بسم الله

MongoDB

HOSSEIN FORGHANI

MAKTAB SHARIF

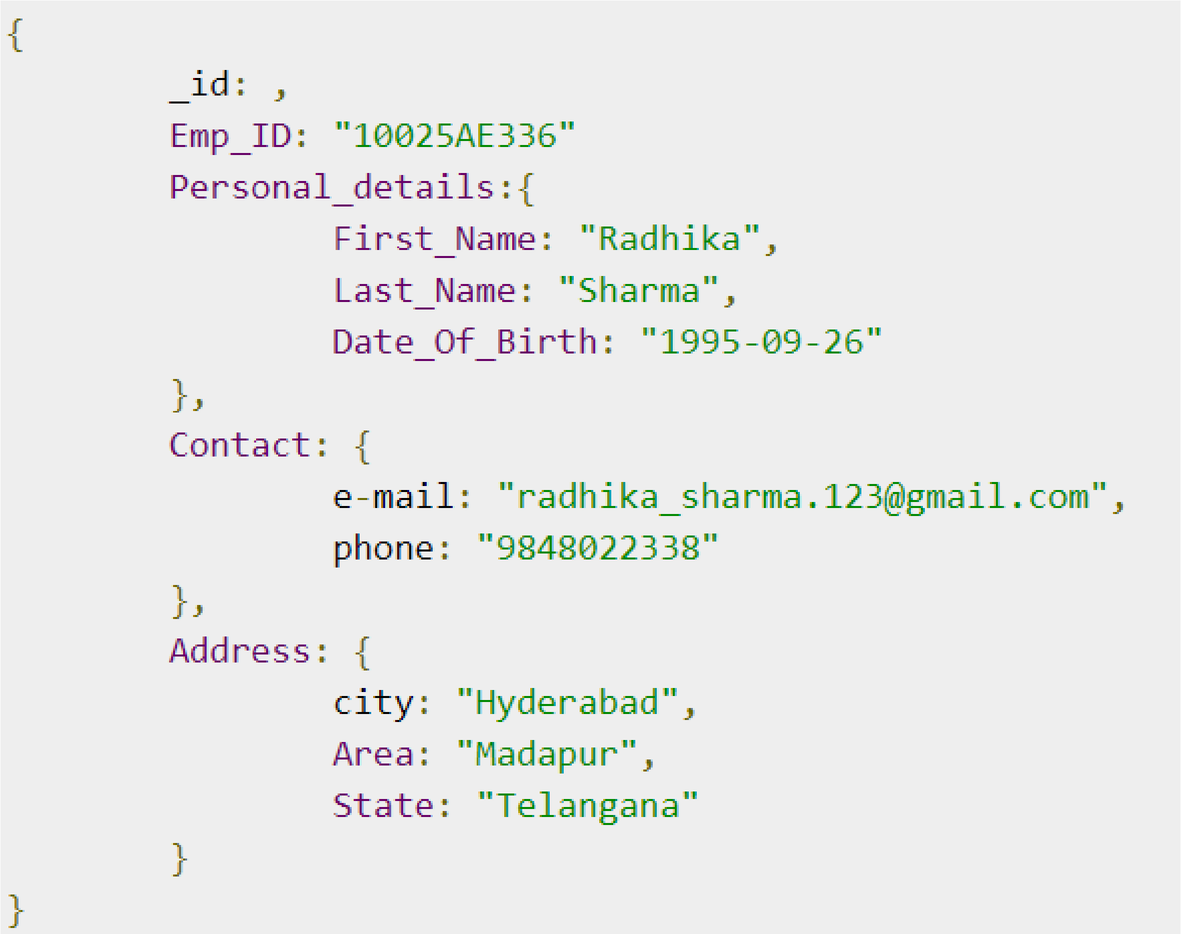
contents

- How to design Mongo database?
- Index
- Aggregation & pipelines
- Dump & restore
- Replication & sharding

Data Modeling

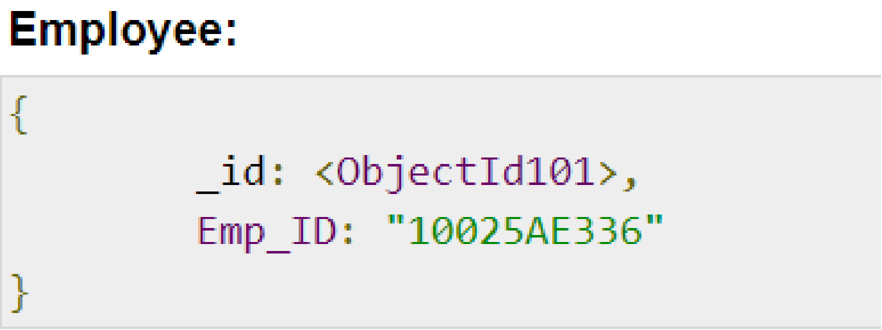
- Embedded data model
- Normalized data model
- Best practice: mixture of the above!

Embedded Data Model



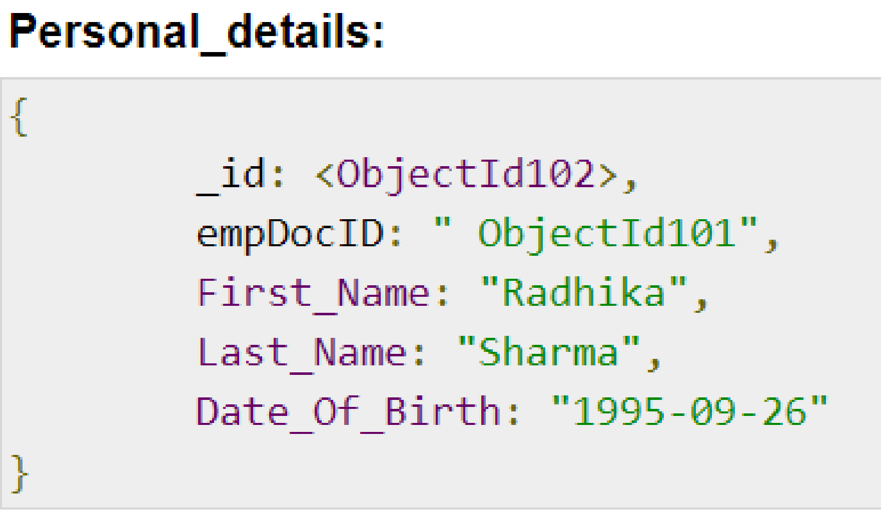
```
{
        _id: ,
        Emp_ID: "10025AE336"
        Personal_details:{
                First_Name: "Radhika",
                Last_Name: "Sharma",
                Date_Of_Birth: "1995-09-26"
        },
        Contact: {
                e-mail: "radhika_sharma.123@gmail.com",
                phone: "9848022338"
        },
        Address: {
                city: "Hyderabad",
                Area: "Madapur",
                State: "Telangana"
        }
}
```

Normalized Data Model



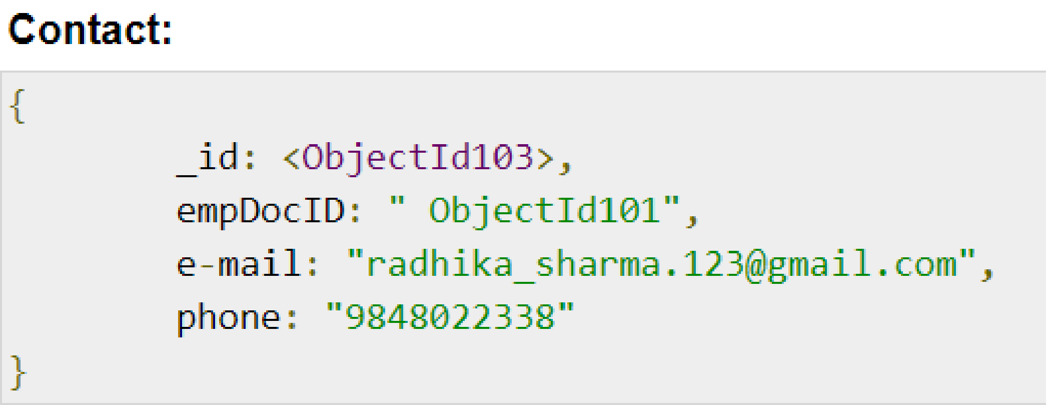
Employee:

```
{
        _id: <ObjectId101>,
        Emp_ID: "10025AE336"
}
```



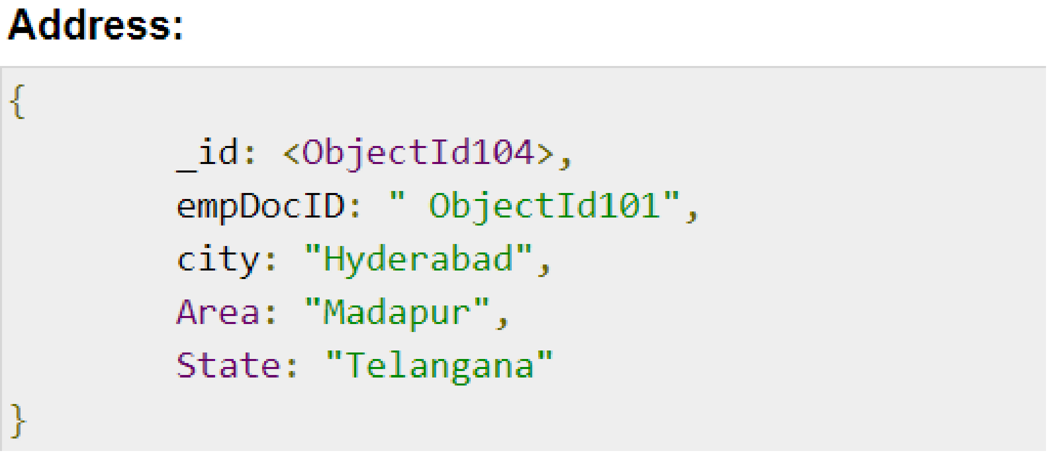
Personal_details:

```
{
        _id: <ObjectId102>,
        empDocID: " ObjectId101",
        First_Name: "Radhika",
        Last_Name: "Sharma",
        Date_Of_Birth: "1995-09-26"
}
```



Contact:

```
{
        _id: <ObjectId103>,
        empDocID: " ObjectId101",
        e-mail: "radhika_sharma.123@gmail.com",
        phone: "9848022338"
}
```



Address:

```
{
        _id: <ObjectId104>,
        empDocID: " ObjectId101",
        city: "Hyderabad",
        Area: "Madapur",
        State: "Telangana"
}
```

Design Notes

- Design your schema according to user requirements
- Combine objects into one document if you will use them together
- Since Mongodb does not have JOIN, make sure you do not need join between collections!
- Optimize your schema for most frequent use cases
- Pre-compute complex aggregations

Indexing

- Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form
- To create an index on the field “KEY” in an ascending order:

```
db.COLLECTION_NAME.createIndex({KEY:1})
```

- Also multiple fields:

```
db.mycol.createIndex({"title":1,"description":-1})
```

- This index can support a sort on `{"title":1,"description":-1}`
- createIndex has some options: background, unique, name, …

Indexing – cont.

- To drop index:

```
db.COLLECTION_NAME.dropIndex({KEY:1})
```

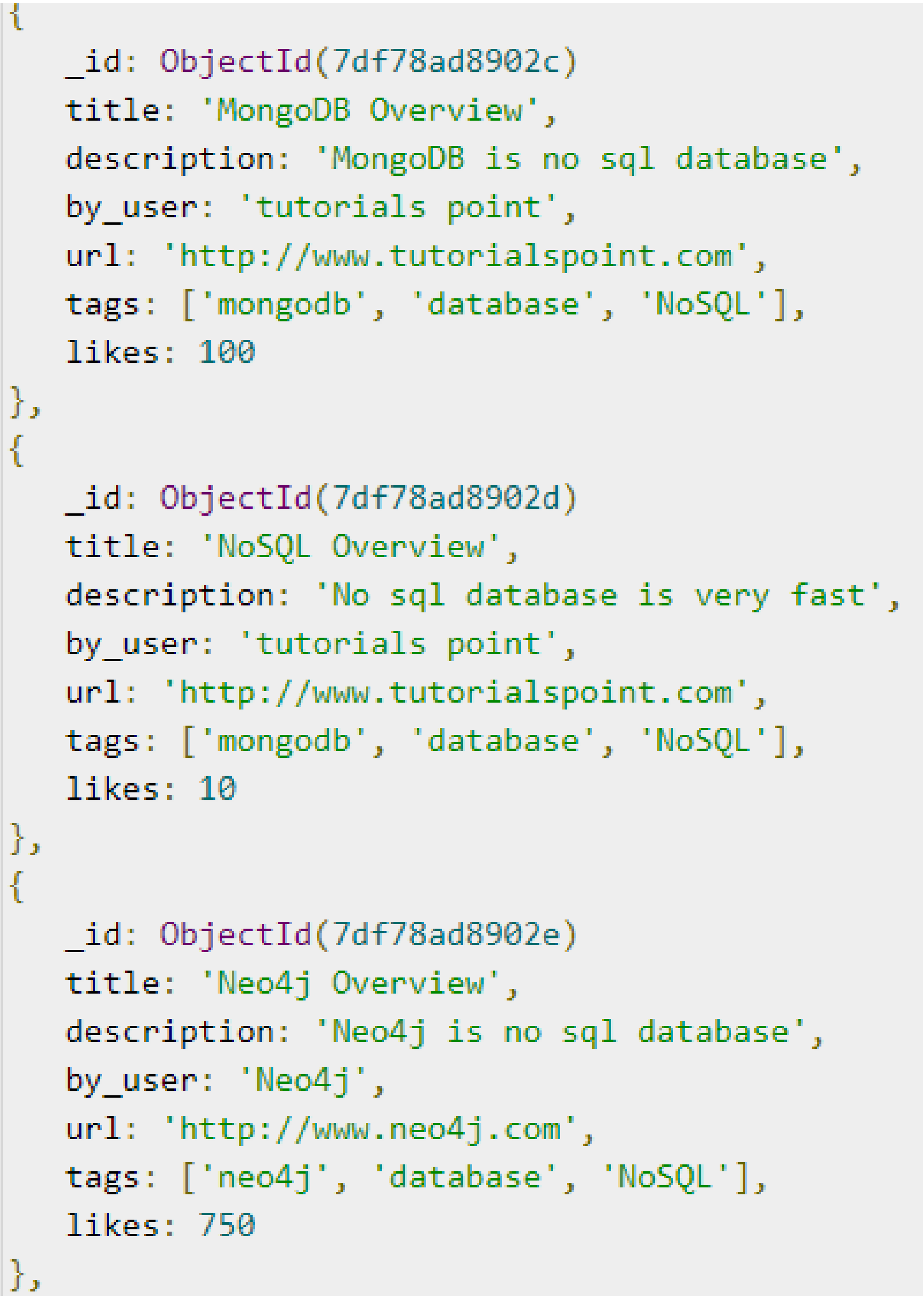
- See also:
  - createIndexes
  - dropIndexes
  - getIndexes

Aggregation

- Aggregations operations process data records and return computed results
- Equivalent to SQL "group by" and aggregation functions

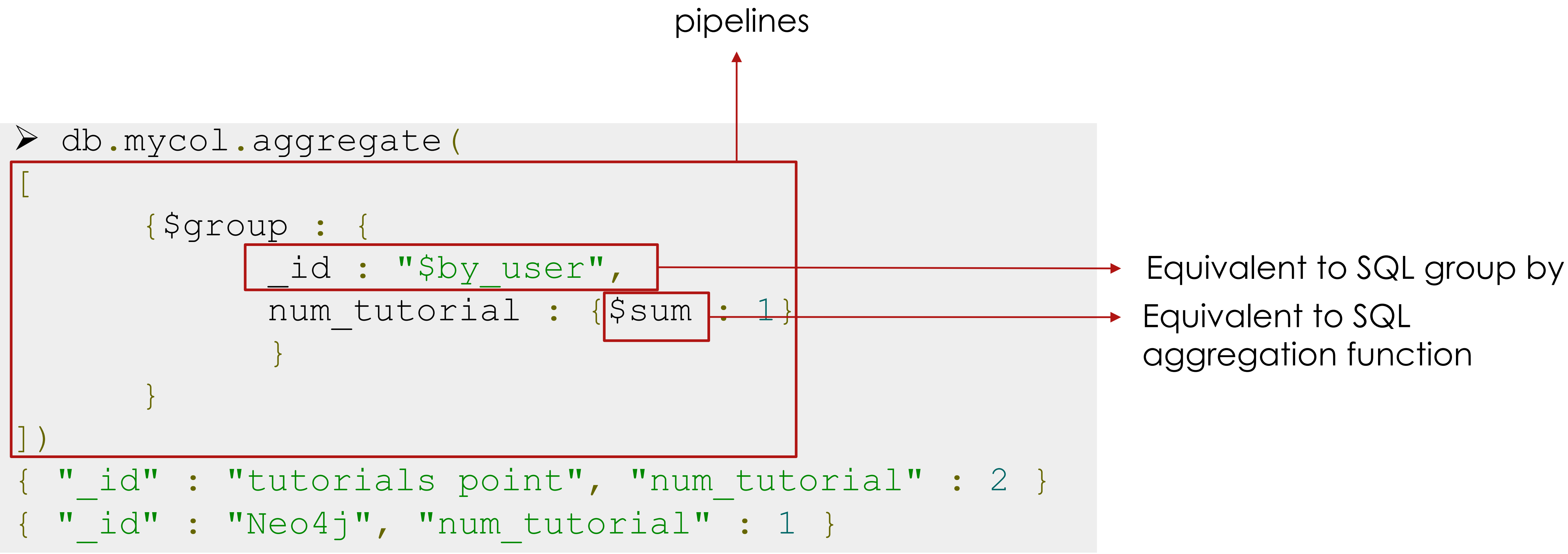
```
db.COLLECTION_NAME.aggregate(pipeline, options)
```

In a collection you have the following data:



```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
},
{
   _id: ObjectId(7df78ad8902d)
   title: 'NoSQL Overview',
   description: 'No sql database is very fast',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 10
},
{
   _id: ObjectId(7df78ad8902e)
   title: 'Neo4j Overview',
   description: 'Neo4j is no sql database',
   by_user: 'Neo4j',
   url: 'http://www.neo4j.com',
   tags: ['neo4j', 'database', 'NoSQL'],
   likes: 750
},
```

Aggregation example:
Counts the number of “by_user” values



pipelines

```
➢ db.mycol.aggregate(
[
      {$group : {
            _id : "$by_user",
            num_tutorial : {$sum : 1}
            }
      }
])
{ "_id" : "tutorials point", "num_tutorial" : 2 }
{ "_id" : "Neo4j", "num_tutorial" : 1 }
```

Equivalent to SQL group by

Equivalent to SQL aggregation function

Aggregation – cont.

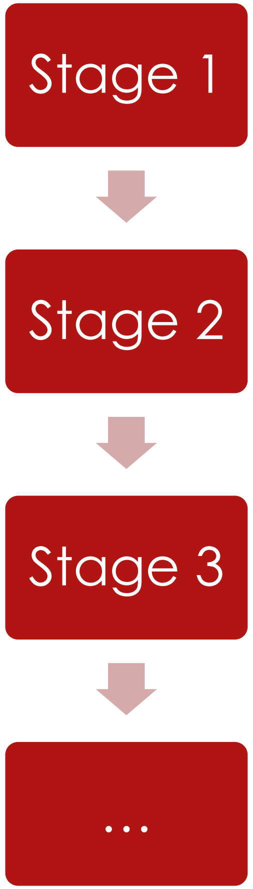
- Sums up “likes” values for each “by_user” value

```
db.mycol.aggregate([{$group : {
   _id : "$by_user",
   likes : {$sum : "$likes"}
}}])
```

- Also $avg, $min, $max

Aggregation Pipeline

- Aggregation steps
  - **$project**: same as find() project
  - **$match**: same as find() filter
  - **$group**: grouping and aggregation
  - **$sort**: same as sort()
  - **$skip**: same as skip()
  - **$limit**: same as limit()
  - **$unwind**: unpack the arrays
- All are optional



Aggregation Pipeline – cont.

- Selects documents with status equal to "A“
- Groups them by the cust_id field and calculates the total for each cust_id field from the sum of the amount field
- And sorts the results by the total field in descending order

```
db.orders.aggregate([
      { $match: { status: "A" } },
      { $group: { _id: "$cust_id", total: { $sum: "$amount" } } },
      { $sort: { total: -1 } }
])
```

Aggregation Pipeline – cont.

- To handle large datasets, set allowDiskUse option to true to enable writing data to temporary files

```
db.stocks.aggregate(
      [
            { $project : { cusip: 1, date: 1, price: 1, _id: 0 } },
            { $sort : { cusip : 1, date: 1 } }
      ],
      { allowDiskUse: true }
)
```

Some tips

- Set group null if there is no grouping:

```
db.orders.aggregate([ { $group: { _id: null, total: { $sum: "$amount" } } } ])
```

- You have a wide range of aggregation operators:

$sum	$first	$dayOfWeek	$size	$add	$toInt
$max	$last	$dayOfYear	$floor	$subtract	$toDecimal
$min	$hour	$sin	$ceil	$divide	$toLong
$avg	$dayofMonth	$cos	$multiply	$toDate	$toString

- For more reading:
  - https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/

$unwind

- Deconstructs an array field from the input documents to output a document for each element

```
{ $unwind: <field path> }
```

- Or:

```
{
      $unwind:
      {
            path: <field path>,
            includeArrayIndex: <string>,
            preserveNullAndEmptyArrays: <boolean>
      }
}
```

$unwind Example

- Consider this collection:

```
{ "_id" : 1, "item" : "ABC", price: NumberDecimal("80"), "sizes": [ "S", "M", "L"] }
{ "_id" : 2, "item" : "EFG", price: NumberDecimal("120"), "sizes" : [ ] }
{ "_id" : 3, "item" : "IJK", price: NumberDecimal("160"), "sizes": "M" }
{ "_id" : 4, "item" : "LMN", price: NumberDecimal("10") }
{ "_id" : 5, "item" : "XYZ", price: NumberDecimal("5.75"), "sizes" : null }
```

$unwind Example – cont.

```
db.inventory2.aggregate( [
       // First Stage
       { $unwind: { path: "$sizes", preserveNullAndEmptyArrays: true } },
       // Second Stage
       { $group:
              { _id: "$sizes", averagePrice: { $avg: "$price" } }
       },
       // Third Stage
       { $sort: { "averagePrice": -1 } }
] )
```

$unwind Example – cont.

- Stage 1: $unwind

```
{ "_id" : 1, "item" : "ABC", "price" : NumberDecimal("80"), "sizes" : "S" }
{ "_id" : 1, "item" : "ABC", "price" : NumberDecimal("80"), "sizes" : "M" }
{ "_id" : 1, "item" : "ABC", "price" : NumberDecimal("80"), "sizes" : "L" }
{ "_id" : 2, "item" : "EFG", "price" : NumberDecimal("120") }
{ "_id" : 3, "item" : "IJK", "price" : NumberDecimal("160"), "sizes" : "M" }
{ "_id" : 4, "item" : "LMN", "price" : NumberDecimal("10") }
{ "_id" : 5, "item" : "XYZ", "price" : NumberDecimal("5.75"), "sizes" : null }
```

$unwind Example – cont.

- Stage 2: $group

```
{ "_id" : "S", "averagePrice" : NumberDecimal("80") }
{ "_id" : "L", "averagePrice" : NumberDecimal("80") }
{ "_id" : "M", "averagePrice" : NumberDecimal("120") }
{ "_id" : null, "averagePrice" : NumberDecimal("45.25") }
```

- Stage 3: $sort

```
{ "_id" : "M", "averagePrice" : NumberDecimal("120") }
{ "_id" : "L", "averagePrice" : NumberDecimal("80") }
{ "_id" : "S", "averagePrice" : NumberDecimal("80") }
{ "_id" : null, "averagePrice" : NumberDecimal("45.25") }
```

Dump and Restore

```
mongodump --host="mongodb0.example.com:27017" --port=27017 --db=<db_name> --out=<pat
```

```
mongorestore --host="mongodb0.example.com:27017" --port=27017 --db=<db_name> <path>
```

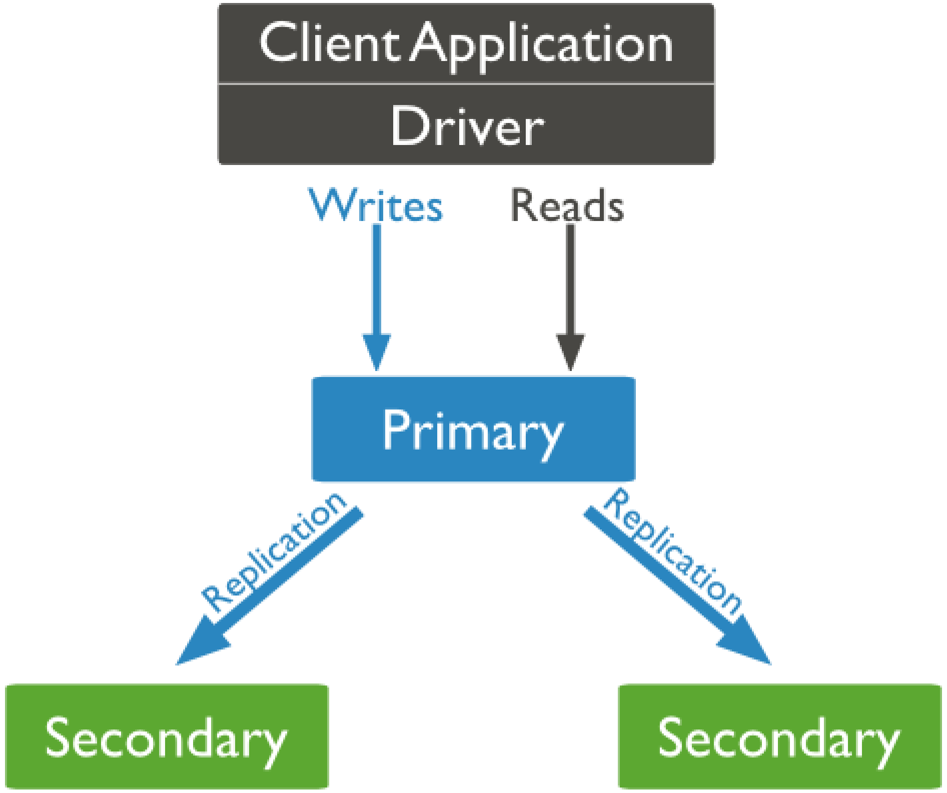
- For more options refer to the references
- See Also:
  - mongoexport
  - mongoimport

Replication

- Replication is the process of synchronizing data across multiple servers
- Cost
  - Provides redundancy
- Benefit
  - Increases data availability
  - Keeps your data safe
  - Disaster recovery
  - No downtime for maintenance

Replication – cont.

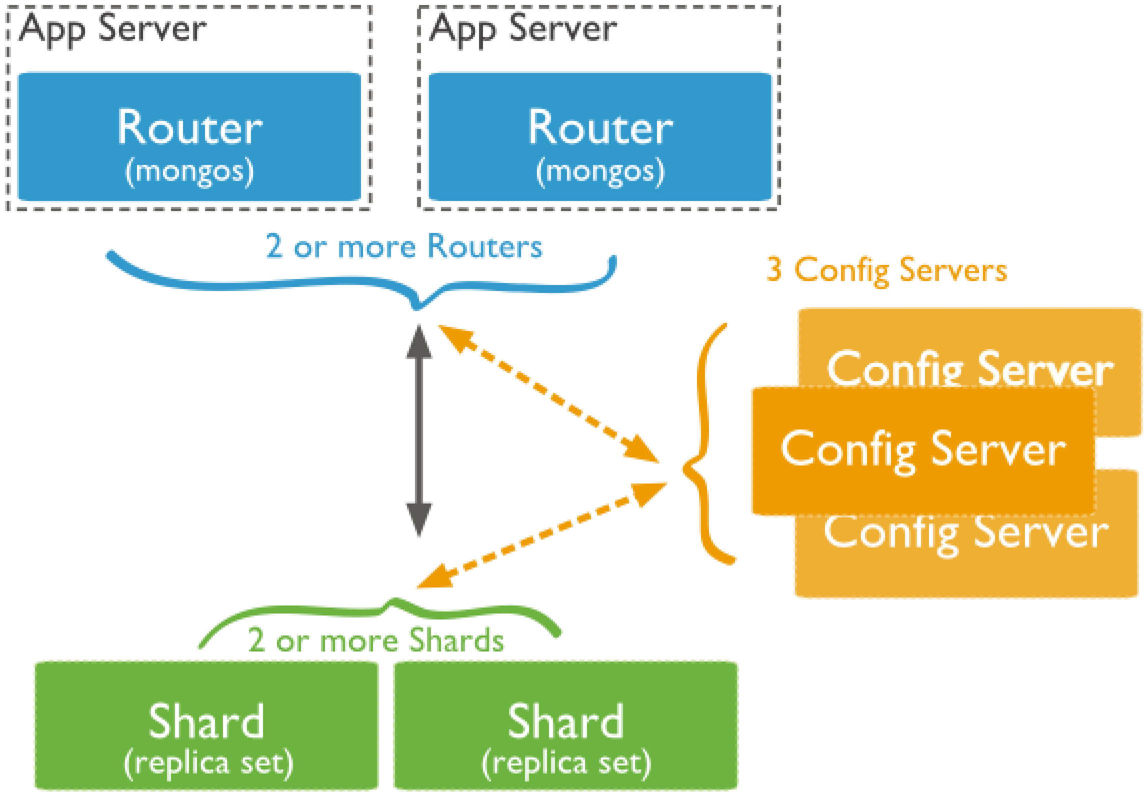
- Replica set:
  - 2 or more nodes
  - A primary node and many secondary nodes
  - At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected
  - After the recovery of failed node, it again join the replica set and works as a secondary node
- To see how to setup a replica set refer to the references



Sharding

- Sharding is the process of storing data records across multiple machines on a single server to handle data growth
- Sharding vs Replication:
  - Benefit: no need to buy any server or add any RAM, CPU, Disk
  - Cost: increased complexity in infrastructure and maintenance

Sharding – cont.



References

- https://www.tutorialspoint.com/mongodb
- https://docs.mongodb.com/manual/reference

Any Question?