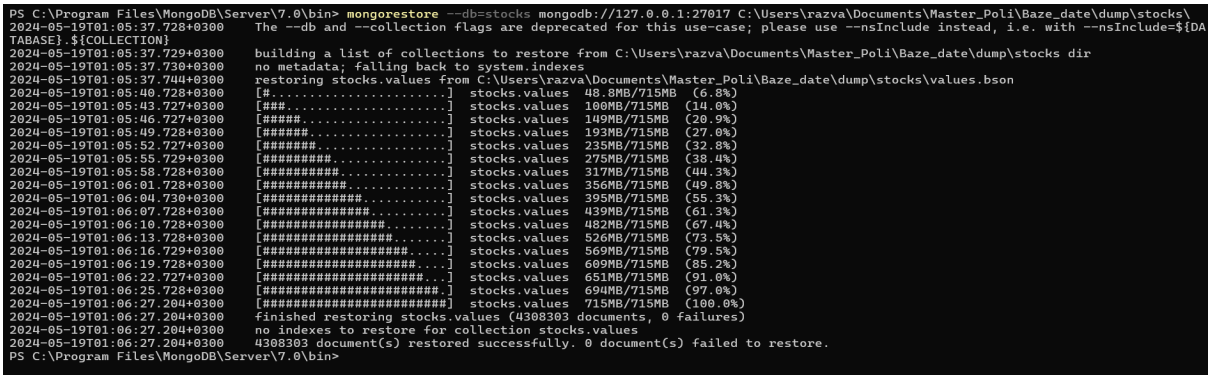
*** Query Optimization with Indexing ***

1. Make sure you have MongoDB Database Tools installed (https://docs.mongodb.com/database-tools/installation/installation/).

2. Make sure you have a local MongoDB instance running (see lab #2) or use the remote MongoDB Atlas instance (see lab #1).

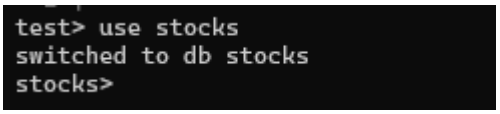
3. Download the sample database containing daily NASDAQ summaries
http://mng.bz/ii49

4. Use mongorestore tool to upload the sample NASDAQ database
Syntax: mongorestore <options> <connection-string> <directory or file to restore>
mongorestore --db=stocks
mongodb://mongoadmin:secret@localhost:27888/?authSource=admin dump/stocks



5. Connect to MongoDB using shell
mongosh mongodb://mongoadmin:secret@localhost:27888/?authSource=admin

6. Find the first occurence of Google's stock price:
> use stocks



> db.values.find({"stock_symbol": "GOOG"}).sort({date: -1}).limit(1)
Notice the slow query and possibly the warning issued by MongoDB.

```
stocks> db.values.find({"stock_symbol": "GOOG"}).sort({date: -1}).limit(1)
[
  {
    _id: ObjectId("4d094f7ec96767d7a02a0af6"),
    exchange: 'NASDAQ',
    stock_symbol: 'GOOG',
    date: '2008-03-07',
    open: 428.88,
    high: 440,
    low: 426.24,
    close: 433.35,
    volume: 8071800,
    'adj close': 433.35
  }
]
stocks>
```

7. Enable profiling
> db.setProfilingLevel(2)

```
stocks> db.setProfilingLevel(2)
{ was: 0, slowms: 100, sampleRate: 1, ok: 1 }
stocks>
```

8. See the execution statistics for the highest closing price in the data set:
> db.values.find({}).sort({close: -1}).limit(1).explain("executionStats")
Notice the "totalDocsExamined" field shows 4308303, which means that all 4 million documents have been scanned.

```
stocks> db.values.find({}).sort({close: -1}).limit(1).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'stocks.values',
    indexFilterSet: false,
    parsedQuery: {},
    queryHash: 'FA091E9B',
    planCacheKey: 'FA091E9B',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'SORT',
      sortPattern: { close: -1 },
      memLimit: 104857600,
      limitAmount: 1,
      type: 'simple',
      inputStage: { stage: 'COLLSCAN', direction: 'forward' }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 2986,
    totalKeysExamined: 0,
    totalDocsExamined: 4308303,
    executionStages: {
      stage: 'SORT',
      nReturned: 1,
      executionTimeMillisEstimate: 151,
      works: 4308306,
      advanced: 1,
      needTime: 4308304,
      needYield: 0,
      saveState: 4308,
      restoreState: 4308,
      isEOF: 1,
      sortPattern: { close: -1 },
      memLimit: 104857600,
      limitAmount: 1,
      type: 'simple',
      totalDataSizeSorted: 0,
      usedDisk: false,
      spills: 0,
      spilledDataStorageSize: 0,
      inputStage: {
        stage: 'COLLSCAN',
        nReturned: 4308303,
        executionTimeMillisEstimate: 84,
        works: 4308304,
        advanced: 4308303,
        needTime: 0,
        needYield: 0,
        saveState: 4308,
        restoreState: 4308,
        isEOF: 1,
        direction: 'forward',
        docsExamined: 4308303
      }
    }
  },
  command: {
    find: 'values',
    filter: {},
```

9. Create an index on the "close" field
> db.values.createIndex({close: 1})

```
stocks> db.values.createIndex({close: 1})
close_1
stocks>
```

10. Let's try the same query again
> db.values.find({}).sort({close: -1}).limit(1).explain("executionStats")
Notice the "totalDocsExamined" field shows 1, which means that just one document has been scanned(!!!) which makes a tremendous difference.

```
stocks> db.values.find({}).sort({close: -1}).limit(1).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'stocks.values',
    indexFilterSet: false,
    parsedQuery: {},
    queryHash: 'FA091E9B',
    planCacheKey: 'FA091E9B',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'LIMIT',
      limitAmount: 1,
      inputStage: {
        stage: 'FETCH',
        inputStage: {
          stage: 'IXSCAN',
          keyPattern: { close: 1 },
          indexName: 'close_1',
          isMultiKey: false,
          multiKeyPaths: { close: [] },
          isUnique: false,
          isSparse: false,
          isPartial: false,
          indexVersion: 2,
          direction: 'backward',
          indexBounds: { close: [ '[MaxKey, MinKey]' ] }
        }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 34,
    totalKeysExamined: 1,
    totalDocsExamined: 1,
    executionStages: {
      stage: 'LIMIT',
      nReturned: 1,
      executionTimeMillisEstimate: 31,
      works: 2,
      advanced: 1,
      needTime: 0,
      needYield: 0,
      saveState: 1,
      restoreState: 1,
      isEOF: 1,
      limitAmount: 1,
      inputStage: {
        stage: 'FETCH',
        nReturned: 1,
        executionTimeMillisEstimate: 31,
        works: 1,
        advanced: 1,
        needTime: 0,
        needYield: 0,
        saveState: 1,
        restoreState: 1,
        isEOF: 0,
        docsExamined: 1,
        alreadyHasObj: 0,
        inputStage: {
          stage: 'IXSCAN',
```

11. Fetch all of Google's closing values greater than 200:

> db.values.find({stock_symbol: "GOOG", close: {$gt: 200}})

```
}
stocks> db.values.find({stock_symbol: "GOOG", close: {$gt: 200}})
[
  {
    _id: ObjectId("4d094f7ec96767d7a02a0e15"),
    exchange: 'NASDAQ',
    stock_symbol: 'GOOG',
    date: '2005-01-03',
    open: 197.4,
    high: 203.64,
    low: 195.46,
    close: 202.71,
    volume: Long("15844200"),
    'adj close': 202.71
  },
  {
    _id: ObjectId("4d094f7ec96767d7a02a0e0b"),
    exchange: 'NASDAQ',
    stock_symbol: 'GOOG',
    date: '2005-01-18',
    open: 200.97,
    high: 205.02,
    low: 198.66,
    close: 203.9,
    volume: Long("13172600"),
    'adj close': 203.9
  },
  {
    _id: ObjectId("4d094f7ec96767d7a02a0dca"),
    exchange: 'NASDAQ',
    stock_symbol: 'GOOG',
    date: '2005-04-21',
    open: 200.42,
    high: 205,
    low: 199.32,
    close: 204.22,
    volume: Long("17751900"),
    'adj close': 204.22
  },
  {
    _id: ObjectId("4d094f7ec96767d7a02a0dfe"),
    exchange: 'NASDAQ',
    stock_symbol: 'GOOG',
    date: '2005-02-04',
    open: 206.47,
    high: 207.75,
    low: 202.6,
    close: 204.36,
    volume: Long("14819300"),
    'adj close': 204.36
  },
  {
    _id: ObjectId("4d094f7ec96767d7a02a0e00"),
    exchange: 'NASDAQ',
    stock_symbol: 'GOOG',
    date: '2005-02-02',
    open: 215.55,
    high: 216.8,
    low: 203.66,
    close: 205.96,
    volume: Long("32799300"),
    'adj close': 205.96
  },
  {
```

Run the query with the "explain" option, like so:

> db.values.find({stock_symbol: "GOOG", close: {$gt: 200}}).explain("executionStats")
Notice that the "nReturned" number of documents is 730 but the "totalDocsExamined" number of examined documents is 5299, even if we have an index on the "close" value.

```
stocks> db.values.find({stock_symbol: "GOOG", close: {$gt: 200}}).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'stocks.values',
    indexFilterSet: false,
    parsedQuery: {
      '$and': [
        { stock_symbol: { '$eq': 'GOOG' } },
        { close: { '$gt': 200 } }
      ]
    },
    queryHash: 'DF39A05A',
    planCacheKey: '9BC06527',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      filter: { stock_symbol: { '$eq': 'GOOG' } },
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { close: 1 },
        indexName: 'close_1',
        isMultiKey: false,
        multiKeyPaths: { close: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { close: [ '(200, inf.0]' ] }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 730,
    executionTimeMillis: 7,
    totalKeysExamined: 5299,
    totalDocsExamined: 5299,
    executionStages: {
      stage: 'FETCH',
      filter: { stock_symbol: { '$eq': 'GOOG' } },
      nReturned: 730,
      executionTimeMillisEstimate: 0,
      works: 5300,
      advanced: 730,
      needTime: 4569,
      needYield: 0,
      saveState: 5,
      restoreState: 5,
      isEOF: 1,
      docsExamined: 5299,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
        nReturned: 5299,
        executionTimeMillisEstimate: 0,
        works: 5300,
        advanced: 5299,
        needTime: 0,
        needYield: 0,
        saveState: 5,
        restoreState: 5,
        isEOF: 1,
        keyPattern: { close: 1 },
```

```
        keyPattern: { close: 1 },
        indexName: 'close_1',
        isMultiKey: false,
        multiKeyPaths: { close: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { close: [ '(200, inf.0]' ] },
        keysExamined: 5299,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
  command: {
    find: 'values',
    filter: { stock_symbol: 'GOOG', close: { '$gt': 200 } },
    '$db': 'stocks'
  },
  serverInfo: {
    host: 'DESKTOP-BCGD7BA',
    port: 27017,
    version: '7.0.9',
    gitVersion: '3ff3a3925c36ed277cf5eafca5495f2e3728dd67'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'trySbeRestricted'
  },
  ok: 1
}
stocks>
```

12. Create a compound index on stock_symbol and close

> db.values.createIndex({stock_symbol: 1, close: 1})

```
stocks> db.values.createIndex({stock_symbol: 1, close: 1})
stock_symbol_1_close_1
stocks>
```

13. Rerun the previous query

> db.values.find({stock_symbol: "GOOG", close: {$gt: 200}}).explain("executionStats")

Notice that the query execution is optimal as the number of returned documents "nReturned" is equal to the number of examined documents "totalDocsExamined", i.e. 730.

```
stocks> db.values.find({stock_symbol: "GOOG", close: {$gt: 200}}).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'stocks.values',
    indexFilterSet: false,
    parsedQuery: {
      '$and': [
        { stock_symbol: { '$eq': 'GOOG' } },
        { close: { '$gt': 200 } }
      ]
    },
    queryHash: 'DF39A05A',
    planCacheKey: '33A3206B',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { stock_symbol: 1, close: 1 },
        indexName: 'stock_symbol_1_close_1',
        isMultiKey: false,
        multiKeyPaths: { stock_symbol: [], close: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: {
          stock_symbol: [ '["GOOG", "GOOG"]' ],
          close: [ '(200, inf.0]' ]
        }
      }
    },
    rejectedPlans: [
      {
        stage: 'FETCH',
        filter: { stock_symbol: { '$eq': 'GOOG' } },
        inputStage: {
          stage: 'IXSCAN',
          keyPattern: { close: 1 },
          indexName: 'close_1',
          isMultiKey: false,
          multiKeyPaths: { close: [] },
          isUnique: false,
          isSparse: false,
          isPartial: false,
          indexVersion: 2,
          direction: 'forward',
          indexBounds: { close: [ '(200, inf.0]' ] }
        }
      }
    ]
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 730,
    executionTimeMillis: 13,
    totalKeysExamined: 730,
    totalDocsExamined: 730,
    executionStages: {
      stage: 'FETCH',
      nReturned: 730,
      executionTimeMillisEstimate: 0,
      works: 731,
```

```
      works: 731,
      advanced: 730,
      needTime: 0,
      needYield: 0,
      saveState: 1,
      restoreState: 1,
      isEOF: 1,
      docsExamined: 730,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
        nReturned: 730,
        executionTimeMillisEstimate: 0,
        works: 731,
        advanced: 730,
        needTime: 0,
        needYield: 0,
        saveState: 1,
        restoreState: 1,
        isEOF: 1,
        keyPattern: { stock_symbol: 1, close: 1 },
        indexName: 'stock_symbol_1_close_1',
        isMultiKey: false,
        multiKeyPaths: { stock_symbol: [], close: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: {
          stock_symbol: [ '["GOOG", "GOOG"]' ],
          close: [ '(200, inf.0]' ]
        },
        keysExamined: 730,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
  command: {
    find: 'values',
    filter: { stock_symbol: 'GOOG', close: { '$gt': 200 } },
    '$db': 'stocks'
  },
  serverInfo: {
    host: 'DESKTOP-BCGD7BA',
    port: 27017,
    version: '7.0.9',
    gitVersion: '3ff3a3925c36ed277cf5eafca5495f2e3728dd67'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'trySbeRestricted'
  },
  ok: 1
}
stocks>
```

NEXT:
We will put relational databases to shame and will explore the aggregation pipeline. Familiarize yourself with aggregation operators (see reference docs at https://docs.mongodb.com/manual/aggregation/).