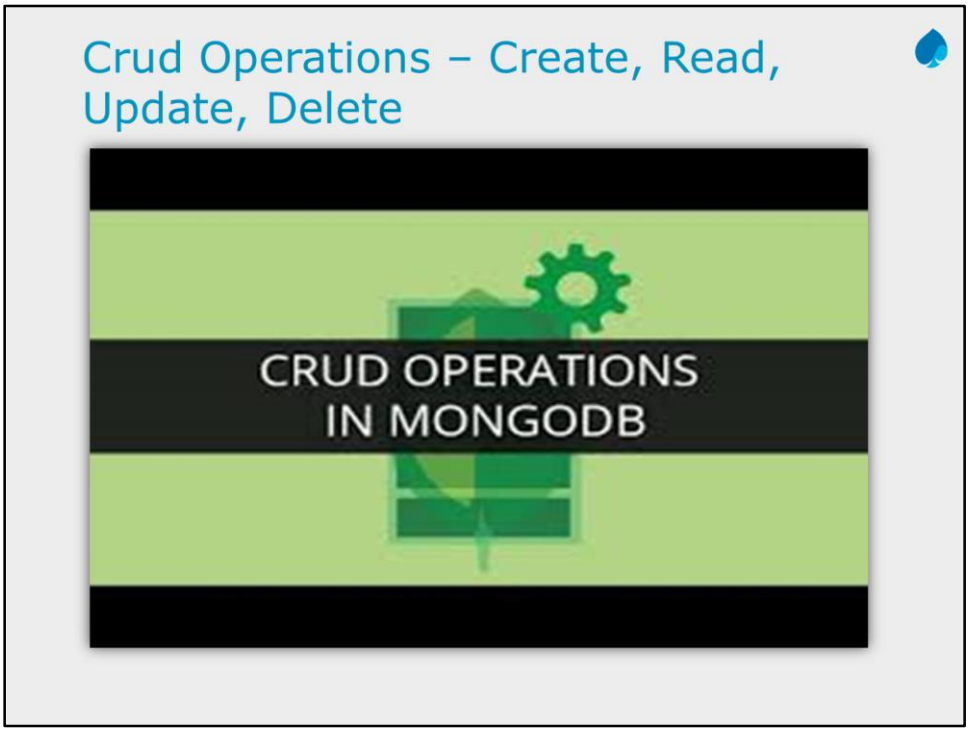
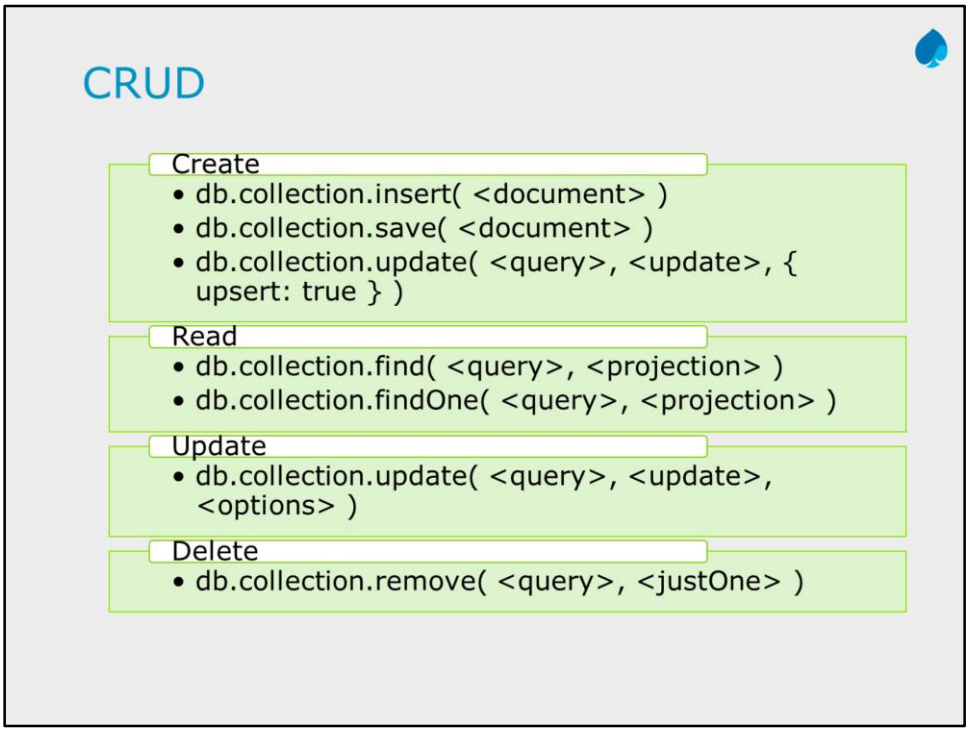

MongoDB – CRUD
Capgemini


Crud Operations – Create, Read, Update, Delete
CRUD OPERATIONS
IN MONGODB



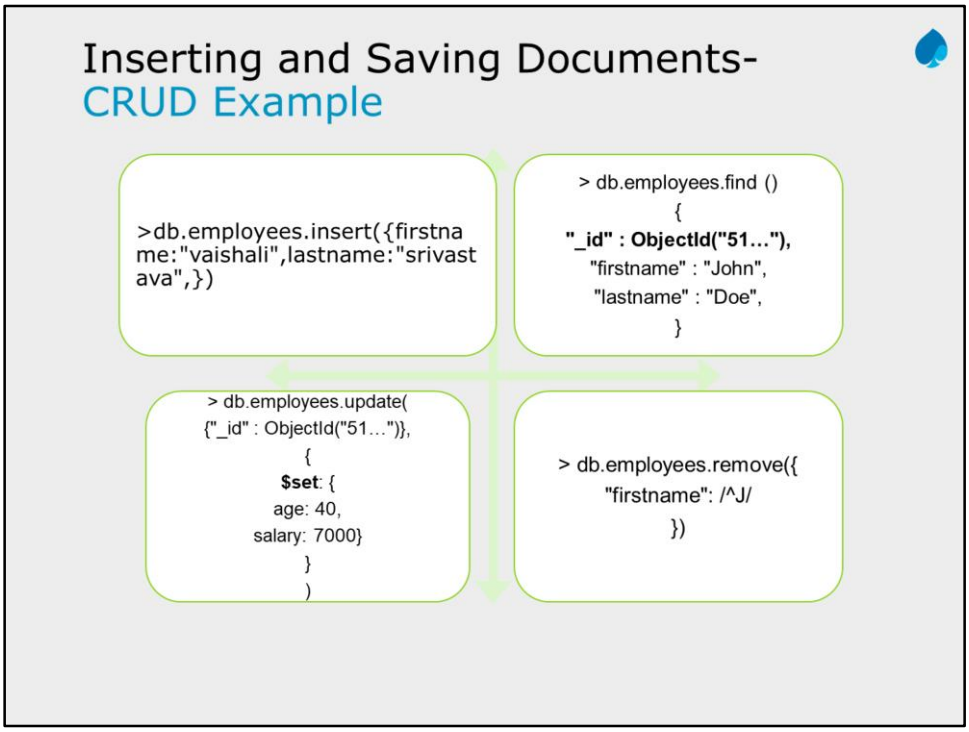
Create

- The field name _id is reserved for use as a primary key; its value must be unique in the collection, is immutable, and may be of any type other than an array.
- The field names **cannot** start with the $ character.
- The field names **cannot** contain the . character.

Create with save

- If the <document> argument does not contain the _id field or contains an _id field with a value not in the collection, the save() method performs an insert of the document.
- Otherwise, the save() method performs an update.

sds


Inserting and Saving Documents-
CRUD Example
>db.employees.insert({firstname:"vaishali",lastname:"srivastava",})
> db.employees.find ()
{
"_id" : ObjectId("51…"),
"firstname" : "John",
"lastname" : "Doe",
}
> db.employees.update(
{"_id" : ObjectId("51…")},
{
$set: {
age: 40,
salary: 7000}
}
)
> db.employees.remove({
"firstname": /^J/
})

Inserting and Saving Documents
Insert Document – insert() method

To insert data into MongoDB collection, we have a method

called insert() or save() method.

Basic syntax of **insert()** command is as follows:

Example:

- db.tutorialspoint.insert({"name":"tutorialspoint"})
- db.nextTable.save({column1:"value",column2:"value",...})'

Removing Documents- Remove document – remove() method

MongoDB's **remove()** method is used to remove document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

Deletion criteria: (Optional) deletion criteria according to documents will be removed.

justOne: (Optional) if set to true or 1, then remove only one document.

- db.collection_name.remove(DELETION_CRITERIA)
- db.employees.remove({firstname:"tanmaya"})

Remove document – remove only one

If there are multiple records and you want to delete only first record, then set **justOne** parameter in **remove()** method

```
db.collection_name.remove(deletion_criteria,1)
```

Remove All documents

If you don't specify deletion criteria, then mongodb will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**

- db.employees.remove()
- >db.employees.find()

Updating Documents-
Update document – update() method

MongoDB's **update()** and **save()** methods are used to update document into a collection. The update() method update values in the existing document while the save() method replaces the existing document with the document passed in save() method.

The update() method updates values in the existing document.

Example:

```
>db.employees.update({firstname:"vaishali"},{first
name:"vaishali",lastname:"shrivastava",age:30})
```

Updating Documents-Update with Options

Update (document) – Specifies the modifications to apply.

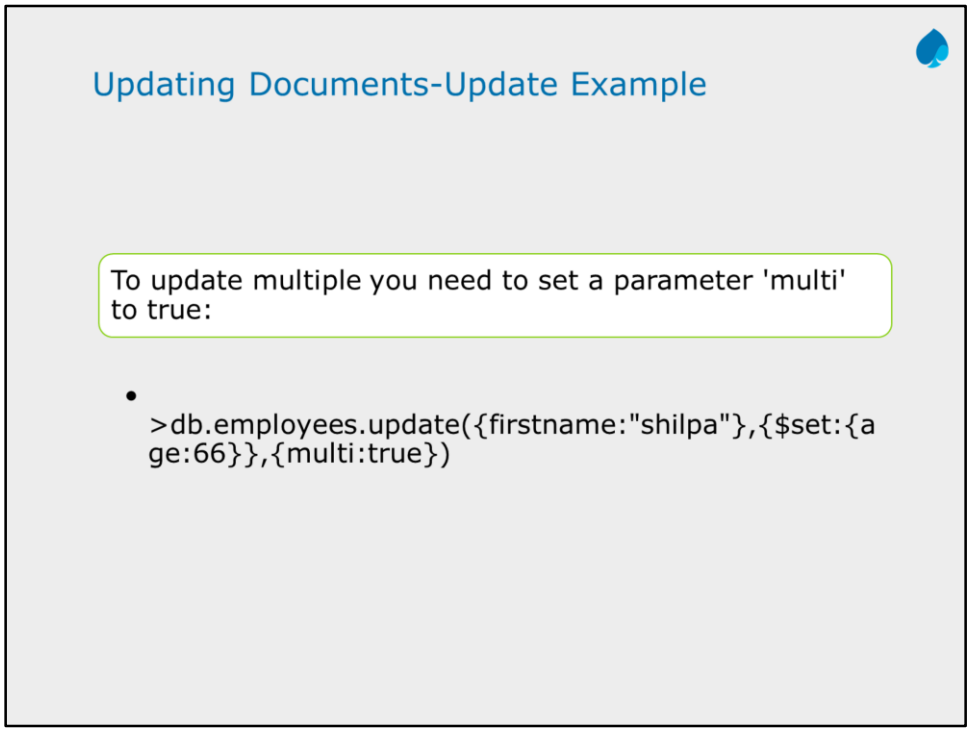
If the **update** parameter contains any update operators expressions such as the $set operator expression, then:

- The update parameter must contain only update operators expressions.
- The update method updates only the corresponding fields in the document.

If the **update** parameter consists only of field: value expressions, then:

- The update method replaces the document with the updates document. If the updates document is missing the **_id field**, MongoDB will add the _id field and assign to it a unique object Id .
- The update method updates cannot update multiple documents.

Options : (optional) Specifies whether to perform an upsert and/or a multiple update. Use the options parameter instead of the individual upsert and multi parameters.


Updating Documents-Update Example
To update multiple you need to set a parameter 'multi' to true:
>db.employees.update({firstname:"shilpa"},{$set:{age:66}},{multi:true})

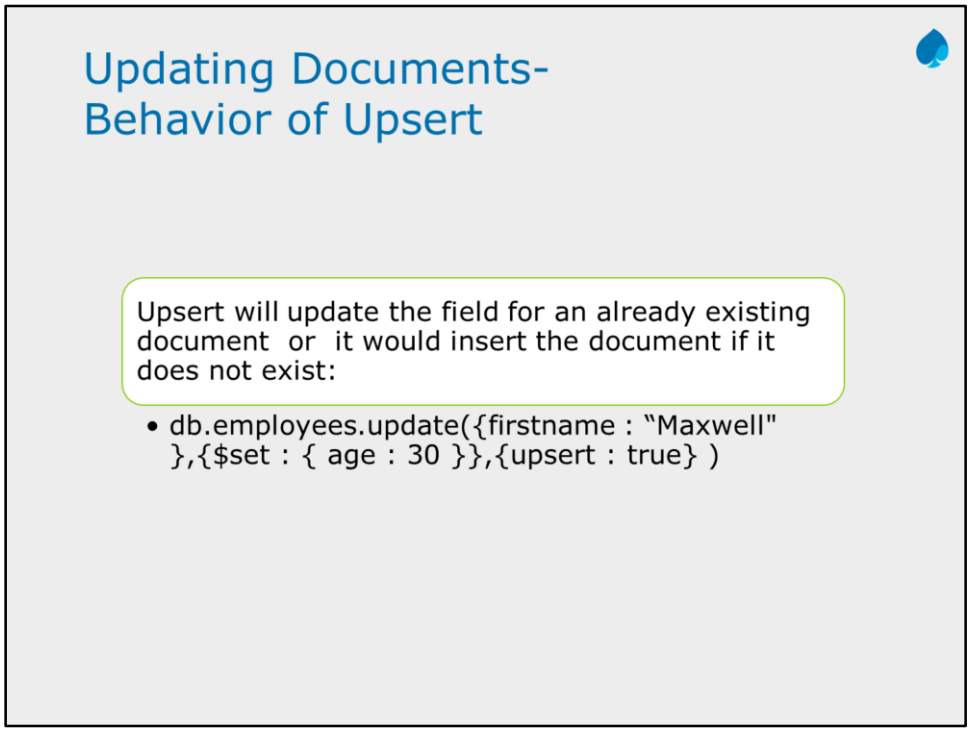
Updating Documents- Upsert

Upsert: A kind of update that either updates the first document matched in the provided query selector or, if no document matches, inserts a new document having the fields implied by the query selector and the update operation. The default value is false. When true, the update() method will update an existing document that matches the query selection criteria **or** if no document matches the criteria, insert a new document with the fields and values of the update parameter and if the update included

Multi (optional): Specifies whether to update multiple documents that meet the query criteria.

When not specified, the default value is false and the update() method updates a single document that meet the query criteria.

When true, the update() method updates all documents that meet the query criteria.


Updating Documents-
Behavior of Upsert
Upsert will update the field for an already existing document or it would insert the document if it does not exist:
• db.employees.update({firstname : "Maxwell" },{$set : { age : 30 }},{upsert : true})

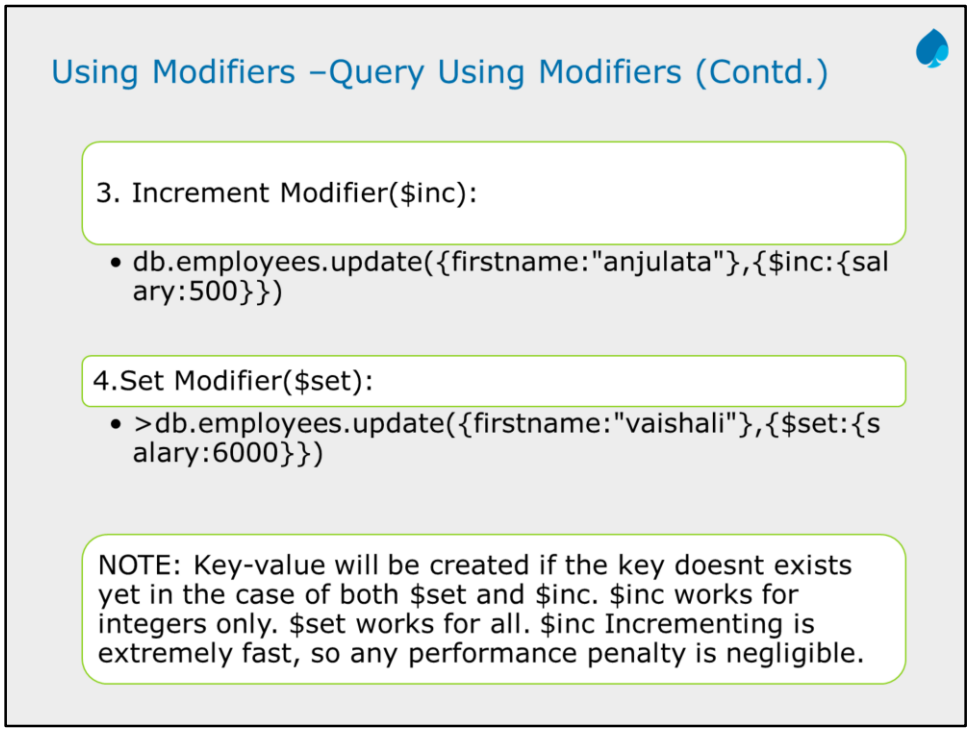
Using Modifiers – Query Using Modifiers

1.Not Equal Modifier($ne):

- db.employees.find({age : {$ne : 55}});

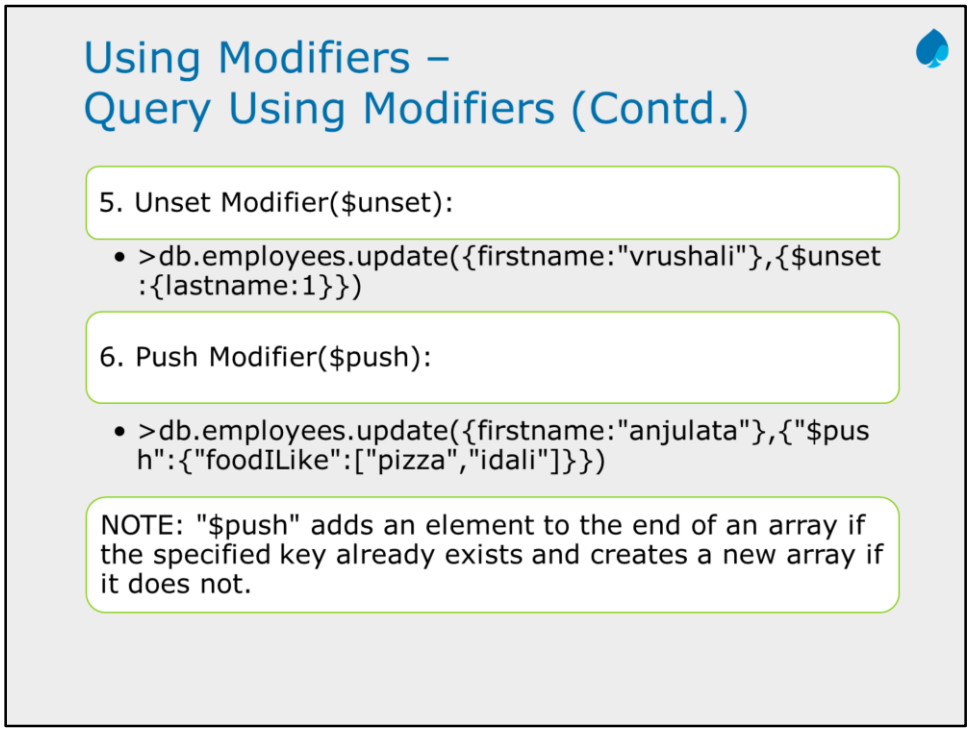
2. Greater/Less than Modifier($gt, $lt, $gte, $lte):

- db.employees.find({age : {$gt : 55}});
- db.employees.find({age : {$lt : 55}});
- db.employees.find({age : {$gte : 55}});
- db.employees.find({age : {$lte : 55}});

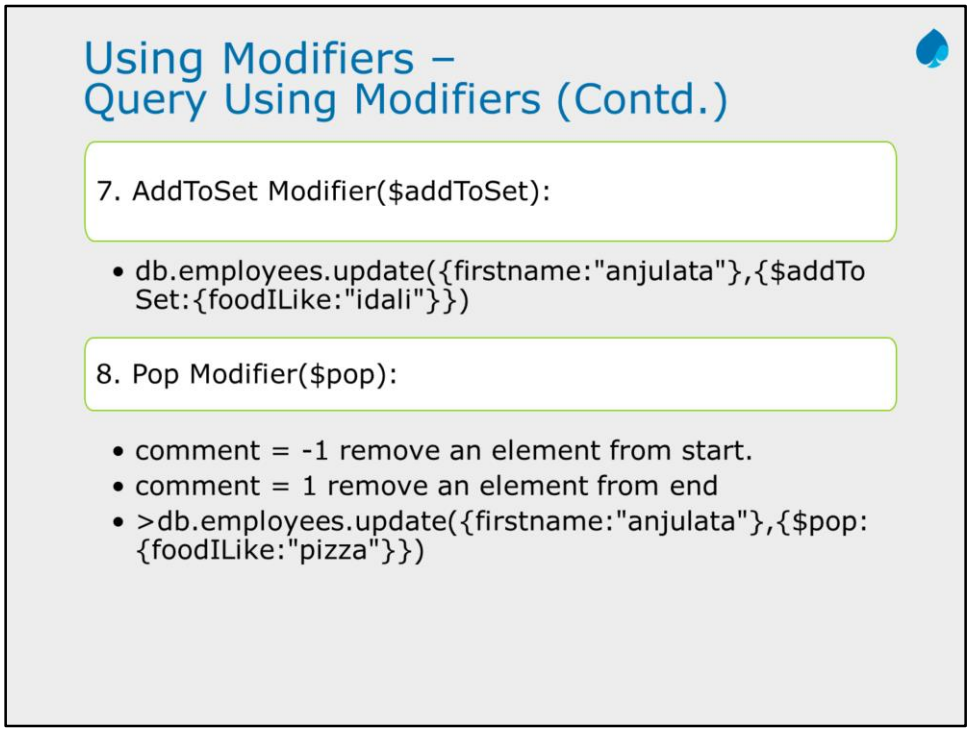


>db.employees.update({firstname:"anjulata"},{$inc:{salary:500}})- Update Salary Of Anjulata by 500

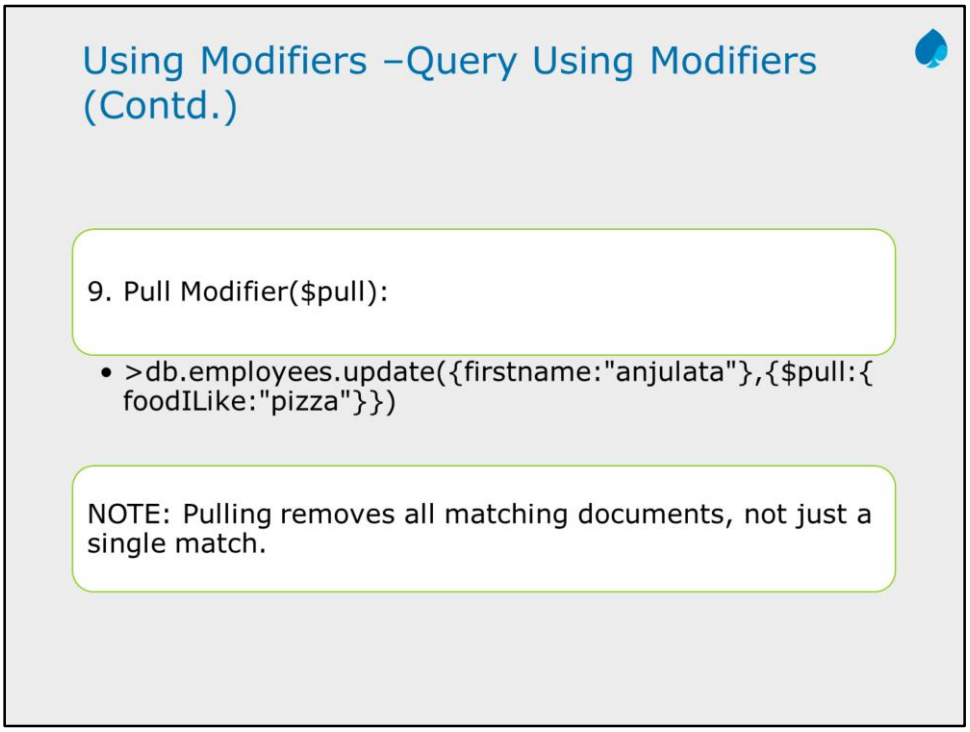
> db.employees.update({firstname:"vaishali"},{$set:{salary:6000}}) - Update employee and add only salary in existing record



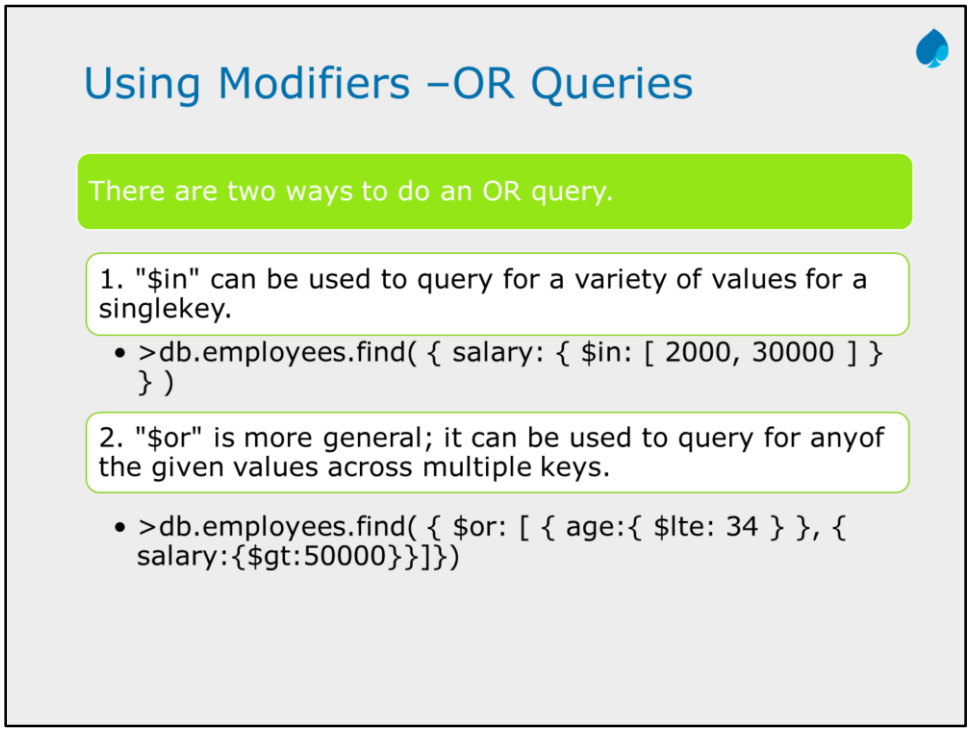
- db.employees.update({firstname:"vrushali"},{$unset:{lastname:1}}) - Use Unset Operator to remove the field
- > db.employees.update({firstname:"anjulata"},{"$push":{"foodILike":["pizza","idali"]}}) - Update existing record with name "anjulata" with an array of favourate food



db.employees.update({firstname:"anjulata"},{$addToSet:{foodILike:"idali"}}) ---- If elements are already existing $addToSet does nothing and do not add. But if it is not existing then it is added.



The all positional operator $[] indicates that the update operator should modify all elements in the specified array field
The $[] operator can be used for queries which traverse more than one array and nested arrays.
$Acts as a placeholder to update the first element that matches the query condition.
$[]Acts as a placeholder to update all elements in an array for the documents that match the query condition.
The positional $ operator has the form: { "<array>.$" : value }



Find the employees having matching salary as 2000 or 32000

->db.employees.find({ salary: { $in: [2000, 30000] } })
Note:-This is as good as IN operator Of oracle

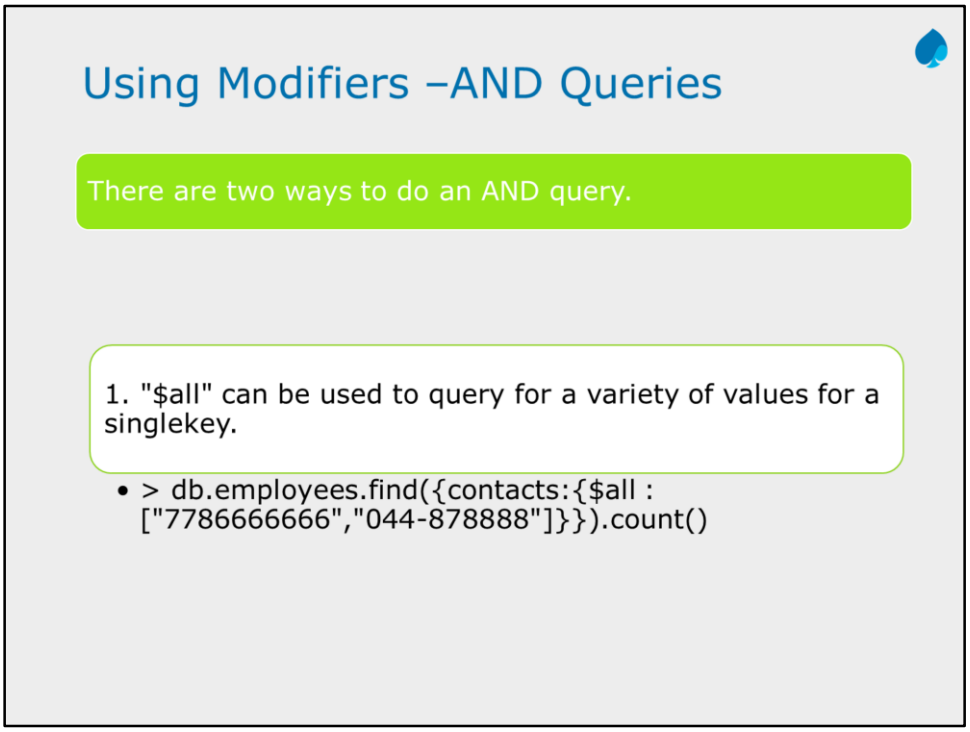
Find record having age <34 Or salary greater than 5000

>db.employees.find({ $or: [{ age:{ $lte: 34 } }, { salary:{$gt:50000}}]})

OR

Find record having age <34 and salary greater than 5000

>db.employees.find({ $and: [{ age:{ $lte: 34 } }, { salary:{$gt:50000}}]})



The $all is equivalent to an $and operation of the specified values; i.e. the following statement:

Find all employee having given all contacts-

> db.employees.find({contacts:{$all : ["7786666666","044-878888"]}}).count()

Using Modifiers –AND Queries (Contd.)

Simple queries are and queries.

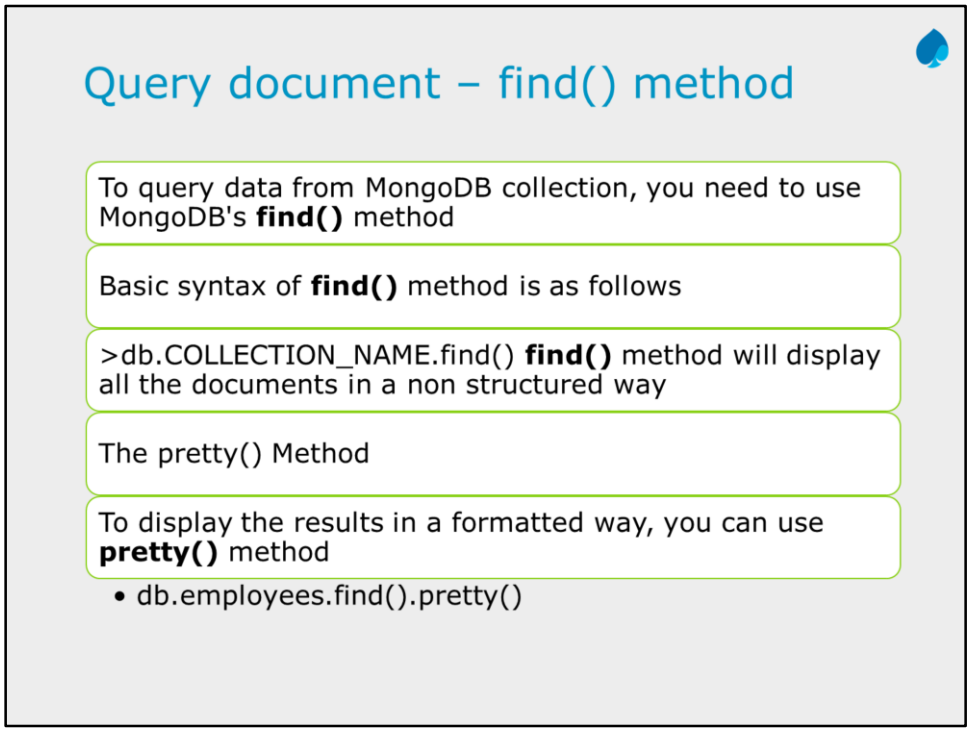
It can be used to query for any of the given values across single/multiple

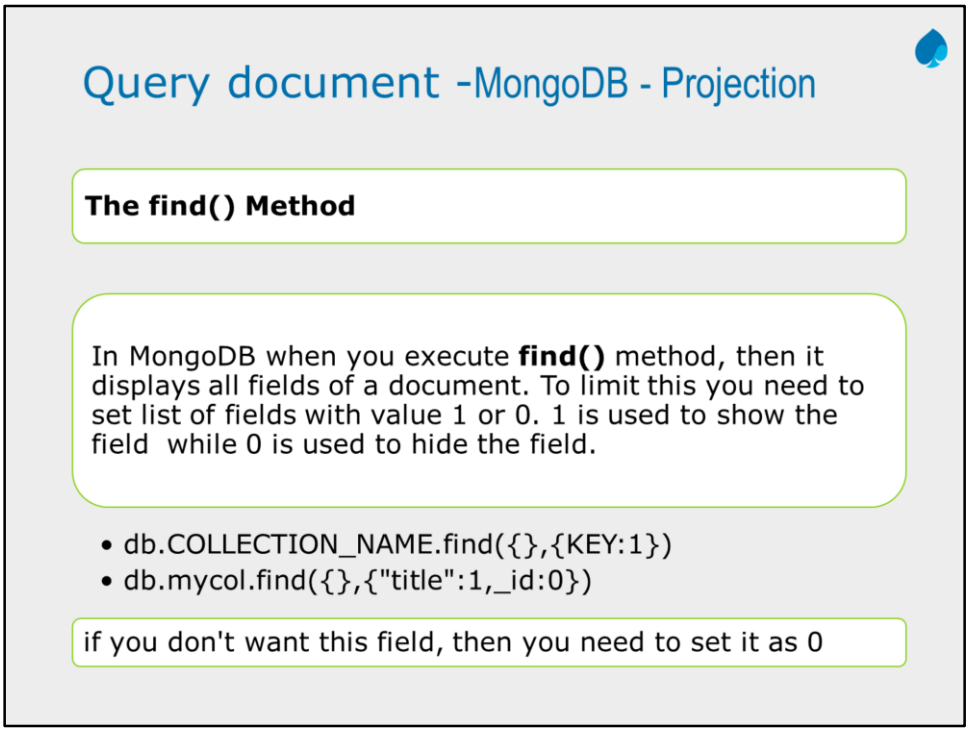
- >db.employees.find({"address.city":"Pune","gender":" Male"}).pretty()

Find all male employees staying in "Pune"

>

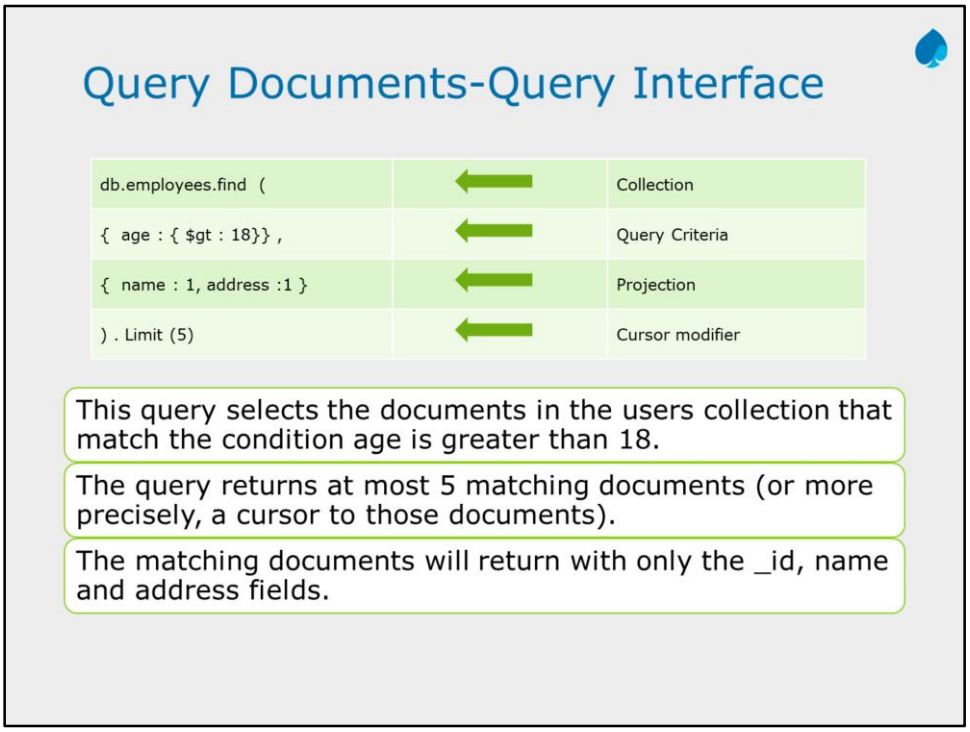
db.employees.find({"address.city":"Pune","gender":"Male"}).pretty()


Query document – find() method
To query data from MongoDB collection, you need to use MongoDB's **find()** method
Basic syntax of **find()** method is as follows
>db.COLLECTION_NAME.find() **find()** method will display all the documents in a non structured way
The pretty() Method
To display the results in a formatted way, you can use **pretty()** method
• db.employees.find().pretty()



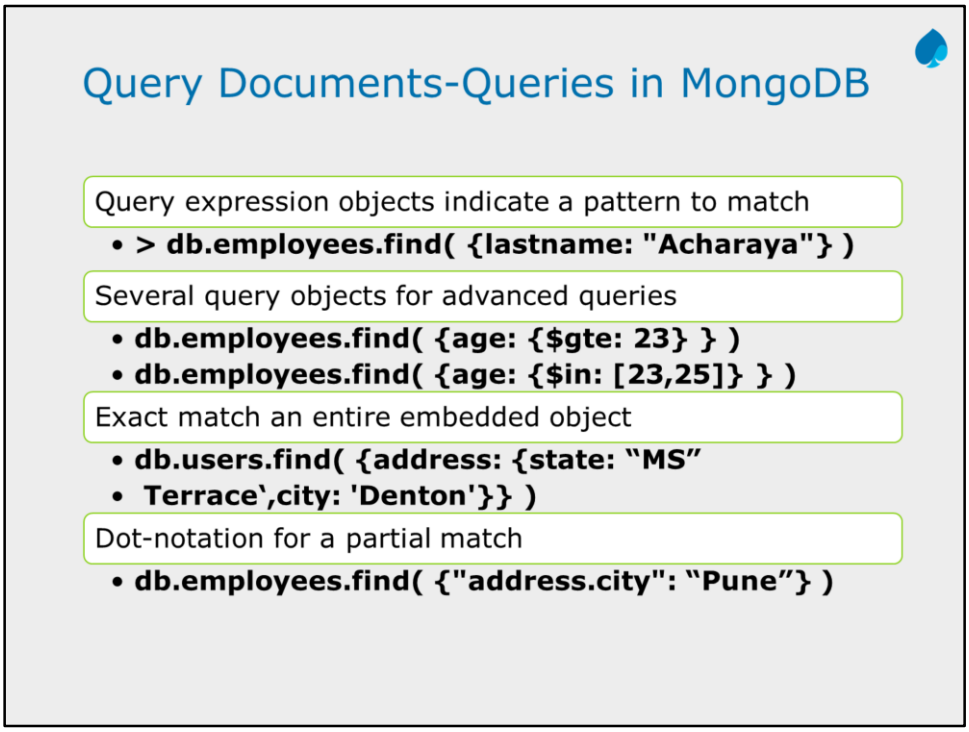
OR show all employee details without firstname,lastname,address field

>db.employees.find({},{"firstname":0,"lastname":0,"address":0}
).pretty()



Find firstname lastname and age for those employee having age greater than 50 with first 5 record>

```
db.employees.find({age:{$gt:50}},{firstname:1,lastname:1,age:1}).limit(5).pret
ty()
{
        "_id" : ObjectId("5ae7ee6465cef006130f1abb"),
        "firstname" : "shilpa",
        "lastname" : "bhosale",
        "age" : 55
}
{
        "_id" : ObjectId("5ae8862d65cef006130f1abe"),
        "firstname" : "shilpa",
        "lastname" : "sharma",
        "age" : 55
}
{
        "_id" : ObjectId("5aeb01bab190985244cff581"),
        "firstname" : "shilpa",
        "lastname" : "bhosale",
        "age" : 60
}
{
        "_id" : ObjectId("5aeb01c5b190985244cff584"),
        "firstname" : "shilpa",
        "lastname" : "bhosale",
        "age" : 60
}
>
```



Find employees between age of 20 and 60- db.employees.find({age: {$in:[20,60]}}).pretty()

db.employees.find({"address.city":"Pune"})

Query Documents-Comparison Operators

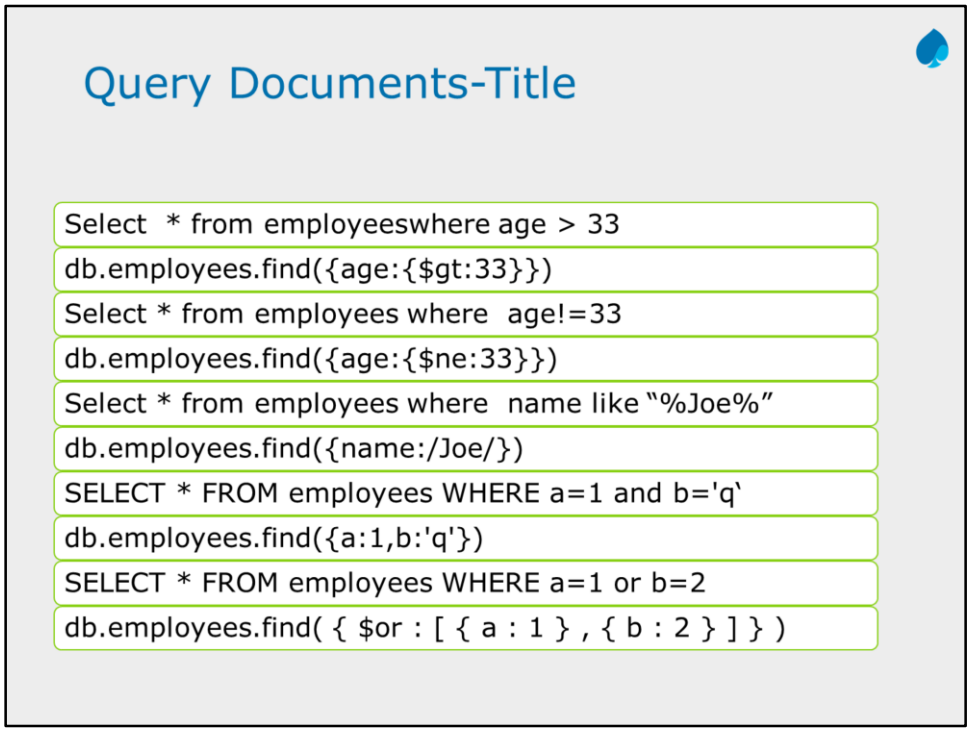
Operators	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$lt	Matches values that are less than a specified value.

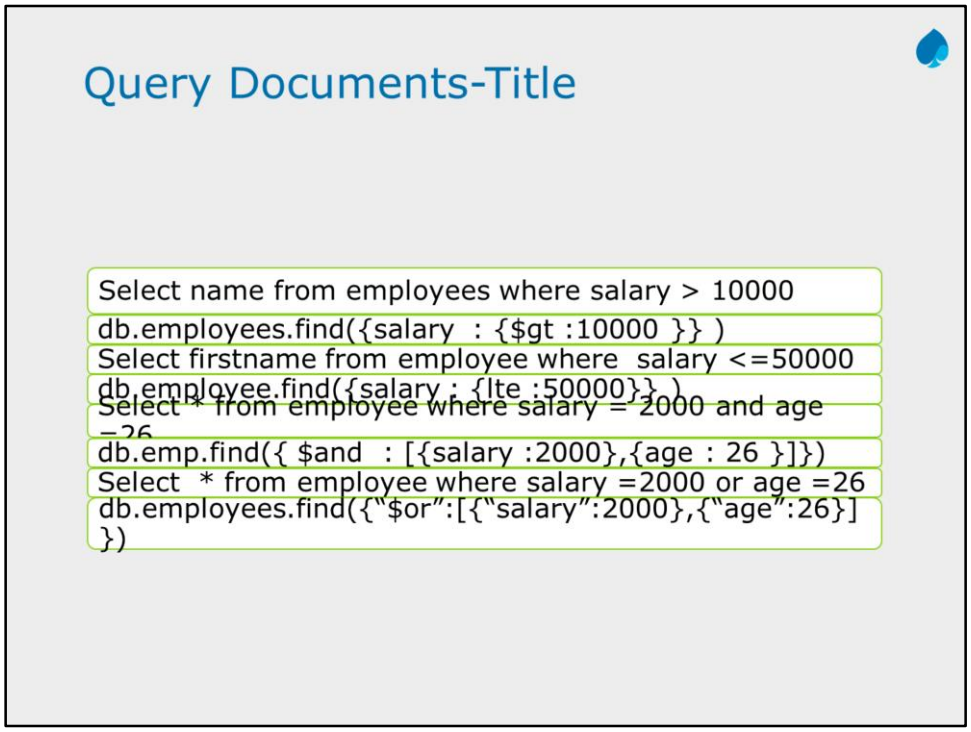
Query Documents-Comparison Operators

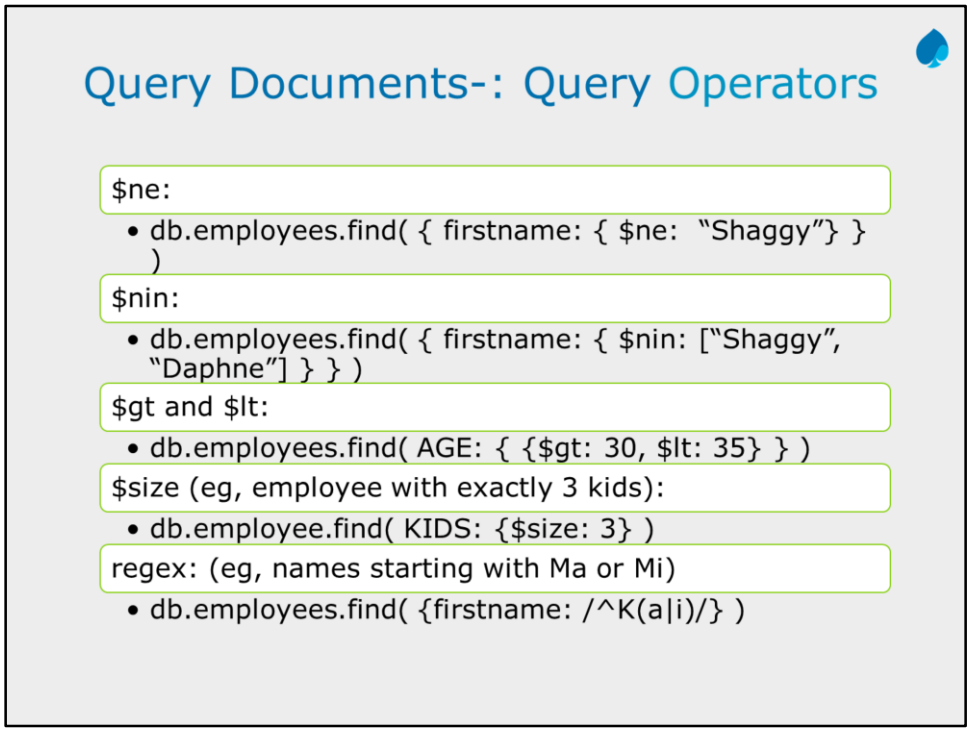
Operators	Description
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$in	Matches any of the values specified in an array.
$nin	Matches none of the values specified in an array.

Query Documents-Logical Operators

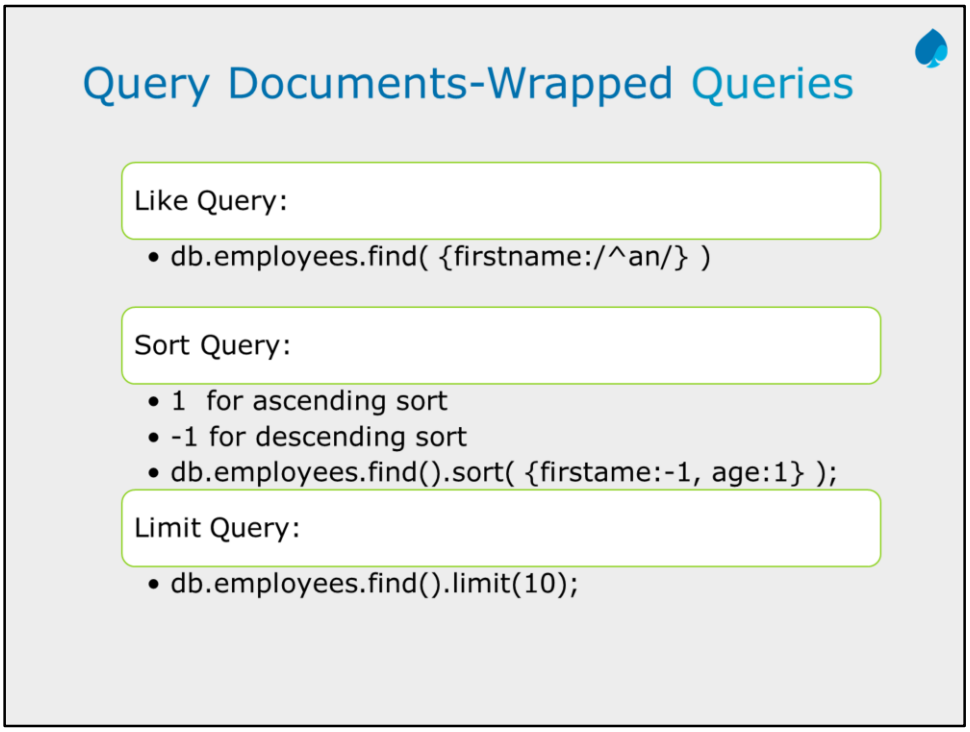
Name	Description
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$in	Matches any of the values specified in an array.
$nin	Matches none of the values specified in an array.


Query Documents-Title
Select * from employeeswhere age > 33
db.employees.find({age:{$gt:33}})
Select * from employees where age!=33
db.employees.find({age:{$ne:33}})
Select * from employees where name like "%Joe%"
db.employees.find({name:/Joe/})
SELECT * FROM employees WHERE a=1 and b='q'
db.employees.find({a:1,b:'q'})
SELECT * FROM employees WHERE a=1 or b=2
db.employees.find({ $or : [{ a : 1 } , { b : 2 }] })


Query Documents-Title
Select name from employees where salary > 10000
db.employees.find({salary : {$gt :10000 }})
Select firstname from employee where salary <=50000
db.employee.find({salary : {lte :50000}})
Select * from employee where salary =2000 and age =26
db.emp.find({ $and : [{salary :2000},{age : 26 }]})
Select * from employee where salary =2000 or age =26
db.employees.find({"$or":[{"salary":2000},{"age":26}] })



db.employees.find({firstname: /^M(a|i)/})



db.employees.find({firstname:/^an/})
db.employees.find().sort({lastname:-1,age:1}).pretty()

Query Documents-Wrapped Queries (Contd.)

Count Query:

- db.employees.find().count();

Skip Query:

- db.employees.find().skip(5);

Query Documents-Querying on Embedded Document

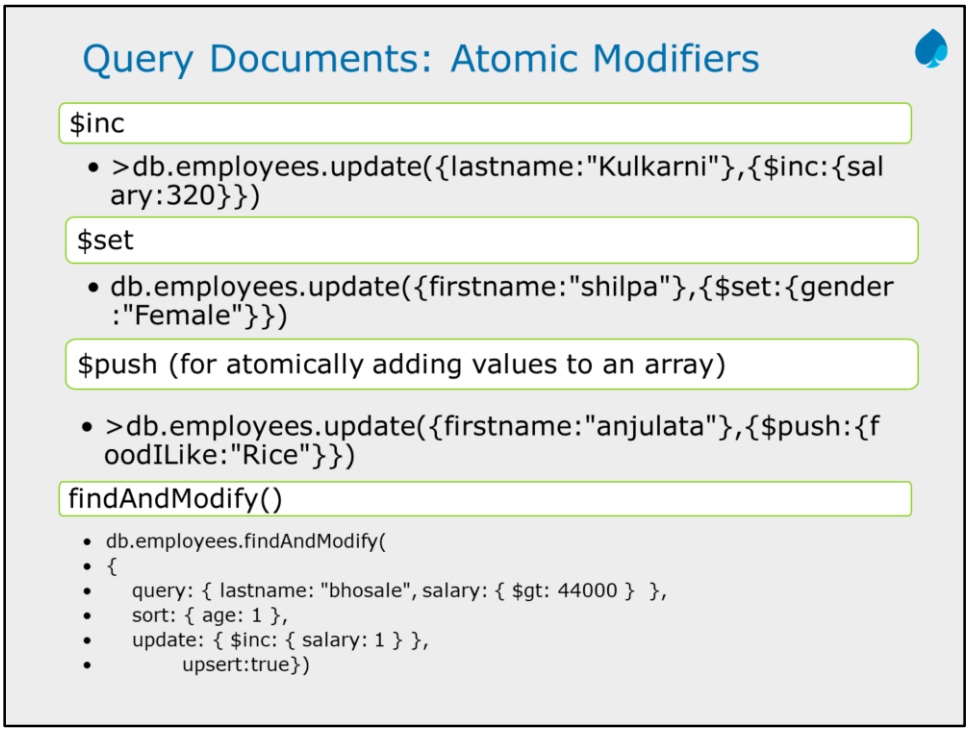
There are two ways of querying for an embedded document.

1. Querying for an entire embedded document works identically to a normal query.

- db.employees.find({name: { first:"Amit", last:"Kumar" } })
- This query will do exact match and order too matters, if order will change then it will not find.

2. Querying for its individual key/value pairs.

- db.employees.find({"name.first" : "Amit", "name.last" : "Kumar"})



Increase Kulkarni salary by 320 ->
db.employees.update({lastname:"Kulkarni"},{$inc:{salary:320}})

Update Shilpa's Gender -> >
db.employees.update({firstname:"shilpa"},{$set:{gender:"Female"}})

db.collection.findAndModify(*document*)¶

Modifies and returns a single document. By default, the returned document does not include the modifications made on the update.
To return the document with the modifications made on the update, use the new option.
The findAndModify() method is a shell helper around the findAndModify command.

EX
```
db.employees.findAndModify(
{
    query: { lastname: "bhosale",
                                        salary: { $gt: 44000 }
                        },
    sort: { age: 1 },
    update: { $inc: { salary: 1 } },
                        upsert:true
})
```

- The query finds a document in the employees collection where the last name field has the value bhosale, And salary field has a value greater than 44000.
- The sort orders the results of the query in ascending order. If multiple documents meet the query condition, the method will select for modification the first document as ordered by this sort.
- The update increments the value of the salary field by 1.
- The method returns the original (i.e. pre-modification) document selected for this update
- upsert: true -returns modified document

Query Documents-: Atomic Modifiers (Contd.)

```
SELECT * FROM employees WHERE fnamename='bob' and
(age=30 or age=50 )
```

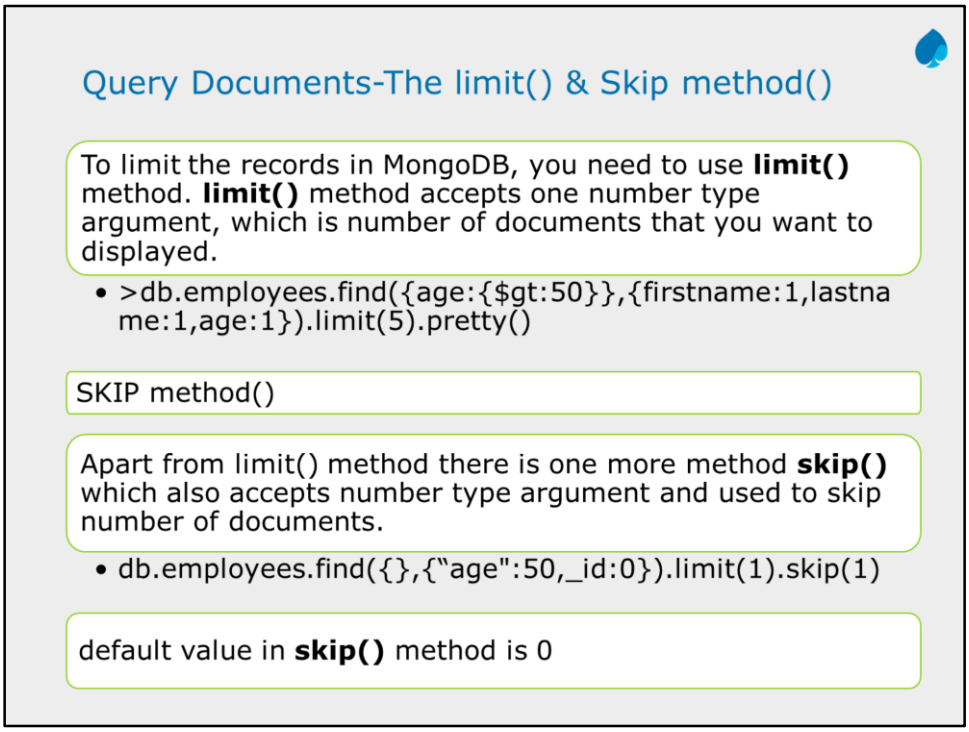
- db.employees.find({ firstname : "bob" , $or : [{ age : 30 } , { age : 50 }] })

```
SELECT * FROM employees WHERE age>33 AND age<=40
```

- db.employees.find({"age":{$gt:33,$lte:40}})

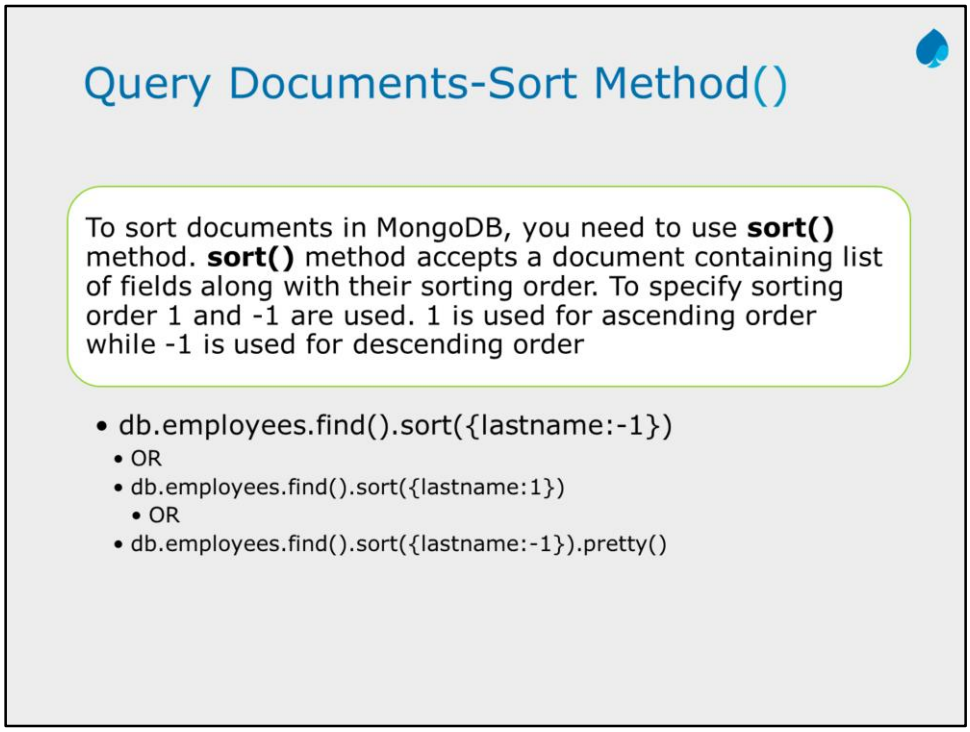
Query Documents-Query Behavior

- All queries in MongoDB address a single Collection
- Modify the query to impose limits ,skips and sort orders
- The order of documents returned by a query is not defined unless you specify a sort()
- Operations that modify existing documents (i.e. updates) use the same query syntax as queries to select documents to update
- In aggregation pipeline , the $match pipeline stage provides access to MongoDB queries



Find firstname lastname and age for those employee having age greater than 50 with first 5 record

```
>
db.employees.find({age:{$gt:50}},{firstname:1,lastname:1,age:1}).limit(5).pretty()
{
        "_id" : ObjectId("5ae7ee6465cef006130f1abb"),
        "firstname" : "shilpa",
        "lastname" : "bhosale",
        "age" : 55
}
{
        "_id" : ObjectId("5ae8862d65cef006130f1abe"),
        "firstname" : "shilpa",
        "lastname" : "sharma",
        "age" : 55
}
{
        "_id" : ObjectId("5aeb01bab190985244cff581"),
        "firstname" : "shilpa",
        "lastname" : "bhosale",
        "age" : 60
}
{
        "_id" : ObjectId("5aeb01c5b190985244cff584"),
        "firstname" : "shilpa",
        "lastname" : "bhosale",
        "age" : 60
}
```



Sort Document By lastname

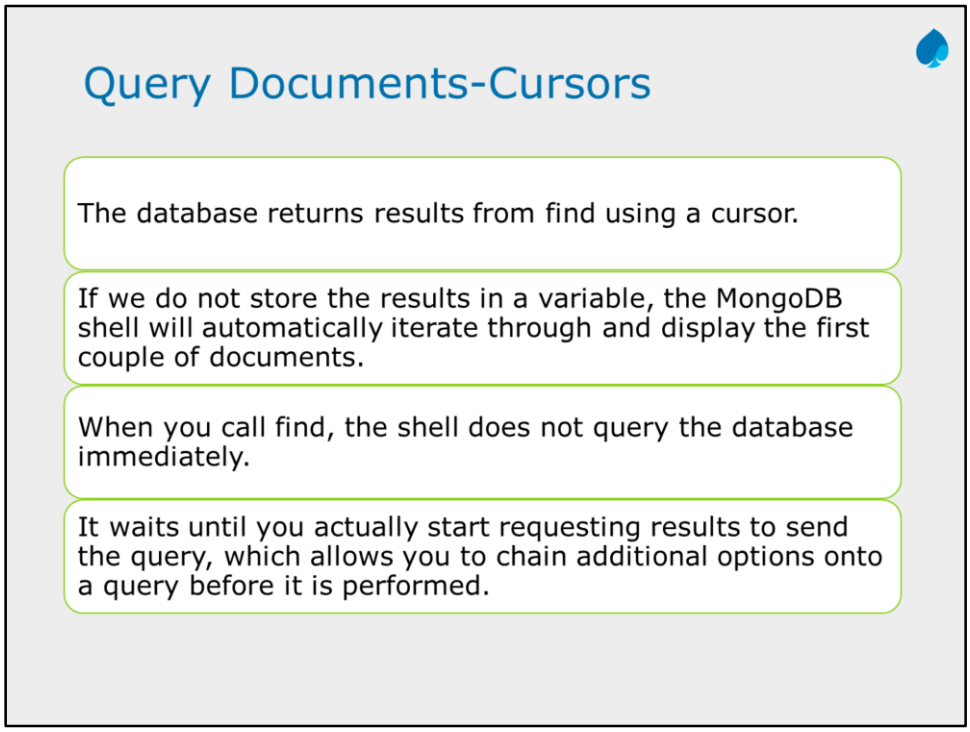
> db.employees.find().sort({lastname:-1})

OR

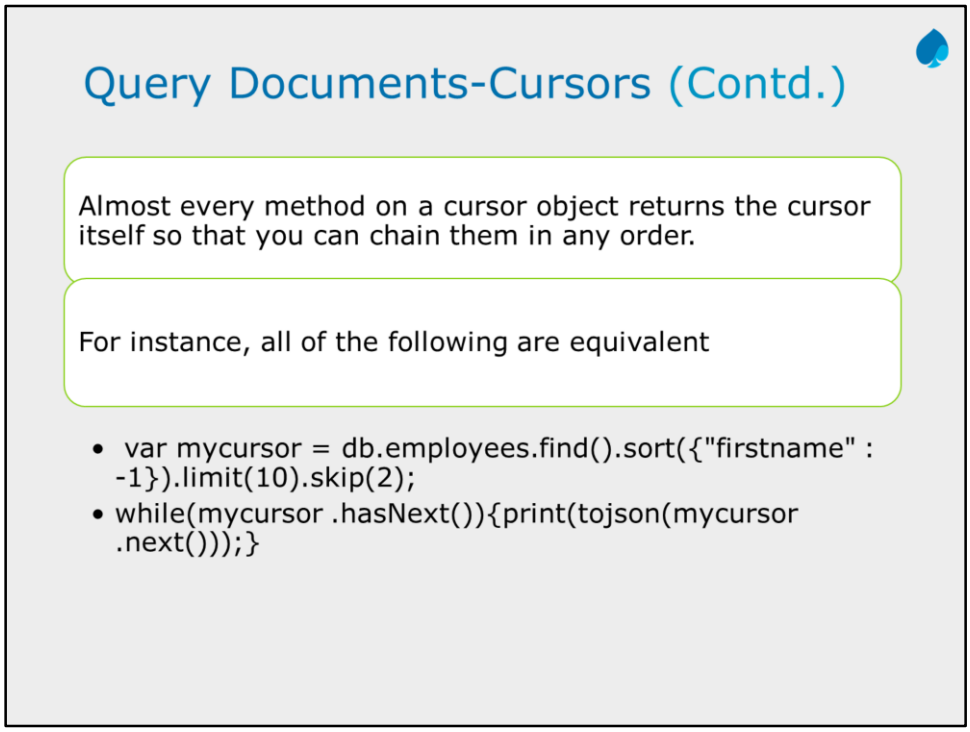
db.employees.find().sort({lastname:1})

OR

db.employees.find().sort({lastname:-1}).pretty()



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.



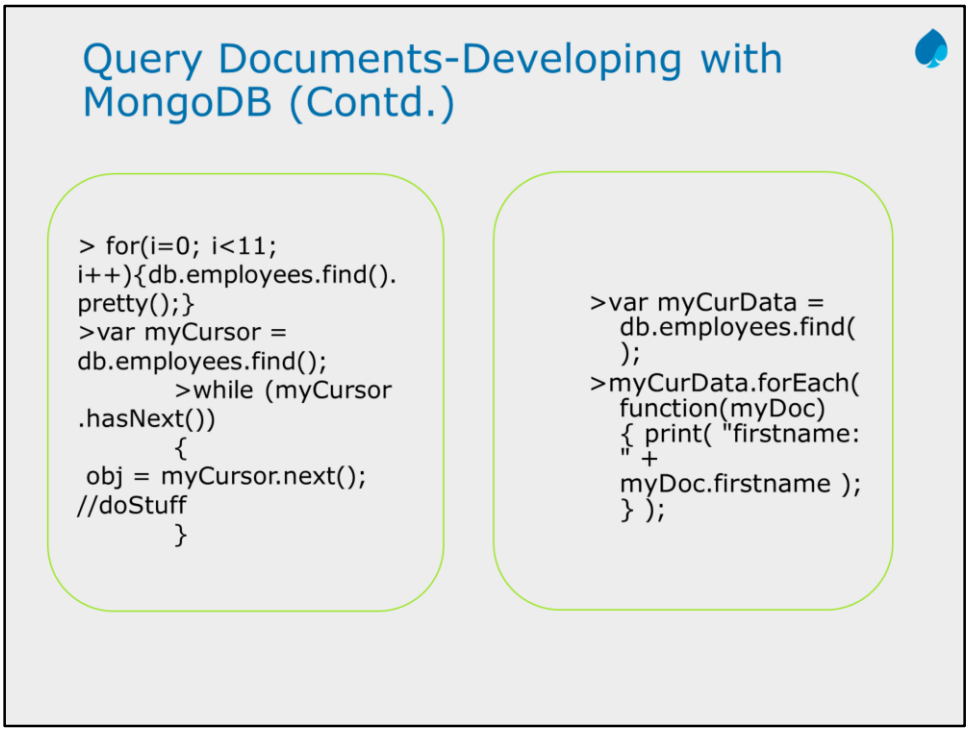

```
var myEmployee = db.employees.find( { salary : { $gt:70000 }});
while(myEmployee.hasNext()){print(tojson(myEmployee.next()));}
---------------------------------------------------------------------------

var mycursor = db.employees.find().sort({"firstname" : -1}).limit(10).skip(2);
while(mycursor .hasNext()){print(tojson(mycursor .next()));}
```

Query Documents-Developing with MongoDB

Behind Find() : Cursor

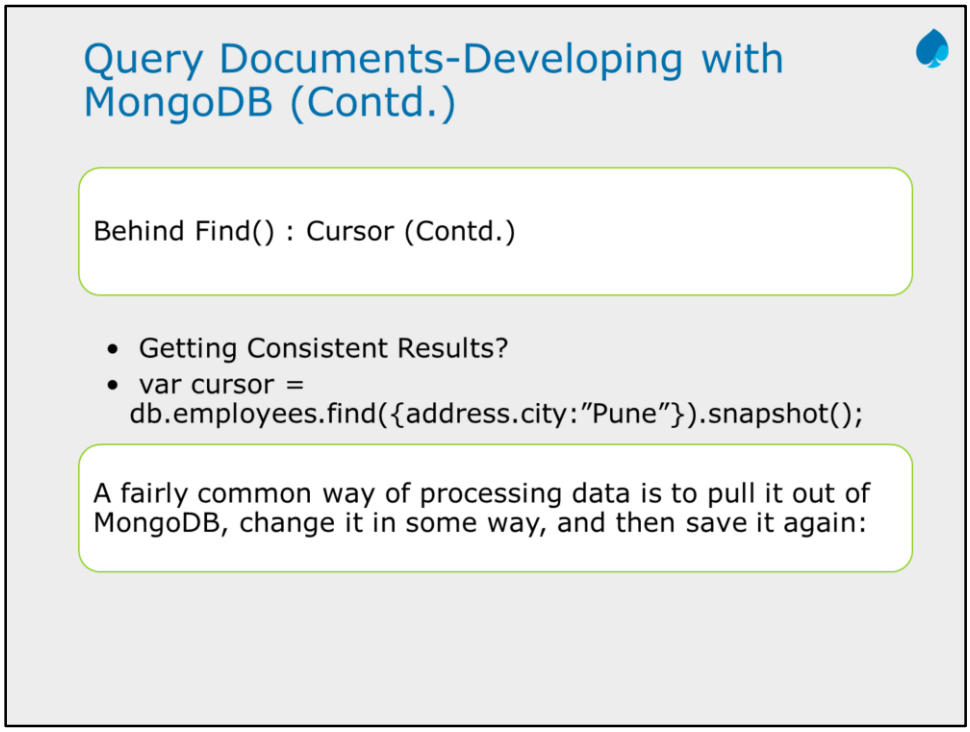
- The database returns results from find using a *cursor*.
- The client-side implementations of cursors generally allow you to control a great deal about the eventual output of a query.



cursor.forEach(*function*)-
Iterates the cursor to apply a JavaScript function to each document from the cursor.
The forEach() method has the following prototype form:
db.collection.find().forEach(<function>)
The forEach() method has the following parameter:
Parameter - function
Type - JavaScript
Description- A JavaScript function to apply to each document from the cursor. The <function> signature includes a single argument that is passed the current document to process.

Fetch all employee Data One By One

```
> for(i=0; i<11; i++){db.employees.find().pretty();}
OR
>var myCursor = db.employees.find();
>while (myCursor .hasNext())
{
        obj = myCursor.next();
}
```



http://www.mongodb.org/display/DOCS/How+to+do+Snapshotted+Queries+in+the+Mongo+Database

cursor.snapshot()-
cursor.snapshot()- Append the snapshot() method to a cursor to toggle the "snapshot" mode. This ensures that the query will not return a document multiple times, even if intervening write operations result in a move of the document due to the growth in document size.

Warning
You must apply snapshot() to the cursor before retrieving any documents from the database.
You can only use snapshot() with **unsharded** collections.

The snapshot() does not guarantee isolation from insertion or deletions.
The snapshot() **cannot** be used with sort() or hint().

Snapshot Mode

snapshot() mode assures that objects which update during the lifetime of a query are returned once and only once. This is most important when doing a find-and-update loop that changes the size of documents that are returned ($inc does not change size).

> // mongo shell example
>var cursor = db.employees.find({address.city:"Pune"}).snapshot();

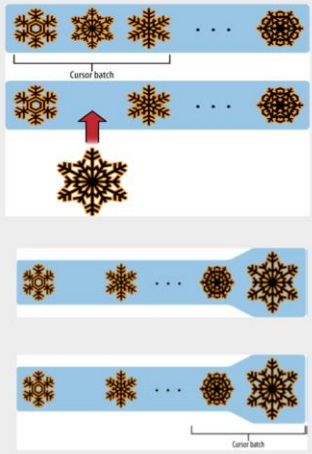
Even with snapshot mode, items inserted or deleted during the query may or may not be returned; that is, this mode is not a true point-in-time snapshot.

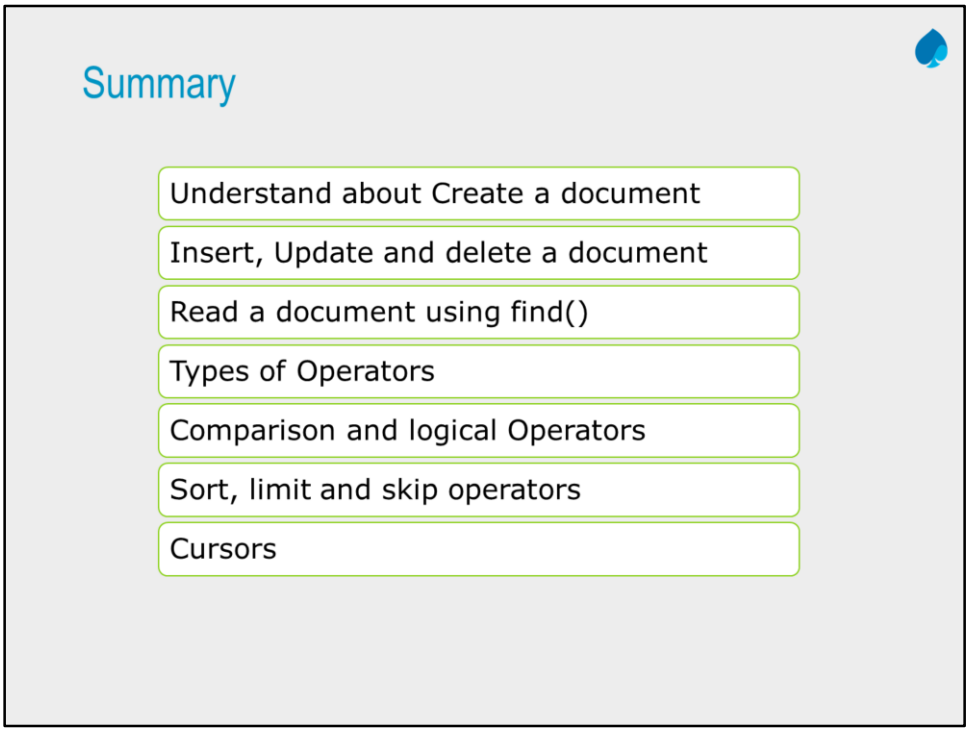
Because snapshot mode traverses the _id index, it may not be used with sorting or explicit hints. It also cannot use any other index for the query.

You can get the same effect as snapshot by using any unique index on a field(s) that will not be modified (probably best to use explicit hint() too).
If you want to use a non-unique index (such as creation time), you can make it unique by appending _id to the index at creation time.

Query Documents-Developing with MongoDB (Contd.)

```
cursor = db.foo.find();
while
   (cursor.hasNext())
   {
  var doc =
   cursor.next();
  doc = process(doc);
  db.foo.save(doc);
}
```





Create

- The field name _id is reserved for use as a primary key; its value must be unique in the collection, is immutable, and may be of any type other than an array.
- The field names **cannot** start with the $ character.
- The field names **cannot** contain the . character.

Create with save

- If the <document> argument does not contain the _id field or contains an _id field with a value not in the collection, the save() method performs an insert of the document.
- Otherwise, the save() method performs an update.

sds