→ Stored Procedure vs. Stored Function

Stored Procedure	Stored Function
It can return multiple values	It can return only one value which is mandatory
You can used transactions in Stored Procedure	You cannot use transactions in stored function
We can pass three kind of directional parameters, input, output, INOUT	We can only pass input parameters
We cannot call stored procedure from stored function	We can call stored function from stored procedure
Exception handling can be used in stored procedure	Exception handling cannot be used in stored Function
In Stored Procedure, we can use SELECT, INSERT, UPDATE, DELETE statements	In Stored Function, we can only use SELECT statement

→
→
→

→ Module end exam Pattern

Create. Insert
Queries on Hr database, mostly using where group by … having,
Stored procedures & stored functions, triggers, etc.

→ Pending

a. Windows function,
b. Normalization
c. ER-diagrams

→ interview important

a. Windows function : Lag(), Rank(), Lead(), dense_rank()
b. Joins
c. Sub-Queries

→ Do Not do IMDB assignment right now, use windows function, Joins & SubQueries

→ MongoDB

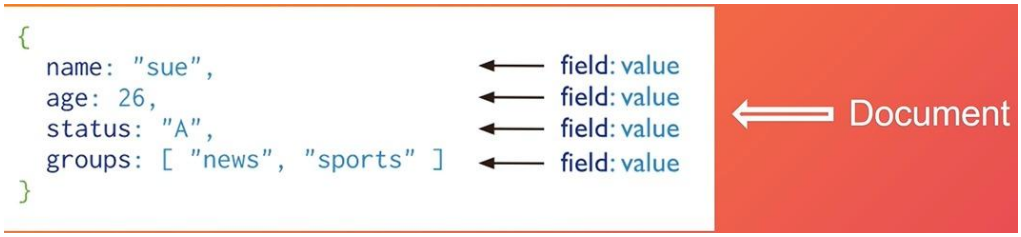
a. `>mongod --version` `#for newer version 6.0+`
   `>mongo –version` `#for older version`
   i. It shows version of MongoDB

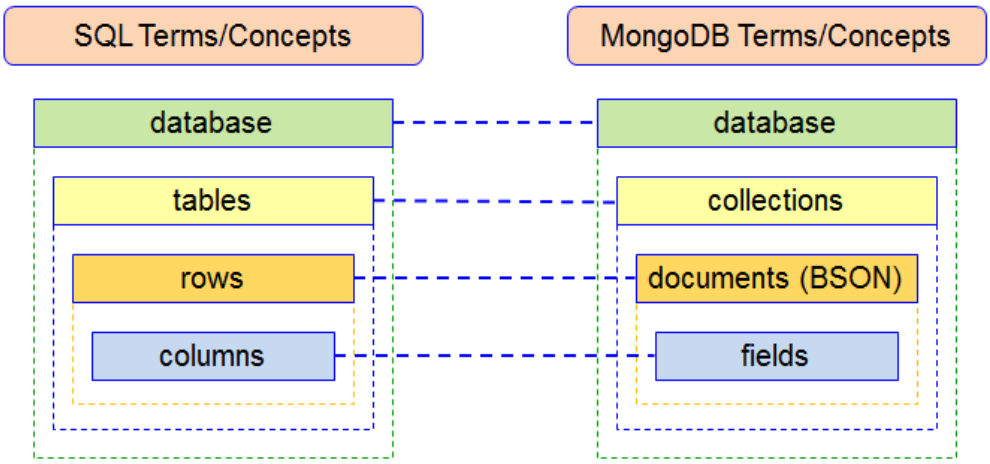
b. `>mongod` `#for new versions 6.0+`
   `>mongo` `#for older versions`
   i. It starts a mongod shell process/service.
   ii. Now, read the outputmessage carefully to check if there is some error regarding path C:\data\db.
   iii. If you path from error does not exist, create that path first and then run command "`mongod`" again
   iv. starts mongodb database process, and makes available to clients like mongo, mongosh or any application
   v. It tries to find configuration files in C:\data\db path. If it does not find the path, it'll give an error related to the path not found, and it'll not start the server. So, we need to create this path before running mongod command
   vi. In mongoDB, data is store in collection of documents which can have dynamic schema
   vii. A Collection is a collection/group of documents and documents within a collection can have different fields

c. Commands
   i. Install mongodb
      `choco install mongodb`
      https://community.chocolatey.org/packages/mongodb
   ii. Install mongoDB compass
      `choco install mongodb-compass`
      https://community.chocolatey.org/packages/mongodb-compass
   iii. Install mongodb shell
      `choco install mongodb-shell`
   iv. https://community.chocolatey.org/packages/mongodb-shell

d. RDBMS vs MongoDB analogy
   i. Database is database
   ii. Table is collection
   iii. Rows/Tuples is documents
   iv. Column is Fields

e. In MongoDB,
   i. Collection is equivalent to table in RDBMS
   ii. Documents is equivalent to rows in RDBMS
   iii. Fields are equivalent to column in RDBMS





Relational Database

Student_Id	Student_Name	Age	College
1001	Chaitanya	30	Beginnersbook
1002	Steve	29	Beginnersbook
1003	Negan	28	Beginnersbook

MongoDB

```
{
 "_id": ObjectId("....."),
 "Student_Id": 1001,
 "Student_Name": "Chaitanya",
 "Age": 30,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("....."),
 "Student_Id": 1002,
 "Student_Name": "Steve",
 "Age": 29,
 "College": "Beginnersbook"
}
{
 "_id": ObjectId("....."),
 "Student_Id": 1003,
 "Student_Name": "Negan",
 "Age": 28,
 "College": "Beginnersbook"
}
```

→ Starting with using MongoDB

a. `>use sample`
   Creates / Switches to database sample

b. `>db.createCollection("employee")`
   Createscollection named employee in current DB

c. `> show dbs`
   Lists databases

d. `>show collections`
   Lists collection in current database

e. `>db`
   Shows current/selected database

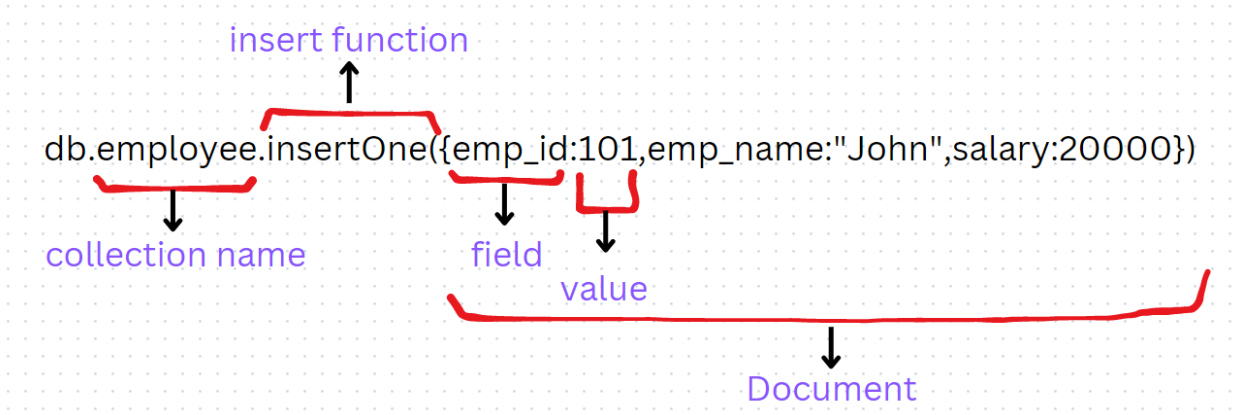
f. You cannot have `NULL` characters in your field name
g. The server permites storage of the field name that contains `'.'` or `'$'` sign
h. The field name `_id` is reserved to be used as a primary key, it has a value that is unique to the collection
i. If a user don't define `_id`, then Mongo will create an `ObjectId` that can be uniquely identified throughout the collection

→ CRUD ops in MongoDB

a. C – Create – CREATE – insert()
b. R – Read – SELECT – find()
c. U – Update –UPDATE – update()
d. D – Delete– DELETE – remove()

→ `insert()` command

a. `insert()` command is used to insert a document in a collection



b. Syntax to use **`insert(document(s),[,options])`** `# now deprecated`

```
>db.employee.insert({emp_id:101, emp_name:"John", salary:20000})
```

c. Syntax to use **`insertOne(document,[,options])`**

```
>db.employee.insertOne({emp_id:101, emp_name:"John", salary:20000})
```

```
sample> db.employee.insertOne({emp_id:101, emp_name:"John", salary:20000})
{
  acknowledged: true,
  insertedId: ObjectId("6443c5fc7e4dc891eead8af6")
}
```

d. Syntax to use **`find()`**

```
>db.employee.find()
```

```
sample> db.employee.find()
[
  {
    _id: ObjectId("6443c5fc7e4dc891eead8af6"),
    emp_id: 101,
    emp_name: 'John',
    salary: 20000
  }
]
```

e. Syntax to use **`insertMany(ArrayOfDocuments,[,options])`**

```
>db.employee.insertMany([{_id:123, emp_id:102, emp_name:"Smith",
salary:2000}, {_id:122, emp_id:103, emp_name:"Dave", salary:4500}])
```

```
sample> db.employee.insertMany([{_id:123, emp_id:102, emp_name:"Smith", salary:2000}, {_id:122, emp_id:103, emp_name:"Dave", salary
:4500}])
{ acknowledged: true, insertedIds: { '0': 123, '1': 122 } }
```

>db.employee.find()

```
sample> db.employee.find()
[
  {
    _id: ObjectId("6443c5fc7e4dc891eead8af6"),
    emp_id: 101,
    emp_name: 'John',
    salary: 20000
  },
  { _id: 123, emp_id: 102, emp_name: 'Smith', salary: 2000 },
  { _id: 122, emp_id: 103, emp_name: 'Dave', salary: 4500 }
]
```

f. Syntax to use **`insertMany()`** to insert multiple records using a variable

```
>const documents=[{name:"johnny", age:30}, {name:"Ram", age:23}, {name:"RRocky", age:450}]
```

```
sample> const documents=[{name:"johnny", age:30}, {name:"Ram", age:23}, {name:"RRocky", age:450}]
```

```
>db.employee.insertMany(documents, function(err, result){if (err){console.log(err);}else{console.log(result.insertedCount+"document inserted");}});
```

```
sample> db.employee.insertMany(documents, function(err, result){if (err){console.log(err);}else{console.log(result.insertedCount+"document inserted");}});
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("6443cc557e4dc891eead8af7"),
    '1': ObjectId("6443cc557e4dc891eead8af8"),
    '2': ObjectId("6443cc557e4dc891eead8af9")
  }
}
```

```
>db.employee.find()
```

```
sample> db.employee.find()
[
  {
    _id: ObjectId("6443c5fc7e4dc891eead8af6"),
    emp_id: 101,
    emp_name: 'John',
    salary: 20000
  },
  { _id: 123, emp_id: 102, emp_name: 'Smith', salary: 2000 },
  { _id: 122, emp_id: 103, emp_name: 'Dave', salary: 4500 },
  {
    _id: ObjectId("6443cc557e4dc891eead8af7"),
    name: 'johnny',
    age: 30
  },
  { _id: ObjectId("6443cc557e4dc891eead8af8"), name: 'Ram', age: 23 },
  {
    _id: ObjectId("6443cc557e4dc891eead8af9"),
    name: 'RRocky',
    age: 450
  }
]
```

g. Syntax to use **`insertMany()`** to insert multiple records using JSON format

```
>db.employee.insertMany(
[{  "_id": ObjectId("6090196dbb1cfc7842916d61"),
    "name": "John Doe", "position": "Software Engineer",
    "department": "Engineering", "salary": 90000}
,{ "_id": ObjectId("6090196dbb1cfc7842916d62"),
    "name": "Jane Smith", "position": "Project Manager",
    "department": "Management", "salary": 110000}
,{ "_id": ObjectId("6090196dbb1cfc7842916d63"),
    "name": "Bob Johnson", "position": "Marketing Manager",
    "department": "Marketing", "salary": 105000}
,{ "_id": ObjectId("6090196dbb1cfc7842916d64"),
    "name": "Sara Lee", "position": "Human Resources",
    "department": "Human Resources", "salary": 95000}
,{ "_id": ObjectId("6090196dbb1cfc7842916d65"),
    "name": "Tom Williams", "position": "Sales Representative",
    "department": "Sales", "salary": 85000}
,{ "_id": ObjectId("6090196dbb1cfc7842916d66"),
```

```
    "name": "Emily Jones", "position": "Accountant",
    "department": "Accounting", "salary": 100000}
,{
    "_id": ObjectId("6090196dbb1cfc7842916d67"),
    "name": "Mike Davis", "position": "Operations Manager",
    "department": "Operations", "salary": 120000}
,{ "_id": ObjectId("6090196dbb1cfc7842916d68"),
    "name": "Kelly Brown", "position": "IT Specialist",
    "department": "Information Technology", "salary": 95000}
,{ "_id": ObjectId("6090196dbb1cfc7842916d69"),
    "name": "Samuel Kim", "position": "Customer Service
Representative",
    "department": "Customer Service", "salary": 80000}
,{ "_id": ObjectId("6090196dbb1cfc7842916d6a"),
    "name": "Anna Lee", "position": "Graphic Designer",
    "department": "Design", "salary": 85000}])
```

```
... },
... {
...     "_id": ObjectId("6090196dbb1cfc7842916d6a"),
...     "name": "Anna Lee",
...     "position": "Graphic Designer",
...     "department": "Design",
...     "salary": 85000
... }])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("6090196dbb1cfc7842916d61"),
    '1': ObjectId("6090196dbb1cfc7842916d62"),
    '2': ObjectId("6090196dbb1cfc7842916d63"),
    '3': ObjectId("6090196dbb1cfc7842916d64"),
    '4': ObjectId("6090196dbb1cfc7842916d65"),
    '5': ObjectId("6090196dbb1cfc7842916d66"),
    '6': ObjectId("6090196dbb1cfc7842916d67"),
    '7': ObjectId("6090196dbb1cfc7842916d68"),
    '8': ObjectId("6090196dbb1cfc7842916d69"),
    '9': ObjectId("6090196dbb1cfc7842916d6a")
  }
}
sample>
```

h. Syntax to use find record by passing key,value pair to **`find()`**

```
>db.employee.find({name:"John Doe"})
```

```
sample> db.employee.find({name:"John Doe"})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d61"),
    name: 'John Doe',
    position: 'Software Engineer',
    department: 'Engineering',
    salary: 90000
  }
]
```

→ `find()` command

a. Syntax to use **`find()`**

```
#lists only first 10 documents from collection
>db.employee.find()
```

```
sample> db.employee.find()
[
  {
    _id: ObjectId("6443c5fc7e4dc891eead8af6"),
    emp_id: 101,
    emp_name: 'John',
    salary: 20000
  }
]
```

b. Syntax to find record by passing key,value pair to **`find(query,[fields])`**

```
>db.employee.find({name:"John Doe"})
```

```
sample> db.employee.find({name:"John Doe"})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d61"),
    name: 'John Doe',
    position: 'Software Engineer',
    department: 'Engineering',
    salary: 90000
  }
]
```

```
>db.employee.find({position:"Operations Manager"}, {name:"Mike Davis"})
```

```
sample> db.employee.find({position:"Operations Manager"}, {name:"Mike Davis"})
[ { _id: ObjectId("6090196dbb1cfc7842916d67"), name: 'Mike Davis' } ]
```

c. Syntax to find record with **`find(query).limit(n:number)`**

```
>db.employee.find({name:"John Doe"}).limit(2)
```

```
sample> db.employee.find({name:"John Doe"}).limit(2)
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d61"),
    name: 'John Doe',
    position: 'Software Engineer',
    department: 'Engineering',
    salary: 90000
  }
]
```

d. Syntax to find count of documents in a collection using **`find().itcount`**

```
>db.employee.find().itcount()
```

```
sample> db.employee.find().itcount()
16
```

e. Syntax to skip ‘n’ documents in a collection using **`find().skip(n)`**

```
>db.employee.find().skip(12)
```

```
sample> db.employee.find().skip(12)
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d67"),
    name: 'Mike Davis',
    position: 'Operations Manager',
    department: 'Operations',
    salary: 120000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d68"),
    name: 'Kelly Brown',
    position: 'IT Specialist',
    department: 'Information Technology',
    salary: 95000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d69"),
    name: 'Samuel Kim',
    position: 'Customer Service Representative',
    department: 'Customer Service',
    salary: 80000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d6a"),
    name: 'Anna Lee',
    position: 'Graphic Designer',
    department: 'Design',
    salary: 85000
  }
]
sample>
```

f. Syntax to sort in ascending order using **`find().sort(Key:1)`**:

```
>db.emp.find().sort({salary:1})
```

```
sample> db.emp.find().sort({salary:1})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d69"),
    name: 'Samuel Kim',
    position: 'Customer Service Representative',
    department: 'Customer Service',
    salary: 80000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d65"),
    name: 'Tom Williams',
    position: 'Sales Representative',
    department: 'Sales',
    salary: 85000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d6a"),
    name: 'Anna Lee',
    position: 'Graphic Designer',
    department: 'Design',
    salary: 85000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d61"),
    name: 'John Doe',
    position: 'Software Engineer',
    department: 'Engineering',
    salary: 90000
```

g. Syntax to sort in descending order using **`find().sort(Key:-1)`**:

```
>db.emp.find().sort({salary:-1})
```

```
sample> db.emp.find().sort({salary:-1})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d67"),
    name: 'Mike Davis',
    position: 'Operations Manager',
    department: 'Operations',
    salary: 120000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d62"),
    name: 'Jane Smith',
    position: 'Project Manager',
    department: 'Management',
    salary: 110000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d63"),
    name: 'Bob Johnson',
    position: 'Marketing Manager',
    department: 'Marketing',
    salary: 105000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d66"),
    name: 'Emily Jones',
    position: 'Accountant',
    department: 'Accounting',
    salary: 100000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d64"),
```

h. Syntax to find record with OR operator **`'$or:ArrayOfExpressions'`**

```
>db.employee.find({$or:[{position:"Operations Manager"}, {name:"Mike Davis"}]})
```

```
sample> db.employee.find({$or:[{position:"Operations Manager"}, {name:"Mike Davis"}]})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d67"),
    name: 'Mike Davis',
    position: 'Operations Manager',
    department: 'Operations',
    salary: 120000
  }
]
```

i. Syntax to find record using NOR operator **'$nor:ArrayOfExpressions'**:

```
>db.emp.find({$nor:[{name:'Samuel Kim'},{name:'Anna Lee'},{name:'Kelly Brown'}]})
```

```
sample> db.emp.find({$nor:[{name:'Samuel Kim'},{name:'Anna Lee'},{name:'Kelly Brown'}]})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d61"),
    name: 'John Doe',
    position: 'Software Engineer',
    department: 'Engineering',
    salary: 90000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d62"),
    name: 'Jane Smith',
    position: 'Project Manager',
    department: 'Management',
    salary: 110000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d63"),
    name: 'Bob Johnson',
    position: 'Marketing Manager',
    department: 'Marketing',
    salary: 105000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d64"),
    name: 'Sara Lee',
    position: 'Human Resources',
    department: 'Human Resources',
    salary: 95000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d65"),
```

j. Syntax to find record using AND operator **'$and:ArrayOfExpressions'**:

```
>db.emp.find({$and:[{salary:85000},{name:'Anna Lee'}]})
```

```
sample> db.emp.find({$and:[{salary:85000},{name:'Anna Lee'}]})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d6a"),
    name: 'Anna Lee',
    position: 'Graphic Designer',
    department: 'Design',
    salary: 85000
  }
]
```

k. Syntax to find record using greater than equal to operator **'$gte:Val'**

```
>db.emp.find({salary:{$gte:100000}})
```

```
sample> db.emp.find({salary:{$gte:100000}})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d62"),
    name: 'Jane Smith',
    position: 'Project Manager',
    department: 'Management',
    salary: 110000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d63"),
    name: 'Bob Johnson',
    position: 'Marketing Manager',
    department: 'Marketing',
    salary: 105000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d66"),
    name: 'Emily Jones',
    position: 'Accountant',
    department: 'Accounting',
    salary: 100000
```

l. Syntax to find record using greater than equal to operator **'$gt:Val'**

```
>db.emp.find({salary:{$gt:100000}})
```

m. Syntax to find record using less than operator **'$lte:Val'**

```
>db.emp.find({salary:{$lte:100000}})
```

n. Syntax to find record using less than operator **'$lt:Val'**

```
>db.emp.find({salary:{$lt:100000}})
```

```
sample> db.emp.find({salary:{$lt:100000}})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d61"),
    name: 'John Doe',
    position: 'Software Engineer',
    department: 'Engineering',
    salary: 90000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d64"),
    name: 'Sara Lee',
    position: 'Human Resources',
    department: 'Human Resources',
    salary: 95000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d65"),
    name: 'Tom Williams',
    position: 'Sales Representative',
    department: 'Sales',
    salary: 85000
  },
```

o. Syntax to find record using all operator **'$all:Array'**:

```
>db.emp.find({name:{$all:['Anna Lee']}})
```

```
sample> db.emp.find({name:{$all:['Anna Lee']}})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d6a"),
    name: 'Anna Lee',
    position: 'Graphic Designer',
    department: 'Design',
    salary: 85000
  }
]
```

p. Syntax to find record using in operator **'$in:Array'**

i. Returns values specified in the array

```
>db.emp.find({salary:{$in:[80000,850000]}})
```

```
sample> db.emp.find({salary:{$in:[80000,850000]}})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d69"),
    name: 'Samuel Kim',
    position: 'Customer Service Representative',
    department: 'Customer Service',
    salary: 80000
  }
]
```

q. Syntax to find record using in operator **'$nin:Array'**

i. Returns values NOT specified in the array

```
>db.emp.find({salary:{$nin:[80000,850000]}})
```

```
sample> db.emp.find({salary:{$nin:[80000,850000]}})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d61"),
    name: 'John Doe',
    position: 'Software Engineer',
    department: 'Engineering',
    salary: 90000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d62"),
    name: 'Jane Smith',
    position: 'Project Manager',
    department: 'Management',
    salary: 110000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d63"),
    name: 'Bob Johnson',
    position: 'Marketing Manager',
    department: 'Marketing',
    salary: 105000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d64"),
    name: 'Sara Lee',
    position: 'Human Resources',
    department: 'Human Resources',
    salary: 95000
  },
```

r. Syntax to find record using all operator **'$ne:Val'**:

```
>db.emp.find({name:{$ne:{name:"Samuel Kim"}}})
```

```
sample> db.emp.find({name:{$ne:{name:"Samuel Kim"}}})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d61"),
    name: 'John Doe',
    position: 'Software Engineer',
    department: 'Engineering',
    salary: 90000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d62"),
    name: 'Jane Smith',
    position: 'Project Manager',
    department: 'Management',
    salary: 110000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d63"),
    name: 'Bob Johnson',
    position: 'Marketing Manager',
    department: 'Marketing',
    salary: 105000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d64"),
```

s. Syntax to find record using **'$regex'**:

```
>db.emp.find({name:{$regex:/Kim/}})
```

```
sample> db.emp.find({name:{$regex:/Kim/}})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d69"),
    name: 'Samuel Kim',
    position: 'Customer Service Representative',
    department: 'Customer Service',
    salary: 80000
  }
]
```

t. Syntax to find record using **`'$regex'`** with ignore case **`'i'`**:

```
>db.emp.find({name:{$regex:/kim/i}})
```

```
sample> db.emp.find({name:{$regex:/kim/i}})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d69"),
    name: 'Samuel Kim',
    position: 'Customer Service Representative',
    department: 'Customer Service',
    salary: 80000
  }
]
```

u. Syntax to find record using where operator **`'$where'`**:

```
>db.emp.find({$where:'this.salary>100000'})
```

```
sample> db.emp.find({$where:'this.salary>100000'})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d62"),
    name: 'Jane Smith',
    position: 'Project Manager',
    department: 'Management',
    salary: 110000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d63"),
    name: 'Bob Johnson',
    position: 'Marketing Manager',
    department: 'Marketing',
    salary: 105000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d67"),
    name: 'Mike Davis',
    position: 'Operations Manager',
    department: 'Operations',
    salary: 120000
  }
]
```

→ sort() command

a. It helps you to sort any data
b. You've to first use `find()` in order to use `sort()`
c. When you pass '1', it means you're sorting the data in ascending order
d. When you pass '-1', it means you're sorting the data in descending order
e. Syntax to sort in ascending order using **`find()`** along with **`sort(Key:1)`**:

```
>db.emp.find().sort({salary:1})
```

```
sample> db.emp.find().sort({salary:1})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d69"),
    name: 'Samuel Kim',
    position: 'Customer Service Representative',
    department: 'Customer Service',
    salary: 80000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d65"),
    name: 'Tom Williams',
    position: 'Sales Representative',
    department: 'Sales',
    salary: 85000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d6a"),
    name: 'Anna Lee',
    position: 'Graphic Designer',
    department: 'Design',
    salary: 85000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d61"),
    name: 'John Doe',
    position: 'Software Engineer',
    department: 'Engineering',
    salary: 90000
```

f. Syntax to sort in descending order using **`find()`** along with **`sort(Key:-1)`**:

```
>db.emp.find().sort({salary:-1})
```

```
sample> db.emp.find().sort({salary:-1})
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d67"),
    name: 'Mike Davis',
    position: 'Operations Manager',
    department: 'Operations',
    salary: 120000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d62"),
    name: 'Jane Smith',
    position: 'Project Manager',
    department: 'Management',
    salary: 110000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d63"),
    name: 'Bob Johnson',
    position: 'Marketing Manager',
    department: 'Marketing',
    salary: 105000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d66"),
    name: 'Emily Jones',
    position: 'Accountant',
    department: 'Accounting',
    salary: 100000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d64"),
```