Sunbeam Infotech

Mongo Db

RDBMS

- Structured and organized data
- Structured query language (SQL)
- DML, DQL, DDL, DTL, DCL.
- Data and its relationships are stored in separate tables.
- Tight Consistency
- Based on Codd's rules
- ACID transactions.
  - Atomic
  - Consistent
  - Isolated
  - Durable

NoSQL

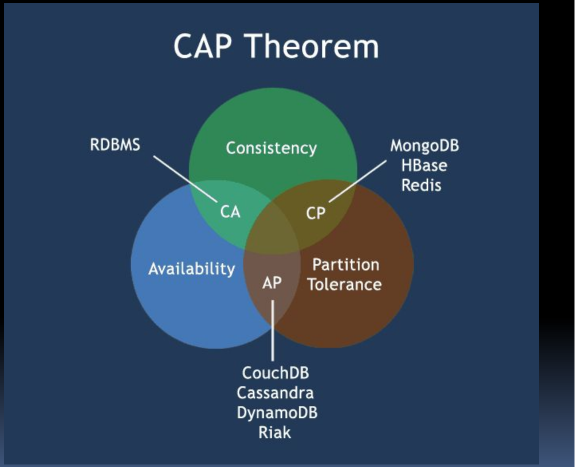
- Stands for Not Only SQL
- No declarative query language
- No predefined schema, Unstructured and unpredictable data
- Eventual consistency rather ACID property
- Based on CAP Theorem
- Prioritizes high performance, high availability and scalability
- BASE Transaction
  - Basically Available
  - Soft state
  - Eventual consistency

Scaling

- Scalability is the ability of a system to expand to meet your business needs.
- E.g. scaling a web app is to allow more people to use your application.
- Vertical scaling: Add resources within the same logical unit to increase capacity. E.g. add CPUs to an existing server, increase memory in the system or expanding storage by adding hard drives.
- Horizontal scaling: Add more nodes to a system. E.g. adding a new computer to a distributed software application. Based on principle of distributed computing.
- NoSQL databases are designed for Horizontal scaling. So they are reliable, fault tolerant, better performance (at lower cost), speed.

CAP Theorem

- **Consistency** - Data is consistent after operation. After an update operation, all clients see the same data.
- **Availability** - System is always on (i.e. service guarantee), no downtime.
- **Partition Tolerance** - System continues to function even the communication among the servers is unreliable.



NoSQL scenarios

- When to use NoSQL?
  - Large amount of data (TBs)
  - Many Read/Write ops
  - Economical Scaling
  - Flexible schema
- Examples:
  - Social media
  - Recordings
  - Geospatial analysis
  - Information processing

- When Not to use NoSQL?
  - Need ACID transactions
  - Fixed multiple relations
  - Need joins
  - Need high consistency
- Examples
  - Financial transactions
  - Business operations

NoSQL Advantages/Problems

- Advantages:
  - High scalability
  - Distributed computing
  - Lower cost
  - Flexible schema/Semi structured
  - No complex relationships
- Disadvantages:
  - No standardization
  - Limited query support (work in progress).
  - Eventual consistency

NoSQL Categories

- Key-value databases - e.g. redis, dynamodb, riak, ...
  - Based on Amazon's Dynamo database.
  - For handling huge data of any type.
  - Keys are unique and values can be of any type i.e. JSON, BLOB, etc.
  - Implemented as big distributed hash-table for fast searching.

- Column-oriented databases - e.g. hbase, cassandra, bigtable, ...
  - Values of columns are stored contiguously.
  - Better performance while accessing few columns and aggregations.
  - Good for data-warehousing, business intelligence, CRM, ...

NoSQL Categories

- Graph databases - e.g. Neo4J, Titan, …
  - Graph is collection of vertices and edges (lines connecting vertices).
  - Vertices keep data, while edges represent relationships.
  - Each node knows its adjacent nodes. Very good performance, when want to access all relations of an entity (irrespective of size of data).
- Document oriented databases - e.g. MongoDb, CouchDb, …
  - Document contains data as key-value pair as JSON or XML.
  - Document schema is flexible & are added in collection for processing.
  - RDBMS tables → Collections
  - RDBMS rows → Documents
  - RDBMS columns → Key-value pairs in document

MongoDb

- Developed by 10gen in 2007. Publically available in 2009.
- Open-source database (github.com) - controlled by 10gen.
- Document oriented database → stores JSON documents.

```
{   "name": "Nilesh Ghule", "age": 34, "salary": 30000.00,
    "permanent": true, hiredate: ISODate("2004-05-31"),
    "skills": [ "Java", "OS", "Hadoop", "C", "C++", "Java EE", "ARM" ],
    "contact" : { "email" : "nilesh@sunbeaminfo.com",
        "mobile" : "9527331338"
}   }
```

MongoDb: Data Types

- In MongoDb, JSON data is stored in binary i.e. BSON.

null	10	
boolean	8	true, false
number	1 / 16 / 18	123, 456.78, NumberInt("24"), NumberLong("28")
string	2	"…."
date	9	new Date(), ISODate("yyyy-mm-ddThh:mm:ss")
array	4	[…, …, …, …]
object	3	{ … }

MongoDb Server & Client

- MongoDb server (mongod) is developed in C, C++ and JS.
- MongoDb data is accessed via multiple client tools
  - mongo : client shell (JS).
  - mongofiles : stores larger files in GridFS.
  - mongoimport / mongoexport : tools for data import / export.
  - mongodump / mongorestore : tools for backup / restore.
- MongoDb data can be accessed in application through client drivers available for all major programming languages e.g. Java, Python, Ruby, PHP, Perl, …
- Mongo shell is follows JS syntax and allow to execute JS scripts.

Mongo - INSERT

- show databases;
- use database;
- db.contacts.insert({name: "nilesh", mobile: "9527331338"});
- db.contacts.insertMany([

```
            {name: "nilesh", mobile: "9527331338"},
            {name: "nitin", mobile: "9881208115"}
      ]);
```

- Maximum document size is 16 MB.
- For each object unique id is generated by client (if _id not provided).
  - 12 byte unique id :: [counter(3) | pid(2) | machine(3) | timestamp(4)]

Mongo - QUERY

- db.contacts.find(); → returns cursor on which following ops allowed:
  - hasNext(), next(), skip(n), limit(n), count(), toArray(), forEach(fn), pretty()
- Shell restrict to fetch 20 records at once. Press "it" for more records.
- db.contacts.find({ name: "nilesh" });
- db.contacts.find({ name: "nilesh" }, { _id:0, name:1 });
- Relational operators: $eq, $ne, $gt, $lt, $gte, $lte, $in, $nin
- Logical operators: $and, $or, $nor, $not
- Element operators: $exists, $type
- Evaluation operators: $regex, $where, $mod
- Array operators: $size, $elemMatch, $all, $slice

Mongo - DELETE

- db.contacts.remove(criteria);
- db.contacts.deleteOne(criteria);
- db.contacts.deleteMany(criteria);
- db.contacts.deleteMany({}); → delete all docs, but not collection
- db.contacts.drop(); → delete all docs & collection as well : efficient

Mongo - UPDATE

- db.contacts.update(criteria, newObj);
- Update operators: $set, $inc, $dec, $push, $each, $slice, $pull
- In place updates are faster (e.g. $inc, $dec, ...) than setting new object. If new object size mismatch with older object, data files are fragmented.
- Update operators: $addToSet
- example: db.contacts.update({ name: "peter" },

```
      { $push : { mobile: { $each : ["111", "222" ], $slice : -3 } } } );
```

- db.contacts.update({ name: "t" }, { $set : { "phone" : "123" } }, true);
  - If doc with given criteria is absent, new one is created before update.

Data Modeling

- Embedded Data Models
  - { name : { fname : "Nilesh", lname : "Ghule" }, age : 34 }
  - Suitable for one-to-one or one-to-many relationship.
  - Faster read operation. Related data fetch in single db operation.
  - Atomic update of document.
  - Document growth reduce write performance and may lead to fragmentation.

Data Modeling

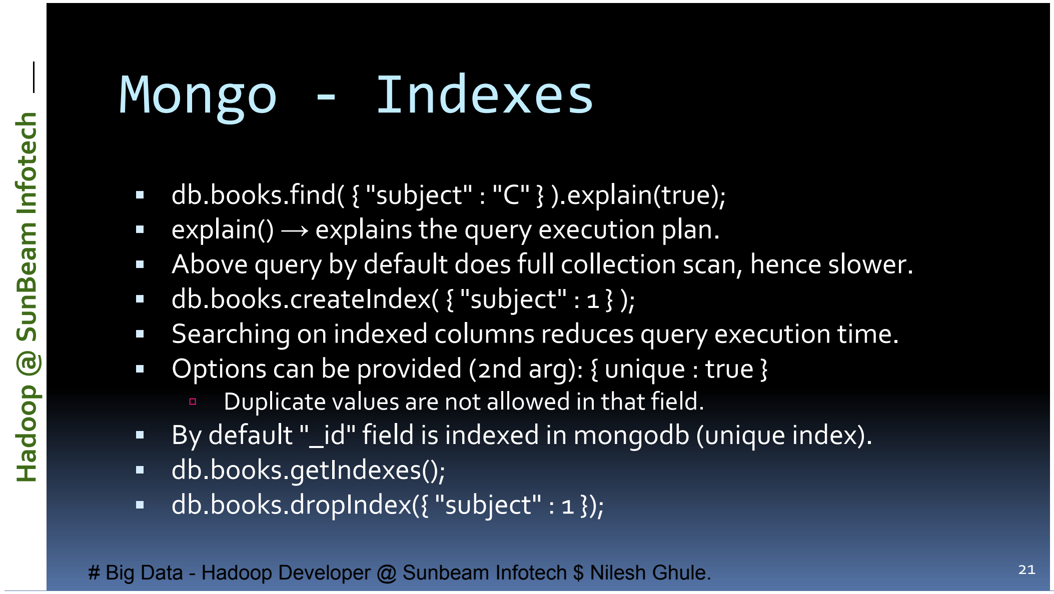
- Normalized Data Models
  - contacts → { _id: 11, email : "nilesh@sun.com", mobile: "9527331338" }
  - persons → { _id: 1, name : "Nilesh Ghule", contact: 11 }
  - Preferred for complex many-to-many relationship.
  - Reduce data duplication.
  - Can use DBRef() to store document reference:
    - contacts → { _id: 11, email : "nilesh@sun.com", mobile: "9527331338" }
    - persons → { _id: 1, name: "Nilesh Ghule", contact: { $db : "test", $ref : "contacts", $id: 11 } }
  - DBRef() are not supported in all client-drivers.

Mongo - MapReduce

- For processing large volumes of data into useful aggregate results.
- Mongodb applies map phase to each input document. The map function emits key-value pairs.
- For keys with multiple values, mongodb applies reduce phase.
- All map-reduce functions written in JS and are executed in mongod server process.
- The map-reduce works on single collection data.
- The output of map-reduce can be written into some collection.
- The input & output collections can be sharded.
- The aggregation framework provides better performance. MR is used for functionalities not available in aggregation framework.

Mongo- Aggregation Pipeline

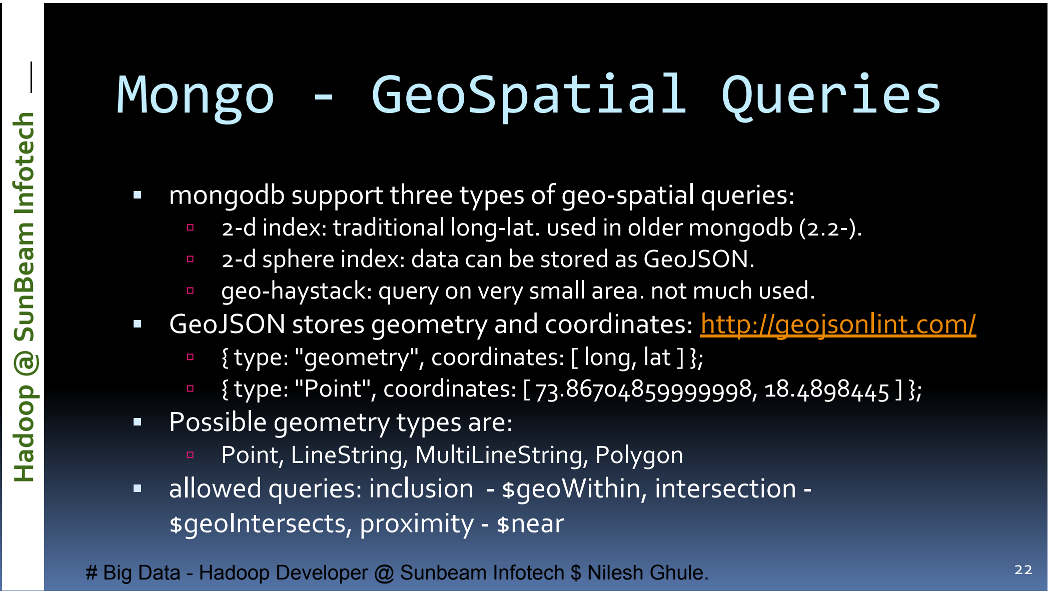
- db.collection.aggregate([{ stage1 }, { stage2 }, ...]);
- $project → select columns (existing or computed)
- $match → where clause (criteria)
- $group → group by
  - { $group: { _id: <expr>, <field1>: { <accum1> : <expr1> }, ... } }
  - The possible accumulators are: $sum, $avg, ...
- $unwind → extract array elements from array field
- $lookup → left outer join
  - { $lookup: { from: other_col, localField: cur_col_field, foreignField: other_col_field, as: arr_field_alias } }
- $out → put result of pipeline in another collection (last operation)



Hadoop @ SunBeam Infotech

Mongo - Indexes

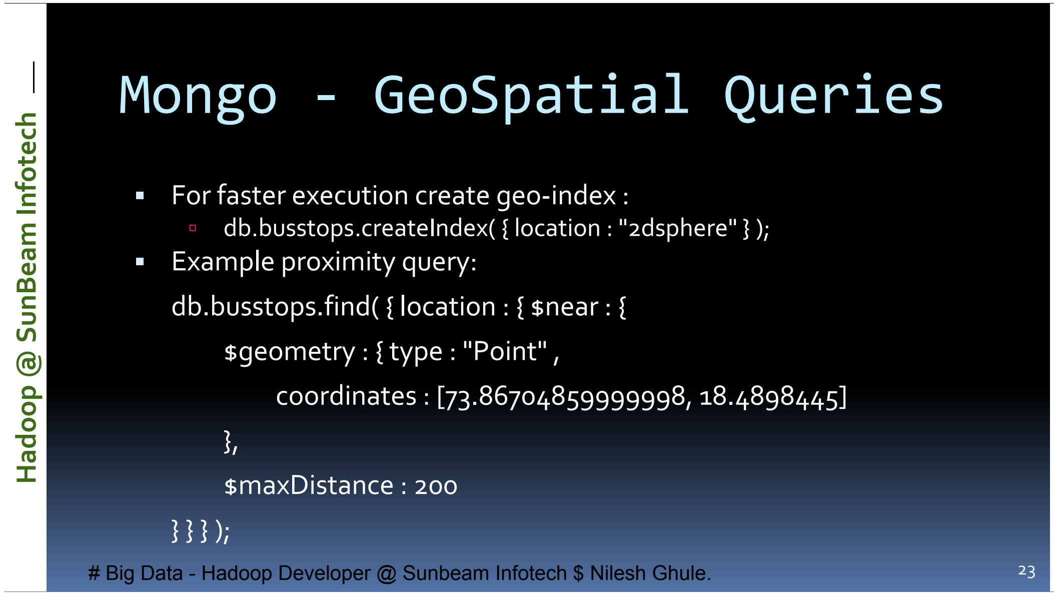
- db.books.find({ "subject" : "C" }).explain(true);
- explain() → explains the query execution plan.
- Above query by default does full collection scan, hence slower.
- db.books.createIndex({ "subject" : 1 });
- Searching on indexed columns reduces query execution time.
- Options can be provided (2nd arg): { unique : true }
  - Duplicate values are not allowed in that field.
- By default "_id" field is indexed in mongodb (unique index).
- db.books.getIndexes();
- db.books.dropIndex({ "subject" : 1 });



Hadoop @ SunBeam Infotech

Mongo - GeoSpatial Queries

- mongodb support three types of geo-spatial queries:
  - 2-d index: traditional long-lat. used in older mongodb (2.2-).
  - 2-d sphere index: data can be stored as GeoJSON.
  - geo-haystack: query on very small area. not much used.
- GeoJSON stores geometry and coordinates: http://geojsonlint.com/
  - { type: "geometry", coordinates: [long, lat] };
  - { type: "Point", coordinates: [73.86704859999998, 18.4898445] };
- Possible geometry types are:
  - Point, LineString, MultiLineString, Polygon
- allowed queries: inclusion - $geoWithin, intersection - $geoIntersects, proximity - $near

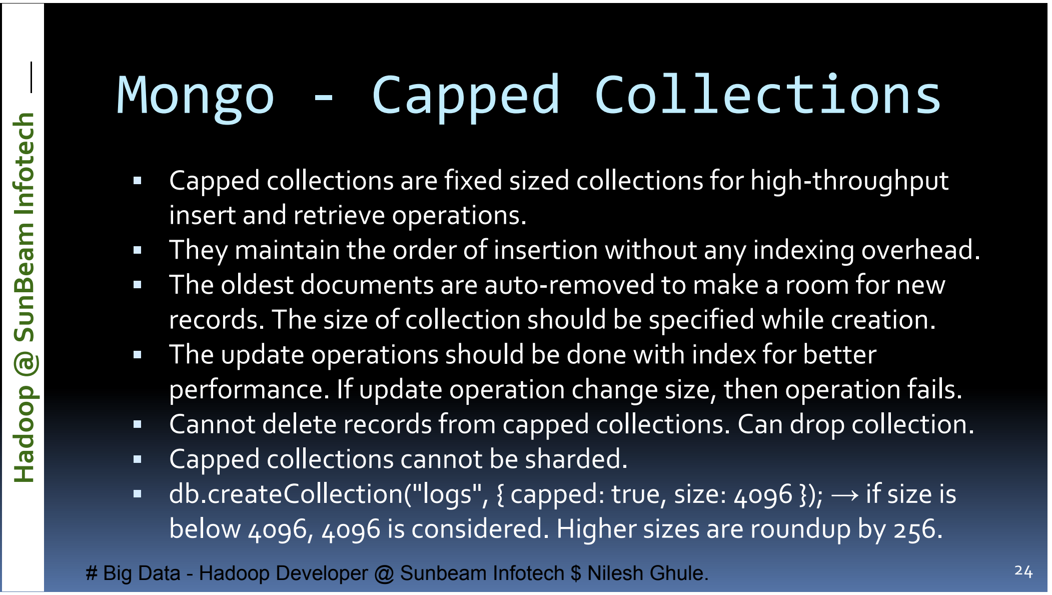


Hadoop @ SunBeam Infotech

Mongo - GeoSpatial Queries

- For faster execution create geo-index :
  - db.busstops.createIndex({ location : "2dsphere" });
- Example proximity query:

```
db.busstops.find( { location : { $near : {
    $geometry : { type : "Point" ,
        coordinates : [73.86704859999998, 18.4898445]
    },
    $maxDistance : 200
} } } );
```



Hadoop @ SunBeam Infotech

Mongo - Capped Collections

- Capped collections are fixed sized collections for high-throughput insert and retrieve operations.
- They maintain the order of insertion without any indexing overhead.
- The oldest documents are auto-removed to make a room for new records. The size of collection should be specified while creation.
- The update operations should be done with index for better performance. If update operation change size, then operation fails.
- Cannot delete records from capped collections. Can drop collection.
- Capped collections cannot be sharded.
- db.createCollection("logs", { capped: true, size: 4096 }); → if size is below 4096, 4096 is considered. Higher sizes are roundup by 256.

Mongo - WiredTiger Storage

- Storage engine is managing data in memory and on disk.
- MongoDB 3.2 onwards default storage engine is **WiredTiger**; while earlier version it was MMAPv1.
- WiredTiger storage engine:
  - Uses document level optimistic locking for better performance.
  - Per operation a snapshot is created from consistent data in memory.
  - The snapshot is written on disk, known as checkpoint → for recovery.
  - Checkpoints are created per 60 secs or 2GB of journal data.
  - Old checkpoint is released, when new checkpoint is written on disk and updated in system tables.
  - To recover changes after checkpoint, enable journaling.

Mongo - WiredTiger Storage

- WT uses write-ahead transaction in journal log to ensure durability.
- It creates one journal record for each client initiated write operation.
- Journal persists all data modifications between checkpoints.
- Journals are in-memory buffers that are synced on disk per 50 ms.
- WiredTiger stores all collections & journals in compressed form.
- Recovery process with journaling:
  - Get last checkpoint id from data files.
  - Search in journal file for records matching last checkpoint.
  - Apply operations in journal since last checkpoint.
- WiredTiger use internal cache with size max of 256 MB and 50% RAM - 1GB along with file system cache.

Mongo - GridFS

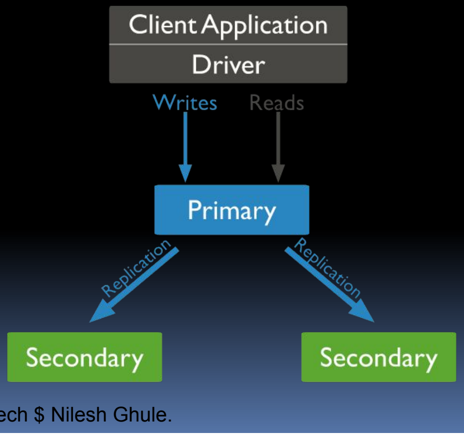
- GridFS is a specification for storing/retrieving files exceeding 16 MB.
- GridFS stores a file by dividing into chunks of 255 kb. When queried back, driver collect the chunks as requested. Query can be range query. Due to chunks file can be accessed without loading whole file in memory.
- It uses two collections for storing files i.e. fs.chunks, fs.files.
- It is also useful to keep files and metadata synced and deployed automatically across geographically distributed replica set.
- GridFS should not be used when there is need to update contents of entire file atomically.
- It can be accessed using **mongofiles** tool or compliant client driver.

Mongo - GridFS

- fs.chunks
  - _id, files_id, n, data
- fs.files
  - _id, length, chunkSize, updateDate, md5, filename, contentType
- Files can be searched using
  - db.fs.files.find({ filename: myFileName });
  - db.fs.chunks.find({ files_id: myFileID }).sort({ n: 1 })
  - GridFS automatically create indexes for faster search.
- mongofiles:
  - mongofiles.exe -d test put nilesh.jpg
  - mongofiles.exe -d test get nilesh.jpg

Mongo - Replication

- A replica set is a group of mongod instances that maintain the same data set.
- Only one member is deemed the primary node, while other nodes are deemed secondary nodes.
- The secondaries replicate the primary's oplog.
- If the primary is unavailable, an eligible secondary will become primary.

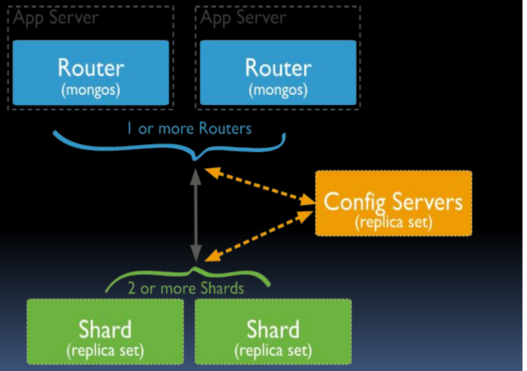


Mongo - Replication

- Secondary servers communicate with each other via heart-beat.
- Secondary applies operations from primary asynchronously.
- When primary cannot communicate a secondary for more than 10 seconds, secondary will hold election to elect itself as new primary. This automatic failover process takes about a minute.
- An arbiter (do not store data) can be added in the system (with even number of secondaries) to maintain quorum in case of election.
- By default client reads from primary, but can set read preference from secondary. Reading from secondary may not reflect state of primary; as read from primary may read before data is durable.

Mongo - Sharding

- Sharding is a method for distributing large data across multiple machines.
- This is mongodb approach for horizontal scaling/scaling out.
- shard: part of collection on each server (replica set).
- mongos: query router between client & cluster.
- config servers: metadata & config settings of cluster.



Mongo - Sharding

- Collections can be sharded across the servers based on shard keys.
- Shard keys:
  - Consist of immutable field/fields that are present in each document
  - Only one shard key. To be chosen when sharding collection. Cannot change shard key later.
  - Collection must have index starting on shard key.
  - Choice of shard key affect the performance.
- Advantages:
  - Read/Write load sharing
  - High storage capacity
  - High availability

Mongo - Sharding

- Sharding strategies:
  - Hashed sharding
    - MongoDB compute hash of shard key field's value.
    - Each chunk is assigned a range of docs based on hashed key.
    - Even data distribution across the shards. However range-based queries will target multiple shards.
  - Ranged sharding
    - Divides data into ranges based on shard key values.
    - mongos can target only those shards on which queried range is available.
    - Efficiency of sharding is based on choosing proper shard key.

Nilesh Ghule
<nilesh@sunbeaminfo.com>

Thank you