



mongoDB

```
{ name: mongo, type: DB }
```

```
{  
  Knowledge Sharing: "MongoDB",  
  BY: "Mg Kyaing(PID department)"  
}
```

Agenda

- ▶ What is NoSQL?
- ▶ SQL vs NoSQL: High-level Differences.
- ▶ NoSQL: 4 Type of NoSQL Databases.
- ▶ What is MongoDB?
- ▶ Advantages of MongoDB over RDMS?
- ▶ Why and Where should use MongoDB?
- ▶ Compare SQL Schema Statements and MongoDB Schema Statements
- ▶ CRUD Operations with MongoDB.
- ▶ Shell and Command Helper in MongoDB.
- ▶ reference websites and gui Tool for mongoDB Download Links



What is NoSQL?

- Schema-less
- Capability to handle semi-structured, and unstructured data
- No need of ORM
- Agility
- Ease of scale-out
- High Availability



► SQL vs NoSQL: High-Level Differences

- ✓ SQL databases are primarily called as Relational Databases (RDBMS); whereas NoSQL database are primarily called as non-relational or distributed database.
- ✓ SQL databases are table based databases whereas NoSQL databases are document based, key-value pairs, graph databases or wide-column stores.
- ✓ SQL databases have predefined schema whereas NoSQL databases have dynamic schema for unstructured data.
- ✓ SQL databases are vertically scalable whereas the NoSQL databases are horizontally scalable.
- ✓ SQL databases are scaled by increasing the horse-power of the hardware. NoSQL databases are scaled by increasing the databases servers in the pool of resources to reduce the load.
- ✓ SQL databases uses SQL (structured query language) for defining and manipulating the data, which is very powerful. In NoSQL database, queries are focused on collection of documents. Sometimes it is also called as UnQL (Unstructured Query Language). The syntax of using UnQL varies from database to database.

► SQL vs NoSQL: High-Level Differences

- ✓ For complex queries: SQL databases are good fit for the complex query intensive environment whereas NoSQL databases are not good fit for complex queries.
- ✓ For the type of data to be stored: SQL databases are not best fit for hierarchical data storage. But, NoSQL database fits better for the hierarchical data storage as it follows the key-value pair way of storing data similar to JSON data. NoSQL database are highly preferred for large data set (i.e for big data).
- ✓ For scalability: In most typical situations, SQL databases are vertically scalable. You can manage increasing load by increasing the CPU, RAM, SSD, etc, on a single server. On the other hand, NoSQL databases are horizontally scalable.
- ✓ For properties: SQL databases emphasizes on ACID properties (Atomicity, Consistency, Isolation and Durability) whereas the NoSQL database follows the Brewers CAP theorem (Consistency, Availability and Partition tolerance)







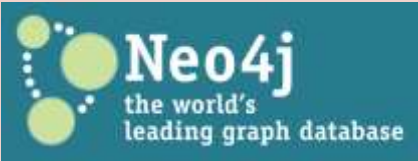


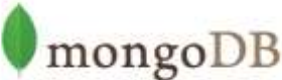



NOSQL:4 TYPE OF NOSQL DATABASES

- Key-Value pair
- Documents
- Column family
- Graph



AVAILABLE DBMS

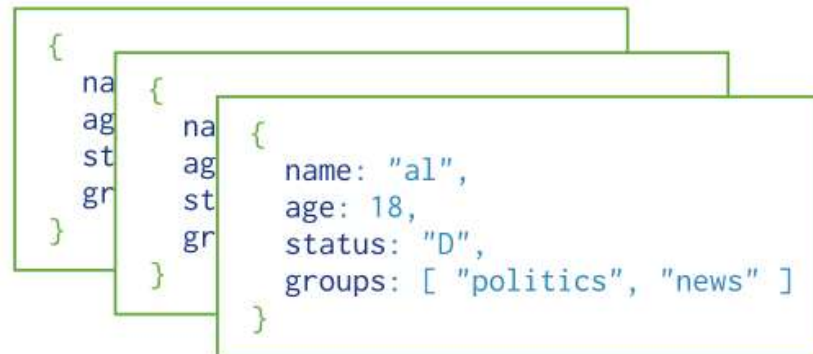
Key/ Value	Column - Family	Graph	Document
 redis  riak 	  Cassandra  HYPERTABLE	  	 

- ▶ What is MongoDB?
- ▶ MongoDB (from "*humongous*") is a document-based open-source database. MongoDB is a leading NoSQL database. It provides high performance, high availability, auto sharding and automatic scaling. MongoDB database is written mainly in C++ . It's drivers and client libraries are typically written in their respective languages, although some drivers use C extensions for better performance. MongoDB differs from relational databases because it is a schema-less database. Such databases usually have lower transaction safety but are faster in accessing data and scale better than relational databases.

History of MongoDB

MongoDB was developed in October 2007 by a New York based organization 10gen that is now called MongoDB Inc. It is written in C , C++ and JavaScript. MongoDB is developed as a Platform as a Service (PAAS) initially. In 2009 MongoDB was introduced as an open-source database server. MongoDB provides backend services to many websites and to other services. For example, eBay, Foursquare , Viacom, Craigslist and the New York Times.

MongoDB stores **BSON documents**, i.e. data records, in **collections**; the collections in databases.



Collection



▶ Advantages of MongoDB over RDBMS

- ▶ Schema less : MongoDB is document database in which one collection holds different different documents. Number of fields, content and size of the document can be 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



- ▶ **Why should use MongoDB?**
- ▶ Document Oriented Storage : Data is stored in the form of JSON style documents
- ▶ Index on any attribute
- ▶ Replication & High Availability
- ▶ Auto-Sharding
- ▶ Rich Queries
- ▶ Fast In-Place Updates
- ▶ Professional Support By MongoDB
- ▶ **Where should use MongoDB?**
- ▶ Big Data
- ▶ Content Management and Delivery
- ▶ Mobile and Social Infrastructure
- ▶ User Data Management
- ▶ Data Hub



The following table presents the various *SQL terminology and concepts* and the corresponding *MongoDB terminology and concepts*

SQL Terms/Concepts	MongoDB Terms/Concepts
Database	Database
Table	Collection
Row	document or BSON document
Column	Field
Index	Index
Table joins	Embedded documents and linking
Primary key	Primary key
Specify any unique column or column combination as primary key	The primary key is automatically set to the _id field.

DataBase Server and Client between MySQL,Oracle,Informax,DB2

	MongoDB	MySQL	Oracle	Informax	DB2
Database Server	Mongod	Mysqld	Oracle	IDS	DB2 Server
Database Client	Mongo	Mysql	Sqlplus	DB-Access	DB2 Client

SQL Schema Statements	MongoDB Schema Statements
CREATE TABLE users (id MEDIUMINT NOT NULL AUTO_INCREMENT, user_id Varchar(30), age Number, status char(1), PRIMARY KEY (id))	Implicitly created on first <u>insert()</u> operation. The primary key <u>_id</u> is automatically added if <u>_id</u> field is not specified. db.users.insert({ user_id: "abc123", age: 55, status: "A" }) ; However, you can also explicitly create a collection: db.createCollection("users") ;
ALTER TABLE users ADD join_date DATETIME	Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level. However, at the document level, <u>update()</u> operations can add fields to existing documents using the <u>\$set</u> operator. db.users.update({ }, { \$set: { join_date: new Date() } }, { multi: true })
ALTER TABLE users DROP COLUMN join_date	Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level. However, at the document level, <u>update()</u> operations can remove fields from documents using the <u>\$unset</u> operator. db.users.update({ }, { \$unset: { join_date: "" } }, { multi: true })
CREATE INDEX idx_user_id_asc ON users(user_id)	db.users.createIndex({ user_id: 1 })
CREATE INDEX idx_user_id_asc_age_desc ON users(user_id, age DESC)	db.users.createIndex({ user_id: 1, age: -1 })
DROP TABLE users	db.users.drop()



SQL Insert Statement	MongoDB Insert Statement
INSERT INTO users(user_id, age, status) VALUES ("bcd001", 45, "A")	db.users.insert({ user_id: "bcd001", age: 45, status: "A" })

SQL SELECT Statements	MongoDB Find() Statements
SELECT * FROM users;	db.users.find();
SELECT id, user_id, status FROM users;	b.users.find({ }, { user_id: 1, status: 1 }) ;
SELECT user_id, status FROM users;	db.users.find({ }, { user_id: 1, status: 1, _id: 0 });
SELECT * FROM users WHERE status = "A"	db.users.find({ status: "A" })
SELECT user_id, status FROM users WHERE status = "A"	db.users.find({ status: "A" }, { user_id: 1, status: 1, _id: 0 })
SELECT * FROM users WHERE status != "A"	db.users.find({ status: { \$ne: "A" } })
SELECT * FROM users WHERE status = "A" AND age = 50	db.users.find({ status: "A", age: 50 })
SELECT * FROM users WHERE status = "A" OR age = 50	db.users.find({ \$or: [{ status: "A" } , { age: 50 }] })
SELECT * FROM users WHERE age > 25	db.users.find({ age: { \$gt: 25 } })
SELECT * FROM users WHERE age < 25	db.users.find({ age: { \$lt: 25 } })



SQL SELECT Statements	MongoDB find() statements
SELECT * FROM users WHERE age > 25 AND age <= 50	db.users.find({ age: { \$gt: 25, \$lte: 50 } })
SELECT * FROM users WHERE user_id like "%bc%"	db.users.find({ user_id: /bc/ })
SELECT * FROM users WHERE user_id like "bc%"	db.users.find({ user_id: /^bc/ })
SELECT * FROM users WHERE status = "A" ORDER BY user_id ASC	db.users.find({ status: "A" }).sort({ user_id: 1 })
SELECT * FROM users WHERE status = "A" ORDER BY user_id DESC	db.users.find({ status: "A" }).sort({ user_id: -1 })
SELECT COUNT(*) FROM users	db.users.count() <i>or</i> db.users.find().count()
SELECT COUNT(user_id) FROM users	db.users.count({ user_id: { \$exists: true } }) <i>or</i> db.users.find({ user_id: { \$exists: true } }).count()
SELECT COUNT(*) FROM users WHERE age > 30	db.users.count({ age: { \$gt: 30 } }) <i>or</i> db.users.find({ age: { \$gt: 30 } }).count()



SQL SELECT Statements	MongoDB find Statements
SELECT DISTINCT(status) FROM users	db.users.distinct("status")
SELECT * FROM users LIMIT 1	db.users.findOne() <i>or</i> db.users.find().limit(1)
SELECT * FROM users LIMIT 5 SKIP 10	db.users.find().limit(5).skip(10)
EXPLAIN SELECT * FROM users WHERE status = "A"	db.users.find({ status: "A" }).explain()

SQL UPDATE Statements	MongoDB Update Statements
UPDATE users SET status = "C" WHERE age > 25	db.users.update({ age: { \$gt: 25 } }, { \$set: { status: "C" } }, { multi: true })
UPDATE users SET age = age + 3 WHERE status = "A"	db.users.update({ status: "A" } , { \$inc: { age: 3 } }, { multi: true })
UPDATE users SET status="D",age=10 WHERE status="A"	db.users.update({status:"A"},{\$set:{status:"D",age:10}});
SQL DELETE Statements	MongoDB remove Statements
DELETE FROM users WHERE status = "D"	db.users.remove({ status: "D" })
DELETE FROM users	db.users.remove({})

BSON supports the following data types as values in documents.

Each data type has a corresponding number and string alias that can be used with the `$type` operator to query documents by BSON type.

Type	Number	Alias	Notes
Double	1	“double”	
String	2	“String”	
Object	3	“object”	
Array	4	“array”	
Binary data	5	“binData”	
Undefined	6	“undefined”	Deprecated
Objectid	7	“objected”	
Boolean	8	“bool”	
Date	9	“date”	
Null	10	“null”	
Regular Expression	11	“regx”	
DBPointer	12	“dbPinter”	
JavaScript	13	“javascript”	
Symbol	14	“symbol”	
JavaScript(with scope)	15	“javascriptWithScope”	
32-bit integer	16	“int”	
Timestamp	17	“timestamp”	
64-bit integer	18	“long”	
Min Key	-1	“minkey”	
Max Key	127	“maxkey”	



In MongoDB, insert operations target a single **collection**. All write operations in MongoDB are **atomic** on the level of a single **document**.

```
db.users.insert (  ← collection
  {
    name: "sue",    ← field: value
    age: 26,        ← field: value
    status: "A"     ← field: value
  }                } document
)
```

Read operations **documents** from a **collection**; i.e. queries a collection for documents. MongoDB provides the following methods to read documents from a collection:

- `db.collection.find()`

You can specify **query filters** or **criteria** that identify the documents to return.

```
db.users.find(                                ← collection
  { age: { $gt: 18 } },                       ← query criteria
  { name: 1, address: 1 }                     ← projection
).limit(5)                                    ← cursor modifier
```

In MongoDB, update operations target a single collection. All write operations in MongoDB are **atomic** on the level of a single document.

You can specify criteria, or filters, that identify the documents to update. These **filters** use the same syntax as read operations.

```
db.users.update(                               ← collection
  { age: { $gt: 18 } },                       ← update criteria
  { $set: { status: "A" } },                  ← update action
  { multi: true }                             ← update option
)
```

In MongoDB, delete operations target a single **collection**. All write operations in MongoDB are **atomic** on the level of a single document.

You can specify criteria, or filters, that identify the documents to remove. These **filters** use the same syntax as read operations.

```
db.users.remove(                               ← collection
  { status: "D" }                             ← remove criteria
)
```



Access the mongo Shell And **Command Helpers**

mongoDB Shell Methods	
db.collection.aggregate()	Provides access to the aggregation pipeline
db.collection.bulkWrite()	Provides bulk write operation functionality.
db.collection.count()	Wraps count to return a count of the number of documents in a collection or matching a query.
db.collection.copyTo()	Deprecated. Wraps eval to copy data between collections in a single MongoDB instance
db.collection.createIndex()	Builds an index on a collection
db.collection.dataSize()	Returns the size of the collection. Wraps the size field in the output of the collStats
db.colletion.deleteOne()	Deletes a single document in a collection
db.collection.deleteMany()	Deletes multiple documents in a collection.
db.collection.distinct()	Returns an array of documents that have distinct values for the specified field
db.collection.drop()	Removes the specified collection from the database
db.collection.dropIndex()	Removes a specified index on a collection
db.collection.dropIndexes()	Removes all indexes on a collection
And much and more	

Command Helpers	
help	Show Help
db.help()	Show help for database method
db.collection.help()	Show help on collection methods
Show dbs	Print a list of all databases on Server
Use <db>	Switch current database
Show collections	Print a list of all collection for current db
Show users	Print a list of all users for current db.
Show roles	Print a list of all roles,both user-defined and built-in for current db
Show profile	Print the five most recent operations that took 1 millisecond or more
Show.databases	Print a list of all available databases
db.Auth()	If running in secure mode, authenticate the user
db.cloneDatabase<host>	Clone the current database from the <host> specified. The <host> database instance must be in noauth mode.
db.repaireDatabase()	Repair and compact the current database. This operation can be very slow on large databases
db.copyDatabase(<from>, <to>, <host>)	Copy the <from> database from the <host> to the <to> database on the current server
And much and more	

<https://docs.mongodb.com/manual/reference/method/>

Let's see CRUD,find,Sort Operations with me 😊

Use EmployeeDB;//Create the databae(EmployeeDB) using “use” command

```
db.Employee.insert({  
  Employeeid:1,  
  EmployeeName:"Jone Smith",  
  PhoneNo:{ HandPhone:099562365412,HomePhone:0126541230,OfficePhone:01652315}  
});//Create a collection(Employee) using insert() command
```

```
Var objemp=[{  
  Employeeid:2,  
  EmployeeName:"Marry"},  
  {  
    Employeeid:3,  
    EmployeeName:"Steave:"},  
  {Employeeid:4  
    EmployeeName:"akog02"}];  
db.Employee.insert(objemp);//Adding an array of documents using insert() command
```



```
db.Employee.find().forEach(printjson); or db.Employee.find().pretty();//show the EmployeeCollection.
```

```
=====
```

```
db.Employee.find({}); //show all of the EmployeeCollection.
```

```
=====
```

```
db.Employee.find( {EmployeeName:"smith"} ).foreach(printjson); //show by EmployeeName is Smith.
```

```
=====
```

```
db.Employee.find( {Employeeid: { $gt:2 } } ).forEach(printjson); //show by Employeeid is greater than 2.
```

```
=====
```

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

```
while(myEmployee.hasNext())
```

```
{
```

```
  print(tojson(myCursor.next()));
```

```
}
```

```
=====
```

```
db.Employee.update(
```

```
  {Employeeid:1} {$set: {EmployeeName:"David Jone"}}
```

```
); //Update Document using update()
```

```
=====
```

```
db.Employee.update(
```

```
  {Employeeid:2}, {$set: {EmployeeName:"Mory",Employeeid:33}}
```

```
); //Updating Multiple Values using update()
```




```
db.Employee.find().limit(2).forEach(printjson);
db.Empooyee.find().sort({Employeeid:-1}).forEach(printjson);//Sort by the descending order of Employee id
db.Employee.count();
db.Employee.remove({Employeeid:1});
db.Employee.find(
$or:[{Employeeid:2},{EmployeeName:"Jone Smith"}
]);
});//OR Operator
db.Employee.find(
{
Employeeid:1,EmployeeName:"Jone Smith"
});// AND Operator
db.Employee.find({},Employeeid:1);//only show the Employee id BSON DOC
db.Employee.find({Employeeid /2/});// same LIKE Operator in RDBMS
db.Employee.find().limit(10);
db.Employee.find().skip(2).limit(1);
// specify the search criteria
db.Employee.find({EmployeeName:{ $regex:"jon"}}).forEach(printjson); or
db.Employee.find({EmployeeName:{ $regex:"jon"}}).pretty();
//specify the extrace text match
db.Employee.find({EmployeeName:{ $regex:"^jon$"}}).pretty();
```



reference website:

<https://docs.mongodb.com/manual/>

<http://www.guru99.com/mongodb-tutorials.html>

<http://www.tutorialspoint.com>

gui tool for mongodb

<http://www.mongovue.com/>(type:commercial,platform:windows only)

<http://mongobooster.com>

<https://robomongo.org/>(type:free,platform:windows,linux,mac)

<http://www.mongodbmanager.com>

<https://github.com/bububa/mongohub>(type:free,platform:mac)

<https://github.com/iwind/rockmongo>(type:browserbased)

<http://edgytech.com/umongo/>(type:free,platform:windows,linux,mac osx)

<http://edgytech.com/umongo/>(type:commercial,platform:android)

<http://3t.io/mongochef/download/core/platform/>(type:free,commercial,platform:windows,mac,linux)



Next Topic>>MongoDB Replication and Why Replication?

