Lets us look at Create, Read, Update and Delete in MongoDB

Insert Documents

In MongoDB, the `db.collection.insertOne()` method adds new documents into a collection. In addition, both the `db.collection.update()` method and the `db.collection.save()` method can also add new documents through an operation called an **upsert**. An upsert is an operation that performs either an update of an existing documentS or an insert of a new document if the document to modify does not exist.

Insert a Document with insert() Method

The following statements insert documents with three fields into the collection inventory:

```
db.inventory.insertOne( { "_id": 10, "type": "electronic", "item": "ipad",
"qty": 15, "price":550 } )
db.inventory.insertOne( { "_id": 11, type: "electronic", item: "iphone",
qty: 13, "price": 400} )
db.inventory.insertOne( { "_id": 12,type: "consumables", "item": "print
cartridge", "qty": 5 } )
db.inventory.insertOne( { "_id": 13, type: "electronic", "item": "imac",
"qty": 10, "price": 400} )
db.inventory.insertOne( { "_id": 14, type: "book", "item": "yes to noSQL",
"qty": 10, "price": 40} )
db.inventory.insertOne( { "_id": 15, type: "book", "item": "Mongo Mongo",
"qty": 10, "price": 40} )
db.employee.insertOne( { "firstname": "Peter", "lastName": "Smith" } )

db.order.insertOne(
   {  "_id": 99999,
     "name": { "first": "John", "last": "McCarthy" },
     "orderDate": new Date("Sep 04, 2013"),
     "ShippingAddress":{ "line1":"Belgard Road", "line2":"Tallaght",
"city":"Dublin"},
     "lineItems": [
               { "product": "screws",
                 "qty": 10,
                 "uom": "kg",
                 "unitPrice": 10.00
               },
               { "product": "fuel",
                 "qty": 30,
                 "uom": "litres",
                 "unitPrice":3.00
               },
               { "product": "tiles",
                 "qty": 2,
                 "uom": "pallots",
                 "unitPrice": 259.00,
                 "colour": "cream"
               }
             ]})
```

EXERCISE: Try out the above inserts. You want these collections to be stored in a db called **mybusinessdocs** . Check that your documents were inserted okay. Rectify any errors you get if any! What overall observations did you make?

Sol: Fixed above

- **Cannot have duplicate _id**
- `"product": "screws",`
- `"ShippingAddress":{ "line1":"Belgard Road", "line2":"Tallaght",`

Flexible schema: price name value pair for some inventory documents for example.

Support for embedded documents as well as arrays of embedded documents

Insert a Document with update() Method

Call the update() method with the upsert flag to create a new document if no document matches the update's query criteria. The following example attempts to carry out a piece-wise update to an existing inventory document setting the quantity to 10 if it already exists. It creates a new document if no document in the inventory collection contains { type:"books", item : "journal" }: Read the following statements

```
1.
db.inventory.update(
{ "type": "book", item : "journal","price":10},
{ $set : { "qty": 10 } },
{ "upsert" : true })

2.
db.inventory.update(
{ "type": "book", "item" : "journal","price":10 },
{ $set : { "qty": 999 } },
{ "upsert" : true })
3.
db.inventory.update(
{"type": "book"},{$inc: {"qty" : 5} })
-- Finds first match and increments
4.
db.inventory.update(
{"type": "book"},{$inc: {"qty" : 5} }, {"multi": true } )
```

You must use dot notation to update values in subdocuments. For Example: T

```
db.order.update(
   { _id: 9999},
   { $set: {
          "name.first": "John James"
      }
   }
)
```

MongoDB adds the _id field and assigns as its value a unique ObjectId. The new document includes the item and type fields from the <query> criteria and the qty field from the <update> parameter.

EXERCISES:

1. Try out the above updates. Execute them singly and check the changes using an appropriate find command made to the documents in the collection

ANS

```
db.inventory.find({"type":"book"}).pretty()

db.inventory.update(
{ "type": "book", item : "journal","price":10},
{ $set : { "qty": 10 } },
{ "upsert" : true }
)
-- Note No match in the inventory collection so adds a new document inserted with the name value pairs used in the predicate as well as the name value par we wanted to update
i.e.
{
        "_id" : ObjectId("5a1d7f839a366a5854be7d8d"),
        "item" : "journal",
        "price" : 10,
        "type" : "book",
        "qty" : 10
}
UPsert – update documents(s) that meet the condition. Otherwise inert a new document


db.inventory.update(
{ "type": "book", "item" : "journal","price":10 },
{ $set : { "qty": 999 } },
{ "upsert" : true }
)
-- A match! updates the existing document i.e.
{
        "_id" : ObjectId("5a1d7f839a366a5854be7d8d"),
        "item" : "journal",
        "price" : 10,
        "type" : "book",
        "qty" : 999
}
```

```
db.inventory.update(
{"type": "book"},{$inc: {"qty" : 5} })
```
-- Finds the first match ONLY and increments qty. Other matching documents are not updated!

```
db.inventory.update(
{"type": "book"},{$inc: {"qty" : 5} }, {"multi": true } )
-- Updates all matching documents
```

2. The Govt have put a put a €20 tax on all electronic equipment. Write appropriate update.

ANS

```
db.inventory.update(
{type: "electronic"},{$inc: {"price" : 20} }, {"multi": true } )
```

3. Write an update to reflect that journal item price is now €20 and they have just received a delivery of 100.

ANS

```
db.inventory.update(
{"item": "journal"},
{
   $inc: {"qty" : 100} ,
   $set: {"price" : 20}
}
)
```

4. Write an update statement that changes line1 to Belgard Road Lower and adds a PostCode Field value 24

ANS

```
db.order.update(
   { "_id": 9999},
   { $set: {
          "ShippingAddress.line1": "Belgard Road Lower",
          "ShippingAddress.postCode": 24
      }   })
```

Replace All Fields

Given that the following inventory document exists in the inventory collection:

```
{
    "_id" : 22,
    "type" :"book",
    "item" : "The Snapper",
    "author" : "Roddy Doyle",
    "price" : 20,
    "qty" : 4
}
```

The update operation below passes an <update> document that contains only field and value pairs, which means the document **replaces all the fields in the original document**. The operation *does not* replace the_id value. The operation contains the same value for the item field in both the <query> and <update>documents, which means the field does not change:

```
db.inventory.update(
{item : "The Snapper"},
{item : "The Snapper", price : 20, qty : 4 }

)
```

Exercise

Try out the above scenario and provide mongo statements you used

ANS

```
db.inventory.insertOne( {
    "_id" : 22,
    "type" :"book",
    "item" : "The Snapper",
    "author" : "Roddy Doyle",
    "price" : 20,
    "qty" : 4
})

db.inventory.find({"item":"The Snapper"}).pretty()

db.inventory.update(
{"item" : "The Snapper"},
{"item" : "The Snapper", "price" : 20, "qty" : 4 }

)

db.inventory.find({"item":"The Snapper"}).pretty()
```

--Note as we did not use $set: the existing document is replaced by the fields provided in the update statement. i.e field type, author are olst!

Querying Documents in MongoDB

Select All Documents in a Collection

An empty query document ({}) selects all documents in the collection:

```
-- equivalent to SQL SELECT *

db.inventory.find( {} )
```

Conditions

To specify equality condition, use the query document { <field>: <value> } to select all documents that contain the <field> with the specified <value>. The following example retrieves from the inventory collection all documents where the type field has the value book. You have seen some of these already! Try them out.

-- equivalent to the SQL WHERE clause

```
db.inventory.find( { "type": "book" } ).pretty()
```

-- equivalent to the SQL OR clause

```
db.inventory.find( { "type": { $in: [ "book", "electronic" ] } }
).pretty()
```

```
db.inventory.find({ $or: [{"qty": { $gt: 100 } },{ "price": {
$lt: 450 } }] } ) .pretty()
```

-- equivalent to the SQL AND clause

```
db.inventory.find( { "type": "electronic", "qty": { $gt: 5 } } )
.pretty()
```

-- equivalent to the SQL OR and AND clause

```
db.inventory.find( { "type": "book", $or: [ { "qty": { $gt: 10 }
},{ "price": { $lt: 450 } } ]} ).pretty()
```

In the following example, the query uses the dot notation to match all documents where the value of the field **name** is a subdocument that contains a field first with the value John and may contain other fields:

```
db.order.find( { "name.first":"John" }).pretty()
```

Remove All Documents

If you do not specify a query, remove() removes all documents from a collection, but does not remove the indexes. The following example removes all documents from the inventory collection:

```
db.employee.remove({})
```

To remove all documents from a collection, it may be more efficient to use the drop() method to drop the entire collection, including the indexes, and then recreate the collection and rebuild the indexes.

`db.employee.drop()`

Remove Documents that Matches a Condition

To remove the documents that match a deletion criteria, call the remove() method with the <query> parameter.

The following example removes all documents from the inventory collection where the type field equals food:

db.inventory.remove({ "type" : "magazines" })

EXERCISES

1. Find the inventory documents that are either electronic or book and have a price less than 200.

2. Find the inventory documents that are type book, with an item name journal and have a price greater than 5

3. Find all the document that have price in the range of 101 to 699 (hint use $and)

4. Find an order that has a firstname of John and last name of Smith whose shipping address is in Dublin

5. Delete inventory documents that are type book or electronic

ANS

1

```
db.inventory.find( { "type": { $in: [ "book", "electronic" ] },
"price":{$lt: 200 } })
```

OR

```
db.inventory.find( { $or:[ {"type":"book"},
{"type":"electronic"}], "price":{$lt: 200 } })
```

2

```
db.inventory.find( { "type": "book" , "item": "journal", "price":
{ $gt: 5 } })
```

3

```
db.inventory.find({ $and: [{"price": { $gt: 100 } },{ "price": {
$lt: 700 } }] } )
```

OR

```
db.inventory.find({ "price": { $gt: 100,$lt: 700 }} )
```

4.

```
db.order.find( { "name.first":"John" , "name.last":"McCarthy",
"ShippingAddress.city":"Dublin"}).pretty()
```

5.

```
db.inventory.remove( { "type": { $in: [ "book", "electronic" ] }
})
```

OR

```
db.inventory.remove( { $or:[ {"type":"book"},
{"type":"electronic"}]})
```