Insert Operations in MongoDB

- In MongoDB, insert operations refer to the process of adding new documents (also known as records or rows in traditional databases) to a collection.
- MongoDB is a NoSQL database that stores data in a flexible, JSON-like format called BSON (Binary JSON).

Before inserting anything, you need to have idea about the following terms:

- **Documents:** MongoDB stores data in collections, which are analogous to tables in relational databases. Each collection contains documents, which are similar to rows in relational databases but are JSON-like objects. Documents consist of key-value pairs, where keys are strings and values can be various data types, including other documents or arrays.
- **Insertion:** To insert a document into a MongoDB collection, you typically use the insertOne() or insertMany() method provided by the MongoDB driver. The insertOne() method inserts a single document, while insertMany() can insert multiple documents at once.
- **Document Structure:** Before inserting documents into a collection, it's essential to define the structure of the documents. While MongoDB is schema-less in nature, meaning each document in a collection can have a different structure, having a consistent structure across documents can make querying and managing data more manageable.
- **Document IDs:** Each document in a MongoDB collection has a unique identifier called an ObjectID by default. This identifier is automatically generated by MongoDB and ensures that each document can be uniquely identified within the collection.
- **Atomicity:** MongoDB provides atomic operations at the document level. This means that when you perform an insert operation, it either fully succeeds or fails. There's no partially completed insert operation that leaves the database in an inconsistent state.
- **Indexes:** MongoDB allows you to define indexes on fields within documents. Indexes can improve query performance by enabling MongoDB to quickly locate documents based on indexed fields.
- **Write Concern:** MongoDB allows you to specify the level of acknowledgment required from the database when performing write operations. Write concern

options include acknowledging the write operation from the primary server, a subset of replica set members, or all replica set members.

- **Validation:** MongoDB provides validation rules that you can apply to collections to enforce data integrity. Validation rules can include enforcing the presence of certain fields, data types, or even custom validation logic using JavaScript functions.

Overall, MongoDB's approach to insert operations provides flexibility, scalability, and performance for handling various types of data in modern applications.

Various ways to perform insert operations in MongoDB

1. Insert a Single Document

To insert a single document into a collection, you can use the insertOne() method. This method takes an object representing the document to be inserted as its parameter.

```
db.collectionName.insertOne({ key: "value", key2: "value2" });
```

Example :

```
db.personal_Details.insertOne({id:45,name:"garima singh", city:"jaipur"})
```

If you want to see the inserted document then use command,

```
db.collectionName.find()
```

Example:

```
db.personal_Details.find()
```

If you want to insert an array as a key_value then use command

```
db.collectionName.insertOne({ key: "value", key2: "value2" , key3:[“value_x”, “value_y”, “value_z”]});
```

Example:

```
db.personal_Details.insertOne({id:32, name:"aman bansal", city:"agra", skills:["java full stack", "mern stack", "DevOps", "UI/Ux"]})
```

If you want to insert an object as a key_value then use command

```
db.collectionName.insertOne({ key: "value", key2: "value2" , key3:{key_x:”value_x”, key_y:”value_y”, key_z:”value_z”}});
```

Example:

```
db.personal_Details.insertOne({id:49,name:"pranjal",status:{frontend:"perfect",backend:"perfect",uiux:"excellent"}})
```

2. Insert Multiple Documents

To insert multiple documents into a collection, you can use the insertMany() method. This method takes an array of objects, with each object representing a document to be inserted.

```
db.collectionName.insertMany([
  { key: "value1", key2: "value2" },
  { key: "value3", key2: "value4" },
  { key: "value5", key2: "value6" }
]);
```

Example:

```
db.personal_Details.insertMany([
        {id:32, name:"tushar gupta", city:"Ghaziabad"},
        {id:47, name:"stuti singh", city:"Lucknow"},
        {id:54,name:"nikhil garg", city:"Agra"},
        {id:23,name:"neha garg", city:"mathura"}
]);
```

If you want to see the inserted document then use same command again,

```
db.collectionName.find()
```

Example:

```
db.personal_Details.find()
```

if you want to see the specific inserted document on the basis of respective key & value then use the command

```
db.collectionName.find({key:"value"})
```

Example:

```
db.personal_Details.find({id:54});
```

or `db.personal_Details.find({name:"garima singh"});`

or `db.personal_Details.find({id:23, name:"neha garg"});`

Inside the query object, if you specify the wrong key field and value pair then you will get nothing

3. Bulk Write Operations: MongoDB allows for bulk write operations, which include multiple insert operations executed at once.

```
db.collection.bulkWrite([
   { insertOne: { document: { ... } } },
   { insertOne: { document: { ... } } },
   // more operations
]);
```

Example:

```
db.personalDetails.bulkWrite([
... {insertOne:{document:{id:23,name:"priya"}}},
... {insertOne:{document:{id:17,name:"rishabh gupta"}}},
... {insertOne:{document:{id:19,name:"kanika sharma", city:"delhi"}}}]);
```

4. Insert with Explicit _id

By default, MongoDB automatically generates an _id field (primary_key/unique_id) for each document if it's not provided. However, you can specify the _id field explicitly for the document being inserted.

```
db.collectionName.insertOne({ _id: ObjectId("unique_id"), key: "value", key2: "value2" });
```

Example:

```
db.personalDetails.insertMany([
   { _id: 1, name: "Neha Rathore", age: 23 },
   { _id: 2, name: "Rajat Sharma", age: 22 },
   { _id: 3, name: "Faisal Iqbal", age: 22 }
]);
```

5. Insert with Validation

You can enable document validation during the insert operation to enforce a schema on the inserted documents. Document validation ensures that all documents meet certain criteria before they are inserted into the collection.

Example

```
db.createCollection("collectionName", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["id",”name”,”city”],
      properties: {
        id: {
          bsonType: "number",
          description: "must be a number"
        },
        name: {
          bsonType: "string",
          description: "must be a string"
        },
        city: {
          bsonType: "string",
          description: "must be a string"
        }
      }
    }
  },
validationAction:’error’
});
```