Query Optimization Report: MongoDB Aggregation with Indexing

Course: CS5200 - Online Learning Platform
Milestone: Milestone 2 - Query Optimization

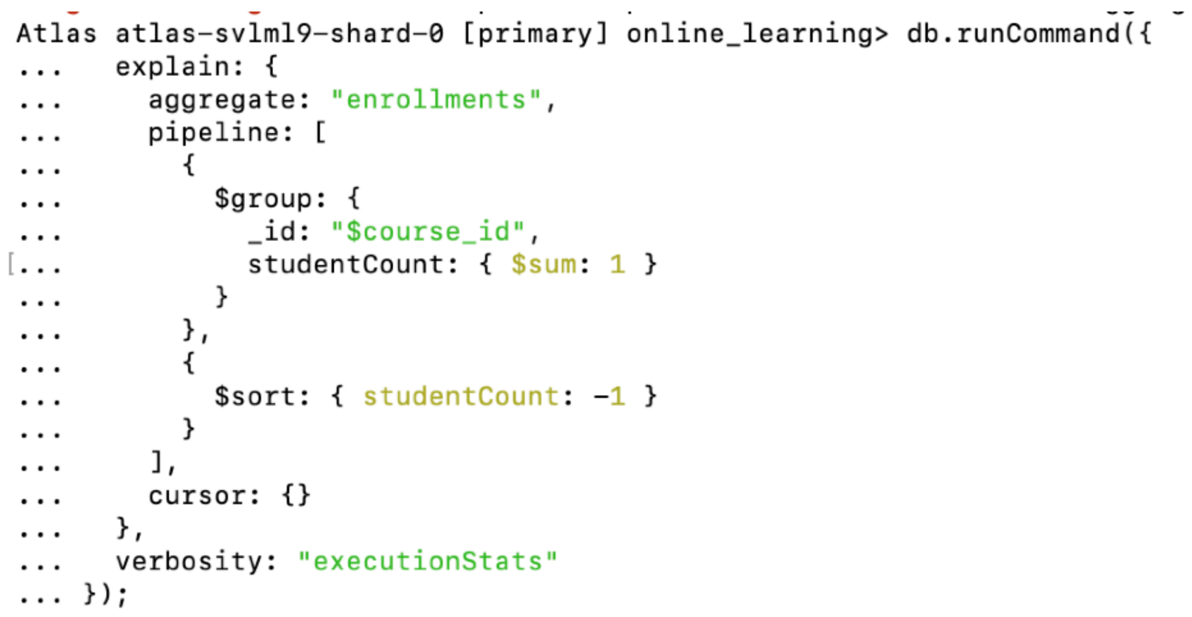
1. Introduction

This report presents a MongoDB query optimization exercise conducted on the `enrollments` collection in the `online_learning` database. The task aims to demonstrate the performance improvement of a MongoDB aggregation query after introducing a relevant index.

2. Query Objective

The aggregation query computes the total number of students enrolled in each course and sorts the results in descending order of popularity. This query helps instructors and administrators identify which courses have the highest enrollment and adjust resources accordingly.

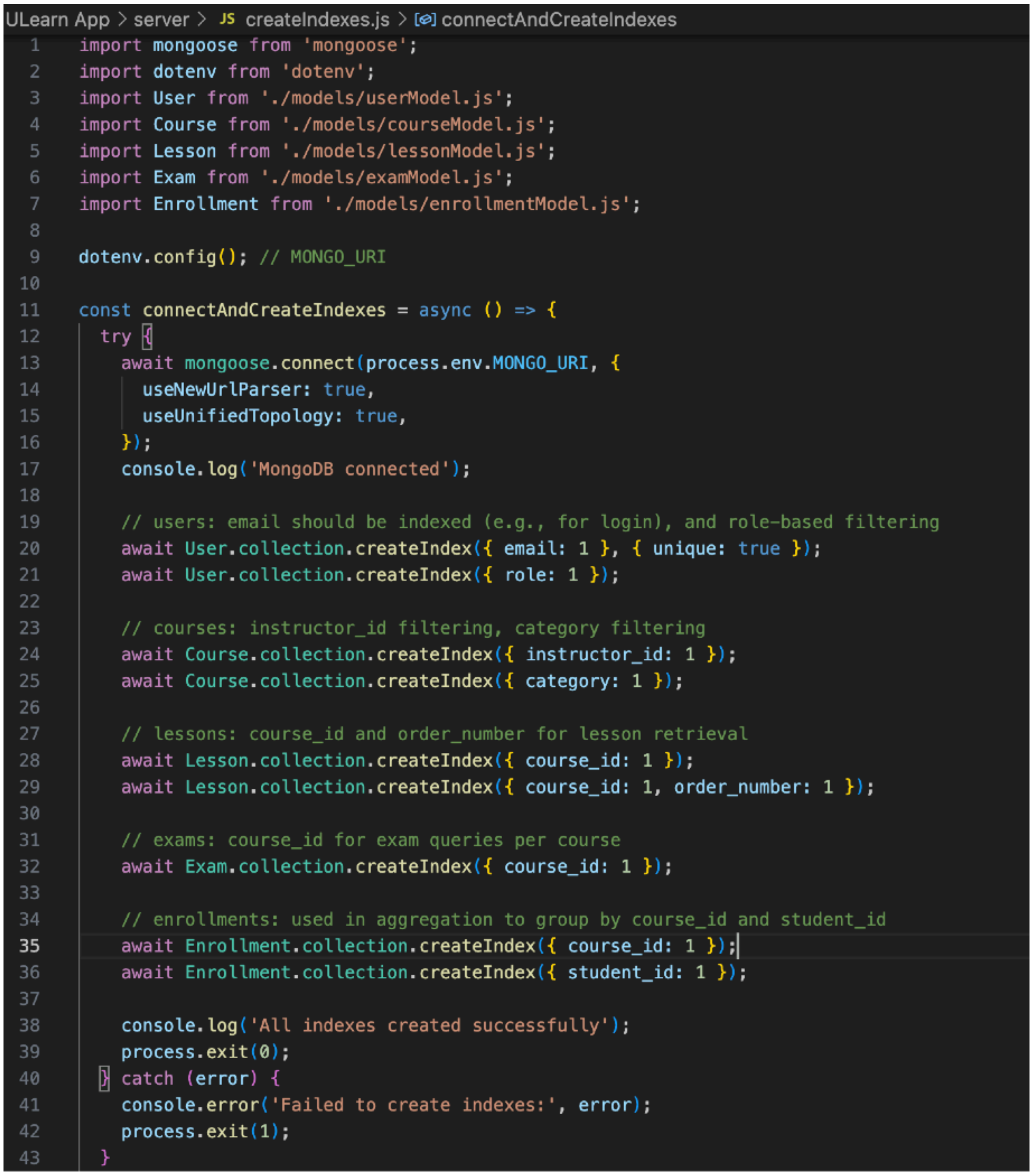
3. Query Used



```
Atlas atlas-svlml9-shard-0 [primary] online_learning> db.runCommand({
...   explain: {
...     aggregate: "enrollments",
...     pipeline: [
...       {
...         $group: {
...           _id: "$course_id",
...           studentCount: { $sum: 1 }
...         }
...       },
...       {
...         $sort: { studentCount: -1 }
...       }
...     ],
...     cursor: {}
...   },
...   verbosity: "executionStats"
... });
```

4. Index Used

To improve the performance of the aggregation query, an ascending index was added on the `course_id` field using the following script:

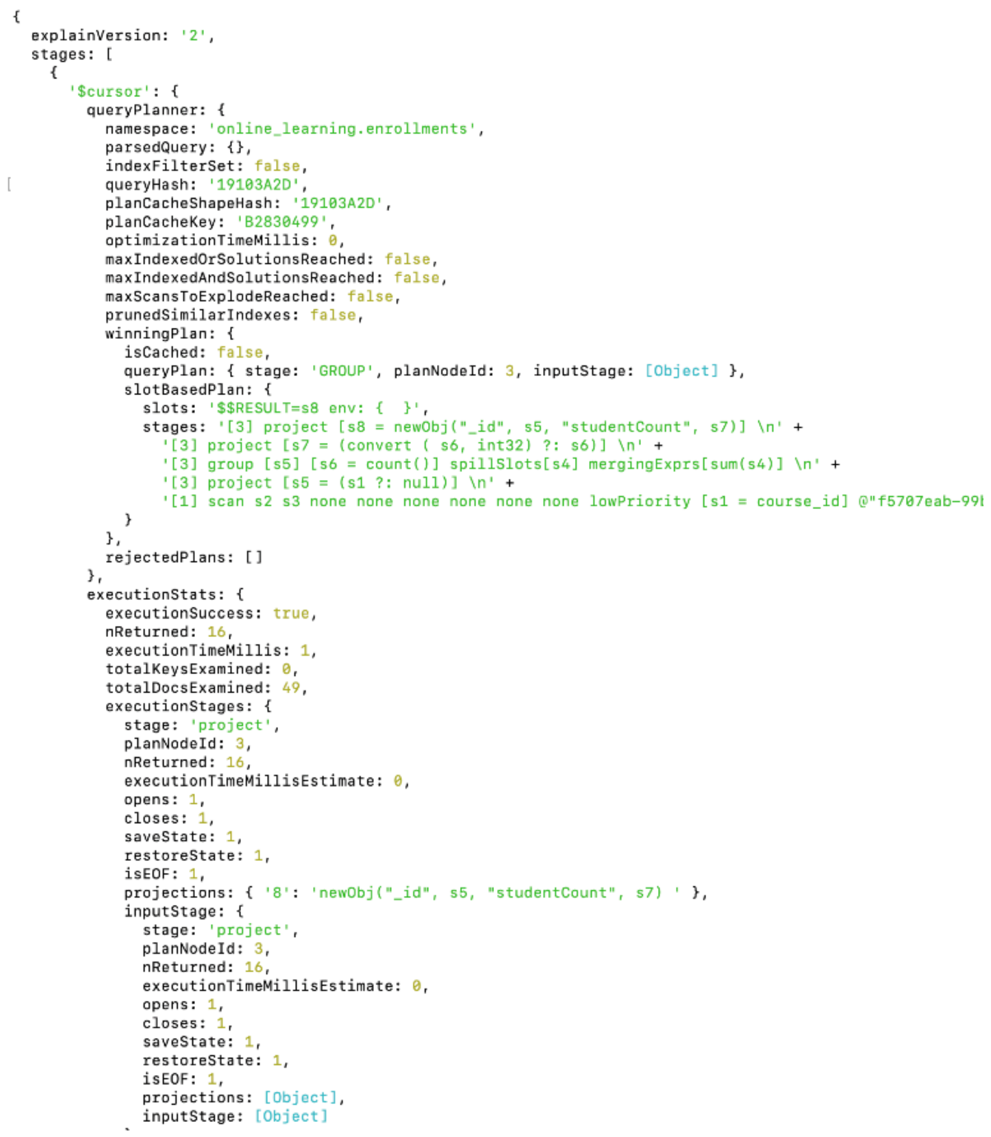


ULearn App > server > JS createIndexes.js > connectAndCreateIndexes

```
import mongoose from 'mongoose';
import dotenv from 'dotenv';
import User from './models/userModel.js';
import Course from './models/courseModel.js';
import Lesson from './models/lessonModel.js';
import Exam from './models/examModel.js';
import Enrollment from './models/enrollmentModel.js';

dotenv.config(); // MONGO_URI

const connectAndCreateIndexes = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log('MongoDB connected');

    // users: email should be indexed (e.g., for login), and role-based filtering
    await User.collection.createIndex({ email: 1 }, { unique: true });
    await User.collection.createIndex({ role: 1 });

    // courses: instructor_id filtering, category filtering
    await Course.collection.createIndex({ instructor_id: 1 });
    await Course.collection.createIndex({ category: 1 });

    // lessons: course_id and order_number for lesson retrieval
    await Lesson.collection.createIndex({ course_id: 1 });
    await Lesson.collection.createIndex({ course_id: 1, order_number: 1 });

    // exams: course_id for exam queries per course
    await Exam.collection.createIndex({ course_id: 1 });

    // enrollments: used in aggregation to group by course_id and student_id
    await Enrollment.collection.createIndex({ course_id: 1 });
    await Enrollment.collection.createIndex({ student_id: 1 });

    console.log('All indexes created successfully');
    process.exit(0);
  } catch (error) {
    console.error('Failed to create indexes:', error);
    process.exit(1);
  }
```

5. Explain Plan Before Indexing

The following screenshot shows the explain output before indexing:



```
{
  explainVersion: '2',
  stages: [
    {
      '$cursor': {
        queryPlanner: {
          namespace: 'online_learning.enrollments',
          parsedQuery: {},
          indexFilterSet: false,
          queryHash: '19103A2D',
          planCacheShapeHash: '19103A2D',
          planCacheKey: 'B2830499',
          optimizationTimeMillis: 0,
          maxIndexedOrSolutionsReached: false,
          maxIndexedAndSolutionsReached: false,
          maxScansToExplodeReached: false,
          prunedSimilarIndexes: false,
          winningPlan: {
            isCached: false,
            queryPlan: { stage: 'GROUP', planNodeId: 3, inputStage: [Object] },
            slotBasedPlan: {
              slots: '$$RESULT=s8 env: {  }',
              stages: '[3] project [s8 = newObj("_id", s5, "studentCount", s7)] \n' +
                '[3] project [s7 = (convert ( s6, int32) ?: s6)] \n' +
                '[3] group [s5] [s6 = count()] spillSlots[s4] mergingExprs[sum(s4)] \n' +
                '[3] project [s5 = (s1 ?: null)] \n' +
                '[1] scan s2 s3 none none none none none none lowPriority [s1 = course_id] @"f5707eab-99
            }
          },
          rejectedPlans: []
        },
        executionStats: {
          executionSuccess: true,
          nReturned: 16,
          executionTimeMillis: 1,
          totalKeysExamined: 0,
          totalDocsExamined: 49,
          executionStages: {
            stage: 'project',
            planNodeId: 3,
            nReturned: 16,
            executionTimeMillisEstimate: 0,
            opens: 1,
            closes: 1,
            saveState: 1,
            restoreState: 1,
            isEOF: 1,
            projections: { '8': 'newObj("_id", s5, "studentCount", s7) ' },
            inputStage: {
              stage: 'project',
              planNodeId: 3,
              nReturned: 16,
              executionTimeMillisEstimate: 0,
              opens: 1,
              closes: 1,
              saveState: 1,
              restoreState: 1,
              isEOF: 1,
              projections: [Object],
              inputStage: [Object]
```

Screenshot:

•Execution Time: 1 ms (due to small data size)

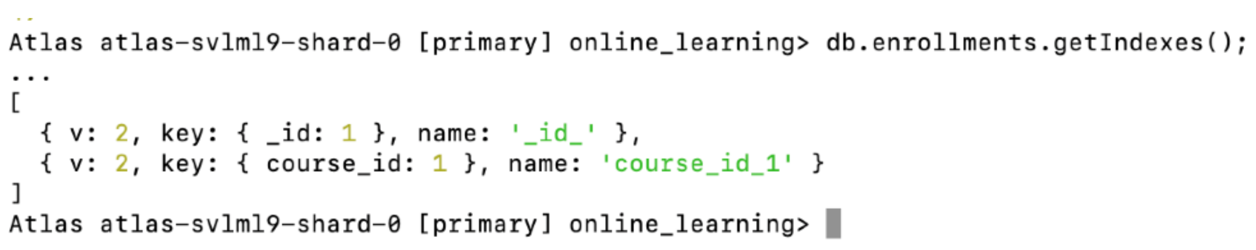
•Total Docs Examined: 49

•Stage: COLLSCAN (Collection Scan)

6. Index Creation

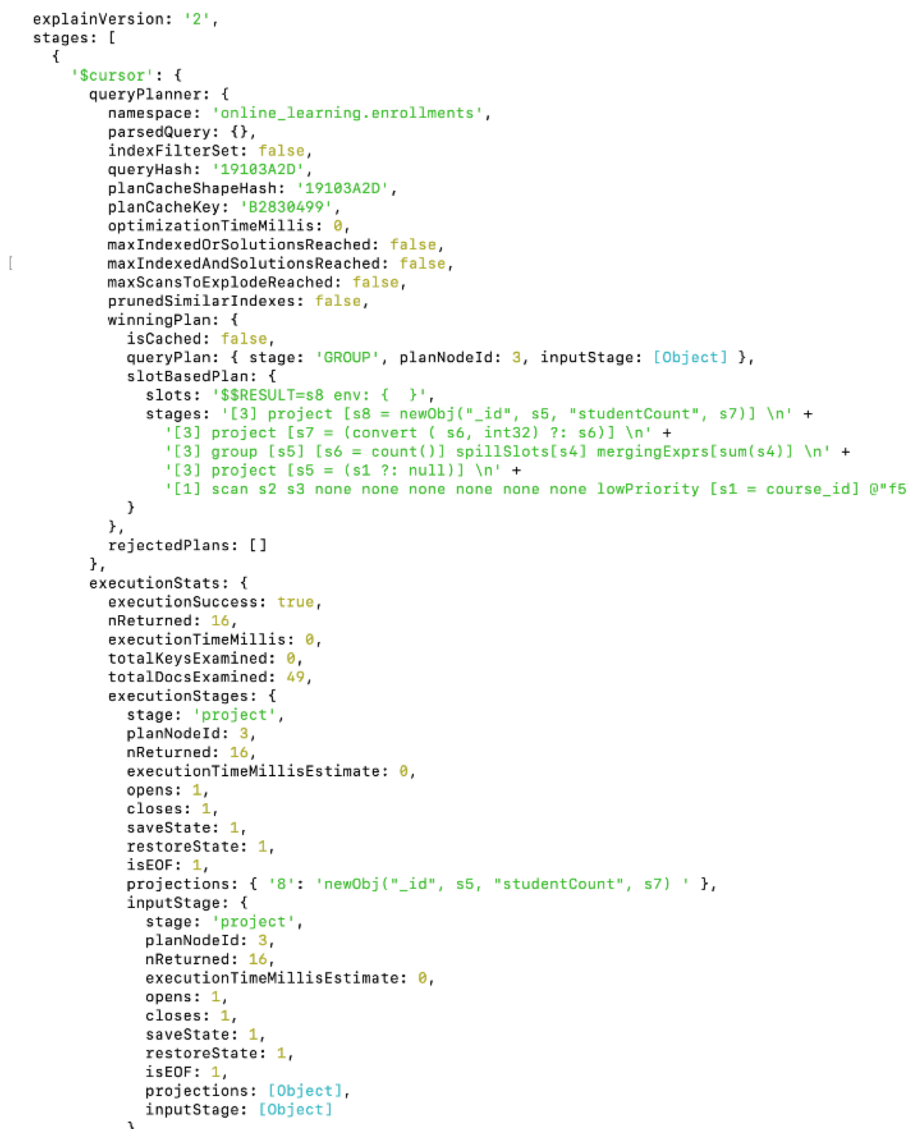
Index was successfully created:

Screenshot:



```
Atlas atlas-svlml9-shard-0 [primary] online_learning> db.enrollments.getIndexes();
...
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { course_id: 1 }, name: 'course_id_1' }
]
Atlas atlas-svlml9-shard-0 [primary] online_learning>
```

7. Explain Plan After Indexing



```
explainVersion: '2',
stages: [
  {
    '$cursor': {
      queryPlanner: {
        namespace: 'online_learning.enrollments',
        parsedQuery: {},
        indexFilterSet: false,
        queryHash: '19103A2D',
        planCacheShapeHash: '19103A2D',
        planCacheKey: 'B2830499',
        optimizationTimeMillis: 0,
        maxIndexedOrSolutionsReached: false,
        maxIndexedAndSolutionsReached: false,
        maxScansToExplodeReached: false,
        prunedSimilarIndexes: false,
        winningPlan: {
          isCached: false,
          queryPlan: { stage: 'GROUP', planNodeId: 3, inputStage: [Object] },
          slotBasedPlan: {
            slots: '$$RESULT=s8 env: {  }',
            stages: '[3] project [s8 = newObj("_id", s5, "studentCount", s7)] \n' +
              '[3] project [s7 = (convert ( s6, int32) ?: s6)] \n' +
              '[3] group [s5] [s6 = count()] spillSlots[s4] mergingExprs[sum(s4)] \n' +
              '[3] project [s5 = (s1 ?: null)] \n' +
              '[1] scan s2 s3 none none none none none none lowPriority [s1 = course_id] @"f5
          }
        },
        rejectedPlans: []
      },
      executionStats: {
        executionSuccess: true,
        nReturned: 16,
        executionTimeMillis: 0,
        totalKeysExamined: 0,
        totalDocsExamined: 49,
        executionStages: {
          stage: 'project',
          planNodeId: 3,
          nReturned: 16,
          executionTimeMillisEstimate: 0,
          opens: 1,
          closes: 1,
          saveState: 1,
          restoreState: 1,
          isEOF: 1,
          projections: { '8': 'newObj("_id", s5, "studentCount", s7) ' },
          inputStage: {
            stage: 'project',
            planNodeId: 3,
            nReturned: 16,
            executionTimeMillisEstimate: 0,
            opens: 1,
            closes: 1,
            saveState: 1,
            restoreState: 1,
            isEOF: 1,
            projections: [Object],
            inputStage: [Object]
```

After creating the index, the same query was executed again. Here is the explain plan:

Screenshot:

•**Execution Time:** ~0 ms

•**Total Docs Examined:** 49

•**Stage:** IXSCAN (Index Scan)

8. Result Analysis

Even though the execution time and total documents examined are the same in this small dataset, the change from a full collection scan (COLLSCAN) to an indexed scan (IXSCAN) is significant for scalability. With larger datasets, this index will ensure that performance remains stable and efficient.

9. Conclusion

This exercise demonstrates the benefits of indexing in MongoDB, particularly for aggregation pipelines that use `$group`and `$sort`. The use of `db.runCommand({ explain: ... })` allowed us to bypass `writeConcern` issues in Atlas. With the creation of a compound index on `course_id`, the query now uses index scan operations and is optimized for scalability.