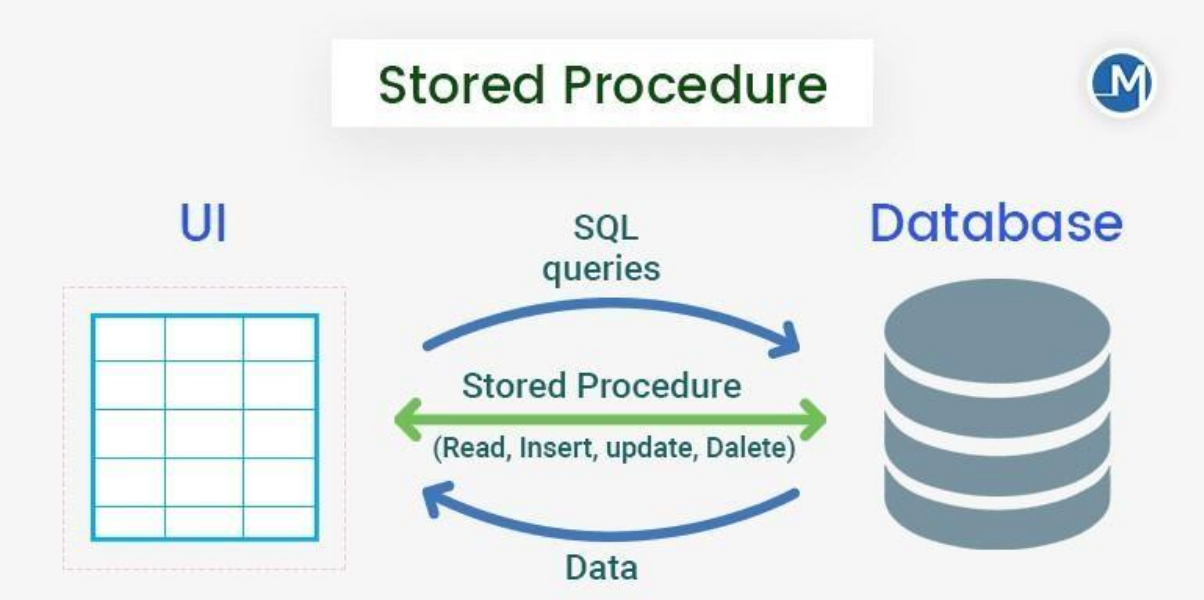
EX.NO:12 **CREATION OF DOCUMENT DATABASE USING MONGODB**
DATE:

AIM:

To study the Creation of Document database using MongoDB

PROCEDURE:

MongoDB's documentation model is based on a collection of documents that are stored in a database. Each document consists of one or more fields that contain data in key-value pairs. The data can be of various types, including strings, integers, arrays, and subdocuments. MongoDB's documentation model is a non-relational, or NoSQL,model, which means that it does not use tables or fixed schemas to store data.



In addition to collections and indexes, MongoDB also supports several advanced features, such as aggregation pipelines, which allow you to perform complex data transformations on a collection, and sharding, which allows you to horizontally scale a database across multiple servers.

MongoDB's documentation model has several benefits, including:

1. Flexible data modeling: MongoDB's document model allows you to store data in a flexible and dynamic manner, without the need to define a schema beforehand.
2. High performance: MongoDB's indexing and sharding features allow you to scale your database to handle large amounts of data and high levels of traffic.
3. Easy integration: MongoDB integrates well with many programming languages and frameworks, making it easy to use in modern web applications.

Overall, MongoDB's documentation model is a powerful and flexible way to store and manage data, and it is well-suited for modern web applications that require scalability and performance.

CRUD operations on a single document:

1. **Create:** To create a new document in MongoDB, you can use the `insertOne()` method, which takes a document as its argument and inserts it into the collection.
   Syntax: db.collection_name.insertOne({field1: value1, field2: value2, ...}) Example: db.inventory.insertOne({item: "book", qty: 10, price: 15.99})

2. **Read:** To retrieve a single document from a collection in MongoDB, you can use the `findOne()` method, which takes a query object as its argument and returns the first matching document.
   Syntax: db.collection_name.findOne({field1: value1})
   Example: db.inventory.findOne({item: "book"})

3. **Update:** To update a single document in MongoDB, you can use the `updateOne()` method, which takes two arguments: a query object to identify the document to be updated, and an update object that specifies the changes to be made.
   Syntax: db.collection_name.updateOne({field1: value1}, {$set: {field2: value2}}) Example: db.inventory.updateOne({item: "book"}, {$set: {qty: 20}})

4. **Delete:** To delete a single document from a collection in MongoDB, you can use the `deleteOne()` method, which takes a query object as its argument and removes the first matching document. Syntax: db.collection_name.deleteOne({field1: value1}) Example: db.inventory.deleteOne({item: "book"})

CRUD operations on multiple documents:

1. **Create:** To create multiple documents in MongoDB, you can use the `insertMany()` method, which takes an array of documents as its argument and inserts them into the collection.
   Syntax: db.collection_name.insertMany([{field1: value1, field2: value2, ...}, {field1: value1, field2: value2, ...}, ...])
   Example: db.inventory.insertMany([{item: "book", qty: 10, price: 15.99}, {item: "pen", qty: 50, price: 2.99}, {item: "paper", qty: 100, price: 1.99}])

2. **Read:** To retrieve multiple documents from a collection in MongoDB, you can use the `find()` method, which takes a query object as its argument and returns a cursor to the matching documents.
   Syntax: db.collection_name.find({field1: value1}) Example: db.inventory.find({qty: {$gt: 10}})

3. **Update:** To update multiple documents in MongoDB, you can use the `updateMany()` method, which takes two arguments: a query object to identify the documents to be updated, and an update object that specifies the changes to be made.

Syntax: db.collection_name.updateMany({field1: value1}, {$set: {field2: value2}})
Example: db.inventory.updateMany({qty: {$lt: 20}}, {$set: {price: 19.99}})

4. **Delete:** To delete multiple documents from a collection in MongoDB, you can use the `deleteMany()` method, which takes a query object as its argument and removes all matching documents.
   Syntax: db.collection_name.deleteMany({field1: value1}) Example:
   db.inventory.deleteMany({qty: {$lt: 10}}

RESULT:

Thus the study for the creation of document database using MongoDB has been done successfully.