DATA MODELLING IN MONGODB

- Data Modelling is one of the crucial steps in building robust databases that can support your application needs for a long time
- Even though NoSQL has a dynamic schema, it doesn't mean that you don't have to think about schema design while developing your database.
- The main consideration here is when we need to build relationships between multiple documents, whether to create **embedded documents** or to use **references**.

DATA MODELLING IN MONGODB

- **Embedded Documents** – One of the most essential features of data modelling in MongoDB is that you can embed one document inside another document . All the embedded documents will be stored as an array with respect to a field.

- Ex – The order related information for all the orders are embedded along with the customer details

```
{
    "_id": 1,
"Cust_id": 10101,
   "Name": "Vishwa",
 "Orders":

        {
           "order_id" : 101,
           "Product Name" : "Plantronics",
           "Sales": 2400
        },

        {
           "order_id" : 102,
           "Product Name" : "Novimex",
           "Sales" : 1800
        }
}
```

DATA MODELLING IN MONGODB

- **References** – References means that a document from one collection contains a reference to another document from a different collection

- This is like how RDBMS work, where we have information stored in multiple tables and are connected by certain keys or fields.

- Ex : The order_id field from the ***Orders*** collection can be stored in an array field called Orders in the ***Customers*** collection, thereby storing the information for all the orders purchased by a customer.

```
#Customers Collection
{
      "_id": 1,
  "Cust_id": 10101,
     "Name": "Vishwa",
  "Orders" : [101,102,....]
}
```

```
#Orders Collection
{
 "order_id" : 101,
 "Product Name": "Plantronics",
 "Sales": 2400
}

{
 "order_id" : 101,
 "Product Name": "Novimex",
 "Sales": 1800
}
```

MODELLING RELATIONSHIPS

- There can be various types of entity-entity relationship between the data that we use in MongoDB. Typically these relationships are:
  - One to One
  - One to Many
  - Many to Many
- Depending on the type of data you have, you can store your data in databases either in a ***denormalised*** embedded structure or a ***normalised*** reference structure.

MODELLING RELATIONSHIPS

- **One to One relationship**
  - One to one relationship exists where each document relates to exactly one other document and vice-versa
  - In such cases, you can use embedded documents to store the information rather than utilizing references to store them.

```
{
    "_id":1,
 "Emp_id": 10101,
   "Name": "Vishwa",
 "Personal Details":
        {
           "Aadhar Card" : 1234,
           "City" : "Bangalore"
        }
}
```

MODELLING RELATIONSHIPS

- **One to One relationship with subsets** :
  - When the embedded documents contain a lot of unnecessary fields, it can slow down your read operations by putting load on the memory.
  - In such cases, subset the entire document into the fields that are more frequently read and less frequently read.
  - The more frequently read fields will remain embedded in the main document in the same collection, whereas the less frequently ones will be stored in another document from a separate collection and will contain a reference.

MODELLING RELATIONSHIPS

- **One to Many Relationship**:
  - One document can relate to many other documents but each of the other document relates back to only one document.

```
{

    "_id": 1,
"Cust_id": 10101,
   "Name": "Vishwa",
 "Orders":

        {
           "order_id" : 101,
           "Product Name  : "Plantronics",
           "Sales": 2400
        },

        {
           "order_id" : 102,
           "Product Name" : "Novimex",
           "Sales" : 1800
        }
}
```

MODELLING RELATIONSHIPS

- **One to Many Relationship**:
  - One document can relate to many other documents but each of the other document relates back to only one document.
  - Both embedded documents as well as references can be utilized to depict such a relationship
  - If the number of documents related to one document is less, we can go for embedded documents, else we can store them in separate collections and use references to relate between them.

```
{
    "_id": 1,
"Cust_id": 10101,
   "Name": "Vishwa",
 "Orders":

        {
           "order_id" : 101,
           "Product Name": "Plantronics",
           "Sales": 2400
        },

        {
           "order_id" : 102,
           "Product Name" : "Novimex",
           "Sales" : 1800
        }
}
```

MODELLING RELATIONSHIPS

- **One to Many Relationship with subsets**:
  - We can also use subsets here as well like in the case of one-one relationship
  - Here we can subset the embedded documents into one that is more frequently read and one that is less frequently read.
  - After that we can store the less frequently read ones in a separate collection and add a reference for the same.

MODELLING RELATIONSHIPS

- **Many - Many Relationship**:
  - One document can relate to many other documents and each of those other documents can relate to multiple other documents as well.
  - Such a relationship is best explained using references else there will be lot of redundancy in data if we use embedded documents

```
{
        "_id": 1,
    "Emp_id": 10101,
      "Name": "Vishwa",
 "Coworkers": [10102,10103,..]

}

{
        "_id": 1,
    "Emp_id": 10102,
      "Name": "Rohan",
 "Coworkers": [10101,10103,..]

}
```

DATA MODELLING IN MONGODB

- **Using $lookup**
  - Earlier in SQL, we had used joins to combine records from multiple tables
  - Like joins, the $lookup aggregation operation in MongoDB helps us in combining collections based on a common field.
  - Performs a left outer join to a collection in the same database to filter in documents from the “joined” collection for processing.
  - Once combined, further data wrangling can be done on the resulting list of documents.

DATA MODELLING IN MONGODB

- **$lookup syntax -**

```
{
    $lookup:
      {
        from: <collection to join>,
        localField: <field from the input documents>,
        foreignField: <field from the documents of the "from" collection>,
        as: <output array field>
      }
}
```

DATATYPES IN MONGODB

- The fields in your documents store the necessary information that you want.
- The fields can store values of several datatypes.

DATATYPES IN MONGODB

- The fields in your documents store the necessary information that you want.
- The fields can store values of several datatypes.
- Let's start with '***_id***' field.
  - Required in every MongoDB document, is unique and acts as a primary key.
  - By default MongoDB uses a 24-digit unique identifier which is of **ObjectId** type.
  - ObjectIDs generally store document IDs, are of 12-byte size, consisting of:
    - 4-byte time stamp value
    - 5-byte random value
    - 3 byte incrementing counter, initialized to a random value.

DATATYPES IN MONGODB

- **String** – This is the most used datatype
- **Integer** – Used to store a numerical value.
- **Boolean** – Used to store a Boolean (true/ false) value.
- **Double** – Used to store floating point values.
- **Object** – Used for embedded documents.
- **Arrays** – Used to store a list of multiple values into one key.
- **Date** – This datatype is used to store the current date or time in UNIX time format.

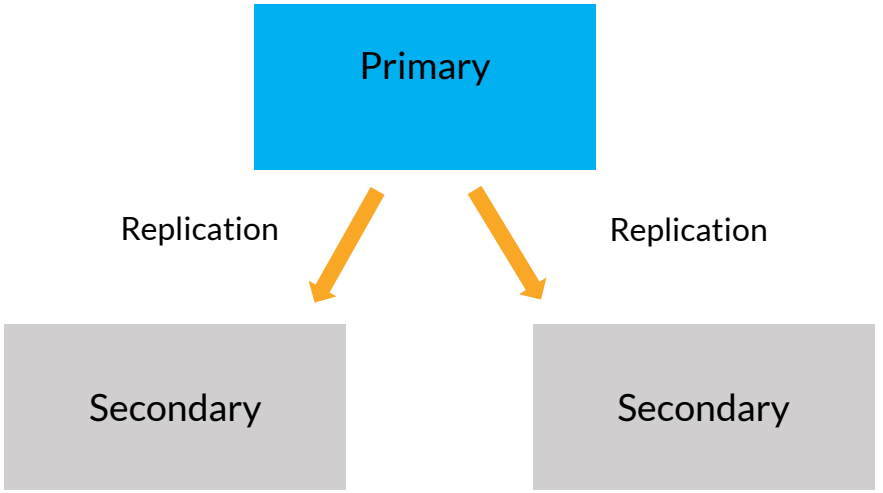
REPLICATION

- Replication in MongoDB essentially refers to the creation and maintaining of replica sets
- A replica set in MongoDB is a group of mongod processes that maintain the same data set
- Highly useful in production deployment as they provide redundancy and high availability

REPLICATION

- As the name suggests, we want to create a ***replica***, or a copy of the data called the secondary data. A replica set consists of a primary and 2 or more secondaries

- All of them have the same collection or documents. The process of creating replica sets on different servers is called **replication.**



REPLICATION

- **Advantages of Replication**
  - **Data availability -** If a server goes down, another server can take the workload. It improves the fault tolerance and if a server goes down, we have backups of the same data on other servers as well.
  - **Improve efficiency -** In case of requests for collections containing huge number of records (say in millions),the client can send requests to multiple replica sets instead of a single server to improve efficiency.

REPLICATION

- **Replica Set Architecture**
  - Only **primary node** receives the **write operations**.
  - The **secondary nodes** only accept **read operations**
  - The replication of data from primary to secondary happens after the write operations are done. The primary records all the write operations in the **operations log** or **oplog**.
  - The secondary nodes then replicate the primary's operation logs and make the exact same changes to their own replica sets so that they also reflect the primary data set.

REPLICATION

- **Election for primary**
  - As we discussed earlier only the primary node accepts the write operations whereas the secondary nodes contain the replica sets and only read operations are allowed on them
  - Now if a primary node goes down due to maintenance or some other issues, the secondary nodes will call for an election to become the primary so that the write operations can continue.
  - This generally takes some time and during the election period, the write operations are stalled and only read operations can be done from the secondary nodes

SHARDING

- **Scaling systems -** Before we get to know what ***sharding*** is, it's important to revisit the concept of scaling systems
  - We need to scale systems because a single server at some time or the other will be overloaded with high query rates and the CPU will be challenged significantly.
  - One way of scaling systems is ***vertical scaling*** - increasing RAM, CPU, storage space of a single server.

SHARDING

- **Scaling systems -** Before we get to know what ***sharding*** is, it's important to revisit the concept of scaling systems
  - We need to scale systems because a single server at some time or the other will be overloaded with high query rates and the CPU will be challenged significantly.
  - One way of scaling systems is ***vertical scaling*** - increasing RAM, CPU, storage space of a single server.
  - ***Horizontal scaling*** involves dividing the entire data and workload over multiple servers to keep up with the demand, which can lead to better performance than just improving a single server.

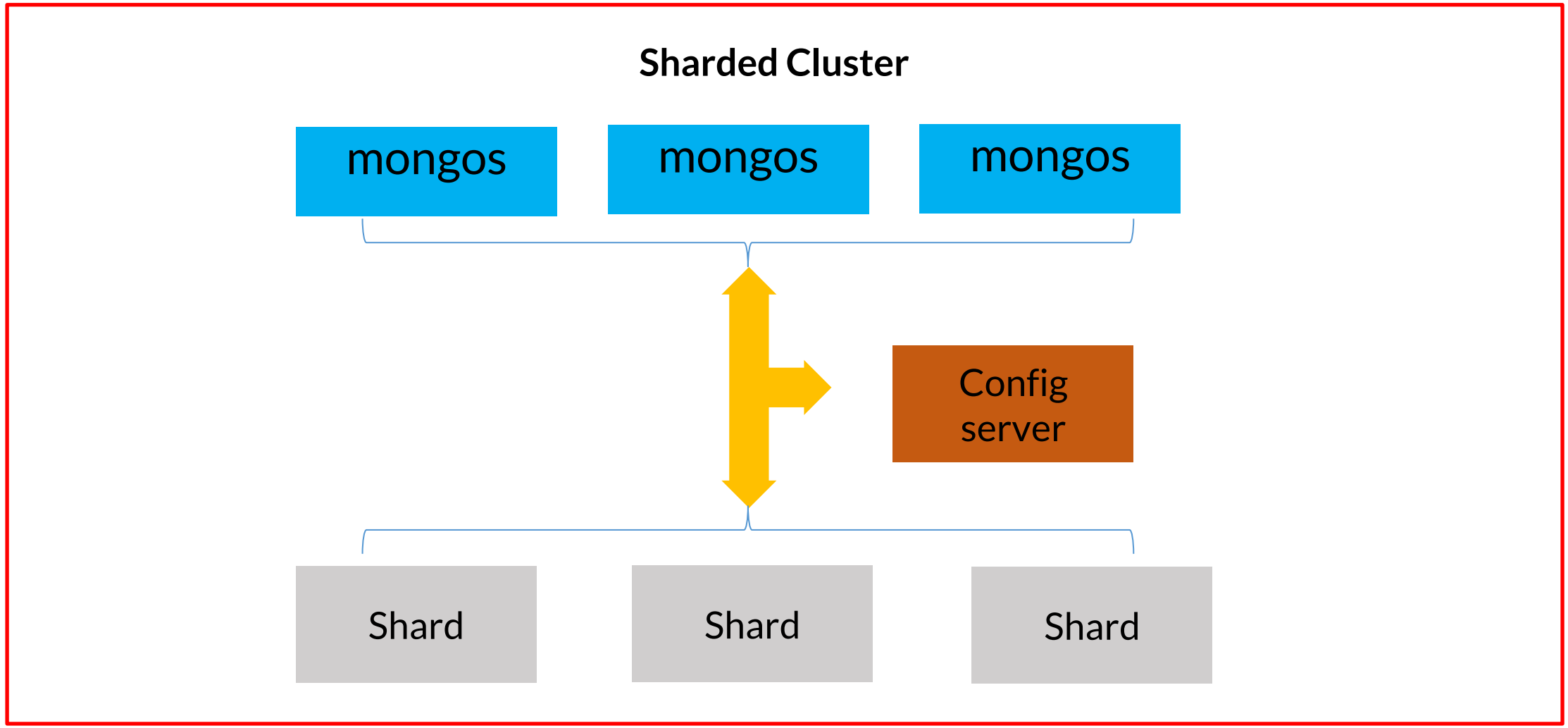
SHARDING

- Sharding is a method utilised by MongoDB to support huge datasets that have high read-write operations.
- Consists of mainly dividing the entire collection into several chunks and then storing each of these chunks in separate replica sets or shards.
- Querying then takes place on a corresponding shard and not on the entire collection.

SHARDING

- **Sharded Cluster in MongoDB**
  - **Shard** – A portion of the entire data is collected in a shard which can be deployed as a replica set.
  - **Mongos** - An interface between the client applications and the sharded cluster and routes the queries from the client to the corresponding shards that are required for an operation.
  - **Config server** - Store metadata and configuration settings for the entire sharded cluster.

SHARDING


Sharded Cluster
mongos
mongos
mongos
Config server
Shard
Shard
Shard

SHARDING

- **Shard key**
  - MongoDB uses the shard key to distribute the collection’s documents across shards.
  - Shard key is crucial for efficiency and scalability purposes and therefore should be done strategically.

SHARDING

- **Shard key**
  - MongoDB uses the shard key to distribute the collection's documents across shards.
  - Shard key is crucial for efficiency and scalability purposes and therefore should be done strategically.
  - A couple of methods exist for creating the shard key
    - **Hash based shard key**: Creates a hash value from the given field chosen
    - **Range based shard key**: This involves dividing data into ranges based on the shard key values

SHARDING

- **Querying from a Sharded Collection**
  - Query is made by the client application which should contain the shard key
  - When a query includes a shard key, the mongos uses the config database to route the queries to the corresponding shard instead of searching the entire collection.
  - The query results are then routed back to the client through the mongos.

SHARDING

- **Advantages of Sharding**
  - The read/write operations can scale horizontally
  - Helps in handling storage capacity for future use.
  - High availability of data.