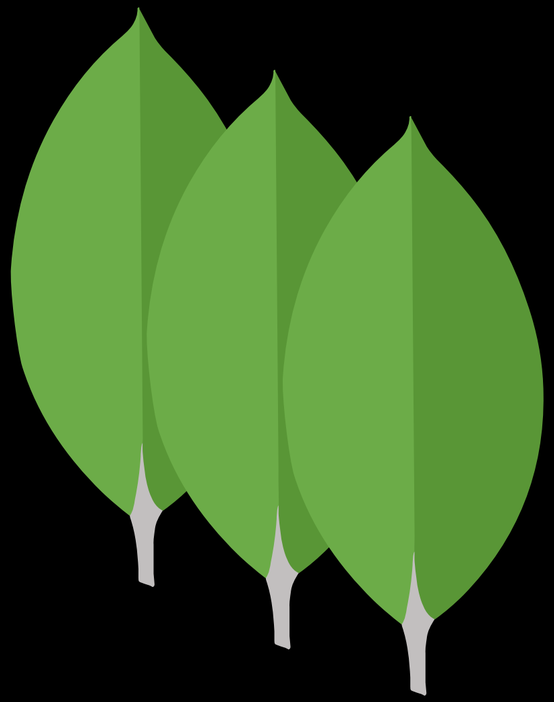

Data Types

MongoDB supports a variety of data types,

- Numbers: MongoDB supports both integer and floating-point numbers.
- Strings: MongoDB supports strings of any length.
- Booleans: MongoDB supports booleans, which are true or false values.
- Arrays: MongoDB supports arrays of any data type, including numbers, strings, booleans, and other arrays.
- Embedded documents: MongoDB supports embedded documents, which are documents that are nested within other documents.

Collections

Collections are the fundamental storage units for data in MongoDB. A collection is a group of documents that share a common schema. Documents in a collection must have the same fields, but the values of those fields can vary.

To create a collection, you can use the createCollection() method in the MongoDB shell.

```
db.createCollection("users")
```

To manage collections, you can use the showCollections() method to list all collections in the current database. You can also use the dropCollection() method to delete a collection

CRUD stands for Create, Read, Update, and Delete. These are the four basic operations that you can perform on documents in a MongoDB collection.

To create a document, use the insertOne() or insertMany() methods.

```
db.users.insertOne({
 "name": "John Doe",
"email": "johndoe@example.com",
 "age": 30
 })
```

To read documents, you can use the find() method. The find() method can take a query parameter to filter the results. For example, the following command finds all documents in the users collection where the age field is greater than 25:

```
db.users.find({ "age": { $gt: 25 } })
```

To update documents, you can use the updateOne() or updateMany() methods. The updateOne() method updates a single document, while the updateMany() method updates all documents that match the query parameter.

```
db.users.updateOne({ "_id": 12345 }, { $set: { "age": 31 } })
```

To delete documents, you can use the deleteOne() or deleteMany() methods. The deleteOne() method deletes a single document, while the deleteMany() method deletes all documents that match the query parameter.

```
db.users.deleteOne({ "_id": 12345 })
```

populate()

Population is the process of automatically replacing the specified paths in the document with document(s) from other collection(s). We may populate a single document, multiple documents, a plain object, multiple plain objects, or all objects returned from a query.

```
findOneAndUpdate(
 { _id: req.params.id, 'rating._id': req.body._id },
 {
 $set: {
 'rating.$.rating': req.body.rating,
 'rating.$.review': req.body.review, },
 }
 );
```

```
findOneAndUpdate(
 { _id: req.params.id },{
 $push: {
 rating: {
 rating: req.body.rating,
 review: req.body.review,},},}
 );
```

```
findOneAndUpdate(
 { _id: req.params.id },
 { $pull: { rating: { _id: req.body._id } } }
 )
```

```
populate('sellerId')

populate('sellerId', 'name address')

populate('sellerId').populate('categoryId')

populate(['categoryId', 'sellerId'])
```

aggregate([])

The aggregate method allows you to perform complex data processing and transformation operations on the documents in a collection. It provides a powerful way to perform operations like filtering, grouping, sorting, and projecting data.

$MATCH

In MongoDB, the $match stage is a crucial part of the aggregation pipeline that allows you to filter and select only the documents that match specified criteria. It's similar to the find() method in MongoDB

```
{ $match: { field1: value1, field2: value2, ... } }
```

$MATCH

In MongoDB, the $match stage is a crucial part of the aggregation pipeline that allows you to filter and select only the documents that match specified criteria. It's similar to the find() method in MongoDB

```
{ $match: { field1: value1, field2: value2, ... } }
```

```
{ $match: { age: { $gte: 18 }, isActive: true } }
```

Logical operators like $and, $or, $not, and $nor can be used for more complex matching

```
{ $match: { $or: [ { age: { $gte: 18 } }, { isAdmin: true } ] } }
```

You can match documents based on the existence of a field.

```
{ $match: { email: { $exists: true } } }
```

$GROUP

In MongoDB, the $group stage is a fundamental part of the aggregation pipeline that allows you to group documents based on certain criteria and perform aggregation operations within each group

```
const aggregationPipeline = [
  {
    $group: {
      _id: '$product', // Group by the '
      totalRevenue: { $sum: '$amount' }
    }
  }
];
```

```
{
      $match: {
        quantity: { $gte: 1 },
        totalAmount: { $gte: 100 },
      },
    },
    {
      $group: {
        _id: '$product',
        totalQuantity: { $sum: '$quantity' },
        totalRevenue: { $sum: '$totalAmount' },
      },
    },
```

```
{
  $group: {
    _id: "$product",
    totalSales: { $sum: "$amount" },
    averageQuantity: { $avg: "$quantity" }
  }
 }
```

- **$sum**: Calculates the sum of a specific field within the group.
- **$avg**: Computes the average of a specific field within the group.
- **$min** and **$max**: Determine the minimum and maximum values of a field within the group.