Course: 501-1: Advanced Web Designing

Unit-1: Concepts of NoSQL: MongoDB

UNIT-1 AWD

Concepts of NoSQL

What is a NoSQL database?

When people use the term “NoSQL database,” they typically use it to refer to any non-relational database. Some say the term “NoSQL” stands for “non SQL” while others say it stands for “not only SQL.” Either way, most agree that NoSQL databases are databases that store data in a format other than relational tables.

Brief history of NoSQL databasesNoSQL databases emerged in the late 2000s as the cost of storage dramatically decreased. Gone were the days of needing to create a complex, difficult-to-manage data model in order to avoid data duplication. Developers (rather than storage) were becoming the primary cost of software development, so NoSQL databases optimized for developer productivity.

Advantages and Disadvantages of NoSQL

Major advantages of NoSQL databases include:

(i) Flexible Data Model:

NoSQL databases are highly flexible as they can store and combine any type of data, both structured and unstructured, unlike relational databases that can store data in a structured way only.

(ii) Evolving Data Model :

NoSQL databases allow you to dynamically update the schema to evolve with changing requirements while ensuring that it would cause no interruption or downtime to your application.

(iii) Elastic Scalability:

NoSQL databases can scale to accommodate any type of data growth while maintaining low cost.

(iv) High Performance:

NoSQL databases are built for great performance, measured in terms of both throughput (it is a measure of overall performance) and latency (it is the delay between request and actual response).

(v) Open-source:

NoSQL databases don't require expensive licensing fees and can run on inexpensive hardware, rendering their deployment cost-effective.

Major disadvantages of NoSQL databases are:

(i) Lack of Standardization:

There is no standard that defines rules and roles of NoSQL databases. The design and query languages of NoSQL databases vary widely between different NoSQL products – much more widely than they do among traditional SQL databases.

(ii) Backup of Database:

Backups are a drawback in NoSQL databases. Though some NoSQL databases like MongoDB provide some tools for backup, these tools are not mature enough to ensure proper complete data backup solution.

(iii) Consistency:

NoSQL puts a scalability and performance first but when it comes to a consistency of the data NoSQL doesn't take much consideration so it makes it little insecure as compared to the relational database e.g., in NoSQL databases if you enter same set of data again, it will take it without issuing any error whereas relational databases ensure that no duplicate rows get entry in databases.

What is MongoDB?

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)

Sample Document

Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB in AWD',
  description: 'MongoDB is no sql database',
  by: 'Ritu Bhatiya',
  url: 'http://www.samplewebsitename.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments: [
    {
      user:'user1',
      message: 'My first comment',
      dateCreated: new Date(2022,1,20,2,15),
```

```
      like: 0
    },
    {
      user:'user2',
      message: 'My second comments',
      dateCreated: new Date(2022,1,25,7,45),
      like: 5
    }
  ]
}
```

_id is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide _id while inserting the document. If you don't provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

MongoDB supports many datatypes. Some of them are –

- String – This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 (Unicode Transformation Format) valid.
- Integer – This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- Boolean – This type is used to store a boolean (true/ false) value.
- Double – This type is used to store floating point values.
- Min/ Max keys – This type is used to compare a value against the lowest and highest BSON elements.
- Arrays – This type is used to store arrays or list or multiple values into one key.
- Timestamp – ctimestamp. This can be handy for recording when a document has been modified or added.
- Object – This datatype is used for embedded documents.
- Null – This type is used to store a Null value.
- Symbol – This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- Date – This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- Binary data – This datatype is used to store binary data.
- Code – This datatype is used to store JavaScript code into the document.

- Regular expression – This datatype is used to store regular expression.

Mongodb commands for database ON WINDOWS POWERSHELL

1. use Command

   MongoDB use DATABASE_NAME is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

   Syntax

   Basic syntax of use DATABASE statement is as follows

   ```
   >use Tybca;
   switched to db Tybca
   ```

2. Show Command

   MongoDB uses show dbs to list all the databases. The command will list all the databases if it does exist.

   Syntax

   Basic syntax of show dbs statement is as follows

   ```
   >Show dbs;
   ```

```
AlienDBex       0.000GB
Tybca           0.000GB
admin           0.000GB
config          0.000GB
db              0.000GB
```

3. db command

   MongoDB uses db to show the current database that we are working with.

   Syntax

   Basic syntax of db statement is as follows

   ```
   >db;
   ```

Tybca

4. The dropDatabase() Method

   MongoDB db.dropDatabase() command is used to drop a existing database. In our case Tybca database will be dropped

   Syntax

   Basic syntax of dropDatabase() command is as follows

   >db.dropDatabase

   >show db;

```
AlienDBex       0.000GB
admin           0.000GB
config          0.000GB
db              0.000GB
```

Mongodb commands for collection ON WINDOWS POWERSHELL

1. In MongoDB, db.createCollection(name, options) is used to create collection. But usually you don?t need to create collection. MongoDB creates collection automatically when you insert some documents.

Syntax:

db.createCollection(name, options)

Name: is a string type, specifies the name of the collection to be created.

Options: is a document type, specifies the memory size and indexing of the collection. It is an optional parameter.

Field	Type	Description
Capped	Boolean	(Optional) If it is set to true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. If you specify true, you need to specify size parameter also.

AutoIndexID	Boolean	(Optional) If it is set to true, automatically create index on ID field. Its default value is false.
Size	Number	(Optional) It specifies a maximum size in bytes for a capped collection. Ifcapped is true, then you need to specify this field also.
Max	Number	(Optional) It specifies the maximum number of documents allowed in the capped collection.

Let's take an example to create collection. In this example, we are going to create a collection name firstSem

```
>use Tybca
```

```
switched to db Tybca
```

```
>db.createCollection("firstSem")
{ "ok" : 1 }
```

To check the created collection, use the command "show collections".

```
>show collections
```

```
firstSem
```

How does MongoDB create collection automatically

MongoDB creates collections automatically when you insert some documents. For example: Insert a document named secondSem into a collection named SyBca. The operation will create the collection if the collection does not currently exist.

```
>db.SyBca.insert({"name" : "secondSem"})
>show collections
SyBca
```

If you want to see the inserted document, use the find() command.

Syntax:

```
db.collection_name.find()
```

MongoDB Drop collection

In MongoDB, db.collection.drop() method is used to drop a collection from a database. It completely removes a collection from the database and does not leave any indexes associated with the dropped collections.

The db.collection.drop() method does not take any argument and produce an error when it is called with an argument. This method removes all the indexes associated with the dropped collection.

Syntax:

```
db.COLLECTION_NAME.drop()
```

MongoDB Drop collection example

Let's take an example to drop collection in MongoDB

First check the already existing collections in your database.

1. >use mydb

```
Switched to db mydb
```

1. > show collections

```
Sudents
classes
```

Note: Here we have a collection named Sudents and classes in our database.

Now drop the collection with the name SSSIT:

1. >db.Sudents.drop()

```
True
```

Now check the collections in the database:

>show collections

```
classes
```

Now, there is only classes collections in your database.

Course: 501-1: Advanced Web Designing

Unit-1: Concepts of NoSQL: MongoDB

CRUD on Documents

1. The insert() Method

To insert data into MongoDB collection, you need to use MongoDB's insert() or save() method.

Syntax

The basic syntax of insert() command is as follows –

```
>db.COLLECTION_NAME.insert(document)
```

```
>db.subjects.insert(
{'name':'Ajava',
'lecturer':'Rushil Panchal',
'Experience':5,
'Jdate':new Date()
})
Write Result({"nInserted":1})
```

Here subjects is our collection name. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

In the inserted document, if we don't specify the _id parameter, then MongoDB assigns a unique ObjectId for this document.

_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows –

_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)

You can also pass an array of documents into the insert() method as shown below:

```
db.subjects.insert(
{'name':'Ajava',
'lecturer':'Rushil Panchal',
'Experience':5,
'class':{
        'div':'second',
        'duration':'45 mins'
        },
```

```
'Batch':[{'size':"small",'no_students':65},{'size':"Medium", 'no_students':90}],
'category':"programming language"
})
Write Result({"nInserted":1})
```

2. Check the inserted documents

If the insertion is successful, you can view the inserted document by the following query.

>db.subjects.find()

```
{"id":ObjectId("56482d3e27e53d2dbc93cef8"),"name":"Ajava",
"lecturer":"Rushil Panchal",
"Experience":5,
{"class":{"div":"second","duration":"45 mins"},
"Batch":[{"size":"small","no_students":65},{"size":"Medium","no_students":90],
"category":"programming language"}
```

The insertOne() method

If you need to insert only one document into a collection you can use this method.

Syntax

The basic syntax of insert() command is as follows –

>db.COLLECTION_NAME.insertOne(document)

Example

Following example creates a new collection named empDetails and inserts a document using the insertOne() method.

```
> db.createCollection("empDetails")
{ "ok" : 1 }
```

```
> db.empDetails.insertOne(
        {
                First_Name: "Radhika",
                Last_Name: "Sharma",
                Date_Of_Birth: "1995-09-26",
                e_mail: "radhika_sharma.123@gmail.com",
                phone: "9848022338"
        })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5dd62b4070fb13eec3963bea")
}
>
```

The insertMany() method

You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

Example

Following example inserts three different documents into the subjects collection using the insertMany() method.

```
> db.subjects.insertMany([
{ 'Name':'Java', 'Lecturer':'Rupal Panchal', 'Experience':5 },
{ 'Name':'IOT', 'Lecturer':'Priyanka Chauhan','Experience':3 },
{ 'Name':'PHP', 'Lecturer':'Heena Patel','Experience':5 }
])
{
      "acknowledged" : true,
      "insertedIds" : [
            ObjectId("62715a28b0ad3e1eac2401ce"),
            ObjectId("62715a28b0ad3e1eac2401cf"),
            ObjectId("62715a28b0ad3e1eac2401d0")
      ]
}
>
```

3. The find() Method

To query data from MongoDB collection, you need to use MongoDB's find() method.

Syntax

The basic syntax of find() method is as follows –

```
>db.COLLECTION_NAME.find()
```

find() method will display all the documents in a non-structured way.

Example

we have created a collection named subjects as –

```
> db.subjects.find()
{ "_id" : ObjectId("627085f4b0ad3e1eac2401c7"), "name" : "Ajava", "lecturer" : "Rushil Panchal", "Experience" : 5, "Jdate" : ISODate("2022-05-03T01:31:32.767Z") }
{ "_id" : ObjectId("62715a28b0ad3e1eac2401ce"), "Name" : "Java", "Lecturer" : "Rupal Panchal", "Experience" : 5 }
{ "_id" : ObjectId("62715a28b0ad3e1eac2401cf"), "Name" : "IOT", "Lecturer" : "Priyanka Chauhan", "Experience" : 3 }
{ "_id" : ObjectId("62715a28b0ad3e1eac2401d0"), "Name" : "PHP", "Lecturer" : "Heena Patel", "Experience" : 5 }
>
```

4. The pretty() Method

To display the results in a formatted way, you can use pretty() method.

Syntax

```
>db.COLLECTION_NAME.find().pretty()
```

Example

Following example retrieves all the documents from the collection named subjects and arranges them in an easy-to-read format.

```
> db.subjects.find().pretty()
{
        "_id" : ObjectId("627085f4b0ad3e1eac2401c7"),
        "name" : "Ajava",
        "lecturer" : "Rushil Panchal",
        "Experience" : 5,
        "Jdate" : ISODate("2022-05-03T01:31:32.767Z")
```

```
}
{
        "_id" : ObjectId("62715a28b0ad3e1eac2401ce"),
        "Name" : "Java",
        "Lecturer" : "Rupal Panchal",
        "Experience" : 5
}
{
        "_id" : ObjectId("62715a28b0ad3e1eac2401cf"),
        "Name" : "IOT",
        "Lecturer" : "Priyanka Chauhan",
        "Experience" : 3
}
{
        "_id" : ObjectId("62715a28b0ad3e1eac2401d0"),
        "Name" : "PHP",
        "Lecturer" : "Heena Patel",
        "Experience" : 5
}
>
```

The findOne() method

Apart from the find() method, there is findOne() method, that returns only one document.

Syntax

>db.COLLECTIONNAME.findOne()

Example

Following example retrieves the document with title MongoDB Overview.

```
> db.subjects.findOne({lecturer:"Rushil Panchal"})
{
        "_id" : ObjectId("627085f4b0ad3e1eac2401c7"),
        "name" : "Ajava",
        "lecturer" : "Rushil Panchal",
        "Experience" : 5,
        "Jdate" : ISODate("2022-05-03T01:31:32.767Z"),


}
```

RDBMS Where Clause Equivalents in MongoDB

To query the document on the basis of some condition, you can use following operations.

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:{$eg;<value>}}	db.subjects.find({"name":"Ajava"}).pretty()	where name = 'Ajava'
Less Than	{<key>:{$lt:<value>}}	db.subjects.find({"Experience ":{$lt:5}}).pretty()	where Experience < 5
Less Than Equals	{<key>:{$lte:<value>}}	db.subjects.find({"Experience ":{$lte:5}}).pretty()	where Experience <= 5
Greater Than	{<key>:{$gt:<value>}}	db.subjects.find({"Experience ":{$gt:5}}).pretty()	where Experience > 5
Greater Than Equals	{<key>:{$gte:<value>}}	db.subjects.find({"Experience ":{$gte:5}}).pretty()	where Experience >= 5
Not Equals	{<key>:{$ne:<value>}}	db.subjects.find({"Experience ":{$ne:5}}).pretty()	where Experience != 5
Values in an array	{<key>:{$in:[<value1>, <value2>,.....<valueN>] }}	db.subjects.find({"name":{$in: ["Java", "IOT", "PHP"] }}).pretty()	Where name matches any of the value in :["Java", "IOT", "PHP"]
Values not in an array	{<key>:{$nin:<value>}}	db.mycol.find({"name":{$nin:["Java", "IOT"]}}).pretty()	Where name values is not in the array :["Java", "IOT"] or, doesn't exist at all

AND in MongoDB

Syntax

To query documents based on the AND condition, you need to use $and keyword. Following is the basic syntax of AND –

```
>db.subjects.find({ $and: [ {<key1>:<value1>}, { <key2>:<value2>} ] })
```

Example

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

For the above given example, equivalent where clause will be ' where Experience= 5 AND name= 'Ajava' '. You can pass any number of key, value pairs in find clause.

OR in MongoDB

Syntax

To query documents based on the OR condition, you need to use $or keyword. Following is the basic syntax of OR –

```
>db.mycol.find(
  {
    $or: [
      {key1: value1}, {key2:value2}
    ]
  }
).pretty()
```

Example

Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

5. Using AND and OR Together

Example

The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is 'where Experience>3 AND (Name= 'Java' OR Lecturer= 'Rushil Panchal')

NOR in MongoDB

Syntax

To query documents based on the NOT condition, you need to use $not keyword. Following is the basic syntax of NOT –

```
>db.COLLECTION_NAME.find(
        {
                $not: [
                        {key1: value1}, {key2:value2}
                ]
        }
)
```

Example

Assume we have inserted 3 documents in the collection empDetails as shown below –

```
db.empDetails.insertMany(
        [
                {
                        First_Name: "Radhika",
                        Last_Name: "Sharma",
                        Age: "26",
                        e_mail: "radhika_sharma.123@gmail.com",
                        phone: "9000012345"
                },
                {
                        First_Name: "Rachel",
                        Last_Name: "Christopher",
                        Age: "27",
                        e_mail: "Rachel_Christopher.123@gmail.com",
                        phone: "9000054321"
                },
                {
                        First_Name: "Fathima",
                        Last_Name: "Sheik",
                        Age: "24",
                        e_mail: "Fathima_Sheik.123@gmail.com",
                        phone: "9000054321"
                }
        ]
)
```

Following example will retrieve the document(s) whose first name is not "Radhika" and last name is not "Christopher"

```
> db.empDetails.find(
        {
                $nor:[
                        40
                        {"First_Name": "Radhika"},
                        {"Last_Name": "Christopher"}
                ]
        }
).pretty()
{
        "_id" : ObjectId("5dd631f270fb13eec3963bef"),
        "First_Name" : "Fathima",
        "Last_Name" : "Sheik",
        "Age" : "24",
        "e_mail" : "Fathima_Sheik.123@gmail.com",
        "phone" : "9000054321"
}
```

NOT in MongoDB

Syntax

To query documents based on the NOT condition, you need to use $not keyword following is the basic syntax of NOT –

```
>db.COLLECTION_NAME.find(
        {
                $NOT: [
                        {key1: value1}, {key2:value2}
                ]
        }
).pretty()
```

Example

Following example will retrieve the document(s) whose age is not greater than 25

```

```

6.

1. Limit the number of rows in output
    a. Db.subjects.find().limit(2);
2. Count the number of rows in the output
    a. Db.subjects.find().count()
    b. Db.subjects.find(name:'Ajava').count()
3. To sort the database (1 for ascending and -1 for descending)
    a. Db.subjects.find().sort({Exp:1}).pretty()
    b. Db.subjects.find().sort({Exp:-1}).pretty()

4. PROJECTION

    In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

    The find() Method

    MongoDB's find() method, explained in MongoDB Query Document accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB, when you execute find() method, then it displays all fields of a document. To limit this, you need to set a list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the fields.

    Syntax

    The basic syntax of find() method with projection is as follows –

    >db.COLLECTION_NAME.find({},{KEY:1})

Example

Consider the collection subjects has the following data –

```
> db.subjects.find().pretty()
{
      "_id" : ObjectId("62616980304d0bff277f23d2"),
      "name" : "java",
      "lecturer" : "Rupal Panchal",
      "Exp." : 5
}
{
      "_id" : ObjectId("62616a6f304d0bff277f23d3"),
```

```
    "name" : "C",
    "lecturer" : "Riya Sen",
    "Exp." : 2
}
{
    "_id" : ObjectId("62616a6f304d0bff277f23d4"),
    "name" : "IOT",
    "Lecturer" : "Priyanka Chauhan",
    "Exp." : 5
}
{
    "_id" : ObjectId("62616e36304d0bff277f23d5"),
    "name" : "AJava",
    "lecturer" : "Rushil Panchal",
    "Exp." : 5,
    "Jdate" : ISODate("2022-04-21T14:46:14.736Z")
}
{
    "_id" : ObjectId("62616ed4304d0bff277f23d6"),
    "name" : "AJava",
    "lecturer" : "Rishi Panchal",
    "Exp." : 5,
    "Jdate" : ISODate("2022-04-21T14:48:52.176Z")
}
```

Following example will display the title of the document while querying the document.

```
> db.subjects.find({},{"name":1,_id:0})
{ "name" : "java" }
{ "name" : "C" }
{ "name" : "IOT" }
{ "name" : "AJava" }
{ "name" : "AJava" }
>
```

Please note **_id** field is always displayed while executing **find()** method, if you don't want this field, then you need to set it as 0.

The Limit() Method

To limit the records in MongoDB, you need to use limit() method. The method accepts one number type argument, which is the number of documents that you want to be displayed.

Syntax

The basic syntax of limit() method is as follows −

>db.COLLECTION_NAME.find().limit(NUMBER)

Example

Consider the collection subjects has the following data.

Following example will display only two documents while querying the document.

```
> db.subjects.find({},{"name":1,_id:0}).limit(2)
{ "name" : "java" }
{ "name" : "C" }
>
```

The sort() Method

To sort documents in MongoDB, you need to use sort() method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

Syntax

The basic syntax of sort() method is as follows –

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

Following example will display the documents sorted by title in the descending order.

```
> db.subjects.find({},{"lecturer":1,_id:0}).sort({"lecturer":-1})
{ "lecturer" : "Rushil Panchal" }
{ "lecturer" : "Rupal Panchal" }
{ "lecturer" : "Riya Sen" }
{ "lecturer" : "Rishi Panchal" }
>
```

MongoDB aggregate examples

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In SQL count(*) and with group by is an equivalent of MongoDB aggregation.

The aggregate() Method

For the aggregation in MongoDB, you should use **aggregate()** method.

Syntax

Basic syntax of **aggregate()** method is as follows −

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Now from the above collection, if you want to display a list stating how many lecturers are teaching a given subject , then you will use the following aggregate() method –

```
> db.subjects.aggregate([{$group : {_id : "$name", no_lecturer4subj: {$sum : 1}}}])

{ "_id" : "AJava", "no_lecturer4subj" : 2 }

{ "_id" : "IOT", "no_lecturer4subj" : 1 }

{ "_id" : "C", "no_lecturer4subj" : 1 }

{ "_id" : "java", "no_lecturer4subj" : 1 }

>
```

MongoDB $match

The `$match` stage allows us to choose just those documents from a collection that we want to work with. It does this by filtering out those that do not follow our requirements.

In the following example, we only want to work with those documents which specify that `Spain` is the value of the field `country`, and `Salamanca` is the value of the field `city`.

In order to get a readable output, I am going to add `.pretty()` at the end of all the commands.

Example:

```
> db.subjects.aggregate([{$match:{name:'AJava',lecturer:'Rishi Panchal'}}]).pretty()
{
	"_id" : ObjectId("62616ed4304d0bff277f23d6"),
	"name" : "AJava",
	"lecturer" : "Rishi Panchal",
	"Exp." : 5,
	"Jdate" : ISODate("2022-04-21T14:48:52.176Z")
}
>
```

Operator	**Meaning**
$count	Calculates the quantity of documents in the given group.
$max	Displays the maximum value of a document's field in the collection.
$min	Displays the minimum value of a document's field in the collection.
$avg	Displays the average value of a document's field in the collection.
$sum	Sums up the specified values of all documents in the collection.
$push	Adds extra values into the array of the resulting document.

The $group stage supports certain expressions (operators) allowing users to perform arithmetic, array, boolean and other operations as part of the aggregation pipeline.

MongoDB $project

It is rare that you ever need to retrieve all the fields in your documents. It is good practice to return only those fields you need so as to avoid processing more data than is necessary.

The `$project` stage is used to do this and to add any calculated fields that you need.

In this example, we only need the fields `name`, `lecturer` and `Jdate`.

In the code that follows, please note that:

- We must explicitly write `_id : 0` when this field is not required
- Apart from the `_id` field, it is sufficient to specify only those fields we need to obtain as a result of the query

This stage …

```
db.subjects.aggregate([

  { $project : { _id : 0, name : 1, lecturer : 1, Jdate : 1 } }

]).pretty()
```

..will give the result …

```
>db.subjects.aggregate([
...   { $project : { _id : 0, name : 1, lecturer : 1, Jdate : 1 } }
... ]).pretty()
{ "name" : "java", "lecturer" : "Rupal Panchal" }
{ "name" : "C", "lecturer" : "Riya Sen" }
{ "name" : "IOT" }
{
        "name" : "AJava",
```

```
        "lecturer" : "Rushil Panchal",
        "Jdate" : ISODate("2022-04-21T14:46:14.736Z")
}
{
        "name" : "AJava",
        "lecturer" : "Rishi Panchal",
        "Jdate" : ISODate("2022-04-21T14:48:52.176Z")
}
>
```