P

ractical - 06

Aim: Implementing Indexing in MongoDB

a) Create an index on a specific field in a MongoDB collection.
b) Measure the impact of indexing on query performance.

Description:

Indexing in MongoDB:
MongoDB uses indexing in order to make the query processing more efficient. If there is no indexing, then the MongoDB must scan every document in the collection and retrieve only those documents that match the query. Indexes are special data structures that store some information related to the documents such that it becomes easy for MongoDB to find the right data file. The indexes are ordered by the value of the field specified in the index.

Execution:

1. Create a MongoDB database named ‘company’

2. Create an employee collection with following attributes: {name, salary,department, position}.

```
company> db.createCollection("employee")
{ ok: 1 }
company> show collections
employee
company> db.employee.insertMany([
...     {
...       name: "Deepak",
...       salary: 45000,
...       department: "Backend Developer",
...       position: "Tech Lead"
...     },
...     {
...       name: "Debashish",
...       salary: 52000,
...       department: "UI/UX Developer",
...       position: "Manager"
...     },
...     {
...       name: "Vinit",
...       salary: 68000,
...       department: "Data Analyst",
...       position: "Manager"
...     },
...     {
...       name: "Pooja",
...       salary: 48000,
...       department: "UI/UX Developer",
...       position: "Developer"
...     },
...     {
...       name: "Rohit",
...       salary: 50000,
...       department: "Data Analyst",
...       position: "Developer"
...     },
...     {
...       name: "Narayan",
...       salary: 70000,
...       department: "Data Analyst",
...       position: "Developer"
...     },
...     {
...       name: "Bijisha",
...       salary: 42000,
...       department: "Web Developer",
...       position: "Developer"
...     },
...     {
...       name: "Sunil",
```

3. Create an index on the salary field

```
company> db.employee.createIndex({salary:1});
salary_1
```

4. Search for employees with a salary greater than 50,000 without using indexes.

```
company> db.employee.find({salary:{$gt:50000}});
[
  {
    _id: ObjectId('6598fa7ee13e860e94713264'),
    name: 'Debashish',
    salary: 52000,
    department: 'UI/UX Developer',
    position: 'Manager'
  },
  {
    _id: ObjectId('6598fa7ee13e860e9471326b'),
    name: 'Ganesh',
    salary: 65000,
    department: 'Web Developer',
    position: 'Developer'
  },
  {
    _id: ObjectId('6598fa7ee13e860e94713265'),
    name: 'Vinit',
    salary: 68000,
    department: 'Data Analyst',
    position: 'Manager'
  },
  {
    _id: ObjectId('6598fa7ee13e860e94713268'),
    name: 'Narayan',
    salary: 70000,
    department: 'Data Analyst',
    position: 'Developer'
  },
  {
    _id: ObjectId('6598fa7ee13e860e9471326a'),
    name: 'Sunil',
    salary: 70000,
    department: 'Data Analyst',
    position: 'Developer'
  }
]
```

5. Search for employees with a salary greater than 50,000 using indexes (use of index using the hint() method. After executing these queries, examine the output's executionStats):

```
company> db.employee.find({salary:{$gt:50000}}).hint({salary:1}).explain("executionStats");
{
  explainVersion: '2',
  queryPlanner: {
    namespace: 'company.employee',
    indexFilterSet: false,
    parsedQuery: { salary: { '$gt': 50000 } },
    queryHash: '29984A79',
    planCacheKey: '9058732A',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      queryPlan: {
        stage: 'FETCH',
        planNodeId: 2,
        inputStage: {
          stage: 'IXSCAN',
          planNodeId: 1,
          keyPattern: { salary: 1 },
          indexName: 'salary_1',
          isMultiKey: false,
          multiKeyPaths: { salary: [] },
          isUnique: false,
          isSparse: false,
          isPartial: false,
          indexVersion: 2,
          direction: 'forward',
          indexBounds: { salary: [ '(50000, inf.0]' ] }
        }
      },
      slotBasedPlan: {
        slots: '$$RESULT=s11 env: { s3 = 1704524936669 (NOW), s2 = Nothing (SEARCH_META), s5 = KS(2D0186A0FE04), s10 = {"salary" :
33FFFFFFFFFFFFFFFFFE04) }',
        stages: '[2] nlj inner [] [s4, s7, s8, s9, s10] \n' +
          '    left \n' +
          '        [1] cfilter {(exists(s5) && exists(s6))} \n' +
          '        [1] ixseek s5 s6 s9 s4 s7 s8 [] @"d8ee9843-28df-43ba-853c-0f0426707559" @"salary_1" true \n' +
          '    right \n' +
          '        [2] limit 1 \n' +
          '        [2] seek s4 s11 s12 s7 s8 s9 s10 [] @"d8ee9843-28df-43ba-853c-0f0426707559" true false \n'
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 5,
    executionTimeMillis: 0,
    totalKeysExamined: 5,
    totalDocsExamined: 5,
    executionStages: {
      stage: 'nlj',
      planNodeId: 2,
      nReturned: 5,
```

6. Query to Find Employees in a Specific Department (use of index using the hint() method. After executing these queries,examine the output's executionStats) :

```
company> db.employee.find({department: "Data Analyst"}).hint({salary:1}).explain("executionStats");
{
  explainVersion: '2',
  queryPlanner: {
    namespace: 'company.employee',
    indexFilterSet: false,
    parsedQuery: { department: { '$eq': 'Data Analyst' } },
    queryHash: '2DD6FB50',
    planCacheKey: '675C919B',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      queryPlan: {
        stage: 'FETCH',
        planNodeId: 2,
        filter: { department: { '$eq': 'Data Analyst' } },
        inputStage: {
          stage: 'IXSCAN',
          planNodeId: 1,
          keyPattern: { salary: 1 },
          indexName: 'salary_1',
          isMultiKey: false,
          multiKeyPaths: { salary: [] },
          isUnique: false,
          isSparse: false,
          isPartial: false,
          indexVersion: 2,
          direction: 'forward',
          indexBounds: { salary: [ '[MinKey, MaxKey]' ] }
        }
      },
      slotBasedPlan: {
        slots: '$$RESULT=s9 env: { s3 = 1704525172478 (NOW), s8 = {"salary" : 1}, s12 = "Data Analyst", s1 = TimeZoneDataba
        stages: '[2] filter {traverseF(s11, lambda(l1.0) { ((l1.0 == s12) ?: false) }, false)} \n' +
          '[2] nlj inner [] [s4, s5, s6, s7, s8] \n' +
          '    left \n' +
          '        [1] ixseek KS(0A0104) KS(F0FE04) s7 s4 s5 s6 lowPriority [] @"d8ee9843-28df-43ba-853c-0f0426707559" @"sa
          '    right \n' +
          '        [2] limit 1 \n' +
          '        [2] seek s4 s9 s10 s5 s6 s7 s8 [s11 = department] @"d8ee9843-28df-43ba-853c-0f0426707559" true false \n'
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 4,
    executionTimeMillis: 0,
    totalKeysExamined: 9,
    totalDocsExamined: 9,
    executionStages: {
      stage: 'filter',
      planNodeId: 2,
      nReturned: 4,
```

7. Query to Find Employees with a Salary Range(use of index using the hint() method. After executing these queries, examine the output's executionStats) :

```
}
company> db.employee.find({salary: {$gte: 50000, $lte: 67000}}).hint({salary:1});
[
  {
    _id: ObjectId('6598fa7ee13e860e94713267'),
    name: 'Rohit',
    salary: 50000,
    department: 'Data Analyst',
    position: 'Developer'
  },
  {
    _id: ObjectId('6598fa7ee13e860e94713264'),
    name: 'Debashish',
    salary: 52000,
    department: 'UI/UX Developer',
    position: 'Manager'
  },
  {
    _id: ObjectId('6598fa7ee13e860e9471326b'),
    name: 'Ganesh',
    salary: 65000,
    department: 'Web Developer',
    position: 'Developer'
  }
]
```

8. Query to Find Employees with a Specific Position(use of index using the hint() method. After executing these queries, examine the output's executionStats)

```
}
company> db.employee.find({position: "Developer"}).hint({salary:1});
[
  {
    _id: ObjectId('6598fa7ee13e860e94713269'),
    name: 'Bijisha',
    salary: 42000,
    department: 'Web Developer',
    position: 'Developer'
  },
  {
    _id: ObjectId('6598fa7ee13e860e94713266'),
    name: 'Pooja',
    salary: 48000,
    department: 'UI/UX Developer',
    position: 'Developer'
  },
  {
    _id: ObjectId('6598fa7ee13e860e94713267'),
    name: 'Rohit',
    salary: 50000,
    department: 'Data Analyst',
    position: 'Developer'
  },
  {
    _id: ObjectId('6598fa7ee13e860e9471326b'),
    name: 'Ganesh',
    salary: 65000,
    department: 'Web Developer',
    position: 'Developer'
  },
  {
    _id: ObjectId('6598fa7ee13e860e94713268'),
    name: 'Narayan',
    salary: 70000,
    department: 'Data Analyst',
    position: 'Developer'
  },
  {
    _id: ObjectId('6598fa7ee13e860e9471326a'),
    name: 'Sunil',
    salary: 70000,
    department: 'Data Analyst',
    position: 'Developer'
  }
]
```

9. Query to Find Employees with a Specific Position(use of index using the hint() method. After executing these queries, examine the output's executionStats)

```
]
company> db.employee.find({position: "Developer"}).hint({salary:1}).explain("executionStats");
{
  explainVersion: '2',
  queryPlanner: {
    namespace: 'company.employee',
    indexFilterSet: false,
    parsedQuery: { position: { '$eq': 'Developer' } },
    queryHash: '4AB7019E',
    planCacheKey: '05BF94DD',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      queryPlan: {
        stage: 'FETCH',
        planNodeId: 2,
        filter: { position: { '$eq': 'Developer' } },
        inputStage: {
          stage: 'IXSCAN',
          planNodeId: 1,
          keyPattern: { salary: 1 },
          indexName: 'salary_1',
          isMultiKey: false,
          multiKeyPaths: { salary: [] },
          isUnique: false,
          isSparse: false,
          isPartial: false,
          indexVersion: 2,
          direction: 'forward',
          indexBounds: { salary: [ '[MinKey, MaxKey]' ] }
        }
      },
      slotBasedPlan: {
        slots: '$$RESULT=s9 env: { s2 = Nothing (SEARCH_META), s3 = 1704744551486 (NOW), s1 = TimeZone
        stages: '[2] filter {traverseF(s11, lambda(l1.0) { ((l1.0 == s12) ?: false) }, false)} \n' +
          '[2] nlj inner [] [s4, s5, s6, s7, s8] \n' +
          '    left \n' +
          '        [1] lxseek KS(0A0104) KS(F0FE04) s7 s4 s5 s6 lowPriority [] @"d8ee9843-28df-43ba-85
          '    right \n' +
          '        [2] limit 1 \n' +
          '        [2] seek s4 s9 s10 s5 s6 s7 s8 [s11 = position] @"d8ee9843-28df-43ba-853c-0f0426707
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 6,
    executionTimeMillis: 0,
    totalKeysExamined: 9,
    totalDocsExamined: 9,
    executionStages: {
      stage: 'filter',
      planNodeId: 2,
      nReturned: 6,
```

10. Query to Find Employees with a Salary Above a Threshold in a SpecificDepartment(use of index using the hint() method. After executing these queries, examine the output's executionStats).

```
company> db.employee.find({salary:{$gt:50000},position: "Developer"}).hint({salary:1}).explain("executionSt
{
  explainVersion: '2',
  queryPlanner: {
    namespace: 'company.employee',
    indexFilterSet: false,
    parsedQuery: {
      '$and': [
        { position: { '$eq': 'Developer' } },
        { salary: { '$gt': 50000 } }
      ]
    },
    queryHash: 'A5F607E0',
    planCacheKey: '4EDD1B16',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      queryPlan: {
        stage: 'FETCH',
        planNodeId: 2,
        filter: { position: { '$eq': 'Developer' } },
        inputStage: {
          stage: 'IXSCAN',
          planNodeId: 1,
          keyPattern: { salary: 1 },
          indexName: 'salary_1',
          isMultiKey: false,
          multiKeyPaths: { salary: [] },
          isUnique: false,
          isSparse: false,
          isPartial: false,
          indexVersion: 2,
          direction: 'forward',
          indexBounds: { salary: [ '(50000, inf.0]' ] }
        }
      },
      slotBasedPlan: {
        slots: '$$RESULT=s11 env: { s3 = 1704744601659 (NOW), s2 = Nothing (SEARCH_META), s5 = KS(2D0186A0F
eveloper", s6 = KS(33FFFFFFFFFFFFFFFFE04) }',
        stages: '[2] filter {traverseF(s13, lambda(l1.0) { ((l1.0 == s14) ?: false) }, false)} \n' +
          '[2] nlj inner [] [s4, s7, s8, s9, s10] \n' +
          '    left \n' +
          '        [1] cfilter {(exists(s5) && exists(s6))} \n' +
          '        [1] ixseek s5 s6 s9 s4 s7 s8 [] @"d8ee9843-28df-43ba-853c-0f0426707559" @"salary_1" true
          '    right \n' +
          '        [2] limit 1 \n' +
          '        [2] seek s4 s11 s12 s7 s8 s9 s10 [s13 = position] @"d8ee9843-28df-43ba-853c-0f0426707559
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 3,
```