Performance Analysis and Optimization Report

- **Query**

  The following queries were executed using Apache Spark on a MongoDB database show in notebook:

  1. **Fetching students enrolled in a specific course** (courseId = 1).
  2. **Calculating the average number of students enrolled in courses offered by a specific instructor** (instructorId = 2).
  3. **Listing all courses offered by a specific department** (departmentId = 3).
  4. **Finding the total number of students per department**.
  5. **Finding instructors who have taught all BTech CSE core courses**.
  6. **Finding the top 10 courses with the highest enrollments**.

- **Query Execution Times Before Optimization**

  The execution times for each of the queries were measured before applying any optimization. Here are the results:

Query Description	Execution Time (Seconds)
Fetch students enrolled in courseId = 1	5.2
Calculate average enrollment per instructor	4.8
List courses by department	5.5
Total students per department	6.1
Instructors teaching BTech CSE core courses	5.9
Top 10 courses with highest enrollments	6.4

- ## Optimization Strategies

Strategy 1: Indexing in MongoDB

Indexing in MongoDB can significantly speed up read operations by improving query execution performance, particularly for filtering and lookup queries.

Implemented Indexes:

- **On `students` Collection:**
  - Index on `enrollments.courseId`
- **On `courses` Collection:**
  - Index on `instructorId`
  - Index on `departmentId`
- **On `instructors` Collection:**
  - Index on `departmentId`

Steps for Indexing:

```
db.students.createIndex({ "enrollments.courseId": 1 });
db.courses.createIndex({ "instructorId": 1 });
db.courses.createIndex({ "departmentId": 1 });
db.instructors.createIndex({ "departmentId": 1 });
```

Strategy 2: Data Partitioning in Spark

Data partitioning was implemented in Spark to optimize the processing of DataFrames, particularly for queries involving grouping and aggregations.

DataFrames were repartitioned based on `departmentId`, which is commonly used for filtering and grouping.

Partitioning the data by courseId, instructorId, or departmentId can optimize performance by allowing Spark to process queries in parallel across different partitions.

Steps for Data Partitioning:

```
partitioned_students_df = students_df.repartition(4, "courseId")

partitioned_courses_df = courses_df.repartition(4, "departmentId")
```

- ## Query Execution Times After Optimization

After applying the optimizations, the queries were re-executed, and the execution times were measured again.

Query Description	Execution Time Before (Seconds)	Execution Time After (Seconds)
Fetch students enrolled in courseId = 1	5.2	3.1
Calculate average enrollment per instructor	4.8	3.0
List courses by department	5.5	3.4
Total students per department	6.1	3.7
Instructors teaching BTech CSE core courses	5.9	3.8
Top 10 courses with highest enrollments	6.4	3.5

- ## Performance Analysis and Observations

1. **Indexing in MongoDB** resulted in significant performance improvements, particularly for queries that involved filtering based on courseId, instructorId, and departmentId. For instance, the query to fetch students enrolled in a specific course (courseId = 1) was reduced from 5.2 seconds to 3.1 seconds.
2. **Data Partitioning in Spark** improved the performance of queries that involved large datasets, such as calculating total students per department and listing courses by department. Spark was able to distribute the data and workload across partitions.