MongoDB – CRUD

Agenda

Module Objectives

ease

the
nt

What you will learn

At the end of this module, you will learn:

- What are Crud Operations

What you will be able to do

At the end of this module, you be able to:

- Understand what are Crud Operations
- Explain what is Upsert
- Describe Query Interface
- List the Comparison Operators and Logical Operators
- State what are Wrapped Queries and Query Operators

9

Crud Operations – Create, Read, Update, Delete

10

CRUD

Create

- db.collection.insert(<document>)
- db.collection.save(<document>)
- db.collection.update(<query>, <update>, { upsert: true })

Read

- db.collection.find(<query>, <projection>)
- db.collection.findOne(<query>, <projection>)

Update

- db.collection.update(<query>, <update>, <options>)

Delete

- db.collection.remove(<query>, <justOne>)

CRUD Example

```
>db.user.insert({
first: "John",
    last : "Doe",
      age: 39
      })
```

```
>       db.user.find ()
            {
"_id" : ObjectId("51…"),
      "first" : "John",
      "last" : "Doe",
        "age" : 39
            }
```

```
>      db.user.update(
{"_id" : ObjectId("51…")},
          {
       $set: {
        age: 40,
      salary: 7000}
          }
          )
```

```
>db.user.remove({
"first": /^J/
         })
```

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:

•>db.COLLECTION_NAME.insert(document)
Example:

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

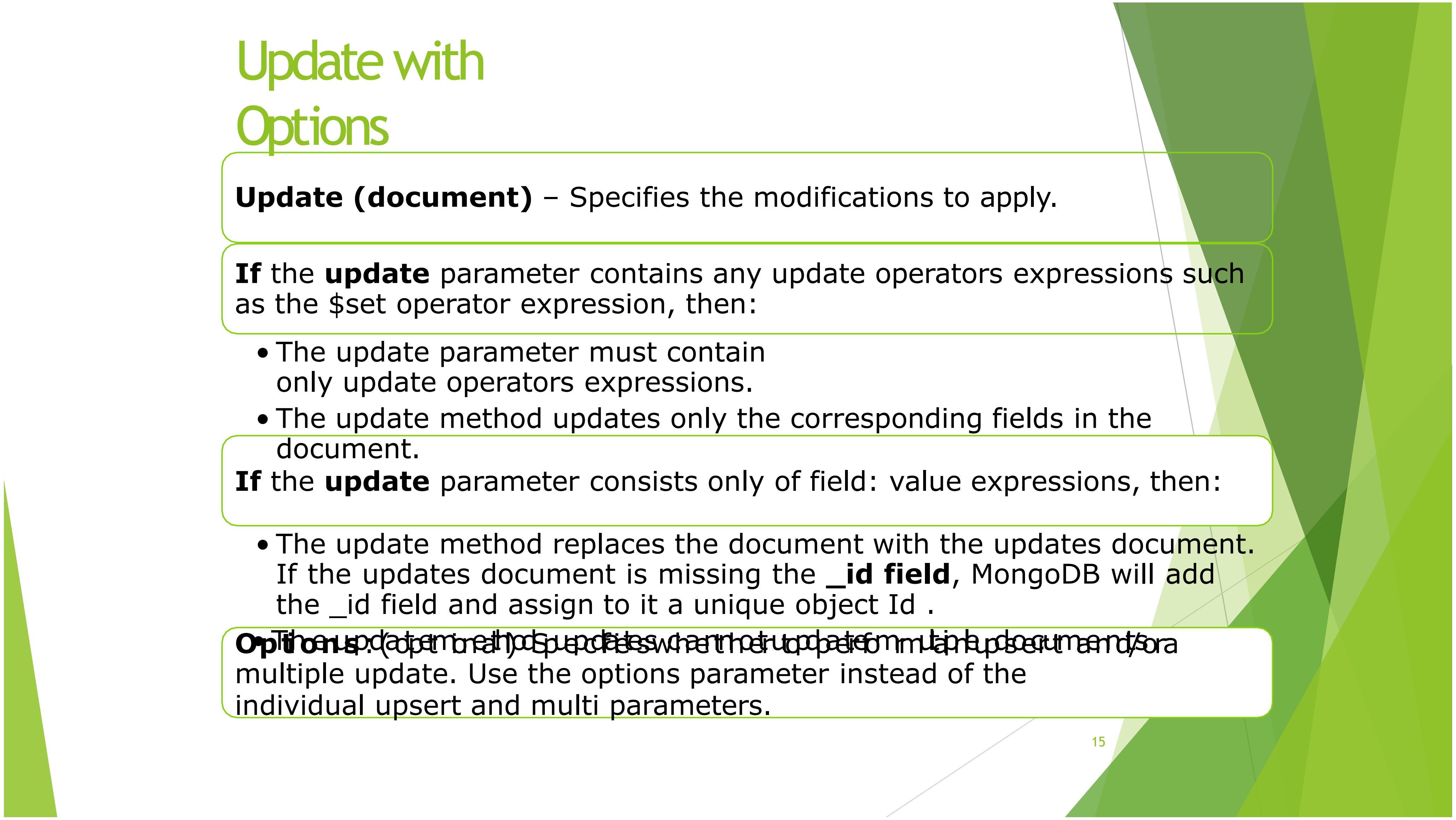
Update document - update() method

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:

- >use mydb
  - >switched to db mydb >db.dropDatabase()
  - >{ "dropped" : "mydb", "ok" : 1 }



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.

15

9

Update Example

To update multiple you need to set a parameter 'multi' to true:

- >db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}},{multi:true})

Upsert

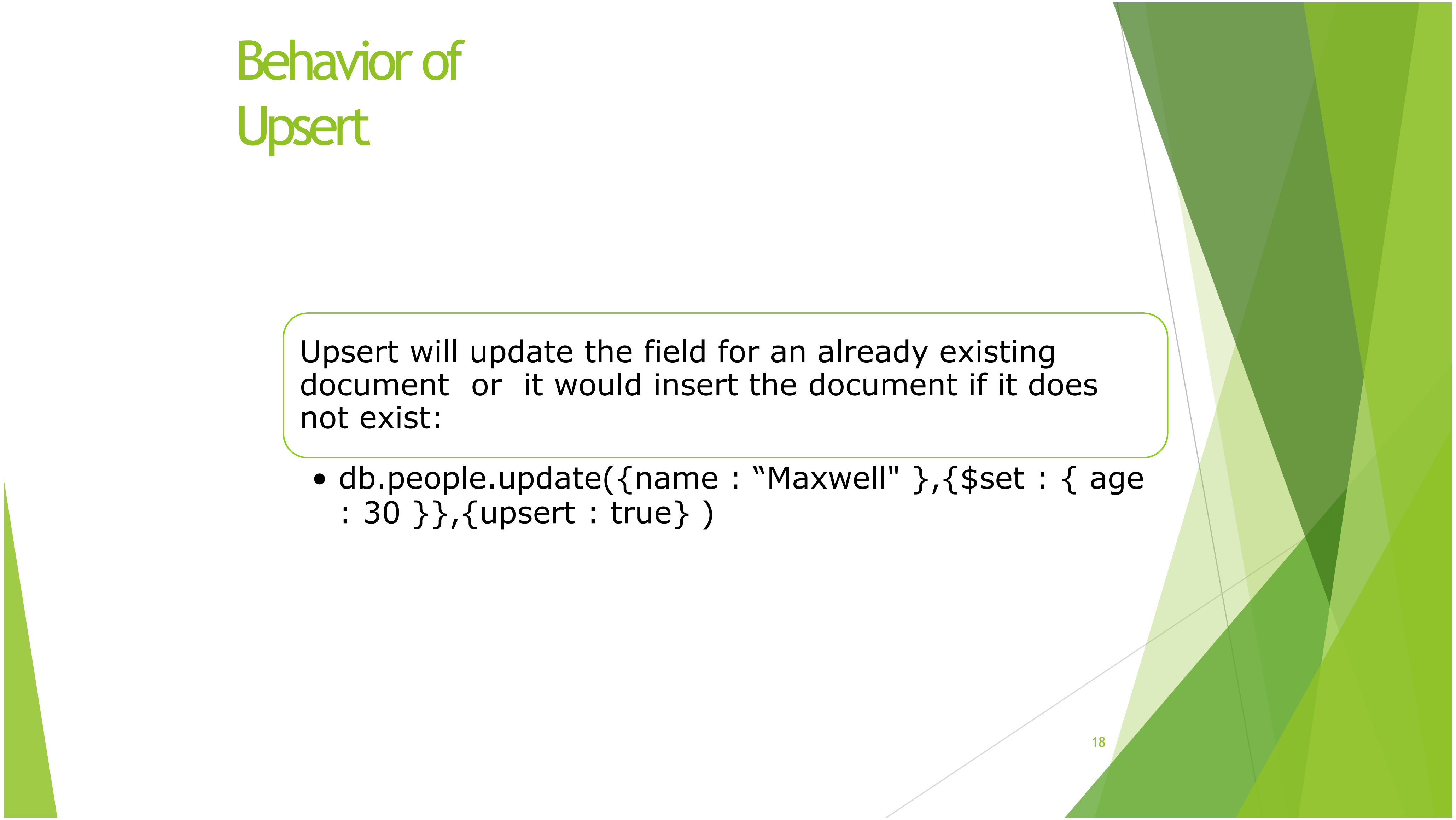
Upsert: A kind of update that either updates the first document matched in the provided query selector or, if no document matches, inserts a new document having the fields implied by the query selector and the update operation. The default value is false. When true, the update() method will update an 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 only update operators, the query parameter as well.

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.

17



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.people.update({name : “Maxwell" },{$set : { age : 30 }},{upsert : true})

18

12

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.mycol.remove({'title':'MongoDB Overview'})

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

20

14

MongoDB - Projection

The find() Method

In MongoDB when you execute **find()** method, then it displays all fields of a document. To limit this you need to set list of fields with value 1 or 0. 1 is used to show the filed while 0 is used to hide the field.

- db.COLLECTION_NAME.find({},{KEY:1})
- db.mycol.find({},{"title":1,_id:0})

if you don't want this field, then you need to set it as 0

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.mycol.find().pretty()

Query Interface

db.users.find (	⬅	Collection
{ age : { $gt : 18}} ,	⬅	Query Criteria
{ name : 1, address :1 }	⬅	Projection
) . Limit (5)	⬅	Cursor modifier

This query selects the documents in the users collection that match the condition age is greater than 18.

The query returns at most 5 matching documents (or more precisely, a cursor to those documents).

The matching documents will return with only the _id, name and address fields.

23

Queries in MongoDB

Query expression objects indicate a pattern to match

- **db.users.find({last_name: 'Smith'})**

Several query objects for advanced queries

- **db.users.find({age: {$gte: 23} })**
- **db.users.find({age: {$in: [23,25]} })**

Exact match an entire embedded object

- **db.users.find({address: {street: 'Oak Terrace‛,city: 'Denton'}})**

Dot-notation for a partial match

- **db.users.find({"address.city": 'Denton'})**

24

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.

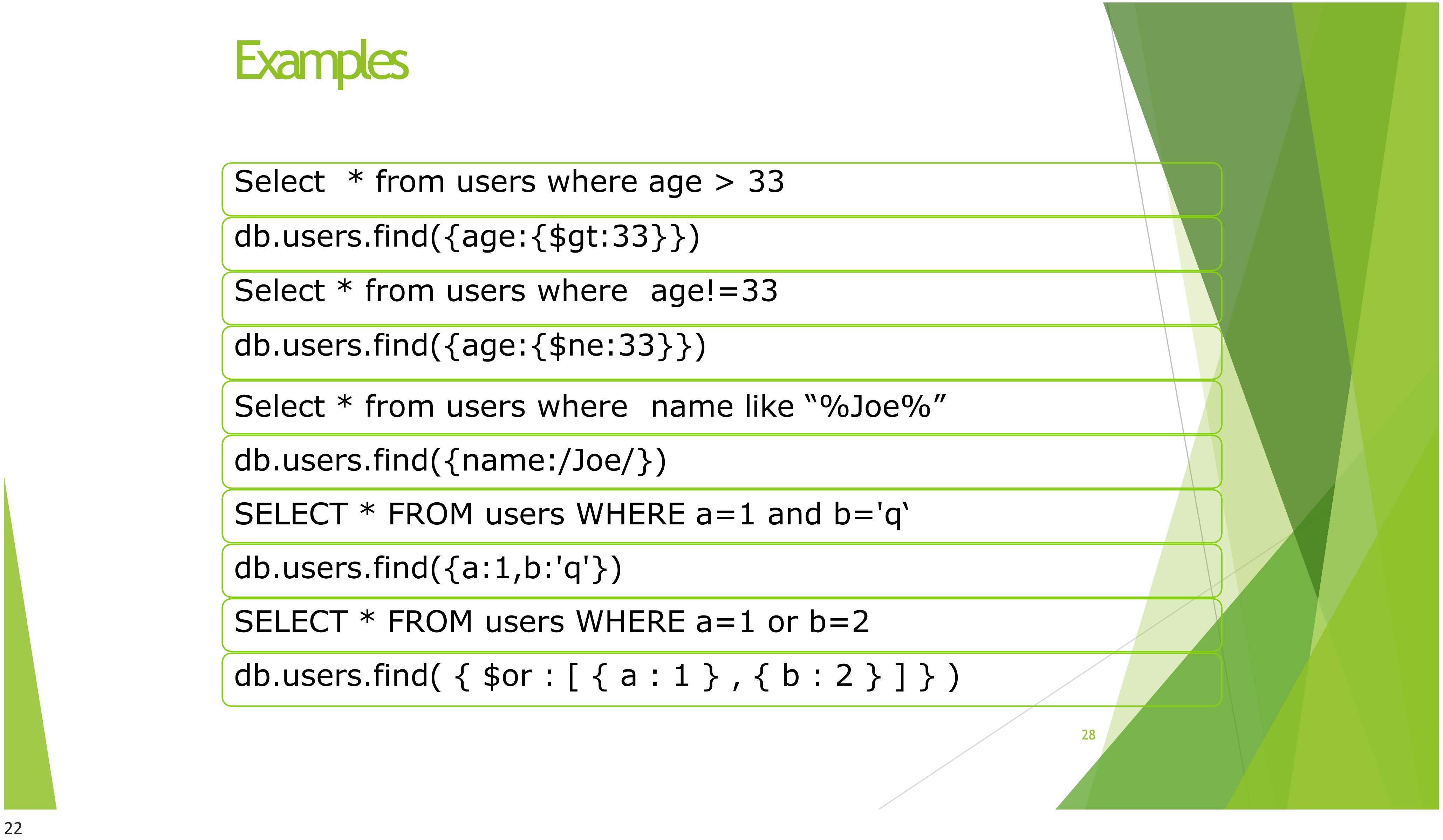
Comparison Operators

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.

26

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.



Examples

```
Select  * from users where age > 33
db.users.find({age:{$gt:33}})
Select * from users where  age!=33
db.users.find({age:{$ne:33}})
Select * from users where  name like “%Joe%”
db.users.find({name:/Joe/})
SELECT * FROM users WHERE a=1 and b='q‘
db.users.find({a:1,b:'q'})
SELECT * FROM users WHERE a=1 or b=2
db.users.find( { $or : [ { a : 1 } , { b : 2 } ] } )
```

28

22

Examples

```
Select name from emp where sal  > 10000
```

```
db.emp.find({sal : {$gt :10000 }} )
```

```
Select name from emp where  sal <=50000
```

```
db.emp.find({sal : {lte :50000}} )
```

```
Select * from emp where salary = 2000 and age =26
```

```
db.emp.find({ $and  : [{salary :2000},{age : 26 }]})
```

```
Select  * from emp where salary =2000 or age =26
```

```
db.emp.find({“$or”:[{“salary”:2000},{“age”:26}]})
```

Awesome: Query Operators

$ne:

- db.people.find({ NAME: { $ne: “Shaggy”} })

$nin:

- db.people.find({ NAME: { $nin: [“Shaggy”, “Daphne”] } })

$gt and $lt:

- db.people.find(AGE: { {$gt: 30, $lt: 35} })

$size (eg, people with exactly 3 kids):

- db.people.find(KIDS: {$size: 3})

regex: (eg, names starting with Ma or Mi)

- db.people.find({NAME: /^M(a|i)/})

Wrapped Queries

Like Query:

- db.Tag.find({name:/^term/})

Sort Query:

- 1 for ascending sort
- -1 for descending sort
- db.Tag.find().sort({userName:-1, age:1});

Limit Query:

- db.Tag.find().limit(10);

Wrapped Queries (Contd.)

Count Query:

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

Skip Query:

- db.Tag.find().skip(50);

Query Using Modifiers

1.Not Equal Modifier($ne):

- db.Tag.find({firstProperty : {$ne : 3}});

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

- db.Tag.find({firstProperty : {$gt : 2}});
- db.Tag.find({firstProperty : {$lt : 2}});
- db.Tag.find({firstProperty : {$gte : 2}});
- db.Tag.find({firstProperty : {$lte : 2}});

33

Query Using Modifiers (Contd.)

3. Increment Modifier($inc):

- db.Tag.update({thirdProperty:"Hello19"}, {"$inc": {firstProperty:2}})

4.Set Modifier($set):

- db.Tag.update({thirdProperty:"Hello19"}, {"$set": {fourthProperty: "newValue"} })

NOTE: Key-value will be created if the key doesnt exists yet in the case of both $set and $inc. $inc works for integers only. $set works for all. $inc Incrementing is extremely fast, so any performance penalty is negligible.

Query Using Modifiers (Contd.)

5. Unset Modifier($unset):

- db.Tag.update({thirdProperty:"Hello19"}, {"$unset": {fourthProperty: "anything"} })

6. Push Modifier($push):

- db.Blog.update({title:"1st Blog"}, { $push: {comment:"1st Comment"} });

NOTE: "$push" adds an element to the end of an array if the specified key already exists and creates a new array if it does not.

Query Using Modifiers (Contd.)

7. AddToSet Modifier($addToSet):

- db.Blog.update({title:"1st Blog"}, { $addToSet: {comment:"2nd Comment"} });

8. Pop Modifier($pop):

- comment = -1 remove an element from start.
- comment = 1 remove an element from end
- db.Blog.update({title:"1st Blog"}, {$pop:{comment:-1} });

Query Using Modifiers (Contd.)

9. Pull Modifier($pull):

- db.Blog.update({title:"1st Blog"}, {$pull:{comment:"2st Comment"}})

NOTE: Pulling removes all matching documents, not just a single match.

10.Position Operator($): The positional operator updates only the first match.

- db.embededDocument.update({"comments.author" : "John"}, {"$inc" : {"comments.0.votes" : 1}})
- db.embededDocument.update({"comments.votes" : 3}, {"$set" : {"comments.$.author" : "Amit Kumar"}})

OR Queries

There are two ways to do an OR query.

1."$in" can be used to query for a variety of values for a singlekey.

- db.Blog.find({comment: {$in:["5th Comment", "1st Comment"]} })

2. "$or" is more general; it can be used to query for anyof the given values across multiple keys.

- db.Tag.find({$or: [{firstProperty:0}, {secondProperty:/J/}] })NOTE: The opposite of "$in" is "$nin".

38

AND Queries

There are two ways to do an AND query.

1."$all" can be used to query for a variety of values for a singlekey.

- db.Blog.find({comment:{$all:["5th Comment", "1st Comment"]} })

NOTE: Below will do exact match and order too matters, if order will change then it will not match.

- db.Blog({"comment" : ["5th Comment", "comment1st Comment"]})

AND Queries (Contd.)

Simple queries are and queries.

It can be used to query for anyof the given values across single/multiple keys.db.Tag.find({firstProperty:0, secondProperty:/J/})

- db.Blog.find({comment:"5th Comment",comment:"1stComment"})

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.User.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.User.find({"name.first" : "Amit", "name.last" : "Kumar"})

Awesome: Atomic Modifiers

$inc

- db.pageviews.update({URL: 'http://myurl.com'}, {$inc: {N: 1}})

$set

- db.people.update({NAME: 'Steve'}, {$set: {Age: 35}})

$push (for atomically adding values to an array)

- db.people.update({NAME: 'Steve'}, {$push: {KIDS: {NAME: 'Sylvia', AGE: 3}}})

findAndModify()

```
• db.tasks.findAndModify(
          query: {STATUS: 'pending'},
•         sort: {PRIORITY: -1},
•         update: {$set: {STATUS: 'running', TS: new Date()}}
  )
```

42

36

Awesome: Atomic Modifiers (Contd.)

SELECT * FROM foo WHERE name=‘bob’ and (a=1 or b=2)

- db.foo.find({ name : "bob" , $or : [{ a : 1 } , { b : 2 }] })

SELECT * FROM users WHERE age>33 AND age<=40

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

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

The limit() & Skip method()

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

- db.mycol.find({},{"title":1,_id:0}).limit(2) {"title":"MongoDB Overview"} {"title":"NoSQL Overview"}

SKIP method()

Apart from limit() method there is one more method **skip()** which also accepts number type argument and used to skip number of documents.

- db.mycol.find({},{"title":1,_id:0}).limit(1).skip(1)

default value in **skip()** method is 0

45

Sort Method()

To sort documents in MongoDB, you need to use **sort()** method. **sort()** method accepts a document containing 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

- db.COLLECTION_NAME.find().sort({KEY:1})
- db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})

Cursors

The database returns results from find using a cursor.

If we do not store the results in a variable, the MongoDB shell will automatically iterate through and display the first couple of documents.

When you call find, the shell does not query the database immediately.

It waits until you actually start requesting results to send the query, which allows you to chain additional options onto a query before it is performed.

Cursors (Contd.)

Almost every method on a cursor object returns the cursor itself so that you can chain them in any order.

For instance, all of the following are equivalent

- > var cursor = db.Tag.find().sort({"x" : 1}).limit(1).skip(10);
- > var cursor = db.Tag.find().limit(1).sort({"x" : 1}).skip(10);
- > var cursor = db.Tag.find().skip(10).limit(1).sort({"x" : 1});

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.

Developing with MongoDB (Contd.)

```
> for(i=0; i<100;
   i++) {
... db.c.insert({x : i});
... }
> var cursor =
  db.collection.find();
– > while
  (cursor.hasNext())
  {
– ... obj =
  cursor.next();
– ... // do stuff
– ... }
```

```
> var cursor =
   db.people.find();
>
   cursor.forEach(func
   tion(x) {
... print(x.name);
... });
adam
matt
zak
```

Developing with MongoDB (Contd.)

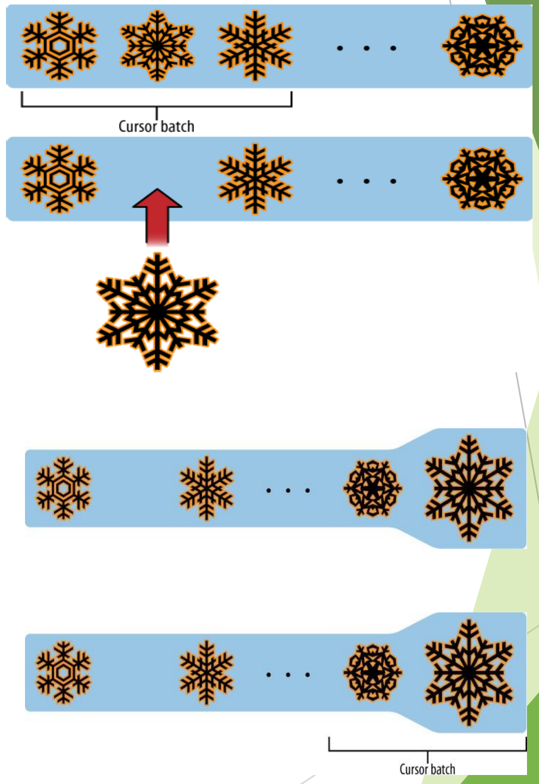
Behind Find() : Cursor (Contd.)

- Getting Consistent Results?
- var cursor = db.myCollection.find({country:'uk'}).snapshot();

A fairly common way of processing data is to pull it out of MongoDB, change it in some way, and then save it again:

Developing with MongoDB (Contd.)

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



52

Summary

- Understand about Create a document
- Insert, Update and delete a document
- Read a document using find()
- Types of Operators
- Comparison and logical Operators
- Sort, limit and skip operators
- Cursors