What is MongoDB?

MongoDB is a document-oriented NoSQL database used for high volume data storage.

Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents.

Documents consist of key-value pairs which are the basic unit of data in MongoDB.

Collections contain sets of documents and function which is the equivalent of relational database tables.

MongoDB is a database which came into light around the mid-2000s.

MongoDB is

- Cross-platform
- Open source
- Non-relational
- Distributed
- NoSQL
- Document Oriented data store

Why MongoDB?

- Auto Sharding
- Document Oriented
- High Performance
- Fast in-place updates
- Replication
- Easy Scalability
- High availability

Sharding

Sharding is akin to horizontal scaling. It means that the large dataset is divided and distributed over multiple servers or shards.

Each shard is an independent database and collectively they would constitute a logic database.

The prime advantages of sharding are as follows:

1. Sharding reduces the amount of data that each shard needs to store and manage. For example, if the dataset was 1 TB in size and we were to distribute this over four shards, each shard would house just 256 GB data. Refer Figure 6.4. As the cluster grows, the amount of data that each shard will store and manage will decrease.
2. Sharding reduces the number of operations that each shard handles. For example, if we were to insert data, the application needs to access only that shard which houses that data.

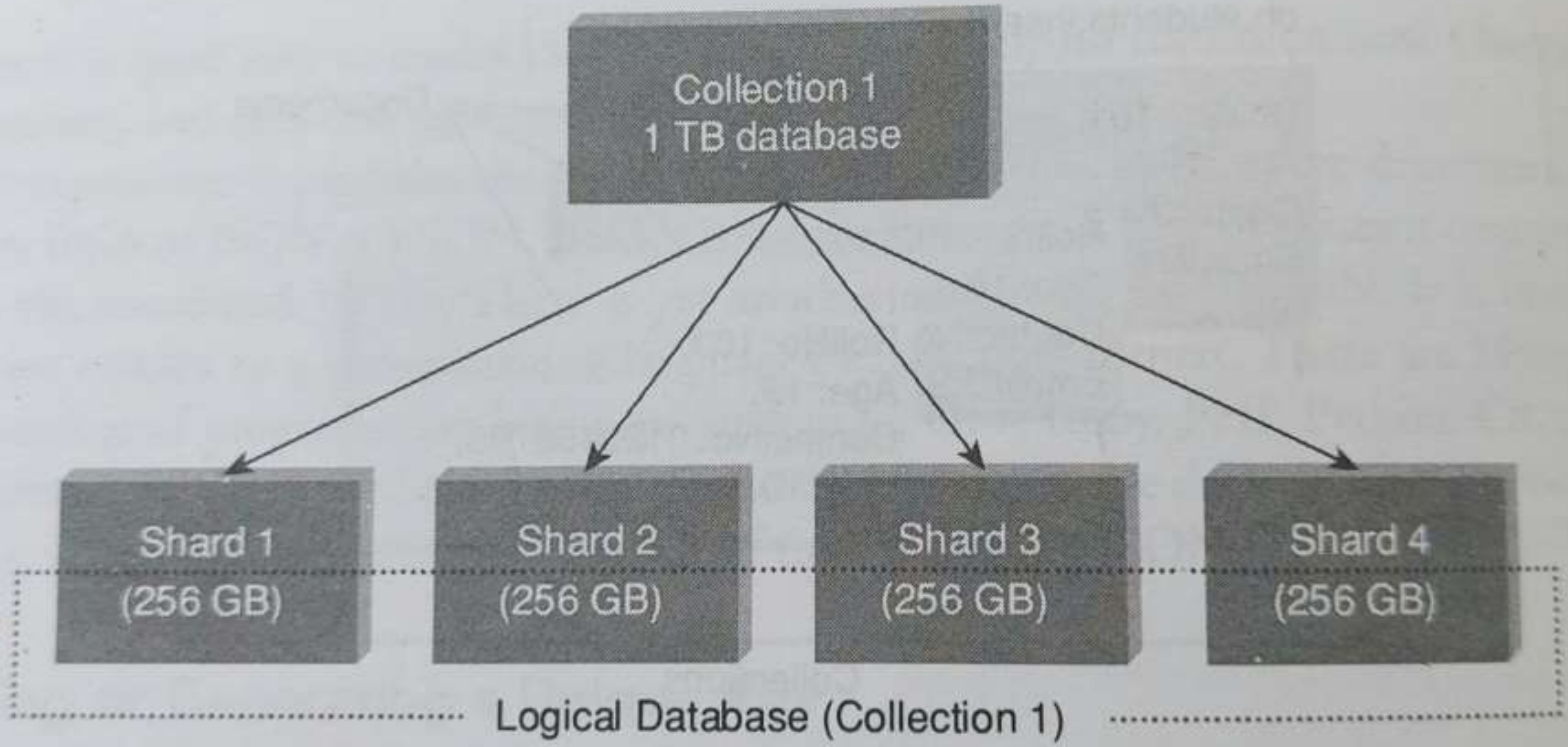

Figure 6.4 The process of **SHARDING** in MongoDB.

Replication

It provides data redundancy and high availability.
It helps to recover from hardware failure and service interruptions.
In MongoDB, the replica set has a single primary and several secondaries.
Each write request from the client is directed to the primary. The primary logs all write requests into its Oplog (operations log).
The Oplog is then used by the secondary replica members to synchronize their data.
This way there is strict adherence to consistency.
Refer Figure 6.3. The clients usually read from the primary.
However, the client can also specify a read preference that will then direct the read operations to the secondary.

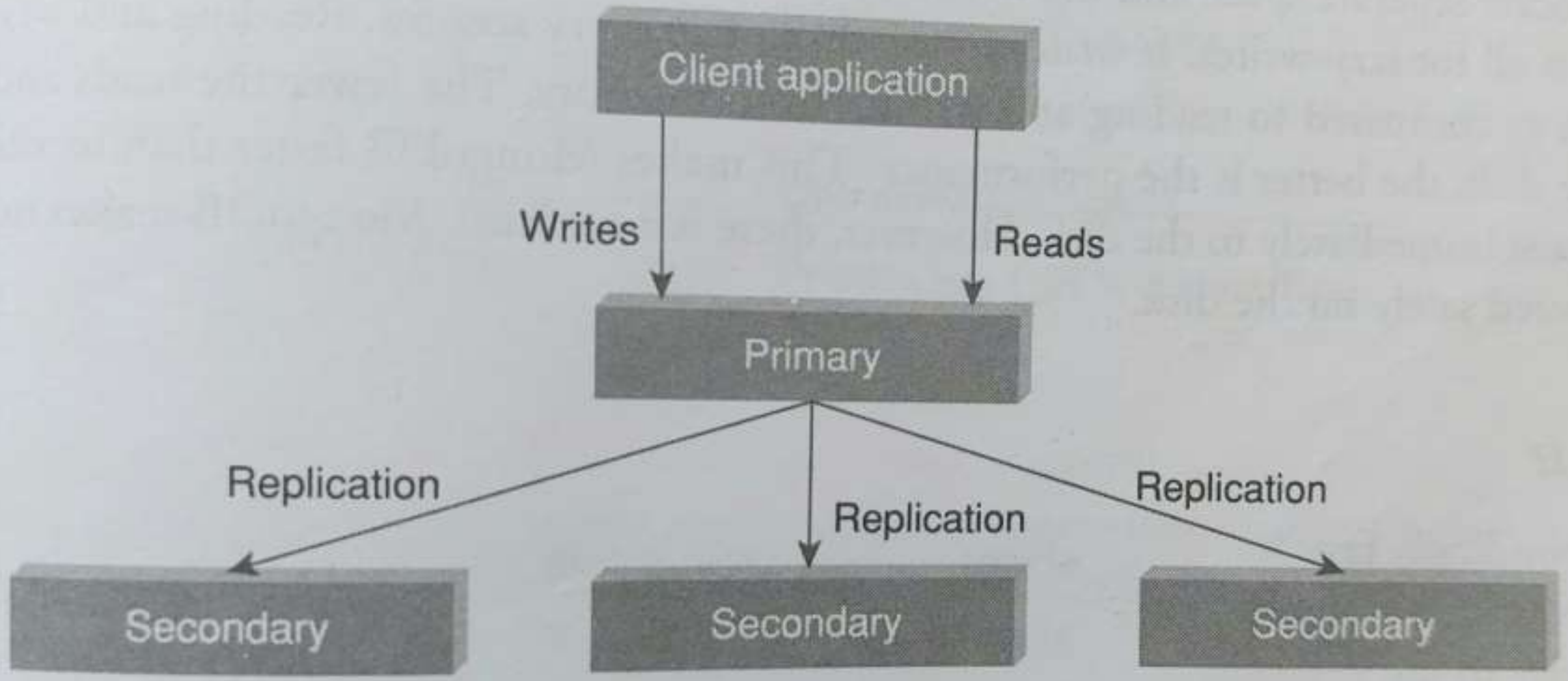

Figure 6.3 The process of **REPLICATION** in MongoDB.

Updating Information In-Place

MongoDB updates the information in-place. This implies that it updates the data wherever it is available It does not allocate separate space and the indexes remain unaltered.

MongoDB is all for lazy-writes. It writes to the disk once every second. Reading and writing to disk s slow operation as compared to reading and writing from memory.

The fewer the reads and writes that perform to the disk, the better is the performance. This makes MongoDB faster than its other competition who write almost immediately to the disk.

However, there is a tradeoff. MongoDB makes no guarantee the data will be stored safely on the disk.

Terms used in RDBMS and MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Record	Document
Columns	Fields/ Key value pair
Index	Index
Joins	Embedded documents

MongoDB database Server is known as **Mongod** and Database client is known as **mongo.**

Basic commands on mongo shell

Show all available databases:

```
show dbs;
```

Select a particular database to access, e.g. mydb. This will create mydb if it does not already exist:

```
use mydb;
```

Show all collections in the database (be sure to select one first, see above):

```
show collections;
```

Show all functions that can be used with the database:
db.mydb.help();

To check your currently selected database, use the command db
> **db**
mydb

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

CRUD Operation:

CREATE

db.people.insert({name: 'Tom', age: 28});

Or

db.people.save({name: 'Tom', age: 28});

The difference with save is that if the passed document contains an _id field, if a document already exists with that _id it will be updated instead of being added as new.

Two new methods to insert documents into a collection, in MongoDB 3.2.x:

Use insertOne to insert only one record:

db.people.insertOne({name: 'Tom', age: 28});

Use insertMany to insert multiple records:

db.people.insertMany([{name: 'Tom', age: 28},{name: 'John', age: 25}, {name: 'Kathy', age: 23}])

UPDATE

db.people.update({name: 'Tom'}, {age: 29, name: 'Tom'})

// New in MongoDB 3.2
db.people.updateOne({name: 'Tom'},{age: 29, name: 'Tom'}) //Will replace only first matching document.

db.people.updateMany({name: 'Tom'},{age: 29, name: 'Tom'})
//Will replace all matching documents.

Or just update a single field of a document. In this case age:
db.people.update({name: 'Tom'}, {$set: {age: 29}})

You can also update multiple documents simultaneously by adding a third parameter.
This query will update all documents where the name equals Tom:
db.people.update({name: 'Tom'}, {$set: {age: 29}}, {multi: true})

// New in MongoDB 3.2
db.people.updateOne({name: 'Tom'},{$set:{age: 30}}) //Will update only first matching document.

db.people.updateMany({name: 'Tom'},{$set:{age: 30}}) //Will update all matching documents.

If a new field is coming for update, that field will be added to the document

DELETE
Deletes all documents matching the query parameter:

// New in MongoDB 3.2
db.people.deleteMany({name: 'Tom'})

// All versions
db.people.remove({name: 'Tom'})

Or just one
// New in MongoDB 3.2
db.people.deleteOne({name: 'Tom'})
// All versions
db.people.remove({name: 'Tom'}, true)

MongoDB's remove() method. If you execute this command without any argument or without empty argument it will remove all documents from the collection. **db.people.remove(); or db.people.remove({});**

READ

Query for all the docs in the people collection that have a name field with a value of 'Tom':
db.people.find({name: 'Tom'})

Or just the first one:
db.people.findOne({name: 'Tom'})

You can also specify which fields to return by passing a field selection parameter. The following will exclude the _id field and only include the age field:
db.people.find({name: 'Tom'}, {_id: 0, age: 1})

By default, the _id field will be returned, even if you don't ask for it. If you would like not to get the _id back, you can just follow the previous example and ask for the _id to be excluded by specifying **_id: 0** (or **_id: false).**

If you want to find sub record like address object contains country, city, etc.

db.people.find({'address.country': 'US'})

& specify field too if required
db.people.find({'address.country': 'US'}, {'name': true, 'address.city': true})

Remember that the result has a `.pretty()` method that pretty-prints resulting JSON: **db.people.find().pretty()**

UPSERT

```
db.people.update({name: 'Raj', age:30},{upsert: true});
```

```
//Also used to add new field to a document using $set
db.people.update({name:'Rahul',age:30},{$set:{City:"Nadiad"}},
{upsert: true});
```

Use unset to unset any attribute value

```
db.people.update({name: 'Rahul', age:30},{$unset:{City:"Nadiad"}});
```

mongoDB MapReduce

```
//to create new db
use mymapreddb;
```

//add documents to the collection named **customer**

```
db.customers.insert([{CustID:"C123",AccBal:500,AccType:"S"},{CustID:"C123",AccBal:800,AccType:"S"},{CustID:"C111",AccBal:1200,AccType:"S"},{CustID:"C123",AccBal:1500,AccType:"C"}]);
```

```
//to display the content of collection
db.customers.find().pretty();
```

```
//create a map function
var map = function(){
... emit(this.CustID,this.AccBal);}

//create a reduce function
 var reduce = function(key,values){
... return Array.sum(values);}
```

//Use **mapReduce() function** to use mapreduce functionality

```
db.customers.mapReduce(map,reduce,{out:"Customer_Totals",quer
y:{AccType:"S"}});
```

//To display the output which is stored in **Customer_Totals** collection

```
db.Customer_Totals.find().pretty();
```