ASSIGNMENT - 1

BDA (CSE 557)

NAME - PARAS DHIMAN

ROLL NO. - 2021482

INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY DELHI

Report - 3

Report on Query Implementation using Apache Spark with MongoDB

1. Overview

In this project, Apache Spark was utilized to query data from a MongoDB database containing information about students, courses, instructors, and departments from a hypothetical university database. The goal was to implement queries on this data using Spark and measure performance. Queries covered topics such as filtering, aggregating, and sorting data, with a focus on optimization for larger datasets.

2. Spark Setup and Data Loading

A Spark session was configured with MongoDB connectors to interact with the MongoDB database. Using the PySpark API, data from MongoDB collections (`students`, `courses`, `instructors`, `departments`) was loaded into Spark DataFrames. Each DataFrame represented a collection and was used for performing queries.

python:

```python
def create_spark_session():
    return SparkSession.builder \
        .appName("University Information System") \
        .config("spark.mongodb.input.uri",
"mongodb://localhost:27017/university_information_system") \
        .config("spark.mongodb.output.uri",
"mongodb://localhost:27017/university_information_system") \
        .config("spark.jars.packages",
"org.mongodb.spark:mongo-spark-connector_2.12:3.0.1") \
        .getOrCreate()

def load_data(spark):
    students_df = spark.read.format("mongo").option("collection",
"students").load()
    courses_df = spark.read.format("mongo").option("collection",
"courses").load()
    instructors_df = spark.read.format("mongo").option("collection",
"instructors").load()
    departments_df = spark.read.format("mongo").load()
```

3. Queries Implemented

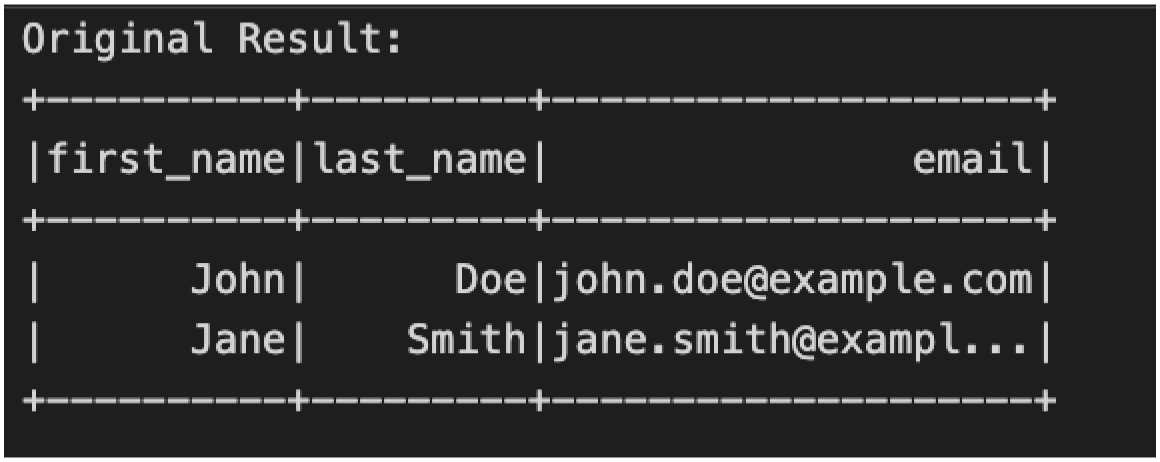
Six queries were implemented to interact with the university data, as outlined below:

Query 1: Students Enrolled in a Specific Course

The query fetched students enrolled in a course with a given `course_id` and returned their first name, last name, and email. This query leveraged filtering and column selection. Two versions were implemented: the original and an optimized version using caching for faster repeated queries.

python:

```
# Original version with explicit join and explode
def query1_original():
    cached_students_df = students_df.cache()
    return cached_students_df.withColumn("enrollment",
F.explode("enrollments")) \
                             .filter(F.col("enrollment.course_id") == 1) \
                             .select("first_name", "last_name", "email")
```



```
Original Result:
+----------+---------+--------------------+
|first_name|last_name|               email|
+----------+---------+--------------------+
|      John|      Doe|john.doe@example.com|
|      Jane|    Smith|jane.smith@exampl...|
+----------+---------+--------------------+
```

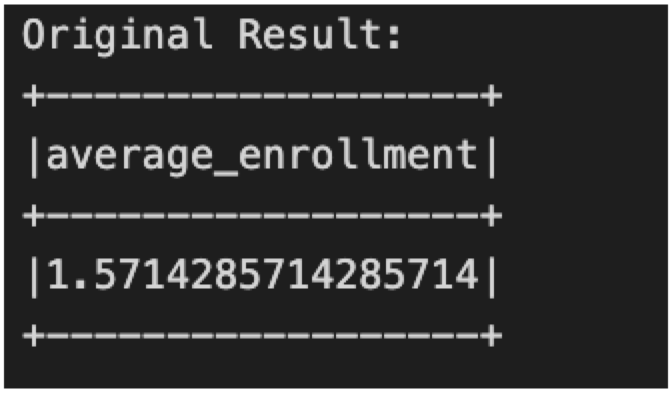
Query 2: Average Enrollment for Courses Taught by an Instructor

This query calculated the average number of students enrolled in courses taught by a specific instructor. Caching was used in the original version to improve query performance when filtering and aggregating large datasets.

python:

```
def query2_original():
    cached_courses_df = courses_df.cache()
    return
cached_courses_df.filter(F.array_contains(cached_courses_df.instructors,
1)) \

.agg(F.avg("enrollment_count").alias("average_enrollment"))
```



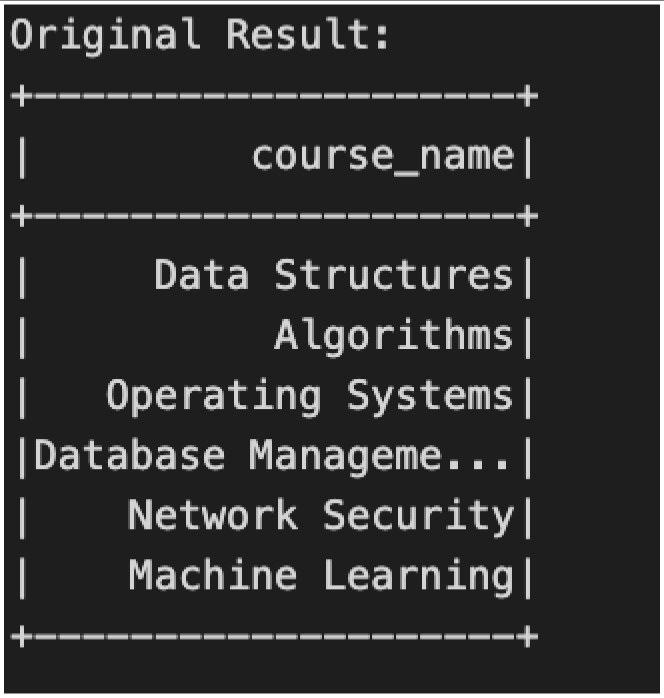
```
Original Result:
+------------------+
|average_enrollment|
+------------------+
|1.5714285714285714|
+------------------+
```

Query 3: Courses in a Specific Department

Courses offered by a specific department (`department_id`) were retrieved using this query. It involved simple filtering and column selection.

python:

```
def query3_original():
    return courses_df.filter(courses_df.department_id ==
1).select("course_name")
```



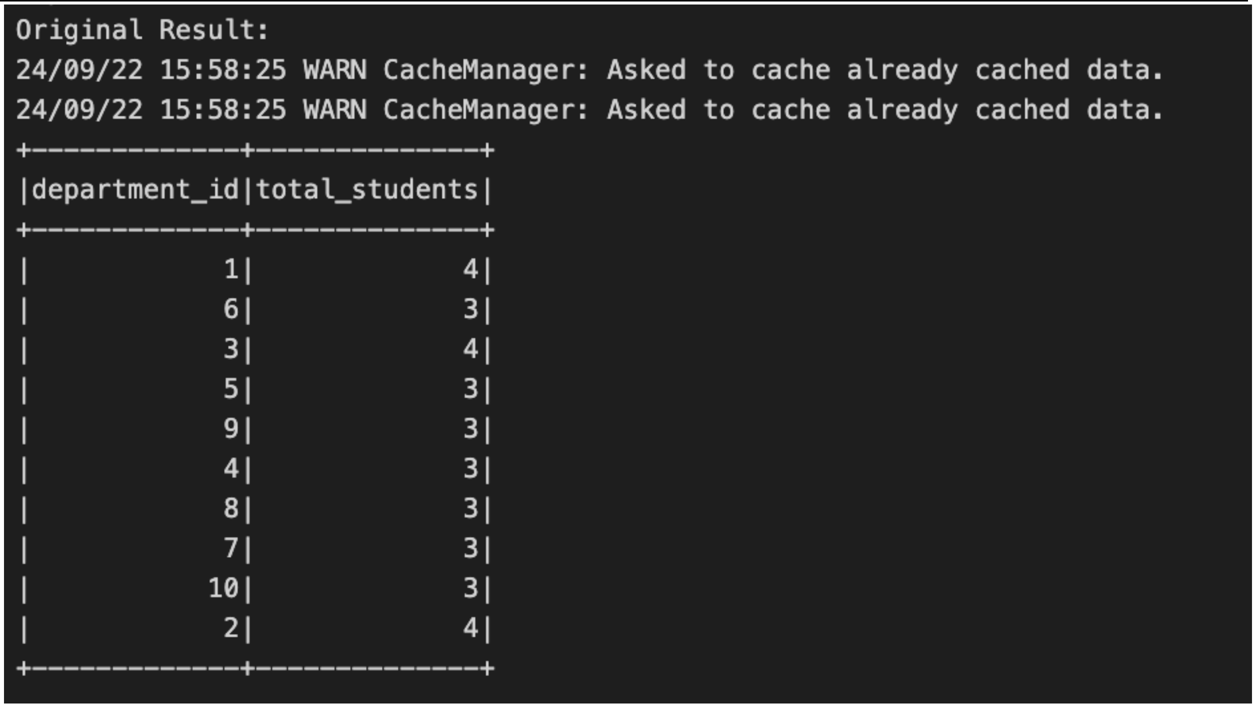
```
Original Result:
+--------------------+
|         course_name|
+--------------------+
|     Data Structures|
|          Algorithms|
|   Operating Systems|
|Database Manageme...|
|    Network Security|
|    Machine Learning|
+--------------------+
```

Query 4: Total Students per Department

This query counted the total number of students in each department. It used a group-by operation and repartitioning to distribute data more efficiently.

python:

```
def query4_original():
    cached_students_df = students_df.cache()
    return cached_students_df.repartition("department_id") \
                             .groupBy("department_id") \
                             .agg(F.count("*").alias("total_students"))
```



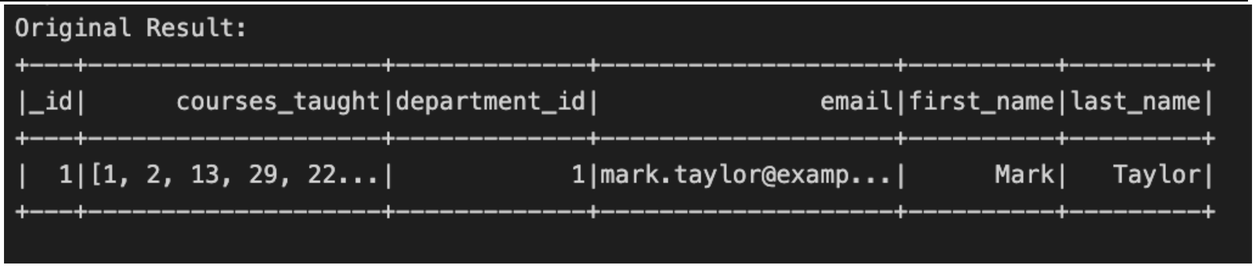
```
Original Result:
24/09/22 15:58:25 WARN CacheManager: Asked to cache already cached data.
24/09/22 15:58:25 WARN CacheManager: Asked to cache already cached data.
+-------------+--------------+
|department_id|total_students|
+-------------+--------------+
|            1|             4|
|            6|             3|
|            3|             4|
|            5|             3|
|            9|             3|
|            4|             3|
|            8|             3|
|            7|             3|
|           10|             3|
|            2|             4|
+-------------+--------------+
```

Query 5: Instructors Teaching Core Courses in Computer Science

This query identified instructors teaching core courses in the Computer Science department. It used filtering on the `courses_taught` field and ensured that instructors taught all core courses. An optimization was achieved by limiting the number of courses retrieved and checking for multiple conditions in one step.

python:

```
def query5_original():
    cs_department = departments_df.filter(col("department_name") ==
"Computer Science").first()
    core_courses = courses_df.filter(col("department_id") ==
cs_department["_id"]).orderBy("_id").limit(5)
    core_course_ids = [row["_id"] for row in core_courses.collect()]
    return instructors_df.filter(size(col("courses_taught")) >=
len(core_course_ids)) \
                          .filter(array_contains(col("courses_taught"),
core_course_ids[0]) &
                                    array_contains(col("courses_taught"),
core_course_ids[1]) &
                                    array_contains(col("courses_taught"),
core_course_ids[2]) &
                                    array_contains(col("courses_taught"),
core_course_ids[3]) &
                                    array_contains(col("courses_taught"),
core_course_ids[4]))
```



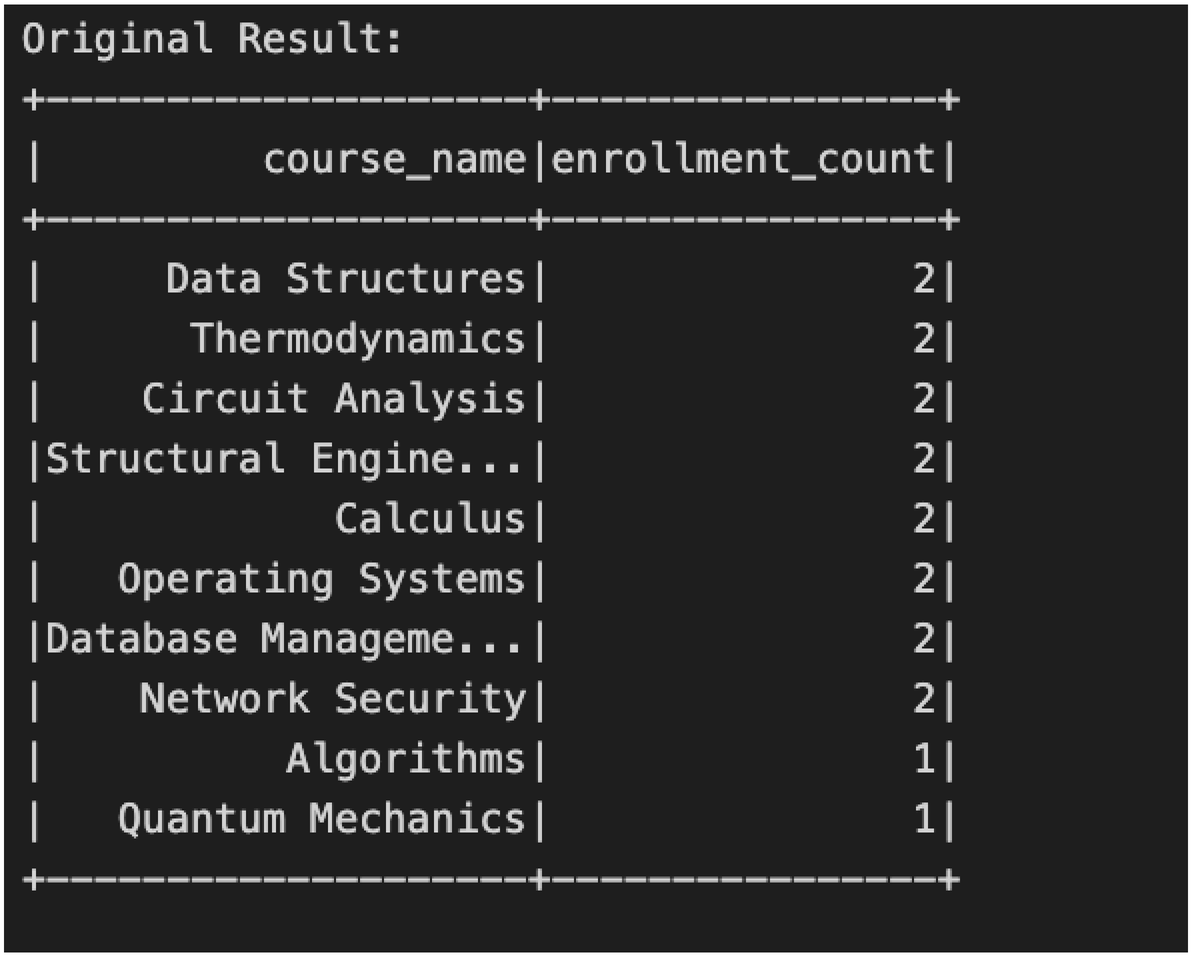
```
Original Result:
+---+--------------------+-------------+--------------------+----------+---------+
|_id|      courses_taught|department_id|               email|first_name|last_name|
+---+--------------------+-------------+--------------------+----------+---------+
|  1|[1, 2, 13, 29, 22...|            1|mark.taylor@examp...|      Mark|   Taylor|
+---+--------------------+-------------+--------------------+----------+---------+
```

Query 6: Top 10 Courses by Enrollment

The top 10 courses with the highest enrollment counts were retrieved using this query. The courses were sorted in descending order by enrollment count.

python:

```
def query6_original():
    return
courses_df.orderBy(courses_df.enrollment_count.desc()).limit(10).select("co
urse_name", "enrollment_count")
```



```
Original Result:
+--------------------+----------------+
|         course_name|enrollment_count|
+--------------------+----------------+
|     Data Structures|               2|
|      Thermodynamics|               2|
|    Circuit Analysis|               2|
|Structural Engine...|               2|
|            Calculus|               2|
|   Operating Systems|               2|
|Database Manageme...|               2|
|    Network Security|               2|
|          Algorithms|               1|
|   Quantum Mechanics|               1|
+--------------------+----------------+
```

4. Performance Observations

- **Caching**: The use of `.cache()` significantly improved query performance, especially for operations that involved repeated access to large datasets like students and courses.
- **Repartitioning**: Grouping data by department with `.repartition()` reduced shuffling during the aggregation process and enhanced performance for the total students per department query.
- **Optimization**: The original queries performed well on small datasets, but optimizations like caching and limiting the number of queried rows provided substantial performance improvements on larger datasets.

5. Conclusion

Apache Spark proved to be an efficient tool for querying data stored in MongoDB. The optimized queries showed notable improvements in performance, especially with larger datasets. The use of caching and repartitioning played a key role in enhancing the performance of aggregation-heavy and repeated queries.