Practice Quiz #4 (30 pts)

Due Dec 5 at 11:59pm	**Points** 30	**Questions** 12	**Available** until Dec 5 at 11:59pm	**Time Limit** None	**Allowed Attempts** Unlimited

Instructions

With each MongoDB question in this Quiz. Run the following commands in the "test" databases to drop and create (recreate) the quiz4 collection as follows. That is, the starting state of the quiz4 collection must be reset with each question.

Please note:

1. You will have a MAX of three attempts on this Quiz.

2. You may simply execute the code to arrive at the answers, but do make sure you understand the questions and will be able to provide the correct output without being able to run code on the Final Exam.

======
DATA:
======

```
db.quiz4.drop() ;
db.quiz4.insert(
[

{
   "id" : 1,
   "ver" : 1,
   "a" : { "a1" : "A", "a2" : "AB" },
   "b" : [ "b1", "b2", "b3", "b4"]
}
,
{
   "id" : 2,
   "ver" : 1,
   "a" : { "a1" : "CA", "a2" : "AD" },
   "b" : [ "b1", "b2"]
}
,
{
   "id" : 3,
   "ver" : 1,
   "a" : { "a1" : "FA", "a2" : "AG" },
   "b" : [ "b1", "b2", "b3" ]
}
,
{
   "id" : 4,
   "ver" : 1,
   "a" : { "a1" : "A", "a2" : "BA" },
   "b" : [ "b1" ]
}
,
{
   "id" : 5,
   "ver" : 1,
   "a" : { "a1" : "F", "a2" : "GA" },
   "b" : [ "b1", "b2", "b3", "b4"]
}

]
) ;
db.quiz4.ensureIndex( { "id" : 1 }, { unique: true } )
```

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	6 minutes	30 out of 30

Correct answers are hidden.

Score for this attempt: **30** out of 30
Submitted Dec 4 at 2:39pm
This attempt took 6 minutes.

Question 1

1 / 1 pts

What is the output of:

```
db.quiz4.insert(
{
    "id" : 1,
    "c" : [ "c1", "c2", "c3", "c4"]
})
```

insertDocument :: caused by :: 11000 E11000 duplicate key error index: test.quiz4.$id_1 dup key: { : 1.0 }

Question 2

1 / 1 pts

What is the output of:

```
db.quiz4.update(
   { b: { $size: 4 } },
   { $set: { "a": ["a", "b", "c"] } },
   { multi: true }
)
db.quiz4.find({ a: { $size: 3 }}).count()
```

2

Question 3

1 / 1 pts

What is the output of:

```
b.quiz4.update(
   { b: { $size: 4 } },
   { $set: { "a": ["a", "b", "c"] } }
)
db.quiz4.find( { a: { $size: 3 } } ).count()
```

1

Question 4

1 / 1 pts

What is the output of:

```
db.quiz4.remove( {b: { $size: 4 }}, 1 )
db.quiz4.count() ;
```

4

Question 5 1 / 1 pts

What is the output of:

```
db.quiz4.remove( {b: { $size: 4 }} )
db.quiz4.count() ;
```

3

Question 6 2 / 2 pts

What is the output of:

```
db.quiz4.findAndModify({
    query: { id: 5 },
    update: { $inc: { ver: 1 } },
    upsert: true
})
db.quiz4.findAndModify({
    query: { id: 5 },
    update: { $inc: { ver: 1 } },
    upsert: true
})
db.quiz4.find( { id: 5, ver: 1 } ).count()
```

0

Question 7 3 / 3 pts

What is the output of:

```
function updateIfCurrent(_id, _ver, _doc) {
    var cur = db.quiz4.find( { id: _id } ) ;
    var obj = cur.next() ;
    if ( obj.ver == _ver ) {
        print ("Updating...") ;
        var nxt = _ver + 1 ;
        db.quiz4.findAndModify({
            query: { id: _id },
            update: { $set : { ver: nxt, a: _doc } }
        })
    }
    else {
        print ("Update error! Try again") ;
    }
}

updateIfCurrent(5, 1, ["a","b","c"] )

db.quiz4.find( { id: 5, ver: 1 } ).count()
```

Line 1: Updating...

Line 2: 0

Answer 1:

0

Question 8

3 / 3 pts

What is the output of the following NodeJS code:

```
var f1 = function () { console.log("1"); }
var f2 = function () { console.log("2"); }
var f3 = function () { console.log("3"); }

s1 = function() { setTimeout( f1, 5000); }
s2 = function() { setTimeout( f2, 1000); }
s3 = function() { setTimeout( f3, 3000); }

s2() ;
s1() ;
s3() ;
```

Line #1: 2

Line #2: 3

Line #3: 1

Answer 1:

2

Answer 2:

3

Answer 3:

1

Question 9

4 / 4 pts

What is the output of the following NodeJS code:

```
var async = require("async") ;

var f1 = function () { console.log ("one...") ; }
var f2 = function () { console.log ("two...") ; }
var f3 = function () { console.log ("three...") ; }

async.series({
    two: function(c){
        setTimeout(function(){
            f1() ;
            c(null,1);
        }, 9000);
    },
    one: function(c){
        setTimeout(function(){
            f2() ;
            c(null,2);
        }, 10000);
    },
    three: function(c){
        setTimeout(function(){
```

```
            f3() ;
            c(null,3);
        }, 3000);
    }
},
function(err, results) {
    console.log( results ) ;
}
);
```

Line 1: one...

Line 2: two...

Line 3: three...

Line 4: { two: 1, one: 2, thr

Answer 1:

one...

Answer 2:

two...

Answer 3:

three...

Answer 4:

{ two: 1, one: 2, three: 3 }

Question 10

5 / 5 pts

What is the output of the following NodeJS code:

```
var async = require("async") ;

async.waterfall([
    function(c){
        c( null, "a" ) ;
    },
    function(arg, c){
        console.log( arg ) ;
        c( null, "b", "c" ) ;
    },
    function(arg1, arg2, c){
        console.log( arg1 ) ;
        console.log( arg2 ) ;
        c(null, "done") ;
    }
],
function(err, results) {
    console.log( "err: " + err ) ;
    console.log( "results: " + results ) ;
}
);
```

Line 1 : a

Line 2: b

Line 3: c

Line 4: err: null

Line 5: results: done

Answer 1:

a

Answer 2:

b

Answer 3:

c

Answer 4:

err: null

Answer 5:

results: done

Question 11 3 / 3 pts

What is the output of the following NodeJS code:

```
var async = require("async") ;

async.waterfall([
    function(c){
        c( null, "a" ) ;
    },
    function(arg, c){
        console.log( arg ) ;
        c( "error", "b", "c" ) ;
    },
    function(arg1, arg2, c){
        console.log( arg1 ) ;
        console.log( arg2 ) ;
        c(null, "done") ;
    }
],
function(err, results) {
    console.log( "err: " + err ) ;
    console.log( "results: " + results ) ;
}
);
```

Line 1: a

Line 2: err: error

Line 3: results: b

Answer 1:

a

Answer 2:

err: error

Answer 3:

results: b

Question 12 5 / 5 pts

```
1.      if ( params?.event == "Turn Crank" )
2.      {
3.          def before = gumballMachine.getCount() ;
4.          gumballMachine.turnCrank();
5.          def after = gumballMachine.getCount() ;
6.
7.          if ( after != before )
8.          {
9.              def gumball = Gumball.findBySerialNumber( machineSerialNum )
10.             Thread.currentThread().sleep(4000)
11.             if ( gumball )
12.             {
13.                 // update gumball inventory
14.                 println( "Updating DB with new Inventory of: $after")
15.                 gumball.countGumballs = after ;
16.                 gumball.save(flush: true); // default optimistic lock
17.             }
18.         }
19.     }
```

In the above Grails Gumball Controller Code which uses the "default" optimistic lock model, which of the following is TRUE about the possible outcome at line #16?

- The update will fail in all cases due to multiple concurrent updates
- **The update will fail in cases where the version stamp managed by GORM was changed**
- The update succeeds in all cases due to the use of optimistic locking model
- The update will fail only in cases if count was changed be some other transaction

Quiz Score: **30** out of 30