# MongoDB Advance Concepts

# Agenda

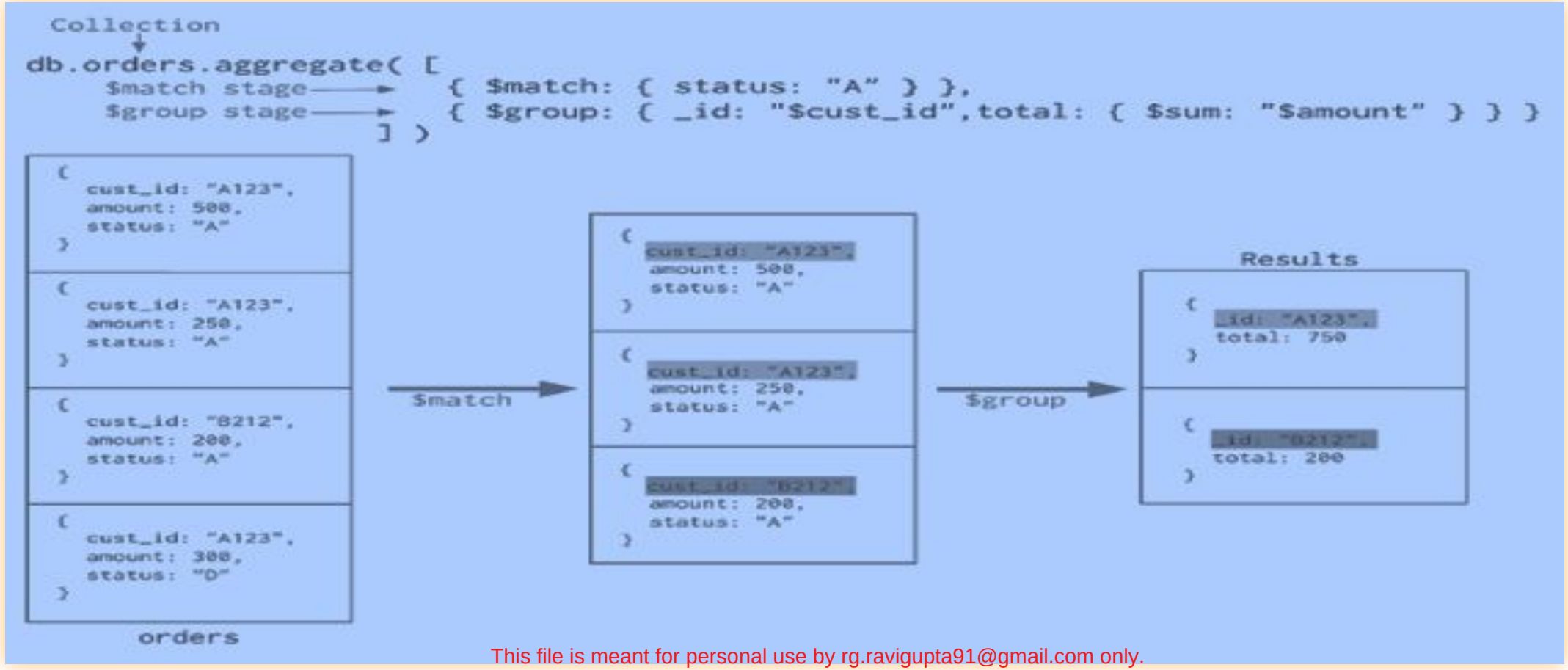
- How to perform Aggregation and MapReduce in MongoDB?
- How to store unstructured data in MongoDB using GridFS
- Concepts of Replication in MongoDB.
- Different members in ReplicaSet
- Importance of Read Preference, Read and Write Concern
- Why to create Sharded Cluster
- How to Shard Database and collections
- Using PyMongo to perform CRUD Operations
- How to perform aggregation, mapReduce using PyMongo

# Aggregation Operation 1/2

- Aggregations operation of MongoDB process data records and return computed results
- MongoDB has rich set of aggregation operations that examine and perform different kinds of calculations on the data sets.
- Aggregation uses documents of collection as an input and return results in the form of one or more documents
- Aggregation Pipelines: In a aggregation framework, documents enter a multi-stage pipeline that transforms the documents into an aggregated result.
- Using aggregation pipeline,we can filter, transform and modify the documents of the collection

# Aggregation Operation 2/2

- Pipeline stages can use differnt operators for performing task such as calculating the sum,average or concatenating a string



# Aggregation pipeline Stages 1/2

- Pipeline stages are configured as an array in an aggregation pipeline
- All the documents goes through the stages in sequence
- We can use same stage multiple times inside an aggregation pipeline except $out and $geoNear stages

```
db.collection.aggregate( [ { <stage> }, ... ] )
```

- $project: is used to select specific fields of the documents
- $match: is used to select specific documents to reduce the amount of documents goes to next stage.
- $group: This does the actual aggregation as discussed above.

# Aggregation pipeline Stages 2/2

- $sort: It is used sorts the documents based on the specific field or condition
- $skip: it is used to skip forward in the list of documents for a given amount of documents.
- $limit: We can limit the amount of documents to look at by providing the number starting from the current position.
- $unwind: This is used to unwind document that are using array data type. Each individual element of the array will be generated as document. Using this stage will increase the amount of documents for the next stage.

# Aggregation pipeline Examples

- The following aggregation operation returns all states with total population greater than 20 million:

```
db.zipcodes.aggregate( [{ $group: { _id: "$state", totalPop: { $sum: "$pop" } } },{ $match: { totalPop: { $gte: 20*1000*1000 } } }] )
```

- The following aggregation operation returns user names sorted by the month they joined. This kind of aggregation could help generate membership renewal notices.

```
db.users.aggregate([{ $project :{month_joined : { $month : "$joined" },name : "$_id",_id : 0}},{ $sort : { month_joined : 1 } }])
```

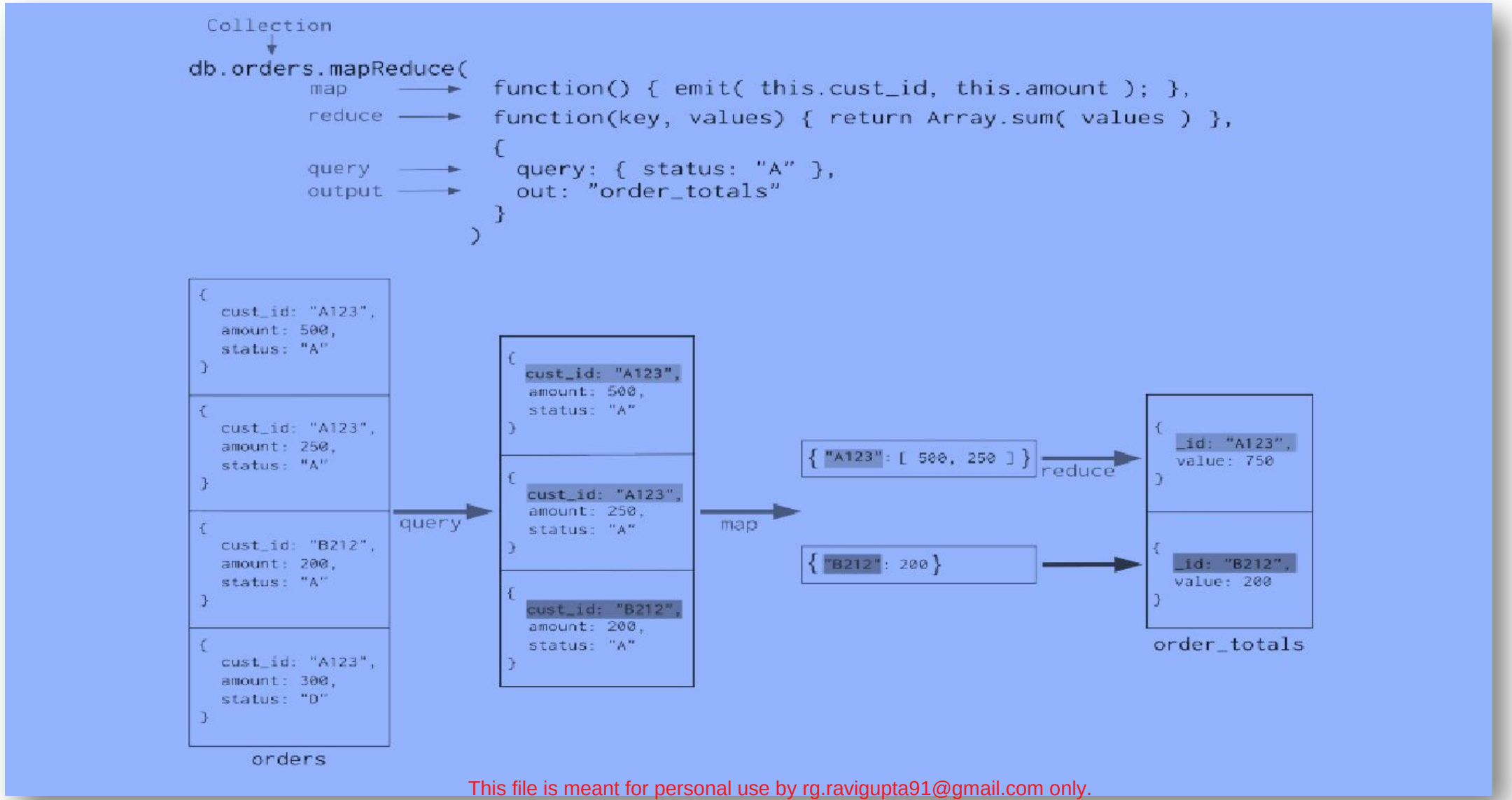
# MapReduce in MongoDB

- Map-Reduce is parallel and distributed computing framework made popular by Hadoop platform.
- Map-reduce in MongoDB is useful for processing large number of documents into useful aggregated results.
- MongoDB provides the mapReduce command  for performing map-reduce operation on the selected collection
- Map Phase is applied to each individual document of the collection that match the query condition

# MapReduce in MongoDB

- The map phase logic has be written using java or python script, reduce function always emit key-value pair as output
- The reduce phase takes each unique key and list of values and apply the java or python script logic written in the custom defined reduce function.
- Output of the map-reduce operation can be returned as a document or may stored in the collection.
- The input and the output collections can be non-sharded or sharded.
- Sharded collection is preferred as it will truly parallelize the computation

# MapReduce Example



# GridFS

- GridFS is a file system abstraction of MongoDB for storing and retrieving files that exceed 16MB

- Whenever we store file in GridFS , it divides a file into chunks and stores each of those chunks as a separate document.

- By default size of the GridFS chunk size is 255k.
- GridFS uses two collections to store files.
- One collection stores the chunks of 255K size, while the other one stores file metadata such as filename and chunkId

# GridFS

- GridFS stores files in two collections “fs.chunks” and “fs.files”

- Each document in the chunks collection represents a distinct chunk of a file in GridFS store.
- Each chunk is identified by its unique ObjectId stored in its _id field.
- Whenever we query a file in GridFS store, the driver or client will group together the chunks as needed.
- we can perform range queries on files stored in the GridFS.
- we also can access specific chunk of the file stotred in GridFS, which hepls you to “skip” into the middle of a video or audio file

# Capped Collection

- Capped collection are fixed-size collection, size can defined in bytes or number of documents.
- Capped collections helps us achieving high throughput insert and fetch operations based on the insertion order
- It works similar to circular buffer data structure
- Capped collections guarantee preservation of the insertion order
- Capped collections only allow updates that fit the original document size
- Once capped collection fully fills its allocated space , It removes oldest document to make a room for new document to be written

# Capped Collection

- db.createCollection( “serverlog", { capped: true, size: 100000 } )
- db.createCollection(“serverlog", { capped : true, size : 5242880, max : 5000 } )

- MongoDB will automatically remove older documents if a capped collection reaches its size or document limits
- If we perform a find() query on a capped collection, then its returns the documents in the same order in which they have been inserted into capped collection.
- If we want to retrieve documents in reverse insertion order, then use the sort() method with the $natural parameter set to -1, as shown in the below example:

**db.cappedCollection.find().sort( { $natural: -1 } )**



- To verify given collection is capped or not then

# Storage Engine

- A storage engine is the component of a database which manages the format and structure of data being stored on disk
- Many databases support multiple storage engines
- Different engines perform better for specific workloads.
- Some storage engine might offer better performance for read-heavy workloads, while some other  might support a higher-throughput for write operations.
- With multiple storage engines, you can decide which storage engine is best for your application
- WiredTiger is the default engine of MongoDB starting version 3.2
- MMAPv1 was the default storage engine in older versions.

# WiredTiger Storage Engine

- WiredTiger engine offers additional flexibility and improved throughput for many workloads
- WiredTiger excels at read and insert workloads as well as more complex update workloads.
- Document Level Locking
- With WiredTiger, all write operations happen within the context of a document level lock.
- As a result, multiple clients can modify more than one document in a single collection at the same time.
- With this very granular concurrency control, MongoDB can more effectively support workloads with read, write and updates as well as high-throughput concurrent workloads.
- .

# Replication in MongoDB

- Replication provides redundancy and increases data availability.

- Replication helps us to recover from hardware failure and service interruptions
- Replication in some cases also help to increase read capacity.
- When data is replicated to multiple servers then Clients have the choice to send read and write operations to different servers.
- Replication can also be done across different data centers to provide higher locality and availability of data for distributed applications.
- Having more copies of the data, we can use one to disaster recovery, reporting, or backup

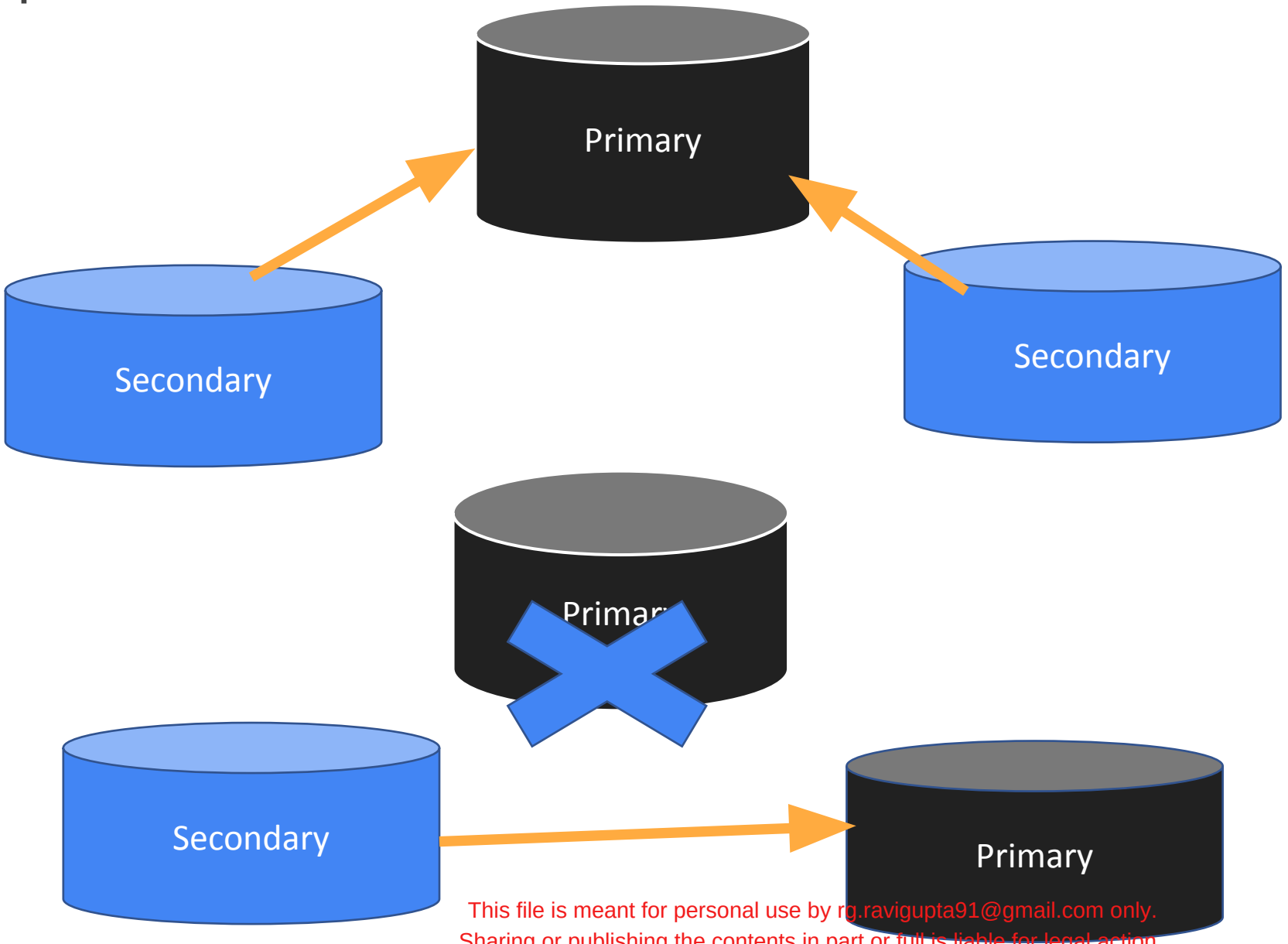
# Replication

- MongoDB provides database replication via a topology known as a replica set
- Replica sets distribute data across machines for redundancy and automate failover in the event of server and network outages

- Additionally, replication is used to scale database reads. If you have a read intensive application, as is commonly the case on the web, it's possible to spread database reads across machines in the replica set cluster

- Replica sets consist of exactly one primary node and one or more secondary nodes.
- a replica set's primary node can accept both reads and writes, but the secondary nodes are read-only

# Replication



# Benefits of Replication

- Redundancy of data and automatic Failover
- No downtime for upgrades and maintaince
- Strong Consistency if availability  can be compromised
- Delayed Consistency if availability can’t be compromised
- Geographical data storage

# Replica Set

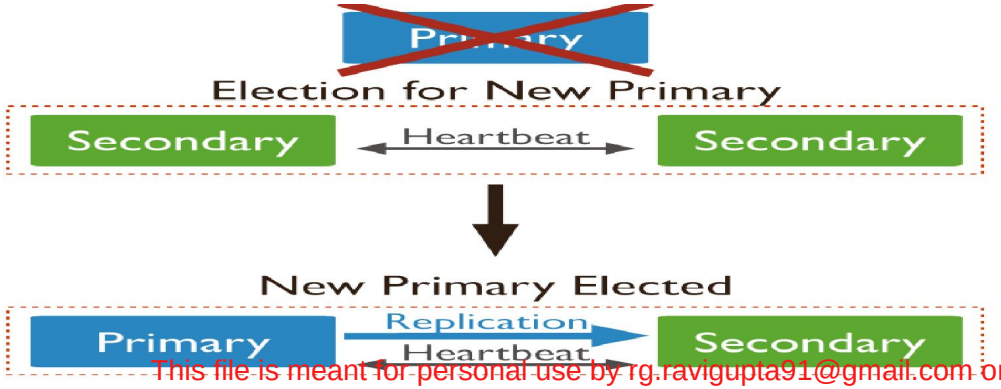
- A replica set is group of separate mongod instances configured together as a cluster.
- All the members of replica set stores same data
- In a replica set,  one mongod will be selected as the primary and it will receive all write operations from the client.
- All other instances will act as secondaries, apply operations from the primary so that they have the same data set as primary
- Strict consistency is guaranteed for all reads operation from the primary.

# Replica Set failover

- To facilitate replication, the primary reocrds all changes to its data sets in its oplog.rs collection in the local database.

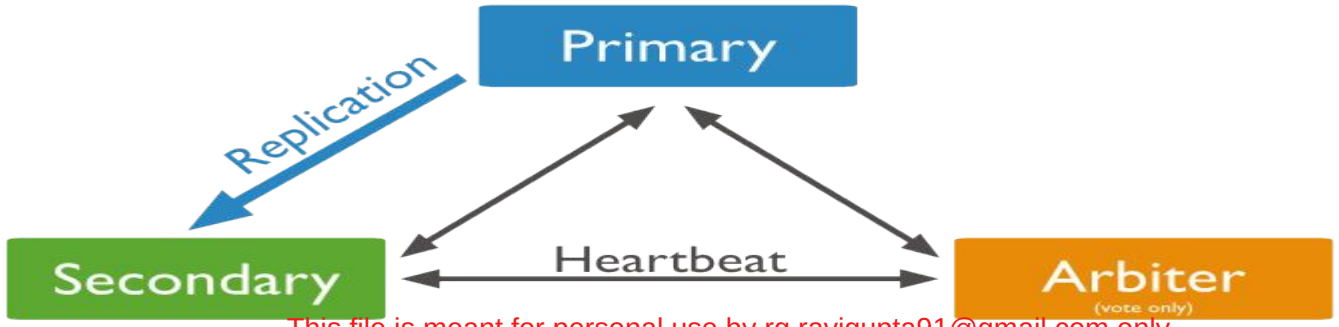
- The secondaries replicate the primary's oplog collection and apply the operations to their  local oplog.rs collection in the local database
- Secondaries apply operations from the primary asynchronously.

- If the primary is down or unavailable , the replica set will conduct election to select a new primary from the existing secondaries.



# Replica Set Members

- Primary : The primary receives all write operations
- Secondaries: Secondaries replicate operations from the primary to maintain an identical data set.
- Arbiters(Optional): Arbiters do not keep a copy of the data. However, arbiters play a role in the elections that select a primary if the current primary is unavailable
- A priority 0 member is a secondary that cannot become primary.
- Starting from version 4.4.0: A replica set can have up to 50 members but only 7 voting members



# Read Preference

- Read preference helps MongoDB client deciding to which member of replica set the read request to be routed.
- By default, All the read requests from an application goes directly to primary member in a replica set
- For an application that does not require fully up-to-date data, you can improve read throughput or reduce latency by distributing some or all reads to secondary members of the replica set
- In mongo shell, the readPref() cursor method provide access to read preference
- Use mongo.setReadPref() to set read preference

# Read Preference Modes

| | |
|---|---|
| **primary** | Default mode. Read from the primary. |
| **primaryPreferred** | Read from the primary, otherwise read from secondary if primary is unavailable |
| **secondary** | Read from Secondary member |
| **Secondary Preferred** | Read from secondary member otherwise read from primary if no secondary members are available |
| **nearest** | Read from any member of the replica set whichever provides least network latency. |

# Write Concern

- Using Write concern, we can control on how many members of the replica set data will be persisted before confirming the success of a write operation
- With stronger write concerns, clients will wait for longer duration to confirm the write operation.
- When inserts, updates and deletes have a weak write concern, then those  operations perform faster.
- MongoDB provides different levels of write concern to better address the specific needs of applications
- Clients may adjust write concern to ensure that the most important operations persist successfully to an whole replica set.
- For other less critical operations, clients can adjust the write concern levels to achieve faster performance rather than ensure persistence to the whole replica set

# Write Concern Levels

- Write Concern can be specified in a below format
- **{ w: <value>, j: <boolean>, wtimeout: <number> }**
- **w Option :** The w option requests acknowledgment that the write operation has persisted the data to the specified number of replica set members.
- **W Option values:**
    - <number>
    - "majority"
    - <custom write concern name>
- You can include a timeout threshold for a write concern:
- **db.items.insert({ item: “Samsung", qty :100, type: “TV" }, { writeConcern: { w: 3, wtimeout: 5000 } })**

# Read Concern

- The readConcern option allows us to control consistency behaviour of the read operation from the replica set .
- By using appropriate write concerns and read concerns, we can adjust the level of consistency and availability we want to achieve in our application
- Read Concern Levels :
    - "local"
    - "available"
    - "majority"
    - "linearizable"
    - "snapshot"

```
db.website.find().readConcern("majority")
```

# Sharding

- Sharding in MongoDB is the process of distributing and storing data across multiple server. In some databases, it is also called as partitioning
- In MongoDB, sharding  is used to meet the demands of data growth with horizontal scaling
- With sharding, we can add more shards to store more data and meet the demands of  increasing read and write operations.
- In MongoDB, Each shard is an independent mongod process or a replica set, and collectively all the shards together make up a single logical database
- Once Sharding  is enabled on the collection, it automatically balances data and load across multiple shards.
- Sharding provides additional write capacity by distributing the write load across multiple shards.

# Advantages of Sharding

- Partitioning the data
- Scaling write throughput
- Increasing data storage capacity
- Auto-balancing of data and load

# When to enable Sharding?

You should consider deploying a sharded cluster, if:

✔Amount of your application data is greater than storage limit of the single node or Replica set

✔the size of your application active working set(data needed to be loaded in memory to serve the queries) is greater than maximum amount of RAM available in the single server.

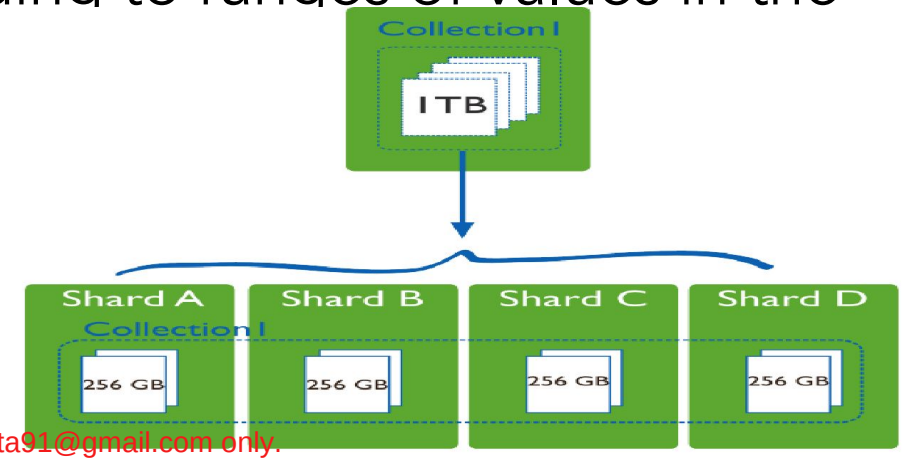
✔If your application needs to perform large amount of write activity and a single MongoDB instance cannot serve those many write queries in one go.

# What is Shard in MongoDB?

- A given shard stores documents which falls within a specific range of values of a shard key.
- Shard key is defined on the collection, it can be either a single field or multiple fields.
- MongoDB further partitions documents of shard into chunks. The default chunk size is 64 MB
- Each chunk represents a smaller range of values within the shard's range.
- MongoDB splits the chunk into smaller chunks Whenever a chunk grows beyond the chunk size .
- The chunk split happens considering the ranges in of shard key

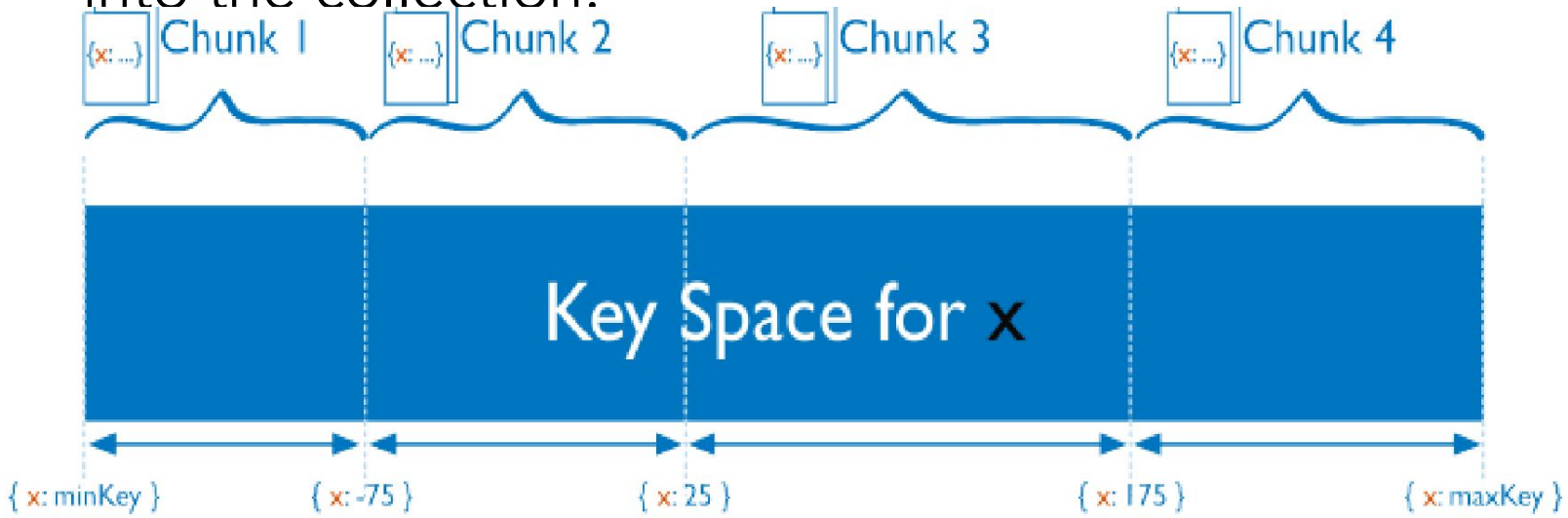
# What is Shard key?

- Inside a sharded cluster, sharding can be enabled on a per-database basis.
- After enabling sharding for a database, you choose which collections to shard. For each sharded collection, you specify a shard key.
- The shard key determines the distribution of the collection's documents among the cluster's shards. The shard key is a field that exists in every document in the collection
- MongoDB distributes documents according to ranges of values in the shard key.



# Impact of Shard key?

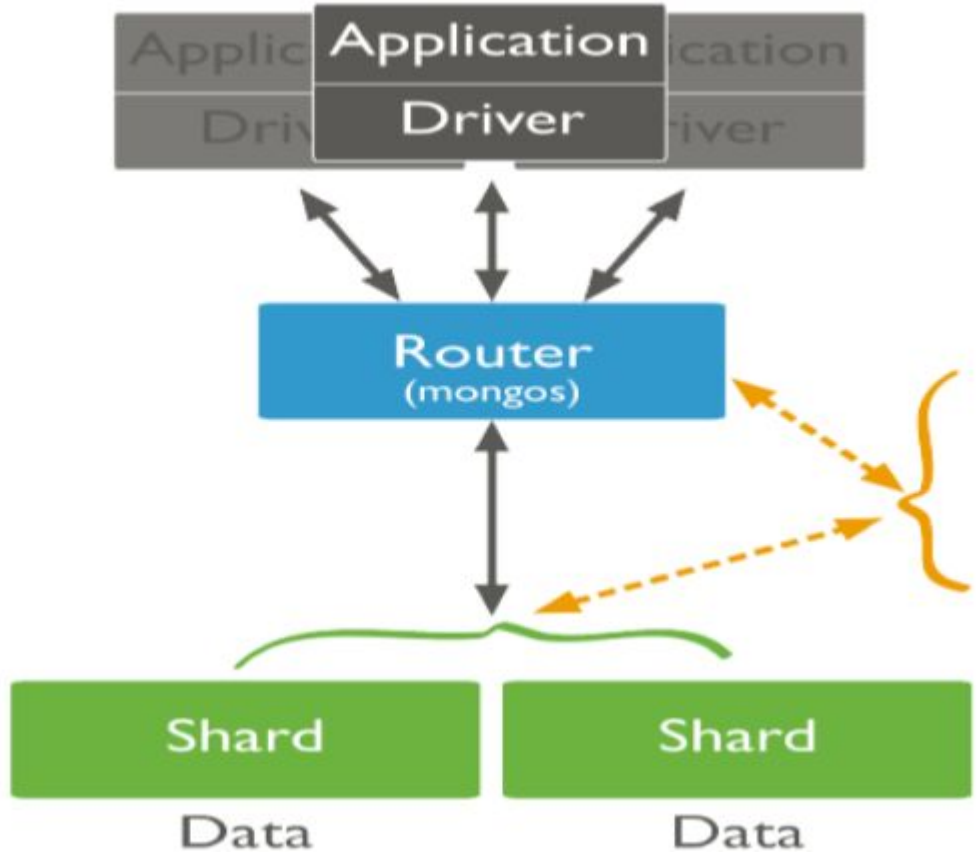
- Selecting the correct shard key can have a huge impact on the scalability, performance and storage capability of mongoDB sharded cluster.
- Appropriate shard key decision mainly depends on the schema of the collection, the way application wants to write and query the data into the collection.

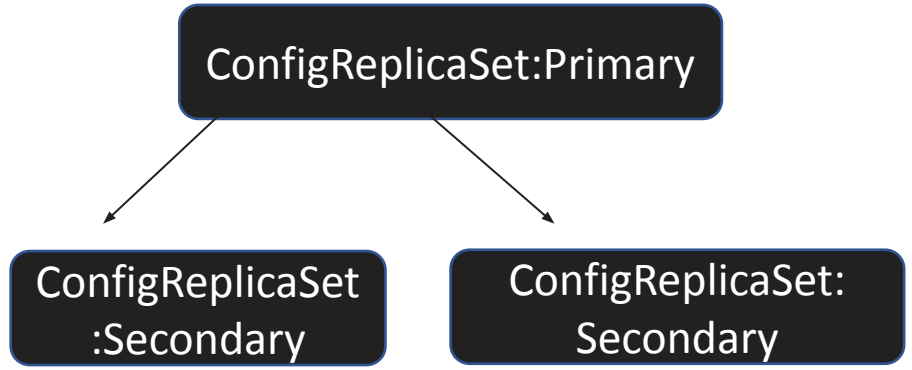


# Different usage of Shard keys in Sharded cluster

- **Write Scaling**: Some possible shard keys distribute write operations more evenly to different shards of the cluster, while others do not.
- MongoDB supports shard key to be hashed shard key
- Hashed shard key uses hashed index and greatly improve write scaling.
- Querying. A mongos receives queries from applications, checks the ranges of shard key from the config replica set, to route queries to the mongod instances having the range which contain the value of the shard key.
- So, The good shard key have a profound affect on query performance.
- Query Isolation: the fastest queries in a sharded environment are those that mongos will route to a single shard, using the shard key and the cluster meta data from the config server

# MongoDB Cluster Architecture



# MongoDB Cluster Details

- Config Replica Set : In the config replica set we should have minimum 3 members .
- Each config server must be on separate machines.
- Like any other replica set, config replica set will have primary and secondarys
- Primary will accept the write operation to store the meta data of the cluster
- This meta data will be replicated on the secondary members of the config replica set
- Shards Two or more replica sets. These replica sets are the shards.
- Query Routers (mongos) One or more mongos instances. The mongos instances used as the routers. Typically, deployments have one mongos instance on each application server.

# How to Enable Sharding on database?

- We need to enable sharding on the database before enabling sharding on the individual collection
- Once sharding is enabled on the database then we can enable sharding on collection inside that database.

1. From a mongo shell, connect to the mongos instance.

**mongo --host <hostname or IP address of server running mongos> --port <port number on which mongos listens on>**

2. Use the following syntax t:

**sh.enableSharding("database name")**

**OR db.runCommand( { enableSharding: <database name> } )**

# How to Enable Sharding for a Collection?

- You enable sharding on a per-collection basis.

  1. Determine which field or set of fields you want to use for the shard key. Your decision of the shard key will greatly affects the efficiency of sharding.

  2. If the collection is not empty and already has data then you must create an index on the shard key.

- If the collection is empty then MongoDB will automatically create the index when we enable the sharding using sh.shardCollection() command.

Command for enabling sharding for a collection :

**sh.shardCollection("<database name>.<collection name>", shard key pattern)**

# Sharded Collection examples

**sh.shardCollection("<database name>.<collection name>", shard key pattern)**

**examples**

sh.shardCollection(“greatlearning.people", { “pincode": 1, "name": 1 } )
sh.shardCollection("greatlearning.addresses", { "state": 1, "_id": 1 } )
sh.shardCollection("greatlearning.chairs", { "typeofchair": 1, "_id": 1 } )
sh.shardCollection(" greatlearning.alerts", { "_id": "hashed" } )


PES UNIVERSITY

- In the sharded cluster , applications will connect to MongoS to submit the request
- MongoS will read the meta data from config replica set to understand whether collection is sharded collection or non sharded
- If sharded then it will check the range of shard keys allocated per shard
- Then mongoS will route the write query to the shard which owns the shard key range
- Once data is written to primary of the shard it will be replicated to remaining secondary members

# Maintaining a balanced data distribution

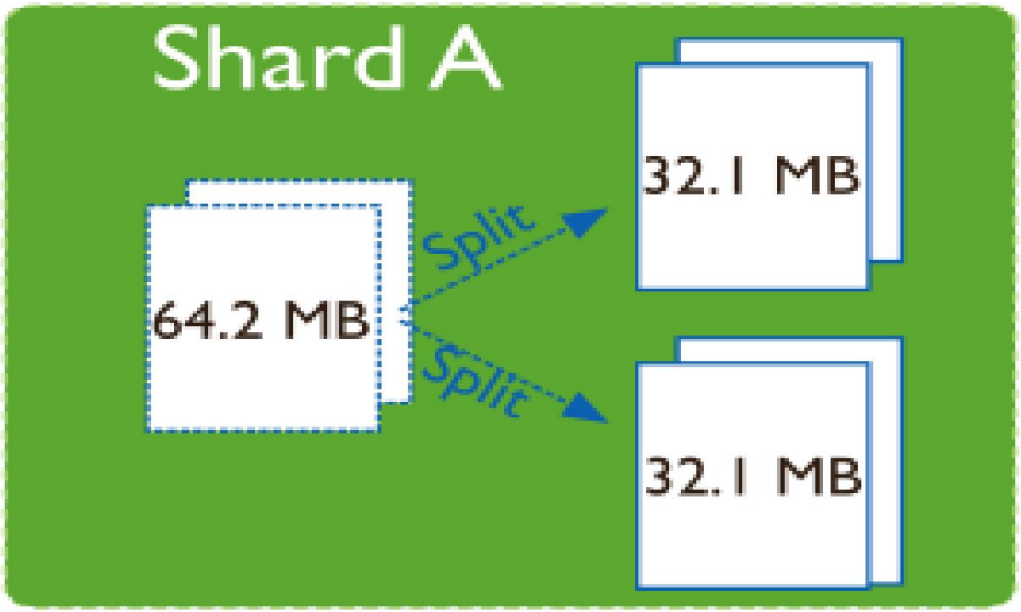
- Storage of new data or the addition of new shards can result in data distribution imbalances within the sharded cluster
- Cluster will be in imbalanced state if a shard contains significantly higher number of chunks than another shard or a size of a chunk is significantly greater than other chunk sizes
- MongoDB automatically maintains a balanced cluster using two different background process: splitting and the balancer

# Splitting 1/2

- Splitting is a background process that keeps chunks from growing too large.
- When a chunk size becomes more than a specified chunk size of 64MB , A mongos instance splits the chunk in half.
- Inserts and updates also triggers splits if chunk grows beyond a specified chunk size.
- Splits creates new token ranges and like other meta-data it gets stored in config replica set.
- MongoDB does not migrate any data while creating splits, migration of data is the responsibility of balancer process.

# Splitting 2/2

- Splits may lead to an uneven distribution of the chunks for a collection across the shards.
- Once splits are created, the mongos instances using balancer will initiate a migration to redistribute chunks across shards.



# Chunk Size

- You can increase or reduce the default chunk size(64MB)
- Changing the chunk size has its effect on the cluster's efficiency
- Small chunks lead to a more even distribution of data at the expense of more frequent migrations.
- Large chunks lead to fewer migrations but at the expense of a potentially more uneven data distribution
- Chunk size also impacts the Maximum Number of Documents to be stored Per Chunk. Whole chunk will be migrated from one shard to another.
- For many production deployments, it is recommended to avoid frequent and potentially spurious migrations by setting larger chunk size

# Special Chunk Type

- Jumbo Chunks: If the split of chunk is unsuccessful, MongoDB classify the chunk as jumbo to skip it in next round of chunk migration

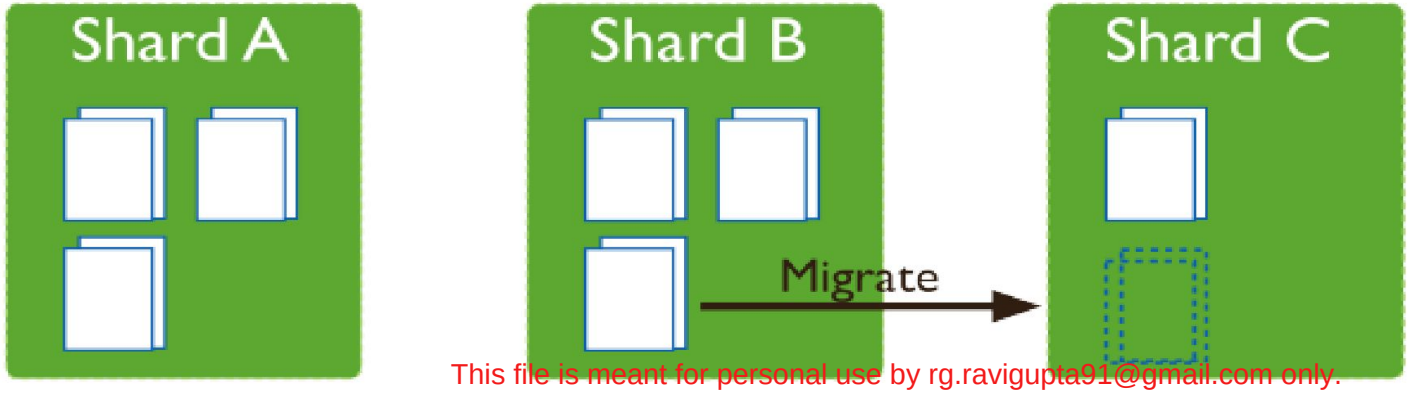
- Indivisible Chunks: In some cases, chunks can grow beyond the specified chunk size without undergoing a split for example if chunk data for single shard key.

# Shard Balancer

- Balancer is the background process MongoDB uses to redistribute data within a sharded cluster.
- The balancer can run from any of the query routers in a cluster.
- When a shard has too many chunks when compared to other shards, MongoDB automatically balances the shards
- Adding a shard to a cluster creates an imbalance cluster since the new shard will be empty and  initially it will not have any chunks.
- Balancer will start migrating some chunks from other shards to the newly added shard to keep the cluster balance.
- Before removing a shard from the cluster, balancer migrates all the chunks to other shards to prevent un-availability of those chunks.

# Shard Balancer

- Migrating chunks can have some negative impact on the cluster, The balancing process minimizes the impact by:
- Moving only one chunk at a time.
- Initiating a balancing round only when the difference in the number of chunks between the shard with the greatest number and the shard with the lowest exceeds the migration threshold.



# Shard Balancer

- We can also disable the balancer on a temporary basis to prevent the balancing process from impacting production traffic
- Below are migration threshold for Balancer to start chunks migration

| No Chunks | Migration Threshold |
|---|---|
| < 20 | 2 |
| 20-79 | 4 |
| >80 | 8 |

# What is PyMongo?

- PyMongo is a Python distribution containing tools for working with MongoDB,
- PyMongo is the recommended way to work with MongoDB from Python.

- Installation of PyMongo:
- **You can install PyMongo 3.3+ by using the following pip command:**
- **pip install pymongo**

# Making a Connection with MongoClient

```
import pymongo

from pymongo import MongoClient

client = MongoClient()

                OR

client = MongoClient('localhost', 27017)

                OR

client = MongoClient('mongodb://localhost:27017/')
```

# Getting a Database: PyMongo

- A single MongoDB cluster can support multiple independent databases.
- Using PyMongo we can access databases using attribute style access on MongoClient instances:

```
db = client.test_greatlearning
```

- We can also use dictionary style access if attribute style access does not work:

```
db = client['test-greatlearning']
```

# Getting a Database: PyMongo

- Accessing a collection in PyMongo works the same as accessing a database:

```
collection = db.test_demo
```

- or (using dictionary style access):

```
collection = db['test-demo']
```

# Documents in PyMongo

- We use dictionaries to represent documents. Below is the example of a document representing a blog post:

```
import datetime
post = {"author": "Mukesh Kumar",
        "text": "My first blog post about PyMongo!",
        "tags": ["mongodb", "python", "pymongo"],
        "date": datetime.datetime.utcnow()}
```

- Python data types such as datetime,String,list etc will be automatically converted to and from the appropriate BSON types.

# Inserting a Documents : PyMongo

- To insert a document into a collection, we can use the insert_one() method:

```
blogs = db.blogs

blog_id = posts.insert_one(blog).inserted_id

post_id

ObjectId('...')
```

- After insertion, every document will have a special key "_id",
- _id is automatically added if the document doesn't already contain an "_id" key.
- The value of "_id" must be unique within the collection.

# Fetching a Document using find_one():PyMongo

- find_one() query is used to fetch single document in MongoDB.
- This method returns single document based on the filter criteria, and none if there are no matches.
- find_one() also supports querying on specific elements
- print.pprint(posts.find_one({"author": "Mukesh Kumar"}))

# Bulk Inserts using PyMongo

- Bulk insert operations can be performed, by passing a list as the first argument to insert_many()
- This will create list of documents and send that list in a single command to the server

```
new_posts = [{"author": "Mukesh Kumar","text": "NoSQL datbases are widely used now!","tags": ["MongoDB", "Cassandra"], "date": datetime.datetime(2019, 11, 12, 11, 14)}, {"author": “Mukesh Kumar","title": "MongoDB is used by many e-Commerce companies", "text": "MongoDB provides flexible schema", "date": datetime.datetime(2019, 11, 10, 10, 45)}]
```

- result = posts.insert_many(new_posts)
- result.inserted_ids

# Fetching more Documents : find() in PyMongo


- We can use the find() method to fetch multiple documents as the result of the query.
- find() returns a Cursor instance, which can be used to iterate over all matching documents.

```
for post in posts.find():
    pprint.pprint(post)
```

# Range Queries:PyMongo

- We can perform many advance query in MongoDB.
- Lets perform a query where we limit results to blogs older than a certain date, but also sort the results by writter

```
d = datetime.datetime(2020, 11, 12, 12)
for blog in blogss.find({"date": {"$lt": d}}).sort("writter"):
    pprint.pprint(blog)
```

# Indexing using PyMongo

- Creating indexes can make certain queries run faster and can also add additional functionality to querying and storing documents.
- Let's create a unique index on a field that rejects duplicate documents in the collection

```
result = db.users.create_index([('useId', pymongo.ASCENDING)],unique=True)


sorted(list(db.users.index_information()))
```

# Aggregation using PyMongo

- Let's perform a simple aggregation to count the number of items for each order in the orders array
- To achieve this we need to pass in three operations to the pipeline.
- First, we need to unwind the orders array, then group by the orders and sum them up
- Finally we sort by count
- As python dictionaries don't maintain order, we can se SON or collections.OrderedDict

# Aggregation using PyMongo

```
from bson.son import SON

pipeline = [ {"$unwind": "$orders"},
             {"$group": {"_id": "$orders", "count": {"$sum": 1}}},
             {"$sort": SON([("count", -1), ("_id", -1)])}
            ]
```

# MapReduce using PyMongo

- MapReduce framework  is another option for doing aggregation in MongoDB
- We will use mapReduce count the number of occurrences for each blog in the blogs array, across the entire collection.

# Map Function in PyMongo

Map function will just emits a single (key, 1) pair for each blog in the array:

```
from bson.code import Code

mapperFn = Code(" " "
            function () {
                this.blogs.forEach(function(z) {
                  emit(z, 1);
                });
             }  " " ")
```

# Reduce Function in PyMongo

The reduce function will sum over all of the emitted values for a given key

```
reducerFn = Code(" " "
            function (key, values) {
              var totalblogs = 0;
              for (var i = 0; i < values.length; i++) {
                 totalblogs += values[i];
              }
              return total;   }   " " ")
```

# Performing MapReduce

Finally, we call map_reduce() and iterate over the result collection:

```
result = db.blogs.map_reduce(mapperFn, reducerFn, "myresults")
for doc in result.find().sort("_id"):
  pprint.pprint(doc)
```

did you know?

PES UNIVERSITY

- MongoDB is the number one NoSQL database in terms of deployment

- MongDB is constantly ranged among top 5 databases  in the last 3 years

https://db-engines.com/en/ranking_trend/system/MongoDB

References:

https://docs.mongodb.com/manual/tutorial/

# Thank You