Advanced Databases

MongoDB CA

Student Number: C19513256

Student Name: Barnet Obeka

Programme Code: TU856

1. Setting up the cluster and replication

The cluster is composed of 3 interconnected nodes and all are part of the replication set

```
yoshibart@Barnets-MacBook-Pro ~ % docker network create C19513256Cluster
eecec69091ca739185e48494f88b6c8e9349eff40e64c7e892dcb6ec3a4b13d6
yoshibart@Barnets-MacBook-Pro ~ %
```

```
yoshibart@Barnets-MacBook-Pro ~ % docker run -d --rm -p 27017:27017 --name C19513256-1 --network C19513256Cluster mongo:latest --replSet C195132
localhost,C19513256-1
58cb91e147082fe3f2f67c0625a652097a77d832bbf6c746031230a180b811d9
yoshibart@Barnets-MacBook-Pro ~ %
```

```
yoshibart@Barnets-MacBook-Pro ~ % docker run -d --rm -p 27018:27017 --name C19513256-2 --network C19513256Cluster mongo:latest --replSet C195132
localhost,C19513256-2
6a6e3e5ba23d0dbebea31e03ca2d619858f8d49c943088f0d4659c415d273206
```

```
yoshibart@Barnets-MacBook-Pro ~ % docker run -d --rm -p 27019:27017 --name C19513256-3 --network C19513256Cluster mongo:latest --replSet C195132
localhost,C19513256-3
f22cbf47906dd500ba20652ec781c6bb0ba70bcf4571fb0d038edf74c7b2a1d2
```

The message ok: 1 shows that the initiation and verification of the replication set were successful

```
yoshibart@Barnets-MacBook-Pro ~ % docker exec -it C19513256-1 mongosh --eval "rs.initiate({
 _id: \"C19513256RepSet\",
 members: [
   {_id: 0, host: \"C19513256-1\"},
   {_id: 1, host: \"C19513256-2\"},
   {_id: 2, host: \"C19513256-3\"}
 ]
})"
Current Mongosh Log ID: 639ce0a3371a151ecb664f82
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.6.1
Using MongoDB:          6.0.3
Using Mongosh:          1.6.1

For mongosh info see: https://docs.mongodb.com/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting
   2022-12-16T21:12:23.937+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb
-filesystem
   2022-12-16T21:12:24.619+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricte
   2022-12-16T21:12:24.620+00:00: vm.max_map_count is too low
------

------
   Enable MongoDB's free cloud-based monitoring service, which will then receive and display
   metrics about your deployment (disk utilization, CPU, operation statistics, etc).

   The monitoring data will be available on a MongoDB website with a unique URL accessible to you
   and anyone you share the URL with. MongoDB may use this information to make product
   improvements and to suggest MongoDB products and deployment options to you.

   To enable free monitoring, run the following command: db.enableFreeMonitoring()
   To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
------

{ ok: 1 }
yoshibart@Barnets-MacBook-Pro ~ %
```

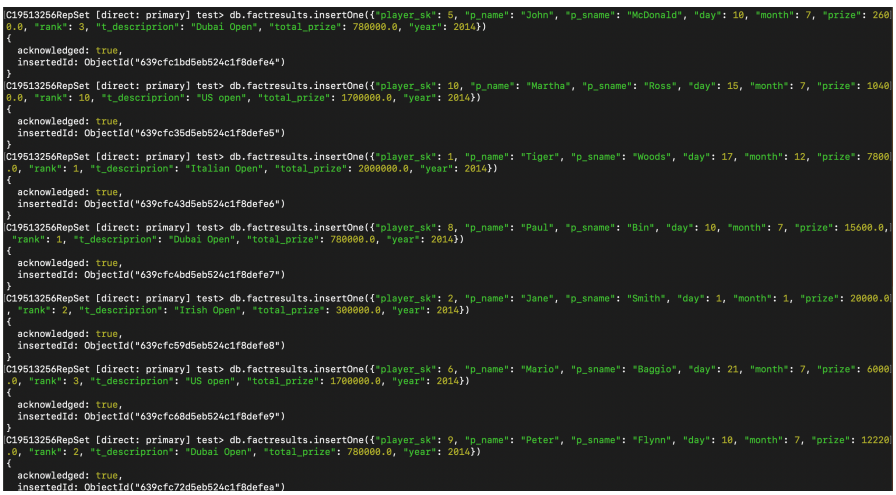
The replication set is up and running with 3 members. The number of votes required to win a vote e.g. a vote to become the primary node is 2 and the majority vote for writing is 2. In this case node, C19513256-1 won and it's the primary node

```
{
  set: 'C19513256RepSet',
  date: ISODate("2022-12-16T21:46:38.957Z"),
  myState: 1,
  term: Long("1"),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long("2000"),
  majorityVoteCount: 2,
  writeMajorityCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1671227195, i: 1 }), t: Long("1") },
    lastCommittedWallTime: ISODate("2022-12-16T21:46:35.877Z"),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1671227195, i: 1 }), t: Long("1") },
    appliedOpTime: { ts: Timestamp({ t: 1671227195, i: 1 }), t: Long("1") },
    durableOpTime: { ts: Timestamp({ t: 1671227195, i: 1 }), t: Long("1") },
    lastAppliedWallTime: ISODate("2022-12-16T21:46:35.877Z"),
    lastDurableWallTime: ISODate("2022-12-16T21:46:35.877Z")
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1671227155, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'electionTimeout',
    lastElectionDate: ISODate("2022-12-16T21:18:39.104Z"),
    electionTerm: Long("1"),
    lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1671225507, i: 1 }), t: Long("-1") },
    lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1671225507, i: 1 }), t: Long("-1") },
    numVotesNeeded: 2,
    priorityAtElection: 1,
    electionTimeoutMillis: Long("10000"),
    numCatchUpOps: Long("0"),
    newTermStartDate: ISODate("2022-12-16T21:18:39.146Z"),
    wMajorityWriteAvailabilityDate: ISODate("2022-12-16T21:18:40.632Z")
  },
```

```
  members: [
    {
      _id: 0,
      name: 'C19513256-1:27017',
      health: 1,
      state: 1,
      stateStr: 'PRIMARY',
      uptime: 2055,
      optime: { ts: Timestamp({ t: 1671227195, i: 1 }), t: Long("1") },
      optimeDate: ISODate("2022-12-16T21:46:35.000Z"),
      lastAppliedWallTime: ISODate("2022-12-16T21:46:35.877Z"),
      lastDurableWallTime: ISODate("2022-12-16T21:46:35.877Z"),
      syncSourceHost: '',
      syncSourceId: -1,
      infoMessage: '',
      electionTime: Timestamp({ t: 1671225519, i: 1 }),
      electionDate: ISODate("2022-12-16T21:18:39.000Z"),
      configVersion: 1,
      configTerm: 1,
      self: true,
      lastHeartbeatMessage: ''
    },
```

2. Porting the data to Mongo

The data inserted in the database using the insert statements generated during the connection between Cassandra and Mongo



The mongo database includes 9 documents imported from Cassandra

```
C19513256RepSet [direct: primary] test> db.factresults.find().pretty()
[
  {
    _id: ObjectId("639cfc1bd5eb524c1f8defe4"),
    player_sk: 5,
    p_name: 'John',
    p_sname: 'McDonald',
    day: 10,
    month: 7,
    prize: 2600,
    rank: 3,
    t_descriprion: 'Dubai Open',
    total_prize: 780000,
    year: 2014
  },
  {
    _id: ObjectId("639cfc35d5eb524c1f8defe5"),
    player_sk: 10,
    p_name: 'Martha',
    p_sname: 'Ross',
    day: 15,
    month: 7,
    prize: 10400,
    rank: 10,
    t_descriprion: 'US open',
    total_prize: 1700000,
    year: 2014
  },
  {
    _id: ObjectId("639cfc43d5eb524c1f8defe6"),
    player_sk: 1,
    p_name: 'Tiger',
    p_sname: 'Woods',
    day: 17,
    month: 12,
    prize: 7800,
    rank: 1,
    t_descriprion: 'Italian Open',
    total_prize: 2000000,
    year: 2014
  },
```

3. Working with the Golf collection in MongoDB:
    a. Basic query on golf data involving a text field.

The following query is searching for a document with the surname “Woods”

```
C19513256RepSet [direct: primary] test> db.factresults.find({p_sname:"Woods"})
[
  {
    _id: ObjectId("639cfc43d5eb524c1f8defe6"),
    player_sk: 1,
    p_name: 'Tiger',
    p_sname: 'Woods',
    day: 17,
    month: 12,
    prize: 7800,
    rank: 1,
    t_descriprion: 'Italian Open',
    total_prize: 2000000,
    year: 2014
  }
]
```

The winning plan is using the collection scan property which is similar to a sequential scan in Cassandra where each and every document is searched and results are returned. In this case, 7 documents are examined before a filter “Woods ” is found. And there were 0 rejected plans

```
C19513256RepSet [direct: primary] test> db.factresults.find({p_sname:"Woods"}).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'test.factresults',
    indexFilterSet: false,
    parsedQuery: { p_sname: { '$eq': 'Woods' } },
    queryHash: '81AB4EB3',
    planCacheKey: '81AB4EB3',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { p_sname: { '$eq': 'Woods' } },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 0,
    totalKeysExamined: 0,
    totalDocsExamined: 7,
    executionStages: {
      stage: 'COLLSCAN',
      filter: { p_sname: { '$eq': 'Woods' } },
      nReturned: 1,
      executionTimeMillisEstimate: 0,
      works: 9,
      advanced: 1,
      needTime: 7,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      direction: 'forward',
      docsExamined: 7
    }
  },
```

b. Adding a secondary index to golf data on a text field.

The index is created on the “p_sname” field

```
C19513256RepSet [direct: primary] test> db.factresults.createIndex({p_sname:1})
p_sname_1
C19513256RepSet [direct: primary] test>
```

The index is verified using a similar query as in 3a. The index works because it uses “IXSCAN” for scanning documents. “FETCH” retrieves a single document and the single document is transferred to the Index scan stage where 1 key is explained with success. The estimated time of execution of FETCH and index scan is 1 millisecond

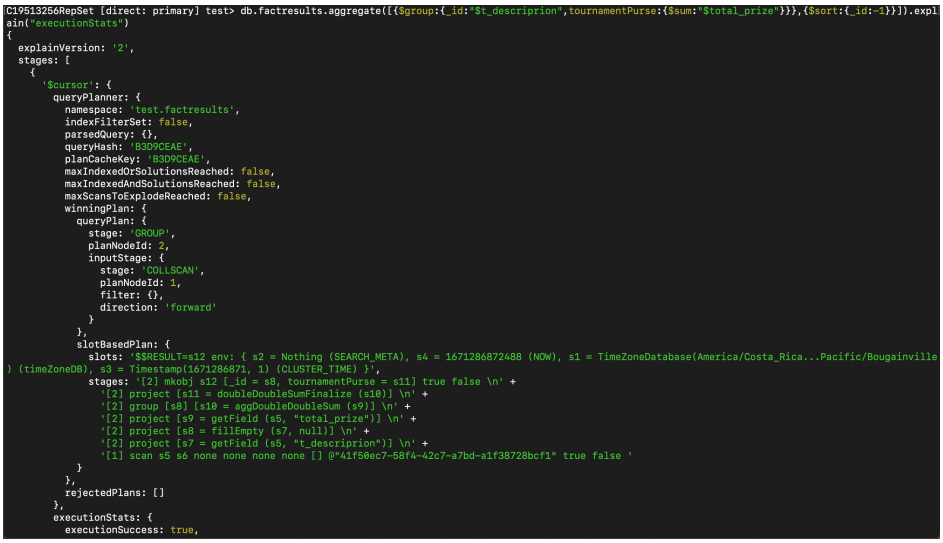
```
C19513256RepSet [direct: primary] test> db.factresults.find({p_sname:"Woods"}).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'test.factresults',
    indexFilterSet: false,
    parsedQuery: { p_sname: { '$eq': 'Woods' } },
    queryHash: '81AB4EB3',
    planCacheKey: '837A6102',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { p_sname: 1 },
        indexName: 'p_sname_1',
        isMultiKey: false,
        multiKeyPaths: { p_sname: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { p_sname: [ '["Woods", "Woods"]' ] }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 1,
    totalKeysExamined: 1,
    totalDocsExamined: 1,
    executionStages: {
      stage: 'FETCH',
      nReturned: 1,
      executionTimeMillisEstimate: 0,
      works: 2,
      advanced: 1,
      advanced: 1,
      needTime: 0,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      docsExamined: 1,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
        nReturned: 1,
        executionTimeMillisEstimate: 0,
        works: 2,
        advanced: 1,
        needTime: 0,
        needYield: 0,
        saveState: 0,
        restoreState: 0,
        isEOF: 1,
        keyPattern: { p_sname: 1 },
        indexName: 'p_sname_1',
        isMultiKey: false,
        multiKeyPaths: { p_sname: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { p_sname: [ '["Woods", "Woods"]' ] },
        keysExamined: 1,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
  command: { find: 'factresults', filter: { p_sname: 'Woods' }, '$db': 'test' },
  serverInfo: {
    host: 'cad377bf1ee5',
    port: 27017,
    version: '6.0.3',
    gitVersion: 'f803681c3ae19817d31958965850193de067c516'
  },
  serverParameters: {
```

4. Working with aggregation in MongoDB:
    a. Create an aggregation pipeline

The following aggregation includes 2 stages. Starts by grouping, and then the total amount of money paid out for each golf tournament is calculated. Next, the output from the group stage is sorted by _id in descending order

```
C19513256RepSet [direct: primary] test> db.factresults.aggregate([{$group:{_id:"$t_descriprion",tournamentPurse:{$sum:"$total_prize"}}},{$sort:{_id:-1}}])
[
  { _id: 'US open', tournamentPurse: 3400000 },
  { _id: 'Italian Open', tournamentPurse: 2000000 },
  { _id: 'Irish Open', tournamentPurse: 300000 },
  { _id: 'Dubai Open', tournamentPurse: 2340000 }
]
C19513256RepSet [direct: primary] test>
```

The aggregation execution winning plan starts in the GROUP stage and involves using the collection scan for all documents and filtering information using the tournament description. There are no rejected plans. There are 4 matches returned for different tournaments after the examination of 7 documents.



```
C19513256RepSet [direct: primary] test> db.factresults.aggregate([{$group:{_id:"$t_descriprion",tournamentPurse:{$sum:"$total_prize"}}},{$sort:{_id:-1}}]).explain("executionStats")
{
  explainVersion: '2',
  stages: [
    {
      '$cursor': {
        queryPlanner: {
          namespace: 'test.factresults',
          indexFilterSet: false,
          parsedQuery: {},
          queryHash: 'B3D9CEAE',
          planCacheKey: 'B3D9CEAE',
          maxIndexedOrSolutionsReached: false,
          maxIndexedAndSolutionsReached: false,
          maxScansToExplodeReached: false,
          winningPlan: {
            queryPlan: {
              stage: 'GROUP',
              planNodeId: 2,
              inputStage: {
                stage: 'COLLSCAN',
                planNodeId: 1,
                filter: {},
                direction: 'forward'
              }
            },
            slotBasedPlan: {
              slots: '$$RESULT=s12 env: { s2 = Nothing (SEARCH_META), s4 = 1671286872488 (NOW), s1 = TimeZoneDatabase(America/Costa_Rica...Pacific/Bougainville) (timeZoneDB), s3 = Timestamp(1671286871, 1) (CLUSTER_TIME) }',
              stages: '[2] mkobj s12 {_id = s8, tournamentPurse = s11] true false \n' +
                '[2] project [s11 = doubleDoubleSumFinalize (s10)] \n' +
                '[2] group [s8] [s10 = aggDoubleDoubleSum (s9)] \n' +
                '[2] project [s9 = getField (s5, "total_prize")] \n' +
                '[2] project [s8 = fillEmpty (s7, null)] \n' +
                '[2] project [s7 = getField (s5, "t_descriprion")] \n' +
                '[1] scan s5 s6 none none none none [] @"41f50ec7-58f4-42c7-a7bd-a1f38728bcf1" true false '
            }
          },
          rejectedPlans: []
        },
        executionStats: {
          executionSuccess: true,
          nReturned: 4,
          executionTimeMillis: 2,
          totalKeysExamined: 0,
          totalDocsExamined: 7,
          executionStages: {
            stage: 'mkobj',
            planNodeId: 2,
            nReturned: 4,
            executionTimeMillisEstimate: 0,
            opens: 1,
            closes: 1,
            saveState: 1,
            restoreState: 1,
            isEOF: 1,
            objSlot: 12,
            fields: [],
            projectFields: [ '_id', 'tournamentPurse' ],
            projectSlots: [ Long("8"), Long("11") ],
            forceNewObject: true,
            returnOldObject: false,
            inputStage: {
              stage: 'project',
              planNodeId: 2,
              nReturned: 4,
              executionTimeMillisEstimate: 0,
              opens: 1,
              closes: 1,
              saveState: 1,
              restoreState: 1,
              isEOF: 1,
              projections: { '11': 'doubleDoubleSumFinalize (s10) ' },
              inputStage: {
                stage: 'group',
                planNodeId: 2,
                nReturned: 4,
                executionTimeMillisEstimate: 0,
                opens: 1,
                closes: 1,
                saveState: 1,
                restoreState: 1,
                isEOF: 1,
                groupBySlots: [ Long("8") ],
                expressions: { '10': 'aggDoubleDoubleSum (s9) ' },
                usedDisk: false,
```

b. Add relevant indexes and reorder your stages.

The index is created on the tournament description field

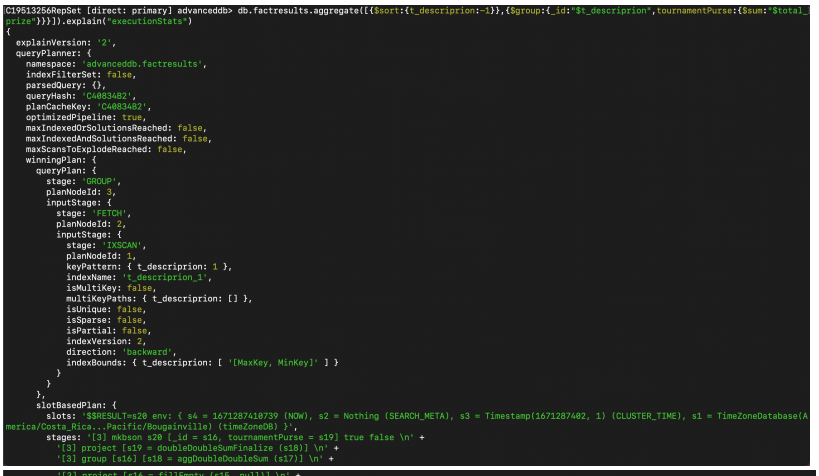
```
C19513256RepSet [direct: primary] advanceddb> db.factresults.createIndex({t_descriprion:1})
t_descriprion_1
C19513256RepSet [direct: primary] advanceddb>
```

The creation of an index is one form of stage optimization. The other in this case is switching the 2 stages in 4a. The first stage now is sorting the tournament description before piping the results to the group stage. This is more efficient because of no filtering to be done in the group stage.

```
C19513256RepSet [direct: primary] advanceddb> db.factresults.aggregate([{$sort:{t_descriprion:-1}},{$group:{_id:"$t_descriprion",tournamentPurse:{$sum:"$total_
prize"}}}])
[
  { _id: 'US open', tournamentPurse: 3400000 },
  { _id: 'Irish Open', tournamentPurse: 300000 },
  { _id: 'Italian Open', tournamentPurse: 2000000 },
  { _id: 'Dubai Open', tournamentPurse: 2340000 }
]
C19513256RepSet [direct: primary] advanceddb>
```

The winning execution plan involved using fetch to retrieve documents, and it's understandable because the aggregation involves grouping all documents and examining them and that's where the 7 documents examined come from. The retrieved documents are passed to the index scan stage. The index on t_descriprion is used by IXSCAN for scanning keys and in this 7 keys were scanned.

There were no rejected plans and a number 4 entries are returned. The execution time is 2 milliseconds



```
C19513256RepSet [direct: primary] advanceddb> db.factresults.aggregate([{$sort:{t_descriprion:-1}},{$group:{_id:"$t_descriprion",tournamentPurse:{$sum:"$total_
prize"}}}]).explain("executionStats")
{
  explainVersion: '2',
  queryPlanner: {
    namespace: 'advanceddb.factresults',
    indexFilterSet: false,
    parsedQuery: {},
    queryHash: 'CA0834B2',
    planCacheKey: 'CA0834B2',
    optimizedPipeline: true,
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      queryPlan: {
        stage: 'GROUP',
        planNodeId: 3,
        inputStage: {
          stage: 'FETCH',
          planNodeId: 2,
          inputStage: {
            stage: 'IXSCAN',
            planNodeId: 1,
            keyPattern: { t_descriprion: 1 },
            indexName: 't_descriprion_1',
            isMultiKey: false,
            multiKeyPaths: { t_descriprion: [] },
            isUnique: false,
            isSparse: false,
            isPartial: false,
            indexVersion: 2,
            direction: 'backward',
            indexBounds: { t_descriprion: [ '[MaxKey, MinKey]' ] }
          }
        }
      },
      slotBasedPlan: {
        slots: '$$RESULT=s20 env: { s4 = 1671287410739 (NOW), s2 = Nothing (SEARCH_META), s3 = Timestamp(1671287402, 1) (CLUSTER_TIME), s1 = TimeZoneDatabase(A
merica/Costa_Rica..Pacific/Bougainville) (timeZoneDB) }',
        stages: '[3] mkbson s20 [_id = s16, tournamentPurse = s19] true false \n' +
          '[3] project [s19 = doubleDoubleSumFinalize (s18)] \n' +
          '[3] group [s16] [s18 = aggDoubleDoubleSum (s17)] \n' +
          '[3] project [s16 = fillEmpty (s15, null)] \n' +
          '[3] project [s15 = getField (s13, "t_descriprion")] \n' +
          '[2] nlj [] [s9, s5, s6, s7, s8] \n' +
          '    left \n' +
          '        [1] nlj [s6, s8] [s10, s11] \n' +
          '            left \n' +
          '                [1] project [s6 = "t_descriprion_1", s8 = {"t_descriprion" : 1}, s10 = KS(F0FE04), s11 = KS(0A0104)] \n' +
          '                [1] limit 1 \n' +
          '                [1] coscan \n' +
          '            right \n' +
          '                [1] project [s5 = s12] \n' +
          '                [1] ixseek s10 s11 s7 s9 s12 [] @"7c32fc7a-5d76-415a-9cc7-534e68792ec9" @"t_descriprion_1" false \n' +
          '            \n' +
          '        \n' +
          '    right \n' +
          '        [2] limit 1 \n' +
          '        [2] seek s9 s13 s14 s5 s6 s7 s8 [] @"7c32fc7a-5d76-415a-9cc7-534e68792ec9" true false \n' +
          '    \n'
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 4,
    executionTimeMillis: 2,
    totalKeysExamined: 7,
    totalDocsExamined: 7,
    executionStages: {
      stage: 'mkbson',
      planNodeId: 3,
      nReturned: 4,
      executionTimeMillisEstimate: 0,
      opens: 1,
      closes: 1,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      objSlot: 20,
      fields: [],
      projectFields: [ '_id', 'tournamentPurse' ],
      projectSlots: [ Long("16"), Long("19") ],
      forceNewObject: true,
```

5. Replication working

In this case, when node C19513256-1 is paused it is reflected by the Replication set with the message “could not get a connection within a time limit “, leading to the conclusion that it's either unhealthy or unreachable. This prompt an election for the next primary node with the winner required to have a majority vote of 2 and in this case node C19513256-2 becomes the primary node