

## Advanced Databases

### MongoDB CA

**Student Number:** C19366191

**Student Name:** Lina Mir

**Programme Code:** TU856

#### 1. a. Setting up the cluster and replication

This cluster has three nodes connected to it.

The three nodes are c19366191-1, c19366191-2, c19366191-3

Rs.status() was used on the cluster

```
C:\Users\Windows>docker exec -it c19366191-1 mongosh --eval "rs.status()"
Current Mongosh Log ID: 63890bd8dd7ad1e29872b536
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.6.0
Using MongoDB:      6.0.3
Using Mongosh:      1.6.0
```

Here it shows that it is connecting to MongoDB and which version it is currently using. It also shows the version of Mongosh and the log ID.

```
members: [
  {
    _id: 0,
    name: 'c19366191-1:27017',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 6657,
    optime: { ts: Timestamp({ t: 1669935628, i: 1 }), t: Long("1") },
    optimeDate: ISODate("2022-12-01T23:00:28.000Z"),
    lastAppliedWallTime: ISODate("2022-12-01T23:00:28.958Z"),
    lastDurableWallTime: ISODate("2022-12-01T23:00:28.958Z"),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1669929287, i: 1 }),
    electionDate: ISODate("2022-12-01T21:14:47.000Z"),
    configVersion: 1,
    configTerm: 1,
    self: true,
    lastHeartbeatMessage: ''
  },
```

This shows the different node members in the cluster. This node is called c19366191-1. It is a primary node. This node c19366191-1 is pointing to port number 27017.

```
  {
    _id: 1,
    name: 'c19366191-2:27017',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 6361,
    optime: { ts: Timestamp({ t: 1669935628, i: 1 }), t: Long("1") },
    optimeDurable: { ts: Timestamp({ t: 1669935628, i: 1 }), t: Long("1") },
    optimeDate: ISODate("2022-12-01T23:00:28.000Z"),
    optimeDurableDate: ISODate("2022-12-01T23:00:28.000Z"),
    lastAppliedWallTime: ISODate("2022-12-01T23:00:28.958Z"),
    lastDurableWallTime: ISODate("2022-12-01T23:00:28.958Z"),
    lastHeartbeat: ISODate("2022-12-01T23:00:36.730Z"),
    lastHeartbeatRecv: ISODate("2022-12-01T23:00:36.873Z"),
    pingMs: Long("0"),
    lastHeartbeatMessage: '',
    syncSourceHost: 'c19366191-1:27017',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 1,
    configTerm: 1
  },
```

In this node with an ID of 1, the name of it is c19366191-2 and this is a secondary node. This secondary node is pointing to the primary node, which is 27017.

Below is the third node with an ID of 2 and is a secondary node called c19366191-3. This third node is also pointing to the primary node, 27107.

```
{
  _id: 2,
  name: 'c19366191-3:27017',
  health: 1,
  state: 2,
  stateStr: 'SECONDARY',
  uptime: 21285,
  optime: { ts: Timestamp({ t: 1669925841, i: 1 }), t: Long("7") },
  optimeDurable: { ts: Timestamp({ t: 1669925841, i: 1 }), t: Long("7") },
  optimeDate: ISODate("2022-12-01T20:17:21.000Z"),
  optimeDurableDate: ISODate("2022-12-01T20:17:21.000Z"),
  lastAppliedWallTime: ISODate("2022-12-01T20:17:21.060Z"),
  lastDurableWallTime: ISODate("2022-12-01T20:17:21.060Z"),
  lastHeartbeat: ISODate("2022-12-01T20:17:28.209Z"),
  lastHeartbeatRecv: ISODate("2022-12-01T20:17:27.582Z"),
  pingMs: Long("0"),
  lastHeartbeatMessage: '',
  syncSourceHost: 'c19366191-1:27017',
  syncSourceId: 0,
  infoMessage: '',
  configVersion: 1,
  configTerm: 7
}
```

### b. Create a replica set

A replica set was created

`docker exec -it c19366191-1 mongosh --eval`

```
"rs.initiate({_id: \"C19366191RepSet\", members: [{_id: 0,
host: \"c19366191-1\"},{_id: 1, host: \"c19366191-2\"},{_id: 2,
host: \"c-19366191-3\"}]})"
```

this creates a replica set called C19366191RepSet

### c. Create a database

A database called c19366191 was created:

```
C19366191RepSet [direct: secondary] c19366191>
switched to db c19366191
```

## 2. Porting the data to Mongo

The python file called C19366191.py converts the data from Cassandra to mongodb.

This is a snip of some of the data that was inserted into the c19366191 database.

It finds the factresult tables data and presents it nicely using pretty().

```

c19366191RepSet [direct: primary] c19366191> db.factresults.find().pretty()
[
  {
    _id: ObjectId("63892581e6bb1865ad9c3b85"),
    player_sk: 5,
    p_name: 'McDonald',
    t_name: 'US Master',
    prize: 2600,
    rank: 3,
    year: 2014
  },
  {
    _id: ObjectId("63892587e6bb1865ad9c3b86"),
    p_name: 'Ross',
    t_name: 'Chiinese Open',
    prize: 10400,
    rank: 10,
    year: 2014
  },
  {
    _id: ObjectId("6389258be6bb1865ad9c3b87"),
    player_sk: 1,
    p_name: 'Woods',
    t_name: 'Dubai Open',
    prize: 20800,
    rank: 2,
    year: 2014
  },
  {
    _id: ObjectId("6389258fe6bb1865ad9c3b88"),
    player_sk: 8,
    p_name: 'Bin',
    t_name: 'US Master',
    prize: 15600,
    rank: 1,
    year: 2014
  },
]

```

### 3. Working with the Golf collection in MongoDB:

#### a. Basic query on golf data involving a text field.

This is the query that was run and the output. It shows document information on tournaments with Irish open.

```

c19366191RepSet [direct: primary] c19366191> db.factresults.find({t_name: "Irish Open"})
[
  {
    _id: ObjectId("63892592e6bb1865ad9c3b89"),
    player_sk: 2,
    p_name: 'Smith',
    t_name: 'Irish Open',
    prize: 9000,
    rank: 2,
    year: 2014
  },
  {
    _id: ObjectId("63892595e6bb1865ad9c3b8a"),
    player_sk: 6,
    p_name: 'Baggio',
    t_name: 'Irish Open',
    prize: 6000,
    rank: 3,
    year: 2014
  }
]

```

This is some of the explain stats output for this query

```

    winningPlan: {
      stage: 'COLLSCAN',
      filter: { t_name: { '$eq': 'Irish Open' } },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 2,
  }
}

```

This query had to perform a collection scan and read all the documents because no optimisation has been applied.

This is showing that the mongodb was moving in a forward direction.

There are no rejected plans because there is no optimization applied.

The number of returned documents is 2.

```

    direction: 'forward',
    docsExamined: 7
  }
},
command: {
  find: 'factresults',
  filter: { t_name: 'Irish Open' },
  '$db': 'c19366191'
},
}

```

Here it just shows that 7 documents were examined and factresults were found. Irish Open was filtered in the name and the database this was done on was c19366191.

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

The index that was created was below

```

c19366191RepSet [direct: primary] c19366191> db.factresults.createIndex({p_name:1})
p_name_11RepSet [direct: primary] c19366191> db.factresults.createIndex({p_name:1})

```

This index was created on the player's name.

This is the query that was done after the index was made

```

c19366191RepSet [direct: primary] c19366191> db.factresults.find({p_name: "Ross"})
[
  {
    _id: ObjectId("63892587e6bb1865ad9c3b86"),
    player_sk: 10,
    p_name: 'Ross',
    t_name: 'Chiinese Open',
    prize: 10400,
    rank: 10,
    year: 2014
  }
]

```

The output from explain stats on this query is below

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'c19366191.factresults',
    indexFilterSet: false,
    parsedQuery: { p_name: { '$eq': 'Ross' } },
    queryHash: '9C5D431D',
    planCacheKey: 'CC6406ED',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { p_name: 1 },
        indexName: 'p_name_1',
        isMultiKey: false,
        multiKeyPaths: { p_name: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { p_name: [ ["Ross", "Ross"] ] }
      }
    },
    rejectedPlans: []
  },
  ...
}
```

docsExamined: 1,

This shows that an index scan was done, and the index is called p\_name\_1. The direction is forward, and the rejected plans are nothing again. The docs that were examined are just 1 because the index was on player names, the search was done on player names.

#### 4. Working with aggregation in MongoDB:

##### a. Create an aggregation pipeline

The aggregation that was created is below:

```
c19366191RepSet [direct: primary] c19366191> db.factresults.aggregate([{$match: {t_name: "Irish Open"}}, {$group: {_id: "$p_name", AvgPrize: {$avg: "$prize"}}}, {$sort: {AvgPrize: -1}}])
[ { _id: 'Smith', AvgPrize: 9000 }, { _id: 'Baggio', AvgPrize: 6000 } ]
c19366191RepSet [direct: primary] c19366191>
```

This gets the average prize for the Irish open tournaments.

```

explainVersion: '2',
stages: [
  {
    '$cursor': {
      queryPlanner: {
        namespace: 'c19366191.factresults',
        indexFilterSet: false,
        parsedQuery: { t_name: { '$eq': 'Irish Open' } },
        queryHash: 'A274ED8C',
        planCacheKey: 'A274ED8C',
        maxIndexedOrSolutionsReached: false,
        maxIndexedAndSolutionsReached: false,
        maxScansToExplodeReached: false,
        winningPlan: {
          queryPlan: {
            stage: 'GROUP',
            planNodeId: 2,
            inputStage: {
              stage: 'COLLSCAN',
              planNodeId: 1,
              filter: { t_name: { '$eq': 'Irish Open' } },
              direction: 'forward'
            }
          }
        },
        slotBasedPlan: {
          slots: '$$RESULT=s13 env: { s1 = TimeZoneDatabase(Asia/Bangkok...Ameri
          stages: '[2] mkobj s13 [_id = s8, AvgPrize = s12] true false \n' +
            '[2] project [s12 = if (s11 == 0, null, doubleDoubleSumFinalize (s10
            '[2] group [s8] [s10 = aggDoubleDoubleSum (s9), s11 = sum (let [11.0
            '[2] project [s9 = getField (s5, "prize")] \n' +
            '[2] project [s8 = fillEmpty (s7, null)] \n' +
            '[2] project [s7 = getField (s5, "p_name")] \n' +
            '[1] filter {applyClassicMatcher (ClassicMatcher({ t_name: { $eq: "I
            '[1] scan s5 s6 none none none none [] @"36afd02c-e5eb-466e-8a82-817
        }
      },
      rejectedPlans: []
    },
    executionStats: {
      executionSuccess: true,
      nReturned: 2,
      executionTimeMillis: 0,
    }
  }
]

```

This is the output for the explain. The flow of this aggregate was collection scan, a few projections, and a filter. It returned 2 documents, and the direction is forward.

## b. Add relevant indexes and reorder your stages.

These indexes were created to try and improve the performance.

```

c19366191RepSet [direct: primary] c19366191> db.factresults.createIndex({prize:1})
prize_1
c19366191RepSet [direct: primary] c19366191> db.factresults.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { p_name: 1 }, name: 'p_name_1' },
  { v: 2, key: { prize: 1 }, name: 'prize_1' }
]

```

The below screenshot is the output from the aggregate again.

### c. Optimize the stage execution

```
{
  explainVersion: '2',
  stages: [
    {
      '$cursor': {
        queryPlanner: {
          namespace: 'c19366191.factresults',
          indexFilterSet: false,
          parsedQuery: { t_name: { '$eq': 'Irish Open' } },
          queryHash: 'A274ED8C',
          planCacheKey: 'A274ED8C',
          maxIndexedOrSolutionsReached: false,
          maxIndexedAndSolutionsReached: false,
          maxScansToExplodeReached: false,
          winningPlan: {
            queryPlan: {
              stage: 'GROUP',
              planNodeId: 2,
              inputStage: {
                stage: 'COLLSCAN',
                planNodeId: 1,
                filter: { t_name: { '$eq': 'Irish Open' } },
                direction: 'forward'
              }
            },
            slotBasedPlan: {
              slots: '$$RESULT$s13 env: { s1 = TimeZoneDatabase(Asia/Bangkok...America/Vancou
              stages: '[2] mkobj s13 [_id = s8, AvgPrize = s12] true false \n' +
                '[2] project [s12 = if (s11 == 0, null, doubleDoubleSumFinalize (s10) / s11)]'
                '[2] group [s8] [s10 = aggDoubleDoubleSum (s0), s11 = sum (let [l1.0 = s9] if
                '[2] project [s9 = getField (s5, "prize")] \n' +
                '[2] project [s8 = fillEmpty (s7, null)] \n' +
                '[2] project [s7 = getField (s5, "p_name")] \n' +
                '[1] filter {applyClassicMatcher (ClassicMatcher({ t_name: { $eq: "Irish Open
                '[1] scan s5 s6 none none none none [] @"36afd02c-e5eb-466e-8a82-817cb091c357
            }
          },
          rejectedPlans: []
        },
        executionStats: {
          executionSuccess: true,
          nReturned: 2,
          executionTimeMillis: 0,
          totalKeysExamined: 0,
          totalDocsExamined: 7,
          executionStages: {
            stage: 'mkobj',
            planNodeId: 2,

```

This performed a collection scan, and the stages are detailed here. It performed a few projections and a filter, and it returned 2 rows. The direction on this one was forward. The number of docs that were examined were 7.

## 5. Replication working

The primary node was stopped. This was the c19366191-1 node.

```
{
  _id: 0,
  name: 'c19366191-1:27017',
  health: 1,
  state: 2,
  stateStr: 'SECONDARY',
  uptime: 6856,
  optime: { ts: Timestamp({ t: 16699358
  optimeDate: ISODate("2022-12-01T23:03
  lastAppliedWallTime: ISODate("2022-12
  lastDurableWallTime: ISODate("2022-12
  syncSourceHost: 'c19366191-2:27017',
  syncSourceId: 1,
  infoMessage: '',
  configVersion: 1,
  configTerm: 2,
  self: true,
  lastHeartbeatMessage: ''
},
{
  _id: 1,
  name: 'c19366191-2:27017',
  health: 1,
  state: 1,
  stateStr: 'PRIMARY',
  uptime: 6559,
  optime: { ts: Timestamp({ t: 16699358
  optimeDurable: { ts: Timestamp({ t: 1
  optimeDate: ISODate("2022-12-01T23:03
  optimeDurableDate: ISODate("2022-12-0
  lastAppliedWallTime: ISODate("2022-12
  lastDurableWallTime: ISODate("2022-12
  lastHeartbeat: ISODate("2022-12-01T23
  lastHeartbeatRecv: ISODate("2022-12-0
  pingMs: Long("0"),
  lastHeartbeatMessage: '',
  syncSourceHost: '',
  syncSourceId: -1,
  infoMessage: '',
  electionTime: Timestamp({ t: 16699358
  electionDate: ISODate("2022-12-01T23:
  configVersion: 1,
  configTerm: 2
},
{
  _id: 2,
  name: 'c19366191-3:27017',
  health: 1,
  state: 2,
  stateStr: 'SECONDARY',
  uptime: 6559
}
```

From this it can be seen that once node 1 stopped, the second node, c19366191-2 became a primary node. The first node turned to a secondary node.