EmployeeName" : "Smith"})**

Cursor

When the **db.collection.find ()** function is used to search for documents in the collection, the result returns a pointer to the collection of documents returned which is called a cursor. By default, the cursor will be iterated automatically when the result of the query is returned. But one can also explicitly go through the items returned in the cursor one by one. The following example shows how this can be done.

**var myEmployee = db.Employee.find( { Employeeid : { $gt:2 }});**

**while(myEmployee.hasNext())**

**{**

**print(tojson(myEmployee.next()));**

**}**

First we take the result set of the query which finds the Employee's whose id is greater than 2 and assign it to the JavaScript variable 'myEmployee'. Next we use the while loop to iterate through all of the documents which are returned as part of the query. Finally for each document, we print the details of that document in JSON readable format.

Mongo prompt is actually a JavaScript prompt. This means you can write individual JavaScript code pieces in the prompt too. JavaScript here is a language using which we work with MongoDB.

Exercises:

Run the “movies” script uploaded on nalanda and write the following queries:

1. List the movies with a metacritic rating greater than 90.

2. List the movies starring Ryan Reynolds.

3. Print the names of the writers of the movies with an imdb rating between 6 and 7. Print them in the following format:

*movie name 1: writer1, writer2*

*movie name 2: writer1*

*....*

ATOMICITY

In MongoDB, an operation on a single document is atomic. Because you can use embedded documents and arrays to capture relationships between data in a single document structure instead of normalizing across multiple documents and collections, this single-document atomicity obviates the need for multi-document transactions for many practical use cases.

Exercises:

1. findAndModify command is used to find documents based on a condition and modify those selected documents. Run the code pieces given and create Product\_Info, Product and Info before attempting the questions.

(a) Can findAndModify be used for embedded collection Product\_Info? If yes, use it to find if items of product with \_id = 1 are still left, reduce available items by 1 and add customer information if items are still left. If no, reason why and provide an alternative method to achieve this.

(b) Can findAndModify be used for multiple collections like Product and Info? If yes, use it to find if items of product with \_id = 1 are still left, reduce available items by 1 and add customer information if items are still left. If no, reason why and provide an alternative method to achieve this.

TRANSACTIONS:

However, in the part (b) of the solution to the above exercise, it is possible that between the executions of the two queries, some other user has purchased the product and it is no more available. Without knowing this, your second query will update the purchase information based on the result of our first query. This will make the database inconsistent because you have sold a product which is not available.

Multi-document transactions can be used across multiple operations, collections, databases, and documents. Multi-document transactions provide an “all-or-nothing” proposition. When a transaction commits, all data changes made in the transaction are saved. If any operation in the transaction fails, the transaction aborts and all data changes made in the transaction are discarded without ever becoming visible. Until a transaction commits, no write operations in the transaction are visible outside the transaction.

The following [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#bin.mongo) shell methods are available for transactions:

[Session.startTransaction()](https://docs.mongodb.com/manual/reference/method/Session.startTransaction/#Session.startTransaction)

[Session.commitTransaction()](https://docs.mongodb.com/manual/reference/method/Session.commitTransaction/#Session.commitTransaction)

[Session.abortTransaction()](https://docs.mongodb.com/manual/reference/method/Session.abortTransaction/#Session.abortTransaction)

Exercises:

1. Find if items of product with \_id = 1 are still left, reduce available items by 1 and add customer information if items are still left. Use transactions and make the action atomic.

## Referenced Relationships

This is the approach of designing normalized relationship. We will see what are normalized relationships in the subsequent sections. In this approach, documents will be maintained separately.

USER DOCUMENT:

{

"\_id":ObjectId("52ffc33cd85242f436000001"),

"contact": "987654321",

"dob": "01-01-1991",

"name": "Tom Benzamin",

"address\_ids": [

ObjectId("52ffc4a5d85242602e000000"),

ObjectId("52ffc4a5d85242602e000001")

]

}

As shown above, the user document contains the array field address\_ids which contains ObjectIds of corresponding addresses. Using these ObjectIds, we can query the address documents and get address details from there. The alternative to this model is embedding(already seen).

NORMALIZATION vs DENORMALIZATION

Normalization:

When you normalize your data, you are dividing your data into multiple collections with references between those collections. Each piece of data will be in a collection, but multiple documents will reference it. This means, to change your data, you only need to update one document, because that data is only defined once. However, MongoDB doesn’t have any join facilities, like SQL does. Therefore, if you need data from several collections, you will need to perform several queries. This inturn requires multiple trips to the database and multiple reads.

Denormalization:

You want to optimize your reads. You don’t want multiple trips to the database to get your informations. For example, one way to store the accounts preferences of each user is using an embedded document, like so:

{

\_id: ObjectId("5977aad83abbae8aef44b47b"),

name: "John Doe",

email: "johndoe@gmail.com",

articles: [

ObjectId("5977aad83abbae8aef44b47a"),

ObjectId("5977aad83abbae8aef44b478"),

ObjectId("5977aad83abbae8aef44b477")

],

accountsPref: {

style: "light",

showFriends: true,

notificationsOn: false

}

}

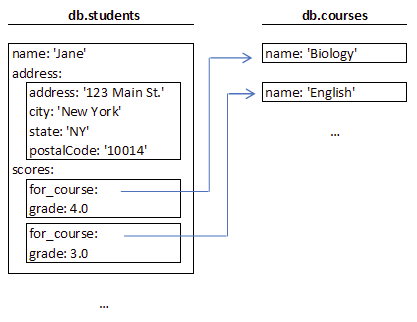
The advantage of this is that you need lesser queries to get the information you need. The downside is that it takes up more space and is more difficult to keep in sync.

JOIN?

MongoDB doesn’t support joins. How to achieve tasks similar to SQL JOINs in MongoDB?

MongoDB is not relational. There is no standard "normal form". You should model your database appropriate to the data you store and the queries you intend to run. Actions similar to JOIN should be performed using multiple queries and references based on the model you used for your database.

Exercises:

1. Design the above scheme using embedded documents. Is this normalization or denormalization?

2. Design the above scheme using references. Is this normalization or denormalization?

3. Write a query to retrieve Jane’s courses and grades information from the first design.

4. Write a query to retrieve Jane’s courses and grades information from the second design.

### **How to choose between Embedding and Referencing?**

### Embedding:

* You have small subdocuments
* Your data does not need to change regularly
* You don’t need immediate consistency ( not up-to-date )
* Your documents grow by a small amount
* You need this data to perform a second query
* You want faster reads

### Referencing:

* You have large subdocuments
* Your data changes frequently
* You need your data to be up-to-date
* Your documents grow by a large amount
* Your data is often excluded from your results
* You want faster writes

KEYS IN MONGODB:

As discussed earlier, primary keys are part of MongoDB. The id field is by default the unique identifier of documents, thus serving as the primary key.

MongoDB does not have explicit foreign keys. The idea of foreign keys can be designed using referencing. Note that however the database does not apply any constraints to the system (i.e.: foreign key contstraints), so there are no "cascading deletes" or "cascading updates". The correctness of referencing is upto the implementation.

NoSQL vs SQL:

Understanding the essence of MongoDB requires an understanding of differences between SQL Server/MySQL and MongoDB. Here are few actions performed in both the DBMSs side by side.

