**3. Execute at least 10 queries on any suitable MongoDB database that demonstrates following:**

- **$ where queries**
- **Cursors (Limits, skips, sorts, advanced query options)**
- **Database commands**

## $ where queries

The MongoDB $where operator is used to match documents that satisfy a JavaScript expression. A string containing a JavaScript expression or a JavaScript function can be passed using the $where operator. The JavaScript expression or function may be referred as this or obj.

```
Db.teab.insert([
{
    "_id" : ObjectId("52873b364038253faa4bbc0e"),
    "student_id" : "STU002",
    "sem" : "sem1",
    "english" : "A",
    "maths" : "A+",
    "science" : "A"
}
{
    "_id" : ObjectId("52873b5d4038253faa4bbc0f"),
    "student_id" : "STU001",
    "sem" : "sem1",
    "english" : "A+",
    "maths" : "A+",
    "science" : "A"
}
{
    "_id" : ObjectId("52873b7e4038253faa4bbc10"),
    "student_id" : "STU003",
    "sem" : "sem1",
    "english" : "A+",
    "maths" : "A",
    "science" : "A+"
}
])
```

**The grade of *english* must be same as *science***

```
>db.teab.find( { $where: function() { return (this.english == this.science)
}}).pretty();
```

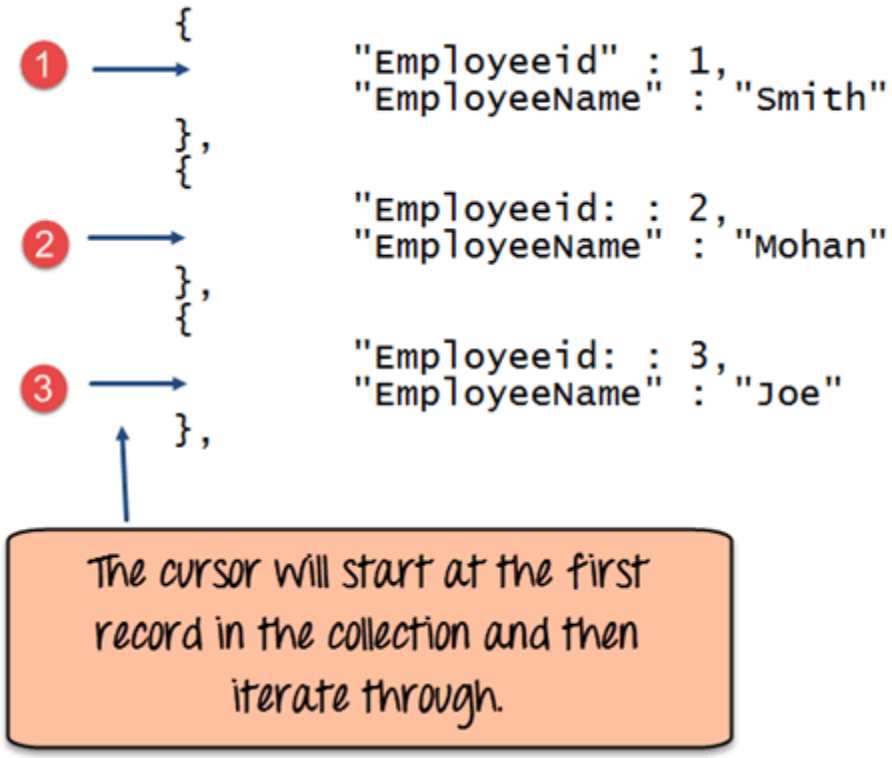
If we want to get the above output the other mongodb statements can be written as below –
>db.teab.find( { $where: function() { return (obj.english == obj.science)}}).pretty(); **OR**

```
>db.teab.find( "this.english == this.science").pretty();
```

## Cursors

When the db.collection.find () function is used to search for documents in the collection, the result returns a pointer to the collection of documents returned which is called a cursor.

By default, the cursor will be iterated automatically when the result of the query is returned. But one can also explicitly go through the items returned in the cursor one by one. If you see **the below** example, if we have 3 documents in our collection, the cursor will point to the first document and then iterate through all of the documents of the collection.



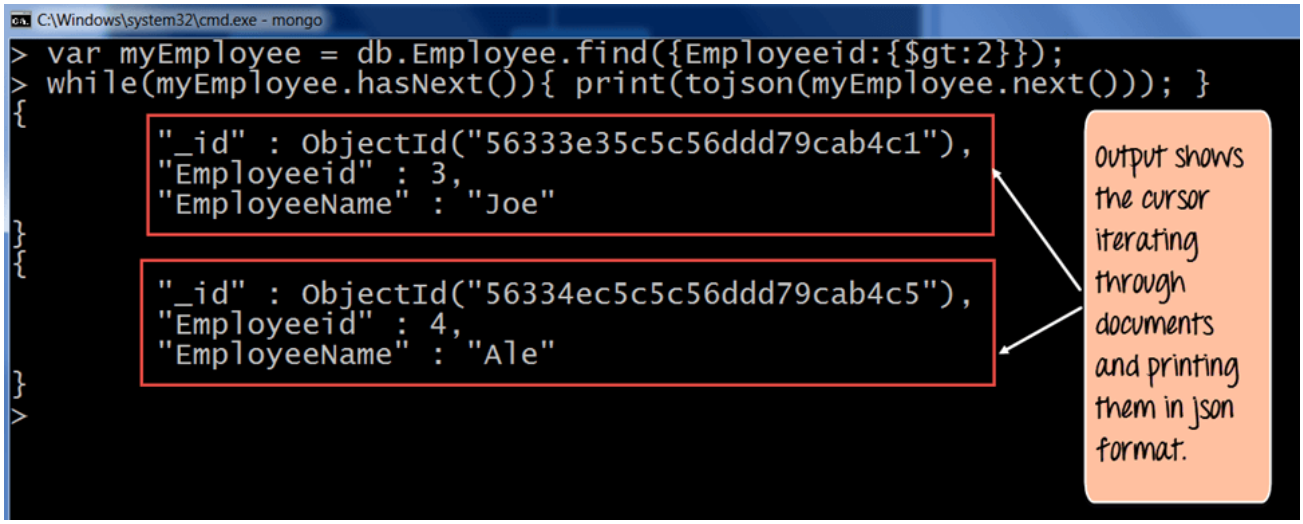

The following example shows how this can be done

```
var myEmployee = db.Employee.find( { Employeeid : { $gt:2 }});

        while(myEmployee.hasNext())

        {

                print(tojson(myCursor.next()));

        }
```

**Code Explanation:**

1. First we take the result set of the query which finds the Employee's whose id is greater than 2 and assign it to the JavaScript variable 'myEmployee'
2. Next we use the while loop to iterate through all of the documents which are returned as part of the query.
3. Finally for each document, we print the details of that document in JSON readable format.

If the command is executed successfully, the following Output will be shown




# MongoDB Query Modifications using limit(), sort()

Mongo DB provides query modifiers such as the 'limit' and 'Orders' clause to provide more flexibility when executing queries.

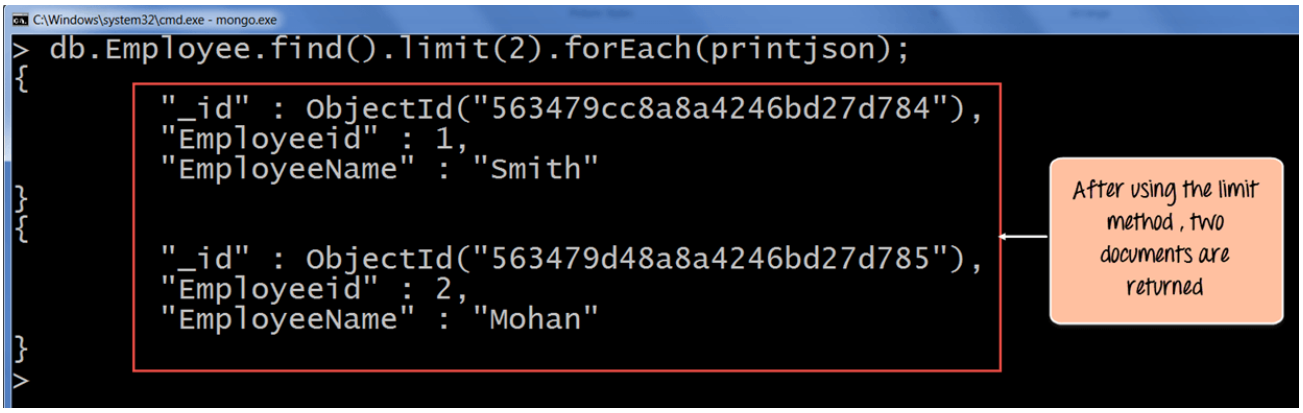
1. Limits

   This modifier is used to limit the number of documents which are returned in the result set for a query. The following example shows how this can be done.

```
db.Employee.find().limit(2).forEach(printjson);
```

**Code Explanation:** The above code takes the find function which returns all of the documents in the collection but then uses the limit clause to limit the number of documents being returned to just 2.

**Output:** If the command is executed successfully, the following Output will be shown




The output clearly shows that since there is a limit modifier, so at most just 2 records are returned as part of the result set based on the ObjectId in ascending order.

A collection `orders` contain the following documents:

```
{ _id: 1, item: { category: "cake", type: "chiffon" }, amount: 10 }
{ _id: 2, item: { category: "cookies", type: "chocolate chip" }, amount: 50 }
{ _id: 3, item: { category: "cookies", type: "chocolate chip" }, amount: 15 }
{ _id: 4, item: { category: "cake", type: "lemon" }, amount: 30 }
{ _id: 5, item: { category: "cake", type: "carrot" }, amount: 20 }
{ _id: 6, item: { category: "brownies", type: "blondie" }, amount: 10 }
```

The following query, which returns all documents from the orders collection, **does not specify a sort order:**
**db.orders.find()**

The following query specifies a sort on the amount field in **descending order.**
**db.orders.find().sort( { amount: -1 } )**

**The following query specifies the sort order using the fields from an embedded document item. The query sorts first by the category field in ascending order, and then within each category, by the type field in ascending order.**

**db.orders.find().sort( { "item.category": 1, "item.type": 1 } )**

The query returns the documents, ordered first by the category field, and within each category, by the type field:

## 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:**

```
cursor.skip()
```

Sample document in the restaurants collection:

```
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```

The following example will return a cursor that **begins returning results for only 5 documents after passing or skipping 12 of documents.**
db.restaurants.find({"cuisine" : "American "}).limit(5).skip(12);