
EAST WEST UNIVERSITY

# CSE479
## Web Programming

**Nishat Tasnim Niloy**

Lecturer

Department of Computer Science and Engineering

Faculty of Science and Engineering

# Topic 8

PHP and NoSQL

# RDBMS Characteristics

- Data stored in columns and tables
- Relationships represented by data
- Data Manipulation Language
- Data Definition Language
- Transactions
- Abstraction from physical layer
- Applications specify what, not how
- Physical layer can change without modifying applications
  - Create indexes to support queries
  - In Memory databases

# ACID Property in DBMS

## Atomic

- All the work in a transaction completes (commit) or none of it completes
- a transaction to transfer funds from one account to another involves making a withdrawal operation from the first account and a deposit operation on the second. If the deposit operation failed, you don’t want the withdrawal operation to happen either.

## Consistent

- A transaction transforms the database from one consistent state to another consistent state. Consistency is defined in terms of constraints.
- a database tracking a checking account may only allow unique check numbers to exist for each transaction

## Isolated

- The results of any changes made during a transaction are not visible until the transaction has committed.
- a teller looking up a balance must be isolated from a concurrent transaction involving a withdrawal from the same account. Only when the withdrawal transaction commits successfully, and the teller looks at the balance again will the new balance be reported.

## Durable

- The results of a committed transaction survive failures
- A system crash or any other failure must not be allowed to lose the results of a transaction or the contents of the database. Durability is often achieved through separate transaction logs that can "re-create" all transactions from some picked point in time (like a backup).

# NoSQL

- NoSQL stands for:
  - No Relational
  - No RDBMS
  - Not Only SQL
- NoSQL is an umbrella term for all databases and data stores that don’t follow the RDBMS principles
  - A class of products
  - A collection of several (related) concepts about data storage and manipulation
  - Often related to large data sets

# NoSQL Definition

**From www.nosql-database.org:**

- **Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontal scalable. The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly.**
- **Often more characteristics apply as:**
  - **schema-free**
  - **easy replication support**
  - **simple API**
  - **eventually consistent / BASE (not ACID)**
  - **a huge data amount, and more.**

# Where does NoSQL come from?

- Non-relational DBMSs are not new
- But NoSQL represents a new incarnation
  - Due to massively scalable Internet applications
  - Based on distributed and parallel computing
- Development
  - Starts with Google
  - First research paper published in 2003
  - Continues also thanks to Lucene's developers/Apache (Hadoop) and Amazon (Dynamo)
  - Then a lot of products and interests came from Facebook, Netfix, Yahoo, eBay, Hulu, IBM, and many more

# Dynamo and BigTable

- Three major papers were the seeds of the NoSQL movement
  - BigTable (Google)
  - Dynamo (Amazon)
    - Distributed key-value data store
    - Eventual consistency
  - CAP Theorem (discuss in a sec ..)

# NoSQL and Big Data

- NoSQL comes from Internet, thus it is often related to the “big data” concept
- How much big are “big data”?
  - Over few terabytes Enough to start spanning multiple storage units
- Challenges
  - Efficiently storing and accessing large amounts of data is difficult, even more considering fault tolerance and backups
  - Manipulating large data sets involves running immensely parallel processes
  - Managing continuously *evolving schema* and metadata for *semi-structured and un-structured* data is difficult

# Why are RDBMS not suitable for Big Data

- The context is Internet
- RDBMSs assume that data are
  - Dense
  - Largely uniform (structured data)
- Data coming from Internet are
  - Massive and sparse
  - Semi-structured or unstructured
- With massive sparse data sets, the typical storage mechanisms and access methods get stretched

# NoSQL Distinguishing Characteristics

- Large data volumes
  - Google's "big data"
- Scalable replication and distribution
  - Potentially thousands of machines
  - Potentially distributed around the world
- Queries need to return answers quickly
- Mostly query, few updates

- Asynchronous Inserts & Updates
- Schema-less
- ACID transaction properties are not needed – BASE
- CAP Theorem
- Open source development

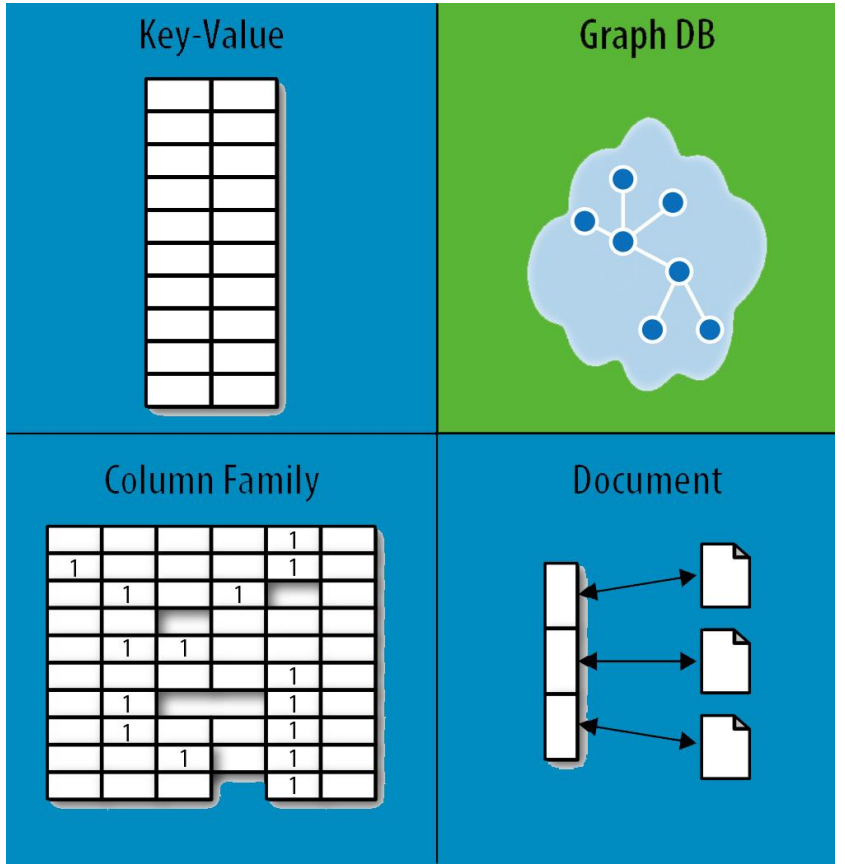
# BASE Property

| | |
|---|---|
| Basically Available | • This property refers to the fact that the database system should always be available to respond to user requests, even if it cannot guarantee immediate access to all data. |
| Soft State | • This property refers to the fact that the state of the database can change over time, even without any explicit user intervention. |
| Eventually Consistent | • This property refers to the eventual consistency of data in the database, despite changes over time. |

# NoSQL Database Types

Discussing NoSQL databases is complicated because there are a variety of types:

- Sorted ordered Column Store
  - Optimized for queries over large datasets, and store columns of data together, instead of rows
- Document databases:
  - pair each key with a complex data structure known as a document.
- Key-Value Store :
  - are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value.
- Graph Databases :
  - are used to store information about networks of data, such as social connections.



# Document Databases (Document Store)

- Documents
  - Loosely structured sets of key/value pairs in documents, e.g., XML, JSON, BSON
  - Encapsulate and encode data in some standard formats or encodings
  - Are addressed in the database via a unique key
  - Documents are treated as a whole, avoiding splitting a document into its constituent name/value pairs
- Allow documents retrieving by keys or contents
- Notable for:
  - MongoDB (used in FourSquare, Github, and more)
  - CouchDB (used in Apple, BBC, Canonical, Cern, and more)

# MongoDB

- MongoDB is a cross-platform, document-oriented database
- It provides high performance, high availability, and easy scalability.
- MongoDB works on concept of collection and document.

# Relationship of RDBMS terminology with MongoDB

| RDBMS | MongoDB |
| --- | --- |
| Database | Database |
| Table | Collection |
| Tuple/Row | Document |
| Column | Field |

# Advantages

- 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.
  - **Ease of scale-out** − MongoDB is easy to scale.

# Why Use MongoDB?

- **Document Oriented Storage** − Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability
- Fast in-place updates
- Professional support by MongoDB

# Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

# Data Modelling

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.

- Some considerations while designing Schema in MongoDB
  - Design your schema according to user requirements.
  - Combine objects into one document if you will use them together. Otherwise separate them (but make sure there should not be need of joins).
  - Duplicate the data (but limited) because disk space is cheap as compared to compute time.
  - Optimize your schema for most frequent use cases.

# Example

- Suppose a client needs a database design for his blog/website and see the differences between RDBMS and MongoDB schema design. Website has the following requirements.
- Every post has the unique title, description and url.
- Every post can have one or more tags.
- Every post has the name of its publisher and total number of likes.
- Every post has comments given by users along with their name, message, data-time and likes.
- On each post, there can be zero or more comments.

# Create Database

- The use Command

  In 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.

  Basic syntax of **use DATABASE** statement is as follows −

  ```
  use DATABASE_NAME
  ```

  Example

  ```
  >use mydb
  switched to db mydb
  ```

# Check Existing Database

- 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.

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

# Drop Database

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

  - 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 }**

# Create Collection

- The createCollection() Method

  Basic syntax of createCollection() command is as follows −

  db.createCollection(name, options)

  Examples

  Basic syntax of createCollection() method without options is as follows −

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

# Check Created Collection

- You can check the created collection by using the command show collections.

**>show collections**
**mycollection**
**system.indexes**

# Drop Collection

- Basic syntax of drop() command is as follows −

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

First, check the available collections into your database **mydb**.
Now drop the collection with the name mycollection.

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

drop() method will return true, if the selected collection is dropped successfully, otherwise it will return false.

# Datatypes

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.

# Datatypes

- **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.

# Insert Document

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

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

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

Example

```
>db.mycollection.insert({
   _id: ObjectId(7df78ad8902c),
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
})
```

# Explanation

- Here mycol is our collection name, as created in the previous chapter. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.
- In the inserted document, if we don't specify the _id parameter, then MongoDB assigns a unique ObjectId for this document.
- _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)
- To insert multiple documents in a single query, you can pass an array of documents in insert() command.

# Example

```
>db.mycollection.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 }
    ] }
])
```

# Query Document

- The find() Method

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

  The basic syntax of find() method is as follows −

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

- The pretty() Method

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

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

# RDBMS Clause Equivalents 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 in MongoDB

In the find() method, if you pass multiple keys by separating them by ',' then MongoDB treats it as AND condition. Following is the basic syntax of AND −

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

Example:

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

```
>db.mycol.find({$and:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty() {
  "_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"
}
```

## ➢ OR in MongoDB

To query documents based on the OR condition, you need to use $or keyword. Following is the basic syntax of OR −

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

Example:

Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

```
>db.mycol.find({$or:[{“title":" MongoDB Overview "},{"title": "MongoDB Overview"}]}).pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "description": "MongoDB is no sql database",
   "url": "http://www.tutorialspoint.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
```

## ➢ Using AND and OR Together

**Example**

**The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is 'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'**

```
>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},
  {"title": "MongoDB Overview"}]}).pretty()
{
  "_id": ObjectId(7df78ad8902c),
  "title": "MongoDB Overview",
  "description": "MongoDB is no sql database",
  "tags": ["mongodb", "database", "NoSQL"],
  "likes": "100"
}
```

# Update Document

- MongoDB Update() Method

  MongoDB's **update()** and **save()** methods are used to update document into a collection. The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

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

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

  ```
  >db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)
  ```

Consider the mycol collection has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'.

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

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

# MongoDB Save() Method

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

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

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

Following example will replace the document with the _id '5983548781331adf45ec5'.

```
>db.mycol.save(
  {
    "_id" : ObjectId(5983548781331adf45ec5), "title":"Tutorials Point New Topic",
     "by":"Tutorials Point"
  }
)
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"Tutorials Point New Topic",
  "by":"Tutorials Point"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

# Delete Document

- The 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.

  Basic syntax of remove() method is as follows −

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

Consider the mycol collection has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will remove all the documents whose title is 'MongoDB Overview'.

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

# Projection

- The find() Method

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.

MongoDB's find() method, explained in MongoDB Query Document accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB, when you execute find() method, then it displays all fields of a document. To limit this, you need to set a list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the fields.

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

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

Consider the collection mycol has the following data −

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will display the title of the document while querying the document.

```
>db.mycol.find({},{"title":1,_id:0})
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Tutorials Point Overview"}
```

Please note _id field is always displayed while executing find() method, if you don't want this field, then you need to set it as 0.

# Limit Records

- The 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.

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

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

Consider the collection myycol has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

- 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.

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

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

Following example will display only the second document.

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

Please note, the default value in skip() method is 0.

# Sort Records

- The 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.

  The basic syntax of sort() method is as follows −

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

Consider the collection myycol has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will display the documents sorted by title in the descending order.

```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Tutorials Point Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

Please note, if you don't specify the sorting preference, then sort() method will display the documents in ascending order.

# MongoDB PHP driver

To use MongoDB with PHP, you need to use MongoDB PHP driver. Download the driver from the url Download PHP Driver. Make sure to download the latest release of it. Now unzip the archive and put php_mongo.dll in your PHP extension directory ("ext" by default) and add the following line to your php.ini file –

```
extension = php_mongo.dll
```

# Make a Connection and Select a Database

- To make a connection, you need to specify the database name, if the database doesn't exist then MongoDB creates it automatically.

Following is the code snippet to connect to the database −

```
<?php
// connect to mongodb
$m = new MongoClient();

echo "Connection to database successfully";
// select a database
$db = $m->mydb;

 echo "Database mydb selected";
?>
```

# Create a Collection

Following is the code to create a collection −

```
<?php
// connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully";

// select a database
$db = $m->mydb;
echo "Database mydb selected";
$collection = $db->createCollection("mycol");
echo "Collection created succsessfully";
?>
```

# Insert a Document

To insert a document into MongoDB, insert() method is used.

Following is the code snippet to insert a document −

```
<?php
   // connect to mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";

   // select a database
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";
```

```
$document = array(
    "title" => "MongoDB",
    "description" => "database",
    "likes" => 100,
    "url" => "http://www.tutorialspoint.com/mongodb/",
    "by" => "tutorials point"
  );

  $collection->insert($document);
  echo "Document inserted successfully";
?>
```

# Find Documents

To select all documents from the collection, find() method is used.

Following is the code snippet to select all documents −

```
<?php
   // connect to mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";

   // select a database
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";
```

```
$cursor = $collection->find();
  // iterate cursor to display title of documents

  foreach ($cursor as $document) {
    echo $document["title"] . "\n";
  }
?>
```

# Update a Document

To update a document, you need to use the update() method.

In the following example, we will update the title of inserted document to MongoDB Tutorial. Following is the code snippet to update a document −

```
<?php
   // connect to mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";

   // select a database
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";
```

```
// now update the document
  $collection->update(array("title"=>"MongoDB"),
    array('$set'=>array("title"=>"MongoDB Updated")));
  echo "Document updated successfully";

  // now display the updated document
  $cursor = $collection->find();

  // iterate cursor to display title of documents
  echo "Updated document";

  foreach ($cursor as $document) {
    echo $document["title"] . "\n";
  }
?>
```

# Delete a Document

To delete a document, you need to use remove() method.

In the following example, we will remove the documents that has the title MongoDB Tutorial. Following is the code snippet to delete a document −

```
<?php
   // connect to mongodb
   $m = new MongoClient();
   echo "Connection to database successfully";

   // select a database
   $db = $m->mydb;
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";
```

```
// now remove the document
  $collection->remove(array("title"=>"MongoDB Tutorial"),false);
  echo "Documents deleted successfully";

  // now display the available documents
  $cursor = $collection->find();

  // iterate cursor to display title of documents
  echo "Updated document";

  foreach ($cursor as $document) {
    echo $document["title"] . "\n";
  }
?>
```