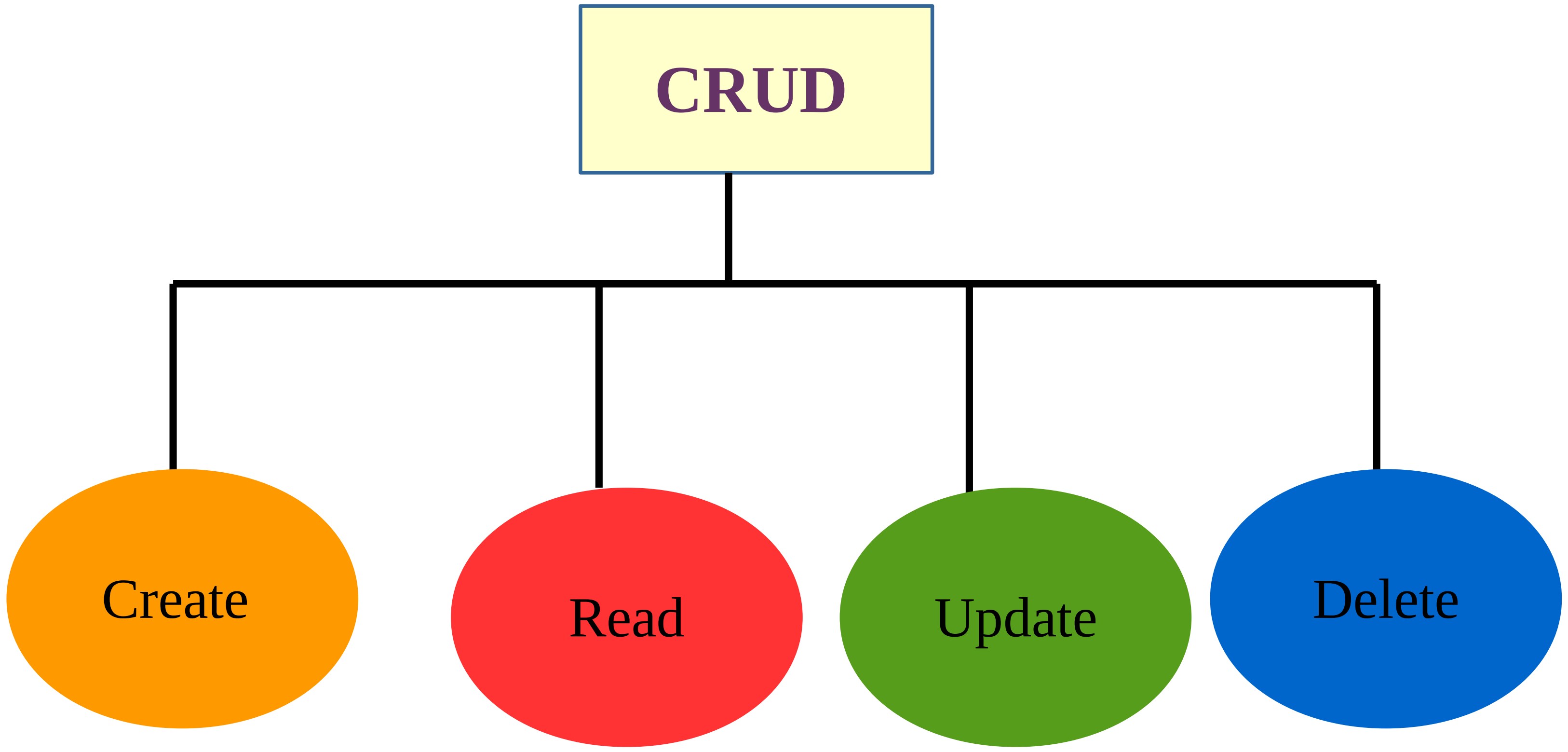
MongoDB
CRUD (Create , read ,Update & delete)

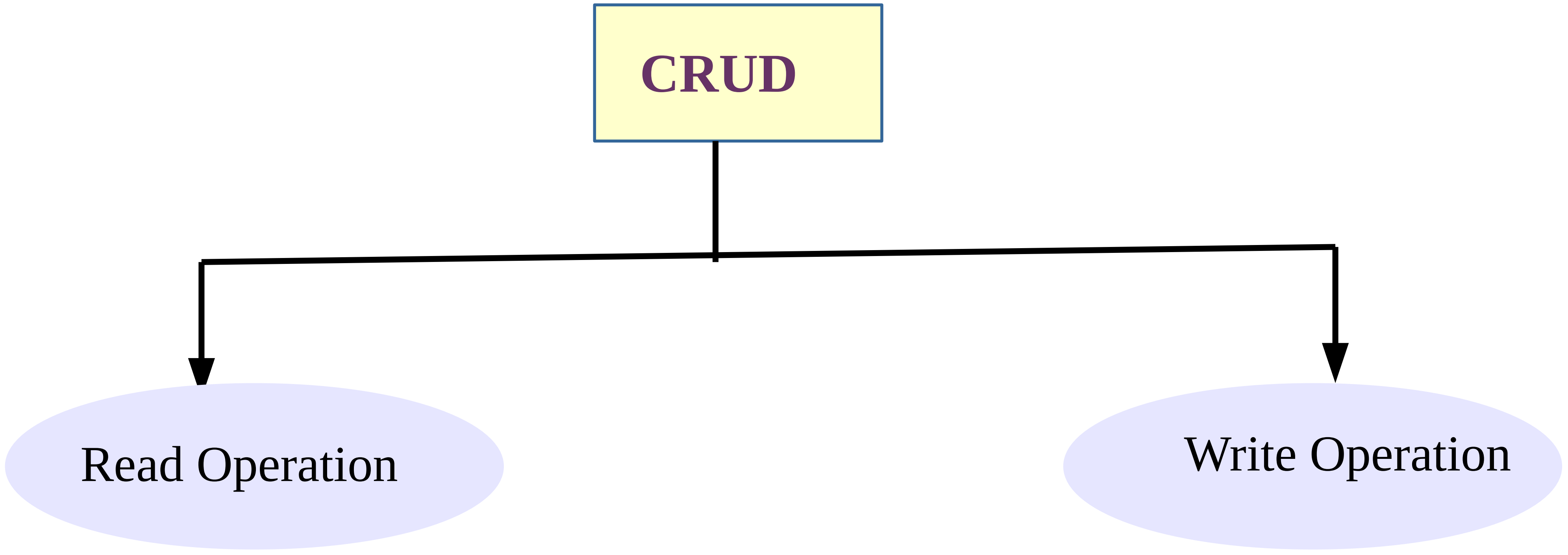
Team Emertxe

Create read update & delete

CRUD

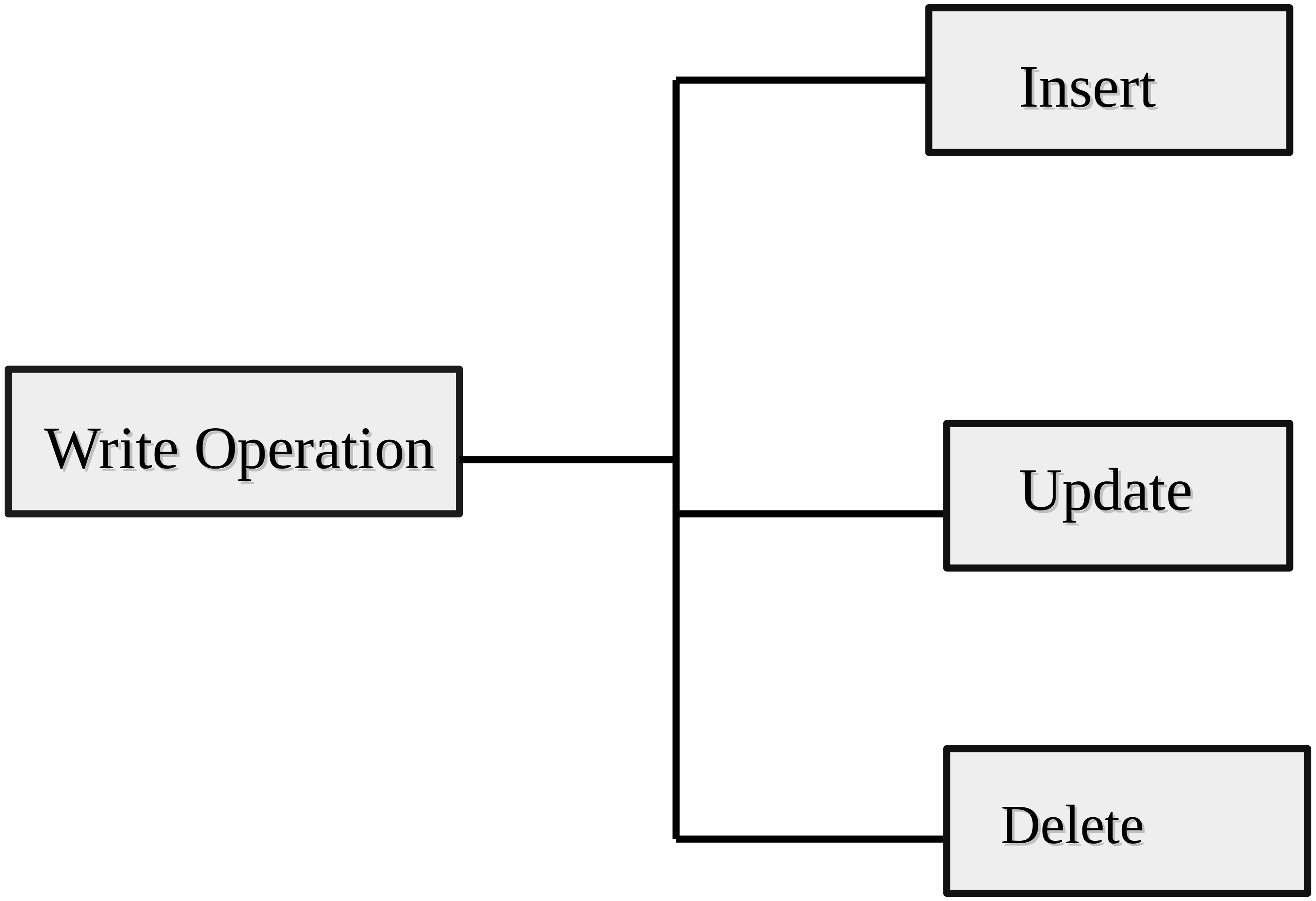


CRUD Concepts



Write Operation

Write Operation



Create Operation

Create or Insert operation adds new documents to a collection .

MongoDB provides following methods to insert document in collection .

- db.collection.insert()
- db.collection.insertOne()
- db.collection.insertMany()

$\left.\begin{array}{l} \text{db.collection.insertOne()} \\ \text{db.collection.insertMany()} \end{array}\right\}$ **Introduced from version 3.2**

Insert operation
Example

```
> db.student.insert( { name  : "john" , id  : 01 , marks : 78 })
WriteResult({ "nInserted" : 1 })
> db.student.insert( { name : "Mac" , id : 02  , marks :68 })
WriteResult({ "nInserted" : 1 })
> db.student.insert( { name : "Smith" , id : 03 , marks : 56 })
WriteResult({ "nInserted" : 1 })
> db.student.find({})
{ "_id" : ObjectId("58dca779bc327815a278d742"),
"name" : "john", "id" : 1, "marks" : 78 }
{ "_id" : ObjectId("58dca7c8bc327815a278d743"),
 "name" : "Mac", "id" : 2, "marks" : 68 }
{ "_id" : ObjectId("58dca7ebbc327815a278d744"),
"name" : "Smith", "id" : 3, "marks" : 56 }
```

Exercise

- Insert Employee name ,Employee Id and salary in the Employee Collection.

EmpName	EmpCode	Salary
Mac	E001	24000
Smith	E002	25000
Allen	E003	29000

- Display the collection Employee.

Update Operation

Update operation modify existing documents in a collection.

MongoDB provide following methods to update documents :

- db.collection.update()
- db.collection.updateOne()
- db.collection.updateMany()
- db.collection.replaceOne()

$\left.\begin{array}{l}\text{updateOne()}\\\text{updateMany()}\\\text{replaceOne()}\end{array}\right\}$ **Introduced from version 3.2**

Query based operations

Operation	Syntax
Equality	{<key> :<value> }
Less than	{<key> : { $lt : <value> }}
Less than Equals	{ <key> : { $lte :<value>}}
Greater Than	{ <key> : { $gt : <value>}}
Greater Than Equals	{ <key> : { $gte : <value>}}
Not Equals	{ <key> : { $ne : <value>}}

The update()

Syntax :

db.collection_name.update(selection_criteria, update_data ,options)

The update()
Example

Key : marks , status

Value : 65 , “A”

```
> db.Student.update (
 { marks : {$gt :65 } },
 { $set : { status : “A” }},
 {multi :true })
```

Selection Criteria

Update Action

Update Option

In the above example it will set the status “A” where marks of student is greater than 65 .

The update()
Example

```
> db.student.update( {  marks : { $gt : 65 } } , { $set : { status : 
"A" }} , {multi :true } )
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 
})
> db.student.find({})
{ "_id" : ObjectId("58dca779bc327815a278d742"), "name" : 
"john", "id" : 1, "marks" : 78, "status" : "A" }
{ "_id" : ObjectId("58dca7c8bc327815a278d743"), "name" : 
"Mac", "id" : 2, "marks" : 68, "status" : "A" }
{ "_id" : ObjectId("58dca7ebbc327815a278d744"), "name" : 
"Smith", "id" : 3, "marks" : 56 }
```

Exercise

- Add the DeptNo 10 where salary is more then 26000.
- Add the DeptNo 20 where salary is less than equal 26000.

Delete Operation

Delete Operation remove documents from collections.

MongoDB provide following methods to update documents :

- db.collection.remove()
- db.collection.deleteOne()
- db.collection.deleteMany()

$\left.\begin{array}{l} \\ \\ \end{array}\right\}$ **Introduced from version 3.2**

ΣMERTXE

Delete Operation
Example

Deletion criteria

```
> db.student.remove( { status : "A"})
  WriteResult({ "nRemoved" : 2 })
```

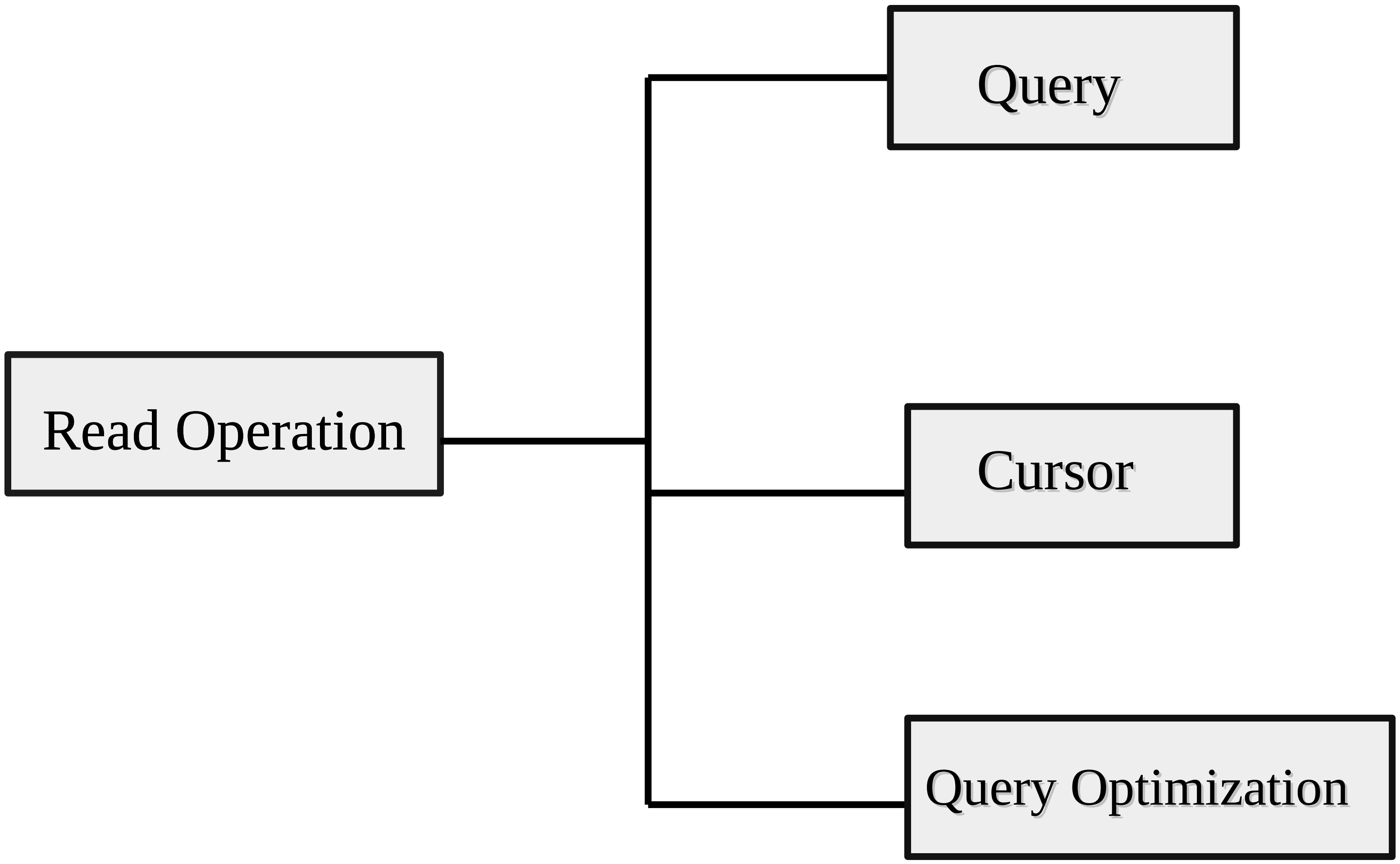
The above example will remove all the documents where status is A. This is equivalent of SQL's truncate command.

Exercise

- Delete first record of Employee collection.

Read Operation

Read Operation



Read Operation
Query

Read operation retrieves documents from collections

The MongoDB provide following method to read documents from a collection.

```
> db.collection.find()
```

The db.collection.find() will retrieve all documents from collection.

To display first document only

```
> db.collection.findOne()
```

Read Operation
Example

This example will display all documents where marks greater than 60.

```
> db.Student.find ( { marks : { $gt :60 }} )
```

This example will display name of students where marks greater than 60.

```
> db.Student.find ( { marks : { $gt :60 }} ,
{name :1 } )
```

Exercise

- Write a MongoDB query to display all the documents in the Student Collection.
- Write a MongoDB query to display the fields Employee Name and Salary in the Student Collections.
- Write a MongoDB query to display the documents where Employee salary is more than 26000.

AND in MongoDB

Syntax

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

AND in MongoDB
Example

```
> db.student.find ( { $and : [ { id : 02 , marks : { $gte : 70 } }] } )
```

The above example retrieves all the documents in the student collection where the id equals 2 and marks is greater than equal to 70.

OR in MongoDB
Syntax

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

OR in MongoDB
Example

```
> db.student.find ( { $or : [  { id : 2 }, { marks  : { $lt : 79 } } ] } )
```

The above example retrieves all documents in the collection where the id equals 2 or marks less than 79.

Pattern Match
Example

```
> db.student.find( { name : /^M/ } )

{ "_id" : ObjectId("58dcc5260f0e6a0d26410fe9") ,
"name" : "Mac", "id" : 2, "marks" : 78 }
```

The above example selects all the documents in the Student collection where name starts with the character 'M'.

Exercise

- Write a MongoDB query to display all documents where Employee Name started with S character and salary more than 24000.
- Write a MongoDB query to display all documents where employee name is “Allen” or Employee code is “E001”.

Query filter

A query filter document can use the query operators to specify conditions in the following form :

{ <field1> : { <operator> : <value1> } , ...}

Query filter
Example

```
> db.student.find( { id : { $in: [ 02 , 03 ] } });

{ "_id" : ObjectId("58dca7ebbc327815a278d744"),
"name" : "Smith", "id" : 3, "marks" : 56 }
{ "_id" : ObjectId("58dcc5260f0e6a0d26410fe9"),
"name" : "Mac", "id" : 2, "marks" : 78 }
```

The above example retrieves all the documents from the student collections where id equals either 02 or 03.

Query on Array

To insert array documents in collection

```
> db.product.insert( { name : "pen" , qty : [ 5 ,7 ] ,
 colour : [ "red" , "green" , "blue" ] } )
WriteResult({ "nInserted" : 1 })

> db.product.insert( { name : "pencil" , qty : [ 6 ,8 ] ,
colour : [ "red" , "green" , "blue" ] } )
WriteResult({ "nInserted" : 1 })

> db.product.insert( { name : "sharpner" , qty : [ 7 ,9 ] ,
colour : [ "black" , "green" , "orange" ] } )
WriteResult({ "nInserted" : 1 })
```

Query on Array

To find an array that contains all elements of array field we can use $all operator

```
> db.product.find( { colour : { $all : [ "red" , "blue", "green" ] } } )

{ "_id" : ObjectId("58dcca8d0f0e6a0d26410feb"),
"name" : "pen", "qty" : [ 5, 7 ], "colour" : [ "red", "green", "blue" ] }
{ "_id" : ObjectId("58dccab20f0e6a0d26410fec"),
"name" : "pencil", "qty" : [ 6, 8 ], "colour" : [ "red", "green", "blue" ] }
```

Query on Array

Specify condition on the element in the array field

{ <array_field> : { <operator1> : <value1> , ... } }

Example

```
> db.product.find( {qty : { $gt : 8 } }).pretty();
```

Query on Array

Specify Multiple conditions for Array of elements :

```
> db.product.find( { qty : { $gt : 8 , $lt : 10 } } )

{ "_id" : ObjectId("58dccaec0f0e6a0d26410fed"),
 "name" : "sharpner",
"qty" : [ 7, 9 ],
 "colour" : [ "black", "green", "orange" ] }
```

The above example will display the element where qty is greater than 8 and other element where qty is less than 10.

Query on Array

Query for an element by the array index position

```
> db.product.find ( { "qty.1" : { $gt : 7 } } )
```

The above example queries for all documents where the second element in the array qty is greater than 7.

Query on Array

Query on Array by Array length

The $size operator is used to find length of array.

```
> db.product.find ( { colour :  { $size : 3 } } )
```

The above example will display the documents where the array colour has three elements.

Cursor

A pointer to the result set of query .

Clients can iterate through a cursor to retrieve results.

Cursor Methods

The cursor.count()

The method used to return the total number of documents in a cursor.

Syntax

```
db.collection.find( <query>).count()
```

The cursor.count()
Example

The given example will count the total number of documents in Student Collection.

```
> db.student.count();
3
```

The given example will count the total number of students where marks is more than 60 in Student collection.

```
> db.student.find( { marks : {$gt : 70 }}).count();
 1
```

The cursor.forEach()

The method iterates the cursor to apply a JavaScript function to each document from the cursor.

Syntax :

```
db.collection.find().forEach(<function>)
```

The cursor.forEach() Example

The given example will display all the student name in student Collection.

```
> db.student.find().forEach(function (mydata)
{ print ( "name : " + mydata.name ) ; } );

name : Smith
name : Mac
name : John
```

The cursor.hasNext()

The cursor.hasNext() returns true if the cursor has more documents to return.

Syntax :

```
db.collection.find( <query>).hasNext()
```

The cursor.next()

The cursor.next() method is used to return the next document in a cursor.

Syntax :

```
db.collection.find( <query>).next()
```

The cursor.limit()

The cursor.limit() method is used to specify the maximum number of documents the cursor will return.

Syntax :

```
db.collection.find( <query>).limit(number)
```

Note : limit(0) or limit() is equivalent to setting no limit.

The cursor.sort()

The cursor.sort() method specifies the order in which query returns matching documents.

Syntax :

Sort parameter

```
db.collection.find().sort(sort)
```

Here the sort parameter contains field and value in following form :

```
{field  : value }
```

value : 1 (ascending order)

value : -1 (descending order)

The cursor.sort() (Ascending order)Example

```
> db.student.find().sort ( { marks :1 } ).pretty();
{
        "_id" : ObjectId("58dca7ebbc327815a278d744"),
        "name" : "Smith",
        "id" : 3,
        "marks" : 56
}
{
        "_id" : ObjectId("58dcc53d0f0e6a0d26410fea"),
        "name" : "John",
        "id" : 1,
        "marks" : 66
}
{
        "_id" : ObjectId("58dcc5260f0e6a0d26410fe9"),
        "name" : "Mac",
        "id" : 2,
        "marks" : 78
}
```

ΣMERTXE

The cursor.sort()
Descending Order)Example

```
> db.student.find().sort ( { marks :1 } ).pretty();
{
        "_id" : ObjectId("58dca7ebbc327815a278d744"),
        "name" : "Smith",
        "id" : 3,
        "marks" : 56
}
{
        "_id" : ObjectId("58dcc53d0f0e6a0d26410fea"),
        "name" : "John",
        "id" : 1,
        "marks" : 66
}
{
        "_id" : ObjectId("58dcc5260f0e6a0d26410fe9"),
        "name" : "Mac",
        "id" : 2,
        "marks" : 78
}
```

The cursor.skip()

The cursor.skip() method is used to return a cursor that begins returning results only after passing or skipping a number of documents.

Syntax :

```
db.collection.find( <query>).skip(number)
```

The cursor.pretty()

The cursor.pretty() method configures the cursor to display results in an easy-to-read format.

Syntax :

```
db.Student.find().pretty()
```

Exercise

- Write a MongoDB query to display the number of documents in Employee collection where salary is more than 25000.
- Write a MongoDB query to display the Employee name using forEach method.
- Write a MongoDB query to display first three Employee Name .
- Write a MongoDB query to display all the Employees in ascending order by salary field.

Query Optimization

Query Optimization

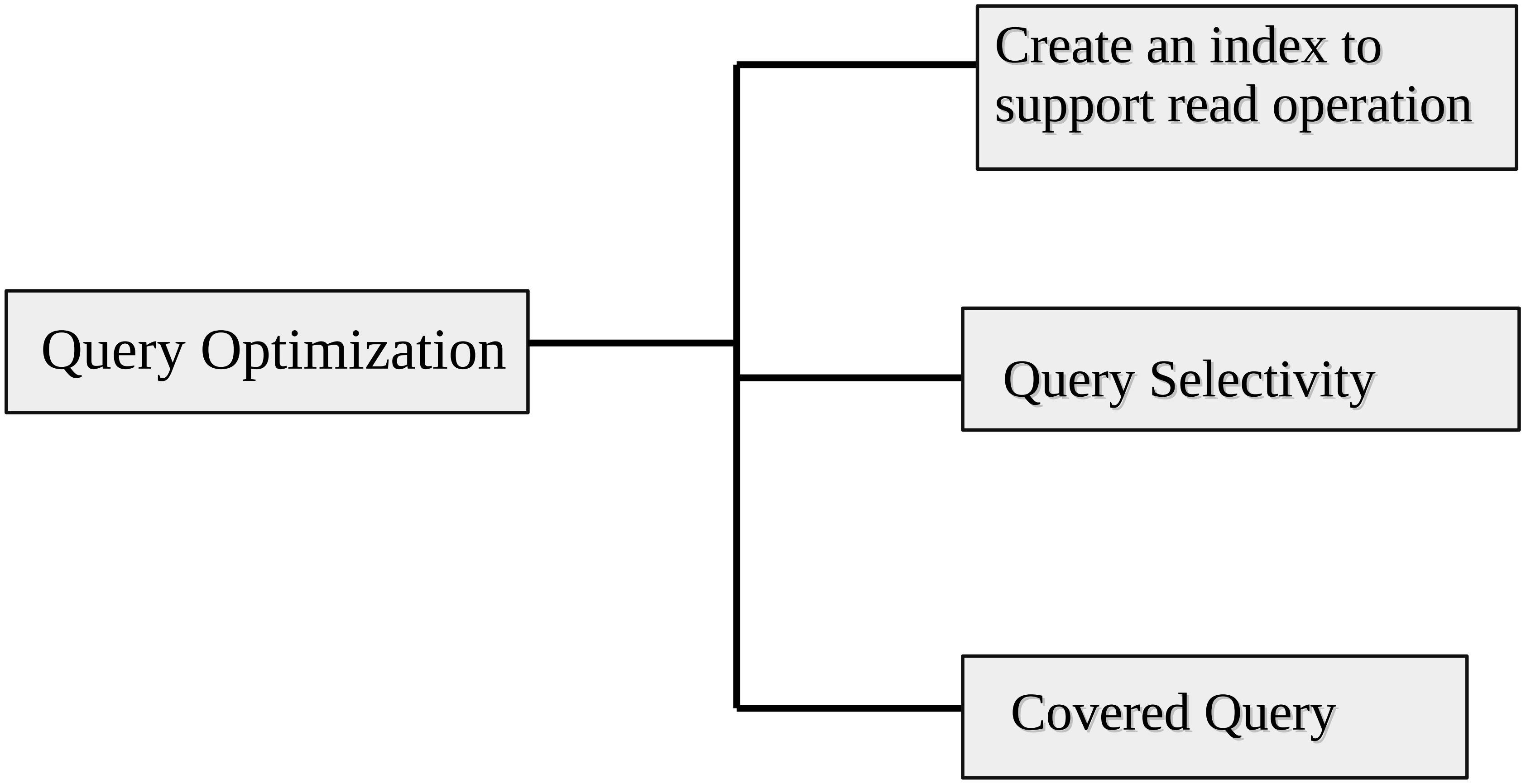
The query optimization is the process of choosing the most efficient way to execute a given query by considering the possible query plans.

Indexing & Query Optimization

MongoDB matches the query conditions using the index.

Indexes improve the efficiency of read operations by reducing the amount of data that query operations need to process.

Query Optimization



Creating Index

Syntax :

db.collection.createindex({ type : typevalue})

Example :

```
> db.student.createIndex( { id : 2} )
```

The above example will prevent scanning of whole documents.

Query Selectivity

Query Selectivity can determine whether or not queries can use indexes effectively.

Less Selective queries match a smaller percentage of documents so less selective queries cannot use indexes effectively.

The inequality operator $nin or $ne are not very selective since they often match the large portion of index.

Covered Query

A Covered query is a query in which

- All the fields in query are part of index.
- All the fields returned results are in same index.

Covered Query
Example

```
> db.student.createIndex( { id : 2} )
> db.student.find( { name : "Mac" ,  marks :
{ $gt :70 } }, { name : 1 , _id :0 }  );

{ "name" : "Mac" }
```

The above example will return only name field . The _id is used to exclude _id field from results.

Limitation

An index cannot cover a query

- If an indexed field is an array , the index becomes a multi-key index and cannot support a covered query.
- If Any of the index fields in the query predicate or returned in the projection are fields in embedded documents.

References

- https://www.wikimedia.org/
- https://docs.mongodb.com/manual/

Stay connected

About us: Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies,
No-1, 9th Cross, 5th Main,
Jayamahal Extension,
Bangalore, Karnataka 560046
T: +91 80 6562 9666
E: training@emertxe.com

https://www.facebook.com/Emertxe

https://twitter.com/EmertxeTweet

https://www.slideshare.net/EmertxeSlides

Thank You